UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CORNÉLIO PROCÓPIO
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

DIONY JOSÉ DE ALMEIDA

PROPOSTA DE UM SISTEMA DE SIMULAÇÃO E DIAGNÓSTICO DE FALHAS APLICADO A UM SISTEMA DE PRODUÇÃO

DISSERTAÇÃO

CORNÉLIO PROCÓPIO
2014

DIONY JOSÉ DE ALMEIDA

PROPOSTA DE UM SISTEMA DE SIMULAÇÃO E DIAGNÓSTICO DE FALHAS APLICADO A UM SISTEMA DE PRODUÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Mestre em Engenharia Elétrica”.

Orientador: Prof. Dr. Marcos Banheti Rabello Vallim

CORNÉLIO PROCÓPIO
2014

Dados Internacionais de Catalogação na Publicação

A447 Almeida, Diony José de
Proposta de um sistema de simulação e diagnóstico de falhas aplicado a um sistema de produção/ Diony José de Almeida. – 2014.
81 f. : il. ; 30 cm

Orientador: Marcos Banheti Rabello Vallim.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica. Cornélio Procópio, 2014.
Referências: p. 76-78.

1. Simulação (computadores). 2. Localização de falhas (Engenharia). 3. Controle automático. 4. Sistemas de tempo discreto. 5. Engenharia elétrica – Dissertações. I. Vallim, Marcos Banheti Rabello, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica. III. Título.

CDD (22. ed.) 621.3

Biblioteca da UTFPR, Câmpus Cornélio Procópio

Minstério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Cornélio Procópio
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Engenharia Elétrica
Mestrado em Engenharia Elétrica

TERMO DE APROVAÇÃO

Proposta de um sistema de simulação e diagnóstico de falhas aplicado a um sistema de produção

por

Diony José de Almeida

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Engenharia Elétrica" e aprovado em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná.
Cornélio Procópio, 26/08/2014.

Paulo Rogério Scalassara, Prof. Dr.
Coordenador do Curso

Banca Examinadora:

Marcos Banheti Rabello Vallim, Prof. Dr.
Orientador

André Bittencourt Leal, Prof. Dr.
UDESC - Joinville

Cristiano Marcos Agulhari, Prof. Dr.
UTFPR - Cornélio Procópio

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

AGRADECIMENTOS

O tempo passou... E uma nova pedra encontra-se pavimentada na estrada da vida, estrada esta que é construida com fé, amor, confiança, dedicação, renúncia e uma somatória do que recebemos com carinho das pessoas que nos cercam.

É momento de agradecer, prefiro não citar nomes, para que não correr o risco de esquecer de alguém...

Em primeiro lugar, agradeço à Deus, porque é Ele que torna tudo possível.

Agradeço à minha família, pela força, sorriso, abraço, carinho, compreensão e por uma infinidade de gestos que me permitiram ficar em pé.

Agradeço ao meu orientador, professores e aos demais funcionários da UTFPR que partilharam não apenas seus conhecimentos mas suas histórias e suas vidas.

Agradeço aos colegas que tornaram-se amigos e a todos os amigos que justificaram o uso desta palavra.

À todos, meu muito obrigado!

RESUMO

ALMEIDA, Diony J. **Proposta de um sistema de simulação e diagnóstico de falhas aplicado a um sistema de produção**. 2014. 82 f. Dissertação – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2014.

Este trabalho apresenta uma proposta de um software de simulação e diagnóstico de falhas aplicado a um sistema de automação. Para realizar esta tarefa foi desenvolvido, um estudo de caso que contem um problema de automação industrial. Deste estudo foi retirado um modelo, usado como base para criar do software de simulação que segue as especificações do modelos, no qual é possível observar a evolução dos eventos através de sua linguagem e diagnosticar falhas, quando estas ocorrem em uma das máquinas da planta. O processo de diagnóstico de falhas tem como base as regras de diagnosticabilidades apresentadas na literatura. As máquinas, assim como suas restrições, são modeladas dentro do próprio software, onde são definidas as estruturas de cada autômato assim como características como observabilidade, controlabilidade e a possibilidade de falhar. Os testes realizados no sistema apresentado mostraram que a linguagem de controle consegue, mesmo permitindo um nível liberdade ao sistema, identificar as falhas verificando somente para os eventos que occorrem após a ocorrência da mesma.

Palavras-chave: Sistemas a eventos discretos. Diagnóstico de falhas. Simulação. Automação. Sistema de produção

ABSTRACT

ALMEIDA, Diony J. **Proposed a simulation system and fault diagnosis applied to a system production.** 2014. 82 f. Dissertação – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2014.

This work we presents a proposal a software of simulation and fault diagnosis applied to an automation system. In that task was developed, a case study that contains a problem of industrial automation. The study was used as the source for creating the simulation that follows the specifications of software models, where you can observe the evolution of events through its language and diagnose faults when they occur in one of the machines of the plant. The process of fault diagnosis is based on the rules of diagnosticabilidades presented in the literature. The machines, as well as its restrictions are modeled within the software itself, where the structures of each automaton are defined the characteristics as observability, controllability and the possibility of failure. The tests in the system showed that of the language used can allowing identify fails, checking only for events that occorrem after the occurrence the fails.

Keywords: Discrete event systems. Fault diagnosis. Simulation. Automation. Production system

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

AC	Adiciona cobertura.
Ac	Componente acessível.
ADEF	Autômato Determinístico de Estados Finitos.
AI	Adiciona Ingredientes.
AN	Adiciona Nozes.
ANDEF	Autômato Não Determinístico de Estados Finitos.
CLP	Controlador lógico programável.
DTE	Diagrama de Transição de Estados.
EC1	Esteira Classificador 1.
EC2	Esteira Classificador 2.
ET	Esteira Transportadora.
FN	Forno.
HDD	Hard Disk Drive.
MT	Misturador.
S2DFA2	Sistema de Simulação e Diagnóstico de Falhas Aplicado à Automação.
SD-VC	Sistemas Dinâmicos com Variáveis Contínuas.
SEDs	Sistemas a Eventos Discretos.
SO	Sistema Operacional.

LISTA DE SÍMBOLOS

Σ	Alfabeto
ϵ	Cadeia vazia
s	Representa uma palavra
t	Prefixo de uma palavra
u	Subcadeia de uma palavra
v	Sufixo de uma palavra
L	Linguagem
Σ^*	Conjunto das possíveis cadeias finitas compostas com elementos de Σ
Σ^+	Conjunto das possíveis cadeias finitas e não nulas compostas com elementos de Σ
G	Autômato que modela o sistema
X	Corresponde ao conjunto de estados finitos
f	Corresponde à função de transição de estado
L_m	Todos os caminhos que começam no estado inicial e terminam em um estado marcado
$\|\|$	Representa a composição síncrona entre dois autômatos
Σ_o	Conjunto dos eventos observáveis.
Σ_{uo}	Conjunto de eventos não observáveis.
P_o	Projeção
P_o^{-1}	Projeção inversa
G_C	Autômato que modela a composição do comportamento normal do sistema
G_{N_k}	Autômato que modela tendo como referência o conjunto de eventos de falha
Σ_{f_k}	Conjunto de eventos de falha
F_k	Novo estado que pode ser atingido por uma falha pertencente ao conjunto Σ_{f_k}

SUMÁRIO

1 INTRODUÇÃO

A evolução de uma forma geral faz com que as pessoas fiquem cada vez mais dependentes da tecnologia. Ela está presente em quase tudo e, como tudo que é feito pelo homem, também está sujeita a falhas. No mundo acadêmico há uma grande preocupação na detecção e diagnóstico de falhas, para que estas não ocorram em equipamentos que controlam eventos críticos de suma importância, pois alguns são vitais (LEAL; PINHO; ALMEIDA, 2006).

As falhas estão ligadas à confiabilidade e têm como característica o término da capacidade de um item de realizar suas funções (LEAL; PINHO; ALMEIDA, 2006 apud RAUSAND; OIEN, 1996). Logo, as falhas representam um problema de confiabilidade e evitá-las é algo fundamental nas indústrias.

A busca por um sistema de produção que atue de forma constante não é mais uma vantagem competitiva, mas uma necessidade (JR; MACHADO; SOUZA, 2010). Para que isso ocorra, as indústrias necessitam de um método que possa determinar quando um sistema caminha para um estado crítico por meio da identificação de falhas. Hammouri P. Kabore (2002) afirma que diagnosticar falhas consiste em fornecer dados referentes ao tempo e ao lugar onde ocorreram as falhas em um processo supervisionado. Assim, métodos para o diagnóstico de falha são ferramentas que permitem a identificação da falha assim que esta ocorra, isso permite que medidas sejam tomadas em tempo hábil, fato que possibilita minimizar prejuízos, aumentar a confiabilidade e evitar acidentes no processo industrial, atuando constantemente e evitando paradas não programadas.

Um dos caminhos usados na solução deste problema é o uso de *softwares* de controle ou simulação, que permitam criar ambientes para simular ou testar as linguagens de controle, avaliando a possibilidade de corrência de falhas. Já existem no mercado *softwares* como o ITS-PLC ou AUTOGEN, que são voltados para o desenvolvimento de linguagens de controle para o CLP (Controlador lógico programável). Estes *softwares*, mesmo tendo um cunho didático não abordam as falhas através de suas características fundamentais de não observabilidade e não controlabilidade, deixando o evento de falha limitado ao acionar de um botão. Este tipo de tratamento acaba comprometendo o diagnóstico de falhas.

Se as falhas forem tratadas da forma correta o diagnóstico pode ser obtido através de uma comparação entre o modelo que representa o comportamento normal e o comportamento de falha de um sistema (SAYED-MOUCHAWEH; BILLAUDEL, 2012 apud DUDA; STORK; HART, 2000).

Geralmente, duas abordagens são usadas para construir o modelo do comporta-

mento do sistema:

(A) **Abordagem baseada em modelo de raciocínio:**

Fundamentada no conhecimento de especialistas, onde são usados dados históricos, reconhecimento de padrões e métodos de processamento de sinais. Esta técnica é usada quando não há um conhecimento completo do sistema.

(B) **Abordagem baseada em modelo:**

Fundamentada em modelos matemáticos ou analíticos sobre o comportamento do sistema, sendo necessário um profundo conhecimento do sistema para definir os modelos.

Determinar uma ferramenta inteligente que possa identificar defeitos através de padrões não é algo fácil, mas é o caminho mais provável na busca de uma solução para as falhas em sistemas (NEVES et al., 2007).

Dentro desta realidade, o diagnóstico de falhas tendo com base a abordagem baseada em modelos é uma proposta em crescimento, tendo como base de pesquisa os Sistemas a Eventos Discretos (SEDs) e utilizando-se de modelos matemáticos para definir o sistema (SAYED-MOUCHAWEH; BILLAUDEL, 2012 apud DUDA; STORK; HART, 2000).

Cassandras e Lafortune (2008) define SEDs como um sistema baseado em eventos de estados discreto, assim a evolução dos estados depende diretamente de eventos discretos, sendo estes independentes do tempo. Apresentam uma diversidade maior do que outras técnicas, uma vez que os SEDs também podem ser aplicados a Sistemas Dinâmicos com Variáveis Contínuas (SD-VC) , dependendo do grau de abstração (BASILIO; CARVALHO; MOREIRA, 2010). Um fator que contribue para o aumento das pesquisas tendo como foco SEDs, esta ligado a sua característica fundamental, ou seja, seu comportamento ser definido por uma sequência de eventos que evolui (CAO; YING, 2006).

1.1 OBJETIVOS

Nesta seção busca deixar claro os objetivos do trabalho gerais e específicos. Um detalhamento destes objetivos é dado a seguir.

1.1.1 Objetivo geral

O objetivo deste trabalho é contribuir para o estudo de falhas em sistemas de automação. Para tal, é proposto um problema real de automação e um software de simulação que permita o diagnóstico de falhas. Este software deve atuar sobre o modelo e permitir

que os usuários criem as representações de cada máquina através de autômatos. Assim, o usuário pode especificar um modelo e avaliar se o modelo proposto é eficiente tanto no controle quanto no diagnóstico de falhas.

1.1.2 Objetivos específicos

Os objetivos específicos são:

- Realizar um estudo comparativo entre abordagens de SEDs que buscam diagnosticar falhas, analisando os algoritmos propostos;
- Propor um problema real de automação que siga as características de uma indústria, contendo os componentes necessários para a automação;
- Modelar o problema com base na teoria dos SEDs, analisando cada componente que compõe o sistema;
- Construir um software que permita a criação dos autômatos, mostrando a evolução dos estados através de uma simulação;
- Avaliar se os modelos de controle propostos permitem o diagnóstico de falhas.

1.2 JUSTIFICATIVA

Falhas representam um custo elevado em todos os setores, e estima-se que 60 bilhões de dólares são gasto anualmente com falhas de estruturas e máquinas em todo o mundo (INMAN et al., 2005 apud SILVA; DIAS JÚNIOR; LOPES JUNIOR, 2006). Já quando relacionamos falhas com prejuízos ao meio ambiente, a situação agrava-se, como no exemplo ocorrido com o navio Vergina II, onde uma falha no sistema de estabilização gerou um erro manobra que provocou o vazamento de 86 mil litros de petróleo em quatro de novembro de 2000, gerando uma mancha que atingiu 21 praias do litoral norte de São Paulo (GABARDO et al., 2003). Neste caso, além do prejuízo direto ao meio ambiente, temos também o prejuízo financeiro indireto uma vez que praias interditadas não viabilizam o turismo e o comércio local.

Dentro de um grande sistema, como um processo industrial, a enorme quantidade de sensores e informações trafegadas exige que um sistema secundário seja acoplado ao principal para fazer uma monitoração (BINH; TUYEN, 2006) e determinar se há falhas no sistema principal, pois a capacidade humana fica limitada diante de um número tão significativo de informações.

Reduzir custos, minimizar acidentes, evitar desperdícios, aumentar a rapidez e a eficiência de projetos são alguns dos pontos que tornam a simulação uma ferramenta útil na gestão de ambientes industriais. Setores como: desenvolvimento, teste de produtos, modelagem de peças e componentes são alguns exemplos que absorveram essa tecnologia. A introdução de falhas aleatórias propicia um incremento substancial na qualidade dos *softwares* de simulação destinados à detecção de falhas.

1.3 ESTRUTURA DO DOCUMENTO

Este documento se estrutura da seguinte forma:

Capítulo 2 apresenta parte da base teórica necessária para a compreensão da proposta.

O capítulo 3 contem algumas técnicas apresentadas na literatura, sendo que estas servirão de base para o trabalho.

O capítulo 4 apresenta um estudo de caso de automação, detalha um exemplo de modelagem com base no formalismo dos SEDs para o modelo apresentado, sendo que esta modelagem é usada no capítulo 5 para ilustrar o funcionamento da ferramenta desenvolvida.

O capítulo 5 o desenvolvimento de uma ferramenta de simulação de SEDs que foi construído com base no modelo proposto no capítulo 4 e com ela são realizados os testes sobre o exemplo verificando o controle e o diagnóstico de falhas.

O capítulo 6 contém a análise dos resultados.

O capítulo 7 a conclusão e sugestões para trabalhos futuros.

2 FORMALISMO E DEFINIÇÕES

Nesta seção são apresentados conceitos necessários para a compreensão do trabalho, tais como SEDs, Autômatos, falhas e simulação.

2.1 SISTEMAS A EVENTOS DISCRETOS

Antes da definição de Sistemas a Eventos Discretos é necessário definir eventos e estados. Segundo Cassandras e Lafortune (2008) os eventos devem ser pensados como uma ações que ocorrem e altera as condições de um sistema, máquina, componente, objeto, etc. levando a uma mudança em suas condições iniciais. Já o estado de uma sistema descreve as características ou comportamento deste sistema em um tempo t. Na teoria dos sistemas, o termo estado assume uma função mais importante no processo de modelagem de muitas técnicas analíticas.

Os Sistema a Eventos Discretos (SED) tem como característica a dinamicidade e sua evolução é regida por eventos físicos que não dependem tempo (CURY, 2001). Neste contexto podemos seguir a abordagem sugerida por Wonham (2012) que classifica SEDs como uma área de pesquisa que busca descobrir princípios gerais comuns a uma ampla gama de domínios de aplicação. Assim, encontramos sistemas a eventos discretos em uma grande quantidade de áreas. Os SEDs estão presentes nas indústrias, nos serviços prestados ao público, nos processos burocráticos, nos softwares, nos bancos de dados e em outras áreas (MORAES; CASTRUCCI, 2007).

A figura 1 apresenta a evolução de um SEDs através dos eventos representados pelas letras gregas $\{ \alpha, \beta, \gamma, \alpha \}$, sendo possível observar que o tempo não é o fator principal e sim os eventos. Também é possível observar que um mesmo evento pode levar o sistema a estado diferentes como é visto na ocorrência de α em t_1 que leva o sistema para o estado X_4 e mesmo evento α quando ocorre em t_4 leva o sistema para o estado X_1 (CURY, 2001).

2.1.1 Modelagem de SEDs

Muitas vezes devido à complexidade, ou mesmo à impossibilidade, um fenômeno não é estudado diretamente, mas sim através de um modelo. Um modelo é uma representação, frequentemente em termos matemáticos, do que parecem ser as principais características do objeto ou sistema sob estudo, sendo que o grau de detalhamento pode variar dependendo dos objetivos (MOURELLE, 2014).

Na modelagem de SEDs, vários paradigmas podem ser considerados. Entretanto, ainda não há um aceito como universal de forma a solucionar todos os problemas referen-

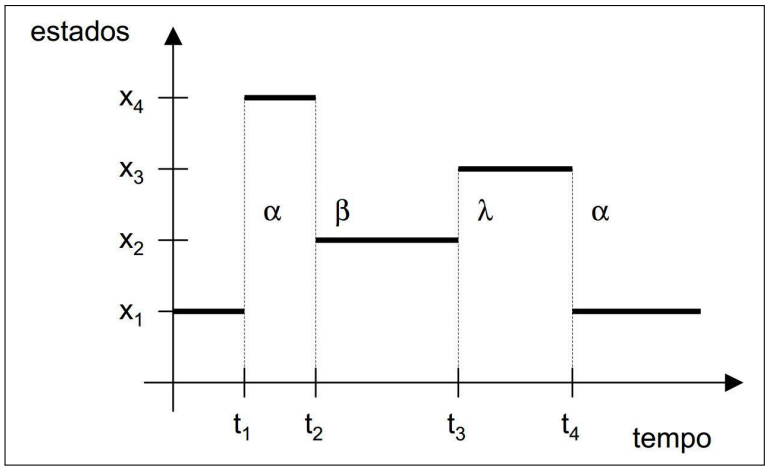

Figura 1 – Trajetória típica de um sistema a eventos discretos
Fonte: (CURY, 2001)

tes aos SEDs. Segundo Montgomery (2004), os paradigmas mais utilizados na representação de SEDs são:

- cadeias de Markov,
- teorias das filas,
- processos semi-markovianos generalizados,
- álgebra de processos,
- álgebra de dióides - E suas formalizações especificas como: álgebra Max-Plus, álgebra Min-Plus, álgebra de caminhos,
- redes de Petri e
- teoria de linguagem formais e autômatos.

Este trabalho tem como base a Teoria de linguagem formais e autômatos. Alguns conceitos necessários para a compreensão do trabalho são apresentados a seguir:

2.1.2 Linguagens

Para uma melhor compreensão do funcionamento dos sistemas, há necessidade de se conhecer as Linguagens Formais (CASSANDRAS; LAFORTUNE, 2008), que proporcionam uma maneira de representar as sequências de eventos por meio de símbolos concatenados. Como o uso do termo linguagem é algo do cotidiano do setor da informática e da computação faz-se necessário expandir os limites desta visão, primeiramente introduzindo conceitos como alfabeto, palavra, linguagem e outros.

2.1.2.1 Diagrama de transição de estados

Através do Diagrama de Transição de Estados (DTE) é possível visualizar todos os estados e eventos de um sistema.

2.1.2.2 Alfabeto

Alfabeto é uma entidade abstrata básica. Letras e dígitos são exemplos de símbolos frequentemente usados. Assim, um alfabeto é um conjunto não vazio e finito de símbolos ou caracteres. Representado geralmente pela letra grega Σ (MENEZES, 2011). São exemplos de alfabeto:

- $\Sigma = \{\alpha, \beta, \gamma, \delta\}$
- $\Sigma = \{a, b, c, d\}$
- $\Sigma = \{a_1, a_2, a_3, a_4\}$

2.1.2.3 Palavra

Uma palavra representa uma cadeia de caracteres ou sentença sobre um alfabeto, é uma sequência finita de símbolos do alfabeto justapostos. Assim tendo como referência o alfabeto apresentado acima $\Sigma = \{a, b, c, d\}$, Menezes (2011) apresenta exemplos de Palavra, Prefixo, Sufixo e Subpalavra:

- Palavra
  $abcb$ Representa uma das palavras do alfabeto $\Sigma = \{a, b, c\}$
- Prefixo
  $\{\epsilon, a, ab, abc, abcb\}$ são todos os prefixos da palavra $abcb$
- Sufixo
  $\{\epsilon, b, cb, bcb, abcb\}$ são todos os sufixos da palavra $\{abcb\}$
- Subpalavra
  Qualquer sequência de símbolos contido na palavra

Uma palavra contem prefixo, sufixo, onde prefixo corresponde a sequência inicial de uma palavra, sufixo a sequência final de uma palavra. A a cadeia vazia que é a ausência de elementos é representada por ϵ .

Se Σ^* contem todas as palavras possíveis de serem obtidas com os simbolos de um alfabeto Σ entre elas s que representa uma palavra e sendo $s = tuv$ podemos também dividir t, u, v em palavras de Σ^* assim pode se dizer que:

- t é chamado de **prefixo** de s
- u é chamado de **subcadeia** de s
- v é chamado de **sufixo** de s

O comprimento de uma palavra s, representada por $|s|$, é o número de símbolos que a compõe (CUNHA, 2003). A palavra vazia, denotada pela letra grega ϵ, é a palavra de comprimento nulo.

2.1.2.4 Linguagens formais

Uma linguagem L definida sobre um alfabeto Σ é um conjunto de cadeias formadas por símbolos pertencentes a Σ (CURY, 2001). Assim, tomando como exemplo o alfabeto já citado em 2.1.2.2 teremos como exemplos de linguagem:

$L_1 = \{\alpha, \alpha\delta\beta\gamma, \beta\delta\alpha\gamma\beta\beta, \beta\delta\beta\alpha\beta\gamma\beta, \beta\beta\gamma\beta\}$

$L_2 = \{\alpha\alpha, \alpha\beta\alpha, \alpha\beta\beta\alpha, \alpha\beta\alpha\beta, \beta\beta\alpha\}$

Também é definido que o conjunto de todas as possíveis cadeias finitas compostas com elementos de Σ é representado por Σ^*, incluindo ϵ. Assim, uma linguagem formada por palavras de Σ é sempre um subconjunto de Σ^*. Em particular $\emptyset$, Σ e Σ^* são linguagens. Outro símbolo que representa um conjunto de linguagens é o Σ^+ que representa o conjunto das palavras de comprimento finito e não nulo tendo com referência o alfabeto Σ. Logo $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$ (CUNHA, 2003).

2.1.2.5 Concatenação

Para definição da concatenação partimos de duas linguagens $L_1, L_2 \subseteq \Sigma^*$, então a concatenação de L_1 e L_2, simbolizada por L_1L_2, é definida por:

$$L_1L_2 := \{s \in \Sigma^* : (s = s_1s_2) \text{ e } (s_1 \in L_1) \text{ e } (s_2 \in L_2)\}$$

Desta forma se uma cadeia está em L_1L_2 somente se e,a poder ser escrita como a concatenação de cadeia de L_1 com uma cadeia de L_2 (CURY, 2001).

2.1.2.6 Prefixo fechamento

Seja uma linguagem $L \in \Sigma^*$, então o prefixo-fechamento de L, denotado por $\overline{L}$, é definido por Cury (2001) como:

- $\overline{L} := \{s \in \Sigma^* : \exists t \in \Sigma^* (st \in L)\}$

Em que, $\overline{L}$ consiste de todas as cadeias de Σ^* que são prefixos de L. Em geral, $L \subseteq \overline{L}$.L é dita prefixo-fechada se $L = \overline{L}$. Uma linguagem L é prefixo-fechada se qualquer prefixo de qualquer cadeia de L é também uma cadeia de L.

2.1.2.7 Fechamento-Kleene

Para definição da Fechamento-Kleene temos: seja $L \subseteq \Sigma^*$, então o fechamento-Kleene de L, simbolizado por L^*, é definido por:

$$L^* := \{\epsilon\} \cup L \cup LL \cup LLL \cup ...$$

Assim temos uma cadeia de L^* formada pela concatenação de um número finitos de cadeias de L, incluindo a cadeia vazia ϵ (CURY, 2001).

2.1.3 Autômatos

Os autômatos são apresentados como um dispositivo com a capacidade de reconhecer uma linguagem, podendo também ser usados para representar certos tipos de sistemas dinâmicos, Autômatos são formados por estados que representam as condições do dispositivo ou do sistema dinâmico (CASSANDRAS; LAFORTUNE, 2008). Quanto aos estados podem ser finitos ou infinitos: sendo chamados de autômatos finitos ou máquina de estados finitos e autômatos infinitos ou máquina de estados infinitos.

Através de uma sequência de símbolos denominada palavra, o autômato transita entre os estados. Assim, o autômato pode ser visto como uma entidade de controle que tem internamente variáveis que representam o estado do autômato. Podem ser definidos em dois tipos:

Autômatos determinísticos quando a função de transição leva a um único estado para cada símbolo,

Um autômato autômato determinístico de estados finitos(ADEF) é uma quintupla $G = (X, \Sigma, f, x_0, X_m)$
em que :

- G Autômato que modela o sistema
- $X = \{x_0; ...x_n\}$ corresponde ao conjunto de estados finitos
- $\Sigma = \{\sigma_1; ...\sigma_m\}$ corresponde ao alfabeto ou conjunto de símbolos

- $f : X \times \Sigma \rightarrow X$ corresponde à função de transição de estado em que:

  $f(x;\epsilon) = x$

  $f(x;\sigma) = \gamma$; para $x, x\epsilon X$ e $\sigma\epsilon\Sigma$;

- $x_0 \ \epsilon \ X$ corresponde ao estado inicial

- $X_m \subseteq X$ corresponde ao conjunto dos estados marcados, onde um estado marcado representa um estado final ou a conclusão de uma tarefa.

Um exemplo de autômato pode ser visto na figura 2 que representa o autômato .

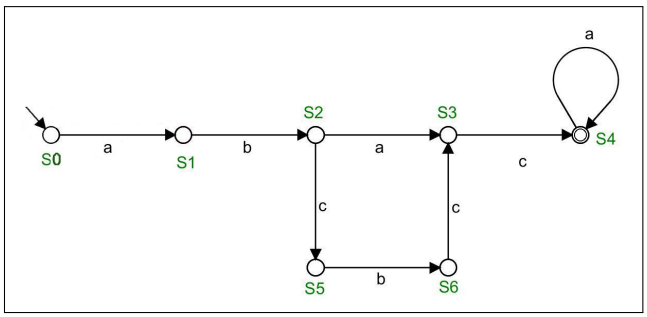

Figura 2 – Exemplo de um ADEF
Fonte: Autoria própria

Autômatos não determinísticos quando a função de transição pode levar a mais de um estado quando um mesmo símbolo é lido.

Um autômato autômato não determinístico de estados finitos(ANDEF) é uma quintupla $G = (X, \Sigma, f, x_0, X_m)$, onde X, Σ, x_0, X_m são os mesmo definido acima mas f que corresponde a função de transferência é definida como:

$f : Xx\Sigma \rightarrow 2^x$

O 2^x corresponde ao conjunto de subconjuntos de X , que também é chamado conjunto das partes de X.

Um ANDEF corresponde a um ADEF reconhecendo assim a sua linguagem. Um exemplo de ANDEF pode ser visto na figura 3 onde a ocorrência do evento **b** pode levar o sistema a mais de um estado.

2.1.3.1 Linguagem gerada e linguagem marcada

Há basicamente dois tipos de linguagens que são associadas ao comportamento de autômatos (BASILIO; CARVALHO; MOREIRA, 2010):

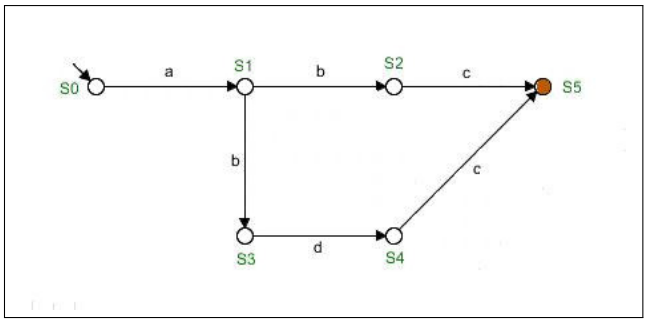

Figura 3 – Exemplo de um ANDEF
Fonte: Autoria própria

Linguagem gerada: definida por L, representa todos os caminhos que podem ser seguidos no diagrama de transições de estado, partindo do estado inicial.

Linguagem marcada identificada como L_m é um sub-conjunto da linguagem gerada L, esta linguagem contem todos os caminhos que começam no estado inicial e terminam em um estado marcado no diagrama de transições de estado.

2.1.3.2 Controlabilidade

Segundo Cury (2001) o conceito de controlabilidade de linguagens pode ser definido tendo como base um autômato G, que tenha comportamento (L(G); L_m(G)) e estrutura de controle Γ, definidos sobre o conjunto de eventos Σ e a linguagem $E \subseteq L(G)$, Onde E é controlável com relação a G, ou simplesmente controlável, se:

$$\overline{E}\Sigma_u \cap L \subseteq \overline{E}$$

2.1.3.3 Acessibilidade e co-acessibilidade

Um autômato pode ter estados inacessíveis, isto é, estados que jamais podem ser alcançados a partir do estado inicial (CURY, 2001). Desta forma um estado $x \in X$ é dito **acessível** se $x = f(x_0, u)$ onde $u \in \Sigma^*$ e o autômato G é dito ser acessível se x é acessível para todo $x \in X$.

Cury (2001) afirma que G é dito ser co-acessível, ou não bloqueante, se e somente se satisfaz as condições da equação (L(G) = $\overline{L_m(G)}$). Assim, cada cadeia ($u \in L(G)$) pude ser completada por algum $w \in \Sigma^*$ tal que $uw \in L_m(G)$, ou seja, se cada cadeia $u \in L(G)$ e um prefixo de uma cadeia em $L_m(G)$.

A componente acessível de um autômato G é definida por G_{ac} pode ser o obtida através eliminação dos estados não acessíveis e das transições associadas a eles.

$G_{ac} = (X_{ac}, \Sigma, f_{ac}, x_0, X_{m_{ac}})$

X_{ac} É o conjunto de estados acessíveis de G

$f_{ac} : f\bar{\sigma}xX_{ac}$

$X_{m_{ac}} : X_{ac} \cap X_m$

Para autômatos acessíveis $G = G_{ac}$ (CURY, 2001).

Quando um autômato é acessível e co-acessível é denominado como $trim$.

2.1.3.4 Composição síncrona

Segundo Cury (2001), a composição síncrona entre dois autômatos representada por $\|$ pode ser mostrada tomando com exemplo o autômato $G_1 = \{X_1, \Sigma_1, f_1, x_{0_1}, X_{m_1}\}$ e o autômato $G_2 = \{X_2, \Sigma_2, f_2, x_{0_2}, X_{m_2}\}$, onde $G_1 \| G_2$ é definido como a composição síncrona entre G_1 e G_2 é representada por:

- $G_1 \| G_2 = Ac(X_1 \text{ x } X_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, (x_{0_1}, x_{0_2}), X_{m_1} \text{ x } X_{m_2})$

  onde Ac é a componente acessível

- $f_{1\|2} = (X_1 \text{ x } X_2) \text{ x } (\Sigma_1 \cup \Sigma_2) \rightarrow (X_1 \text{ x } X_2)$

  Ou seja

$$f_{1\|2}((x_1, x_2), \sigma) := \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{se } \sigma \in \Sigma_1 \cap \Sigma_2 \text{ e } \sigma \in \Sigma_1(x_1) \cup \Sigma_2(x_2) \\ (f_1(x_1, \sigma), x_2)) & \text{se } \sigma \in \Sigma_1 \text{ e } \sigma \notin \Sigma_2 \text{ e } \sigma \in \Sigma_1(x_1) \\ (x_1, f_2(x_2, \sigma))) & \text{se } \sigma \in \Sigma_2 \text{ e } \sigma \notin \Sigma_1 \text{ e } \sigma \in \Sigma_2(x_2) \\ Indefinida & \text{Caso contrário} \end{cases}$$

Um exemplo de composição síncrona é apresentado utilizando os autômatos M1 e M2 mostrados na figura 4.

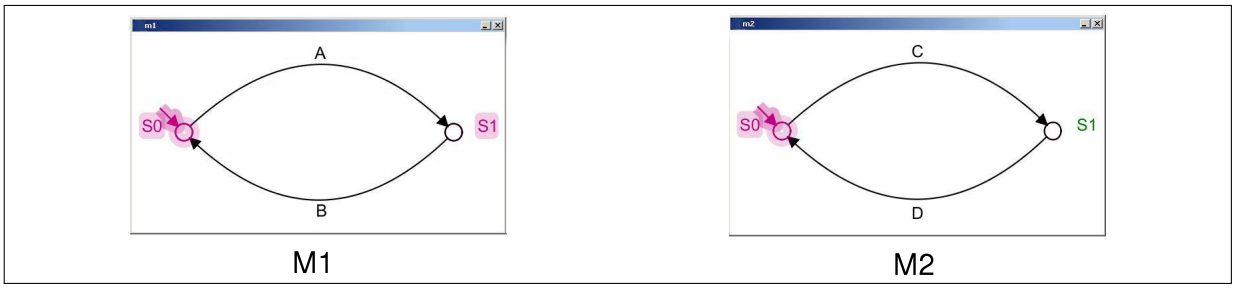

Figura 4 – Autômato M1 e M2
Fonte: Autoria própria

O quadro 1 contem a composição síncrona dos autômatos M1 $\|$ M2 segundo as especificações apresentadas anteriormente.

	A	B	C	D
{S0,S0}	{S1,S0}		{S0,S1}	
{S0,S1}	{S1,S1}			{S0,S0}
{S1,S0}		{S0,S0}	{S1,S1}	
{S1,S1}		{S0,1S}		{S1,S0}

Quadro 1 – Composição síncrona autômato M1 || M2
Fonte: Autoria própria

Os novos estados são formados pela combinação dos estados existentes em M1 e M2 estes estão dispostos na primeira coluna vertical, todos os eventos de M1 e M2 estão dispostos na primeira linha horizontal. Depois disso é analisado o efeito de cada evento sobre cada um dos dois membros que formam o novo estado e se verifica para qual estado o autômato será levado com a corrência de cada evento. Tomando com exemplo o estado$0,0$ é possível observar que o evento A só tem influência sobre o autômato M1. Assim após a ocorrência do evento A o autômato que representa a composição síncrona de M1 e M2 sai do estado $0,0$ e vai para o estado $1,0$

Na figura 5 é possível observar o resultado da composição síncrona através de um autômato.

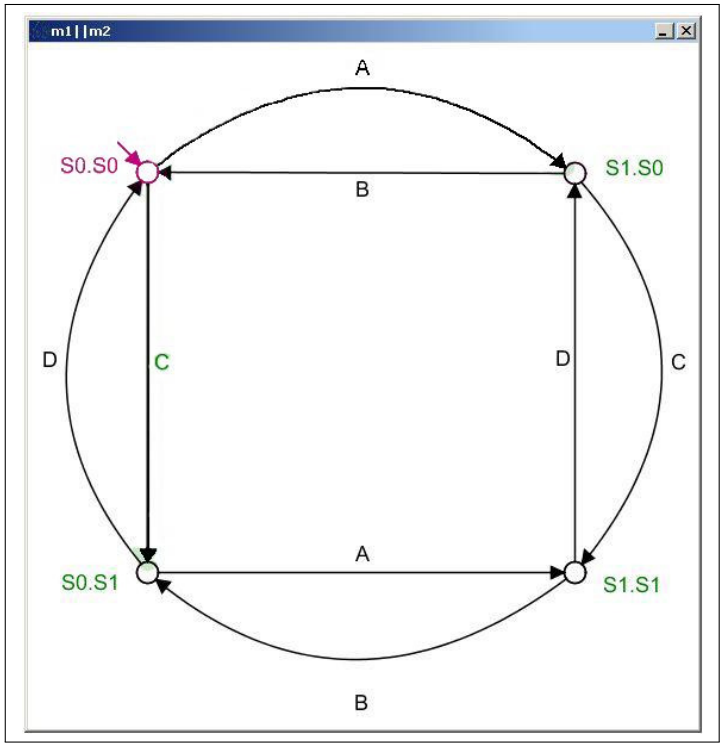

Figura 5 – Autômato da composição síncrona de M1 || M2
Fonte: Autoria própria

2.1.3.5 Observabilidade

Define-se um evento como observável quando sua ocorrência pode ser medida ou detectada através de sensores ou quando estes estiverem associados a comandos. Já os eventos não observáveis são aqueles em que a ocorrência não pode ser detectada por sensores, a esses são somados os eventos de falhas e os eventos que não podem ser detectados devido à característica distribuída do sistema (BASILIO; CARVALHO; MOREIRA, 2010). Estes são definidos como sistema com observabilidade parcial de eventos. Na ocorrência de um evento não observável o sistema não identifica a mudança no ambiente.

No caso de não ser possível identificar todos os eventos que ocorrem em um sistemas podemos particionar Σ em duas partes (CASSANDRAS; LAFORTUNE, 2008; PAOLI; LAFORTUNE, 2003):

- Σ_o - Representa o conjunto dos eventos observáveis;
- Σ_{uo} - Representa o conjunto de eventos não observáveis.

logo :

$$\Sigma = \Sigma_o \cup \Sigma_{uo}$$

Tendo como exemplo um autômato que contem eventos não observáveis, é possível que ocorra duas cadeias S_1, e S_2 que apresente as mesmas características se analisarmos somente os eventos observáveis. Para este caso sera observada a mesma ação de controle, fato que caracteriza um sistema com observabilidade parcial, a figura 6 apresenta um autômato que representa essa ideia. Este autômato contem $\Sigma_o = A, B, C$ e $\Sigma_{uo} = U$. O fato de um evento não ser observável não interfere na sua controlabilidade, assim um evento não observável pode ser ou não controlável Cassandras e Lafortune (2008).

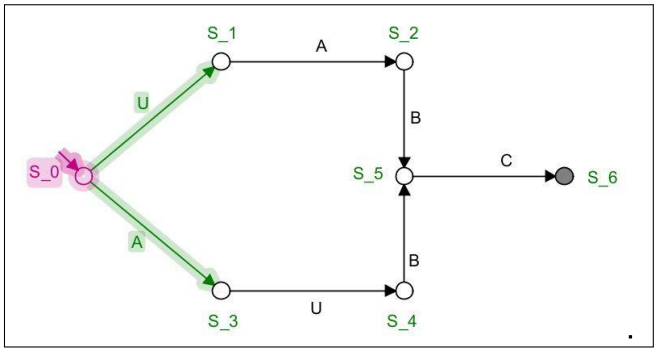

Figura 6 – Autoria com observabilidade parcial
Fonte: Autoria própria

Assim:

$S_1 = U, A, B, C$

$S_1 = A, U, B, C$

Ma se considerarmos somente os eventos observáveis:
$S_1 = A, B, C$

$S_2 = A, B, C$

2.1.3.6 Projeção

A projeção simbolizada por P_o apresentado por Sampath et al. (1995), Ramadge e Wonham (1989) é definida como:

$$\begin{aligned} P_o \quad : \quad & \Sigma^* \to \Sigma_o^*, \text{ sendo } \quad \Sigma_o \subset \Sigma \\ & s \to P_o(s) \end{aligned}$$

e tem como propriedades:

$$\begin{aligned} P_o(\epsilon) &= \epsilon, \\ P_o(\sigma) &= \begin{cases} \sigma, \text{ se } & \sigma \in \Sigma_o, \\ \epsilon, \text{ se } & \sigma \notin \Sigma_o, \end{cases} \\ P_o(s\sigma) &= P_o(s)P_o(\sigma), s \in \Sigma^*, \sigma \in \Sigma. \end{aligned}$$

Como o operador projeção pode ser aplicado as linguagens através das propriedades acima. Sendo $L \subset \Sigma^*$ temos:

$$P_o(L) = \{t \in \Sigma_o^* : (\exists s \in L)[P_o(s) = t]\}$$

Assim, tendo como referência Σ_o, projeção consiste em remover das sequências de L os eventos não observáveis do sistema (BASILIO; CARVALHO; MOREIRA, 2010), podendo gerar ambiguidade, pois também neste caso duas sequências distintas podem apresentar uma mesma projeção.

A projeção inversa é simbolizada por P_o^{-1} é apresentada por (MOREIRA; CA-

BRAL; DIENE, 2012) como sendo:

$$\begin{aligned} P_o^{-1} \quad &: \quad \Sigma_o^* \to 2^{\Sigma^*} \\ &\qquad s \to P_o^{-1}(s) = \{t \in \Sigma^* : P_o(t) = s\}. \end{aligned}$$

Assim a projeção inversa de uma linguagem M restrita à linguagem L é dada por: $P_{OL}^{-1}(M) = s \in L : P_o(t) = s$.

2.1.3.7 Diagnosticabilidade

Segundo Basilio, Carvalho e Moreira (2010) para a analise da diagnosticabilidade de falhas em SEDs, algumas hipóteses são feitas:

- A linguagem gerada por G é viva, i.e., $\Gamma(x_i) \neq \emptyset$ para todo $x_i \in X$
- O autômato G não possui nenhum ciclo formado somente por eventos não observáveis, i.e., $\forall ust \in L, s \in \Sigma_{ou}^*, \exists n_0 \in \mathbb{N}$ tal que $\|s\| \leq n_0$ sendo $\|s\|$ determina o comprimento da sequência s.
- Existe somente um único tipo de falha, i.e., $\prod_f = \{\Sigma_f\}$, em que $\Sigma_f = \{\sigma_f\}$

É possível diagnosticar falhas a partir do comportamento de uma sistema, para tanto é necessário que a linguagem gerada pelo autômato G seja diagnosticável, com isso pode-se utilizar um autômato determinístico denominado diagnosticador.

2.1.3.8 Função Reach

A função reach está vinculada ao uso de um método que forneça os possíveis estados atuais de G após a ocorrência de um evento observável. Isso é possível através do *Reach*(ν), em que $\nu = v\sigma_o = P_o(s)$ e representa a sequência de eventos observáveis pelo diagnosticador após a execução de uma sequência $s \in L$ cujo último evento observável é o σ_o, e pode ser calculado recursivamente como em Qiu e Kumar (2006).

- $Reach(\epsilon) = UR(x_{o,C})$,
- $Reach(\nu\sigma_o) = UR(\delta(Reach(\nu), \sigma_o))$,

Onde $UR(x)$ denota o alcance não observável do estado $x \in X$ com relação ao conjunto Σ_{uo}, é definido como:

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

2.1.4 Sintese de um observador

Algumas etapas são apresentadas por Cury (2001) para encontrar o supervisor minimamente restritivo (SupC).

(A) Obtenção de um modelo para a planta a ser controlada;

(B) Obtenção de um modelo de representação das especificações a serem respeitadas;

(C) Síntese de uma lógica de controle não bloqueante e ótima.

Cury (2001) também apresenta os passos para se gerar o supC no sistema Grail estes são:

(A) Construir a planta livre, através da composição síncrona das máquinas.

- O resultado deste processo é criando um outro arquivo que pode ser dado o nome de **> planta**

(B) Realizar a composição da **planta** com a restrição:

- Normalmente joga-se o resultado da função num outro arquivo **> S**

(C) Encontrar a componente co-acessível de **S** através da função trim:

- Com isso podemos gerar o um outro arquivo geralmente denominado **> Strim**

(D) Criar um arquivo de um único estado com um auto-loop dos eventos não controláveis:

- A esse arquivo é geralmente é dado o nome de **> NControl**

(E) Encontrar o supervisor minimamente restritivo através da função fmsupc:

- fmsupc **planta strim NControl > supervisor**
- Desta forma **Strim** define a especificação a ser seguida pela planta.

2.2 FALHAS

Segundo Amaral (2006) diagnosticar falhas é reconhecer, através da análise de dados como medição teste, um estado que possa ser classificado como um mau funcionamento do sistema. Para se determinar estes estados podemos realizar uma análise buscando por pontos quantitativos ou qualitativos, e sobre essses pontos podemos seguir duas abordagens diferentes: busca topográfica e busca sintomática (RASMUSSEN, 1986 apud MAURYA; RENGASWAMY; VENKATASUBRAMANIAN, 2005).

- Busca Topográfica

  As busca topográficas realizam uma análise buscando o mau funcionamento e tem como base um modelo de operação normal. Assim, a falha pode ser encontrada como uma diferença e identificada pela sua localização no sistema.

- Busca Sintomática

  As busca sintomática buscam por sintomas que permitem direcionar a busca, e assim, localizar as falhas. Deste modo, um conjunto de observações que representam o estado normal do sistema, pode ser utilizado como um modelo de pesquisa para encontrar conjuntos que correspondam a uma biblioteca de sintomas conhecidos, relacionados com diferentes estados de não conformidade do sistema.

Sobre análise do tratamento de falhas, basicamente existem 5 questões chaves (SLACK; CHAMBERS; JOHNSTON, 2002):

- Porque as operações falham?
- Como a falha é medida?
- Como a falha e a falha potencial podem ser detectadas e analisadas?
- Como as operações podem aprimorar sua confiabilidade?
- Como as operações deveriam recuperar as falhas quando elas ocorrem?

Algumas técnicas inteligentes aplicadas à detecção e diagnóstico de falhas são apresentada por (AMARAL, 2006). Como exemplo é apresenta apenas uma vantagem e um desvantagem de cada uma delas mas não necessáriamente o seus pontos mais fortes ou mais fracos:

- **Técnicas Tradicionais de Diagnóstico de Falhas**

  Sistemas baseados em Regras.

  [*] Sistemas baseados em regras do tipo "SE sintoma(s) ENTÃO falha(s)".

  [-] Desvantagem - A dificuldade de adquirir conhecimento.

  Árvores de Decisão de Falhas.

  [*] Uma árvore de decisão de falhas utiliza sintomas ou resultados de testes para decidir que ramo da árvore deve ser seguido. Cada ramo da árvore é composto de ações, novos testes e recomendações de reparo.

  [-] Desvantagem - Para sistemas complexos, esta árvore pode ser muito grande.

- **Abordagens baseadas em modelos (Model Based Reasoning)**

  [*] Permite modelar um sistema a partir dos modelos dos subsistemas que o constituem.

  [-] Desvantagem - Precisam de alto desempenho computacional.

- **Case Based Reasoning (CBR).**

  [*] Consiste em armazenar experiência de soluções anteriores e utilizá-las na solução dos novos casos.

  [-] Desvantagem - Em determinados casos é necessária a intervenção humana para adaptação do modelo anterior para a resolução do problema.

- **Abordagens utilizando Lógica Fuzzy.**

  Geração de modelos.

  [*] Possibilita modelar componentes ou módulos e com ele criar simulações para análise.

  [-] Desvantagem - A necessidade de comparação com um modelo real que já tenha falhado.

  Análise de resíduos.

  [*] Busca informações referentes as falhas não somente nos resíduos, mas também em informações que não podem ser codificadas em modelos matemáticos.

  [-] Desvantagem - É aplicável somente para a indicação da existência de falhas, sendo necessária outra etapa para isolar estas falhas.

- **Abordagens utilizando Redes Neurais.**

  Classificação de padrões.

  [*] A partir de um dicionário de falhas pré-definido é possível treinar uma rede para classificar um vetor de resíduos em classes de falhas.

  [-] Desvantagem - Esta técnica conseguirá bons resultados apenas para circuitos pequenos, pois em grandes malhas haverá uma explosão de dimensionalidade.

  Identificação de parâmetros.

  [*] Pode ser usado para identificar os parâmetros de um sistema dinâmico e utilizá-los em um modelo do sistema que está sendo diagnosticado ou como resíduo.

  [-] Desvantagem - Características pontuais com relação a falhas.

- **Abordagens Sistemas a Eventos Discretos.**

  Como a abordagem voltada a SEDs é o foco neste trabalho, algumas de suas características já foram mostradas nesta seção e outras mais serão mostradas na serão 3.

2.3 SIMULAÇÃO

Simulação é a técnica que busca estudar características e comportamentos de um determinado sistema. Utiliza como base modelos que imitam parte ou todas as propriedades e comportamentos deste sistema, em uma escala menor, e com isso permite sua manipulação e estudo detalhado (TECNOLOGIA, 2002). No entanto existem várias definições de simulação, assim como existem vários mecanismos que permitem simular sistemas. É possível aplicar ferramentas de simulação em indústrias, bancos, softwares, equipamentos eletrônicos e em muitas outras áreas.

A simulação com base em modelos surge como uma alternativa a ser considerada na resolução dos problemas das indústrias, centros de pesquisa e outras organizações, problemas estes que com o passar do tempo tornaram-se maiores e mais complexos, com isso houve a necessidade de um fluxo de investimento cada vez maior em técnicas, procedimentos e recursos (MELLO, 2001).

A necessidade não foi o fator fundamental que possibilitou o aumento significativo de modelos de *software* de simulação (BANKS, 1992), alguns eventos contribuíram e muito para o aumento da quantidade e da qualidade destes, tais como:

- aumento da capacidade computacional;
- evolução das tecnologias que permitem imagens computacionais de maior qualidade;
- novos sistemas operacionais SO que permitem maior acesso à memória;
- redução dos custos dos HDD que possibilitam melhores SO e outros softwares;
- softwares mais amigáveis.

2.3.1 Vantagens e desvantagens das simulações

As aplicações das simulações são muitas assim como as vantagens, mas apresentam também desvantagens quanto ao seu uso (SANTOS, 1999).

Vantagens:

- Possibilidade de estudo de políticas, procedimento, regras de negócio e outras operações sem alterar o mundo real.

- Possibilidade de testar novos *layouts*, equipamentos e sistemas de transporte, sem que estes sejam adquiridos.
- Avaliar a hipótese sobre certos fenômenos.
- O tempo pode ser comprimido ou expandido quando houver necessidade de avaliar certos fenômenos.
- Permite melhor visualização das variáveis que atuam no sistema.
- Permite uma visão geral da interação do sistema com outros sistemas.
- Permite avaliar condições específicas como o caso do "e se...".

Desvantagens:

- A construção de modelos requer pessoal com grande conhecimento do sistema e grande experiência em simulações.
- Em alguns casos é difícil avaliar a saída.
- A construção de modelos completos é complexa, leva tempo, e pode ter um custo elevado.
- A simulação em alguns casos é usada onde uma solução simples com base analítica é possível.

2.3.2 Tarefas não pertinentes

Mesmo com um grade leque de possibilidades, existem tarefas que não podem ser resolvidas por simulação. Não é possível para uma simulação: Prever o futuro, sendo possível somente determinar saídas decorrentes de entradas. Uma simulação não é um modelo matemático, e não pode ser reduzida a uma fórmula matemática. Não se pode considerar uma simulação com uma ferramenta unicamente de otimização, elas podem através de algoritmos realizar otimizações mas suas funções vão alem disso, geralmente voltadas para análise de cenário. Mesmo tendo um grande poder a simulação não substitui um pensamento inteligente, não é possível substituir o poder de decisão humano por uma simulação. Também não deve ser usada como último recurso como foi no passado. E por fim uma simulação não serve para resolver todos os problemas, ela deve ser usada para tratar problemas específicos (CHWIF; MEDINA, 2006).

2.3.3 Simulação e aprendizagem

A introdução do computador como ferramenta de ensino propiciou a criação de *softwares* específicos para este contexto (JUCÁ, 2011). Softwares que são desenvolvidos para outras áreas e que também são usados como ferramentas de ensino-aprendizagem evoluíram significativamente, e a junção destes passou a ser denominada *softwares* educacionais e com uma grande gama de aplicações.

Atualmente, as simulações têm ampliado suas funções e dado suporte ao desenvolvimento da visão geral da prática de pensar estrategicamente, da capacidade de atuar de forma conjunta e distribuir conhecimento buscando uma aprendizado coletivo (BELHOT; FIGUEIREDO; MALAVÉ, 2001). Assim a modelagem pode ser explorada em três aspectos:

(A) como um processo de mapeamento cognitivo que captura o conhecimento e estimula a aprendizagem;

(B) como um meio propício à experimentação e;

(C) como uma forma de aprender a lidar com conflito de interesses.

Há mais de uma década, a simulação é usada no ensino, sendo uma ferramenta que, usada de forma isolada, pode simplificar o ambiente, limita o envolvimento de pessoas e reduz o escopo do problema. Desta forma, ela mostra-se como uma técnica que deve ser valorizada e disseminada entre os estudantes de engenharia, fazendo com que aprimorem o conhecimento, experiência e compreensão da realidade (BELHOT; FIGUEIREDO; MALAVÉ, 2001).

3 REPRESENTAÇÃO DE FALHAS EM SISTEMA A EVENTOS DISCRETOS

As falhas são resultado da ocorrência de eventos não controláveis e não observáveis. Para se determinar que ocorreu um evento falha é preciso veriticar os estados futuros, ou seja, após a falha. Assim, Basilio, Carvalho e Moreira (2010) afirmam que o problema com o diagnóstico de falhas consiste no fato que, o diagnosticador nem sempre sabe exatamente em que estado o sistema se encontra.

Seja, por exemplo, o autômato apresentado na figura 7, onde os estados normais são representados por N_ e os estados após a ocorrência de uma falha são representados por F_.

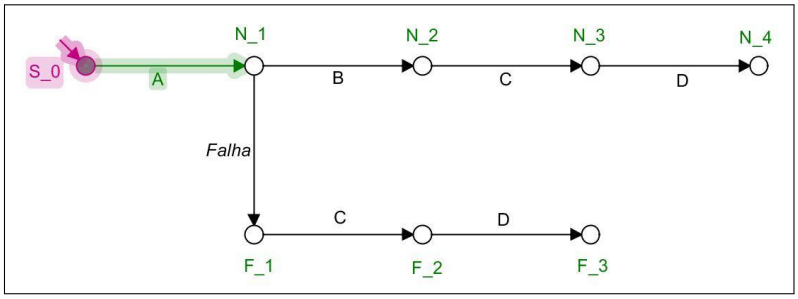

Figura 7 – Autômato com falha
Fonte: Autoria própria

Após ocorrer o evento A o observador não consegue saber se o sistema está no estado N_1 o se ocorreu uma falha e o sistema já se encontra no estado F_1 assim hà necessidade que ocorra um evento observável para saber qual foi o caminho seguido. Se for observado que ocorreu o evento B significa que não hove falhas e o autômato continua executando normalmente, mas se for observado que ocorreu o evento C significa que ocorreu um evento de falhas e todos os estados a partir deste evento são estados de falhas.

O estudo do diagnóstico de falhas em SEDs abrange varias áreas, uma desta é a análise da forma como essa falha podem ocorrer, como é o caso dos estudos voltados para falhas intermitentes e distribuídas, ou utilizando SEDs e outras técnicas. Nesta seção serão abordados alguns dos métodos de diagnóstico presentes na literatura.

3.1 DIAGNOSTICABILIDADE SEGURA

Segundo Paoli e Lafortune (2003) diagnosticabilidade segura deve ser definida sobre uma linguagem L sendo esta prefixo-fechada, que atenda a:

- A1, há para L uma transição definida em cada estado alcançável $x \in X$,

- A2, não existe um ciclo de eventos não observáveis.

Assim, a diagnosticabilidade é segura com relação à projeção P, à partição $\prod_f$, e às línguagens proibidas que podem ser geradas após a ocorrência de uma falha $K_f^i (i = 1, ..., m)$, se a condição de diagnosticabilidade e a de segurança forem satisfeias.

Condição de diagnosticabilidade:

$$(\forall i \in \prod_f)(\exists n_i \in \mathbb{N})(\forall s \in \Psi(\Sigma_{fi}))(\forall t \in L \backslash s)(\|t\| \leq n_i / D)$$

Onde a condição de diagnosticabilidade de D é dada por:

$$w \in P_L^{-1}[P(st)] \leq \Sigma_{fi} \in w$$

Deste modo diagnosticabilidade segura também pode ser definida como a possibilidade de diagnósticar falhas através de uma linguagem L mesmo após a ocorrência de uma falha.

3.2 DIAGNÓSTICO DE FALHAS INTERMITENTES

Lefebvre e Leclercq (2011) apresentam um método de detecção e isolamento de falhas que ocorrem a partir de uma sequência de pares onde a entrada é dada por $U = (u(0), u(1), ..., u(k))$ e a saída por $Y = (y(0), y(1), ..., y(k))$, sendo k o índice de ocorrência. As entradas são geralmente consideradas como eventos e as saídas estão relacionadas com os estados. As falhas são divididas em:

fortemente diagnosticável : resultam em comportamentos anormais imediatos, e nenhum evento intermediário é necessário para o diagnóstico;

fracamente diagnosticável : resultam em comportamentos anormais após um número finito de eventos intermediários;

não-diagnosticável : nenhum comportamento anormal ocorre qualquer que seja a evolução futura do sistema.

O diagnóstico de falhas intermitente é mais complexo que o diagnóstico de falhas permanentes, tendo em vista que estas pode se recuperar e voltar para o comportamento normal (CONTANT; LAFORTUNE; TENEKETZIS, 2002).

3.3 IDENTIFICAÇÃO DE GRUPOS DE FALHAS

A identificação de grupos de falhas é apresentada por Moreira, Cabral e Diene (2012), esta proposta de diagnóstico de falhas tem como base o autômato G_C G_C que é o autômato que modela o comportamento normal do sistema. Para obtenção deste autômato devemos partir do autômato G que modela o comportamento normal do sistema e dos autômatos G_{N_k} que podem assumir $k = 1, ..., r$, onde o autômato G_{N_k} modela o comportamento normal de G tendo como referência o conjunto de eventos de falha Σ_{f_k}. O algoritmo 1 apresenta o método para a obtenção do autômato G_C.

Algoritmo 1 *:*

(A) - Calcule o autômato G_{N_k}*, para cada* $k \in \Pi_f$ *que modela o comportamento normal de* G *com relação ao conjunto de eventos de falha* Σ_{f_k} *, da seguinte forma:*

- *1.1: Defina* $\Sigma_{N_k} = \Sigma \setminus \Sigma_{f_k}$
- *1.2: Construa o autômato* A_{N_k} *composto de um único estado* N_k *com um auto laço rotulado com todos os eventos de* Σ_{N_k}
- *1.3: Faça* $G_{N_k} = GxA_{N_k} = (X_{N_k}, \Sigma, f_{N_k}, \Gamma_{N_k}, x_{0,N_k})$.

(B) - Construa o autômato estendido $G^a_{N_k}$*, para cada* $k \in \Pi_f$*, adicionando um novo estado* F_k *, que indica que um evento de falha do conjunto* Σ_{f_k} *ocorreu. Uma nova transição rotulada com um evento* $\sigma_{f_k} \in \Sigma_{f_k}$ *é adicionada, conectando o estado* (x, N_k) *de* G_{N_k} *ao estado de falha* F_k*, se* $\alpha_{f_k} \in \Gamma(x)$*. Adicione um auto laço rotulado com todos os eventos* $\sigma \in \Sigma$ *ao estado de falha* F_k.

(C) - Calcule o autômato $G_C = (X_C, \Sigma, f_C, \Gamma_C, x_{0,C}) = G^a_{N_1} \| G^a_{N_2} \| \cdots \| G^a_{N_r}$.

Fonte: (MOREIRA; CABRAL; DIENE, 2012)

É importante resaltar nesse ponto que a linguagem original não é preservada para os eventos a partir da ocorrência de uma falha, mas como o autômato que realiza o diagnóstico é passivo, este fato não interfere na observação dos eventos do sistema, bem como no diagnóstico de falhas (MOREIRA; CABRAL; DIENE, 2012).

Após a observação de uma sequência ν , o conjunto dos possíveis estados atuais de G_C , $Reach(\nu)$, pode ser calculado e seus estados podem ser usados para identificar a ocorrência de um evento de falha. O teorema a seguir apresenta a base para o método de diagnose proposto.

Teorema 1 *:*

Seja L *a linguagem gerada por* G *e supondo que* L *é diagnosticável com relação à* P_o *e* Π_f*. Seja* $s \in L \setminus L_{N_k}$ *tal que* $\forall w \in L$*, com* $P_o(w) = P_o(s), w \in L \setminus L_{N_k}$*. Então, a K-ésima coordenada de todos os possíveis estados atuais de* G_C*, alcançados após a ocorrência de* s*, dados por* $Reach(P_o(s))$*, é igual a* F_k *(MOREIRA; CABRAL; DIENE, 2012).*

De acordo com o teorema 1, se L é diagnosticável com relação à P_o e Π_f, então sempre é possível identificar a ocorrência de uma falha do tipo F_k com um número limitado de observações verificando os possíveis estados atuais de G_C. Se após a ocorrência de uma sequência s, todos os estados de $Reach(v)$, em que $v = P_o(s)$, não possuirem uma coordenada (q, N_k), então não é possível que uma sequência normal com relação ao conjunto de eventos de falha Σ_{f_k}, com a mesma projeção que v, tenha sido executada, o que implica que uma falha do tipo F_k ocorreu, ou seja, s contém um evento de falha $\sigma_{f_k} \in \Sigma_{f_k}$. Logo, a diagnose de uma falha do tipo F_k pode ser feita verificando se um estado do comportamento normal descrito por G_{N_k} é uma coordenada de um possível estado atual de G_C.

Observação 1 *:*

O número de estados de G_C *é, no pior caso, igual a* $[(2^r - 1)\ x\ |Q|] + 1$*, em que* r *é o número de tipos de falhas do sistema. Logo, a complexidade computacional da construção de um autômato* G_C *é* $O(2^r\ x\ |Q|\ x\ |\Sigma|)$*, o que mostra que a complexidade é linear com o número de estados e eventos do autômato do sistema e exponencial com o número de tipos de falhas. A complexidade computacional pode ser linear com relação ao número de tipos de falha se cada comportamento normal com relação a um tipo de falha for considerado separadamente. Nesse caso, ao invés de um único autômato* G_C*, têm-se* r *autômatos* $G^a_{N_k}$*, em que cada um leva em consideração apenas a falha do tipo* F_k*, e a complexidade computacional é* $O(r\ x\ |Q|\ x\ |\Sigma|)$*. Embora a análise do pior caso sugira que é vantajoso considerar os autômatos* $G^a_{N_k}$*, para* $k = 1, ..., r$*, ao invés de* G_C*, é importante observar que o número de estados de* G_C *pode ser menor que a soma do número de estados de* $G^a_{N_k}$ *para* $k = 1, ..., r$*, levando a um código de programação menor para a implementação do diagnosticador.(MOREIRA; CABRAL; DIENE, 2012)*

Tomando como exemplo o autômato G do sistema apresentado na figura 8, em que:

Σ = { $a, b, c, \sigma_u, \sigma_{f_1}, \sigma_{f_2}$ }

Σ_o = $\{a, b, c\}$

Σ_{uo} = $\{\sigma_u, \sigma_{f1}, \sigma_{f2}\}$

$\Sigma_f = \{\sigma_{f1}, \sigma_{f2}\}$

Onde:

- Σ - Representa o conjunto de todos os eventos possíveis.
- Σ_o - Representa o conjunto de todos os eventos observáveis.
- Σ_{uo} - Representa o conjunto de todos os eventos não observáveis.
- Σ_f - Representa o conjunto de todos os eventos de falhas.

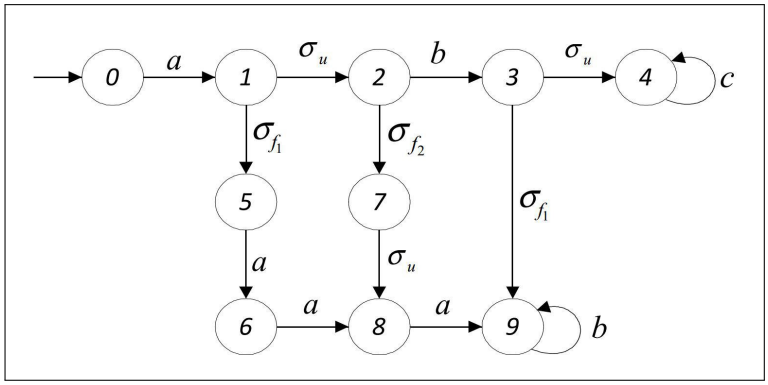

Figura 8 – Autômato G
Fonte: (MOREIRA; CABRAL; DIENE, 2012)

Supondo que o conjunto de falhas possa ser particionado em:

$\Sigma_f = \Sigma_{f_1} \dot{\cup} \Sigma_{f_2}$

onde:

$\Sigma_{f_1} = \{\sigma_{f_1}\}$

$\Sigma_{f_2} = \{\sigma_{f_2}\}$

Para se calcular o autômato G_C de acordo com o algoritmo 1, o primeiro passo é obter os A_{N_k} para $k = 1,2$ através do autômato mostrado na figura 9

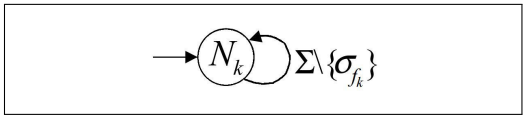

Figura 9 – Autômato A_{N_K}
Fonte: (MOREIRA; CABRAL; DIENE, 2012)

Em seguida são gerados os autômatos $G_{N_k} = G \times A_{N_k}$ para $k = 1,2$ e acrescentando os estados de falhas F_1 e F_2 a fim de obter os autômatos $G^{\alpha}_{N_1}$ e $G^{\alpha}_{N_2}$. O autômato para $G^{\alpha}_{N_1}$ representado pela figura 10.

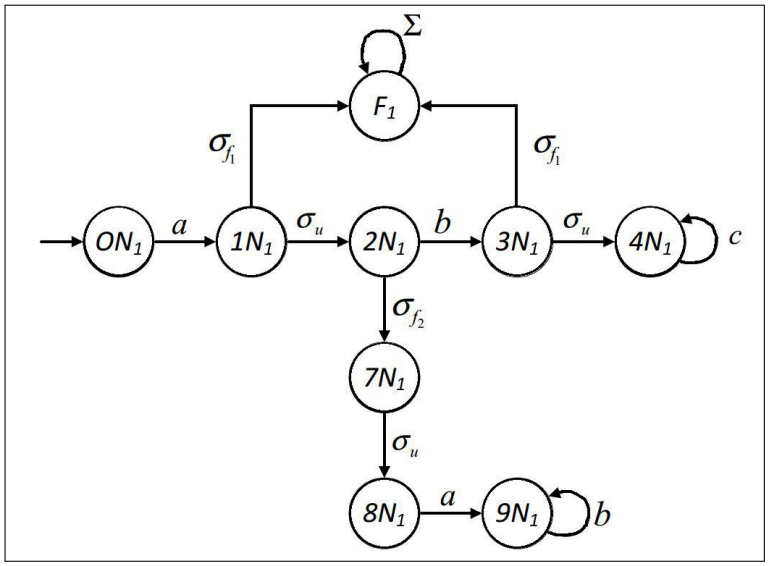

Figura 10 – Autômato $G^{\alpha}_{N_1}$
Fonte: (MOREIRA; CABRAL; DIENE, 2012)

E $G^{\alpha}_{N_2}$ representado pela figura 11.

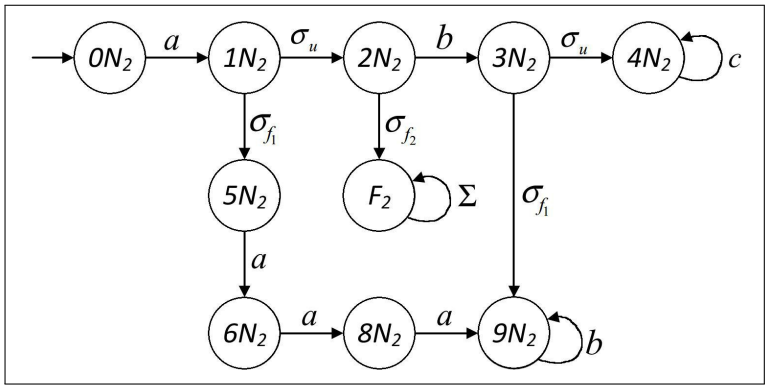

Figura 11 – Autômato $G^{\alpha}_{N_2}$
Fonte: (MOREIRA; CABRAL; DIENE, 2012)

E o último passo do algoritmo é a obtenção do autômato $G_C = G^{\alpha}_{N_1} \parallel G^{\alpha}_{N_2}$, representado pela figura 12.

Para se utilizar o autômato G_C na detecção de falhas tendo como referência a sequência:

$s = \alpha, \sigma_{f_1}, \alpha, \alpha \in L \setminus L_{N_1}$

Logo a sequência observada é:

$P_o(s) = \alpha\alpha\alpha$

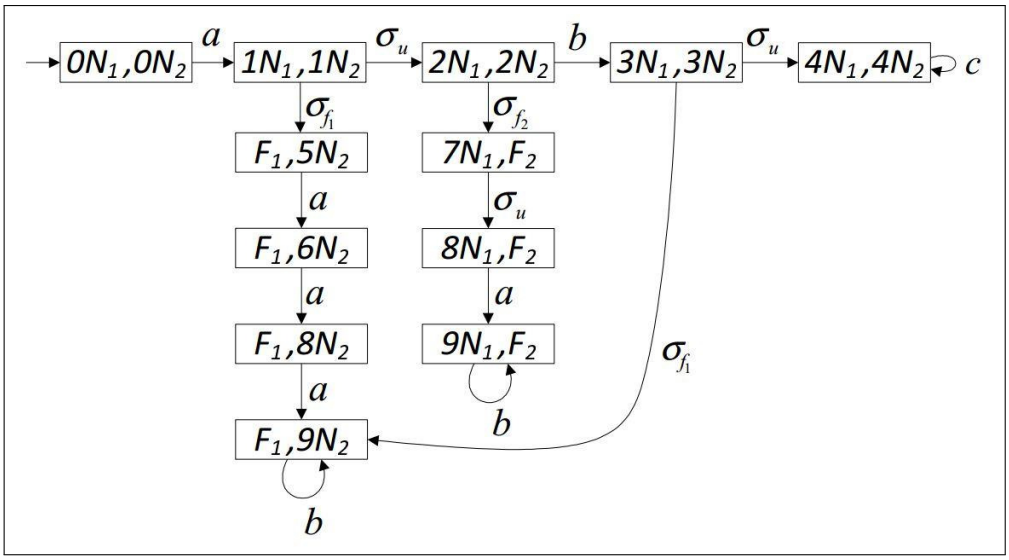

Figura 12 – Autômato G_C
Fonte: (MOREIRA; CABRAL; DIENE, 2012)

De acordo com o teorema 1, se não existir uma sequência $\omega \in L_{N_1}$ tal que $P_o(\omega) = \nu$ então todos os estados no conjunto de estados alcançáveis $Reach(\nu)$ possuem a primeira coordenada igual a F_1. O conjunto de estados alcançáveis $Reach(\nu)$ pode ser obtido recursivamente da seguinte forma:

- $Reach(\epsilon) = \{(0N_1, 0N_2)\}$
- $Reach(a) = \{(1N_1, 1N_2), (2N_1, 2N_2), (F_1, 5N_2), (7N_1, F_2), (8N_1, F_2)\}$
- $Reach(aa) = \{(F_1, 6N_2), (9N_1, F_2)\}$
- $Reach(aaa) = \{(F_1, 8N_2)\}$

Assim, quando analisamos $Reach(aaa)$ é possível identificar que somente após a ocorrência de terceiro evento a é possível ter a certeza que ocorreu uma falha do tipo F_1. Desta forma é possível identificar qual o tipo de falha mesmo esta não sendo observável.

4 ESTUDO DE CASO DE UMA FÁBRICA DE PÃES

Tendo como objetivo avaliar um sistema de diagnóstico de falhas foi feita a escolha de um problema destinado ao estudo de caso que partiu do seguinte princípio, mesmo sendo um sistema de produção construído a partir de eventos reais este deveria ser de fácil entendimento para os usuários, algo que fosse da realidade e do cotidiano da maioria das pessoas. Assim é possível propiciar uma maior liberdade, pois conhecendo as etapas do processo torna-se mais fácil de modelar cada equipamento que faz parte do processo. Isto contribui para o estudo das abordagens relacionadas a falhas e simulação, pois cria uma relação entre estados, eventos e falhas.

Para os proximos passo foi necessário o desenvolvimento de uma modelo modelo que representace as características do problema proposto. Este modelos permite que cada dos componentes sejam modelados de forma individual, deixando os usuários à vontade para escolher as etapas de produção de cada máquina, seu grau de complexidade, eventos controláveis, não controláveis, observáveis e não observáveis e a possibilidade de não falhar ou mesmo de apresentar mais de um tipo de falhas em uma mesma máquina.

Ao sistema foi dado a possibilidade de processos alternativos, a inserção desta característica rompe com o processo completamente linear, dando ao usuário a possibilidade de definir à proporção que deseja de cada um dos produtos finais.

4.1 DESCRIÇÃO DOS ELEMENTOS DA FÁBRICA

O sistema proposto é uma fábrica de pães que tem como produto final: pão comum, pão com nozes, pão comum com cobertura e pão com nozes e cobertura. Para atingir esse objetivo a fábrica conta com oito dispositivos chamados aqui de máquinas:

(A) Esteira Transportadora (ET)

- É uma esteira de transporte que percorre toda a extensão da fábrica.

(B) Adicionar Ingredientes (AI)

- É uma máquina que adiciona o recipiente e os ingredientes comuns a todos os produtos, localizada no início do processo.

(C) Esteira Classificadora 1 (EC1)

- Máquina responsável por destinar parte da produção para a máquina que adiciona nozes.

(D) Adicionar Nozes (AN)

- Máquina que adiciona nozes aos ingredientes iniciais gerando assim um novo produto.

(E) Misturador (MT)

- O misturador cria uma mistura ideal para os ingredientes, independente do fato de ter sido ou não adicionado nozes à mistura.

(F) Forno (FN)

- O forno é necessário para assar os pães.

(G) Esteira Classificadora 2 (EC2)

- Esta esteira destina parte da produção para a máquina que adiciona cobertura.

(H) Adicionar Cobertura (AC)

- Máquina que adiciona uma cobertura cristalizada sobre o pão gerando novos produtos.

O layout desta fábrica é representado pela figura 13.

Sequência	Descrição
ET	Esteira Transportadora
AI	Adicionar Ingredientes
EC1	Esteira Classificadora 1
AN	Adicionar Nozes
MT	Misturador
FN	Forno
EC2	Esteira Classificadora 2
AC	Adicionar Cobertura

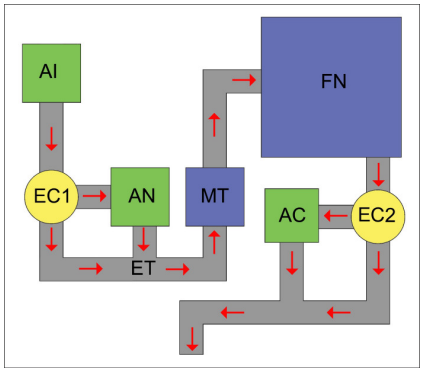

Figura 13 – Layout da fábrica
Fonte: Autoria própria

A partir do layout fica mais fácil compreender o relacionamento de cada máquina com sua antecessora ou sucessora. Cada máquina executa funções que pode ser um simples ligar e desligar outras mais complexas como: virar a direita ou a esquerda executar a função A, B ou C. Estas funções podem representar o acionar de um motor, esteira, braços robóticos e outros dispositivos. Todas as máquinas são representadas por autômatos e cada autômato contem as características de uma única máquina.

O usuário tem a liberdade de escolher qual máquina pode falhar, para isso deve modelar a falha no autômato que representa a máquina respeitando as pontos como a diagnosticabilidade apresentada no item 2.1.3.7.

Cada um dos produtos parte da máquina Adiciona Ingredientes mostrada da figura 13 e evolui através de esteiras, sendo direcionados para as máquinas que determinarão qual o produto em que os ingredientes iniciais irão se transformar.

4.2 EXEMPLO DE PROJETO

Foi criado um modelos contendo os autômatos de cada uma das máquinas que compõe o sistema. Este modelo foi aplicado em um software de simulação. A titulo de ilustração esses autômatos foram criados no *software* Supremica apresentando por Akesson et al. (2006). já passo necessários para a obtenção da maxima linguagem controlável utilizaram os algoritmos do *software* e Grail apresentado por Reiser (2005). A escolha destes softwares se deu pelo fato de serem reconhecidos no mundo academico que trabalha com autômatos. Para representação feita pelo *software* Supremica todos os autômatos terão os estados representados por uma circunferência e as transições representadas por arcos etiquetados ou linhas. O estado inicial será representado por uma seta inclinada sobre a circunferência e o estado marcado por um círculo. Os itens a seguir detalham as características de cada autômato.

4.2.1 Modelagem da Planta

(A) Esteira Transportadora (ET)

O autômato ET, que rege o funcionamento da máquina Esteira Transportadora é representado pela figura 14, onde estão visíveis os estados e os eventos. Esta máquina foi simplificada e não foi modelada com a possibilidade de falha.

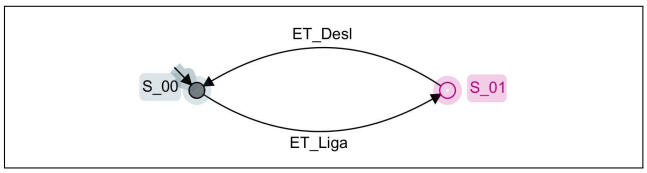

Figura 14 – Modelo do autômato ET no Supremica
Fonte: Autoria própria

O quadro 2 contém informações sobre os estados do autômato ET.

O quadro 3 contém informações sobre os eventos do autômato ET.

Estado	Descrição
S_00	Desligada
S_01	Ligada

Quadro 2 – Estados do autômato ET
Fonte: Autoria própria

Evento	Descrição
ET_Liga	Evento controlável que liga ET
ET_Desl	Evento controlável que desliga ET

Quadro 3 – Eventos do autômato ET
Fonte: Autoria própria

A titulo de ilustração sera mostrado o arquivo que representa o autômato ET no Grail no quadro 4 a representação dos demais autômatos no Grailo seram suprimidos pois segem as mesmas características.

(START)	|-	0
0	ET_Liga	1
1	ET_Desl	0
0	-|	(FINAL)

Quadro 4 – Modelo do autômato ET no Grail
Fonte: Autoria própria

(B) Adiciona Ingredientes (AI)

O autômato AI representado na figura 15, apresenta uma complexidade maior que a do autômato ET, isto se deve ao fato que o AI já conta com a possibilidade de falha através do evento Falha_AI. Por se tratar de uma falha, este evento é não-observável e não-controlável. O autômato AI conta ainda com um outro evento não controlável é o AI_03, e apresenta como estado inicial S_00 e estados marcados { S_00, S_04 }.

O quadro 5 contém informações sobre as estados do autômato AI.

O quadro 6 contém informações sobre os eventos do autômato AI.

(C) Esteira Classificadora 1 (EC1)

A máquina Esteira Classificadora 1, mostrada na figura 16, cria o primeiro desvio condicional do processo, este modelo conta com um número maior de estados se for

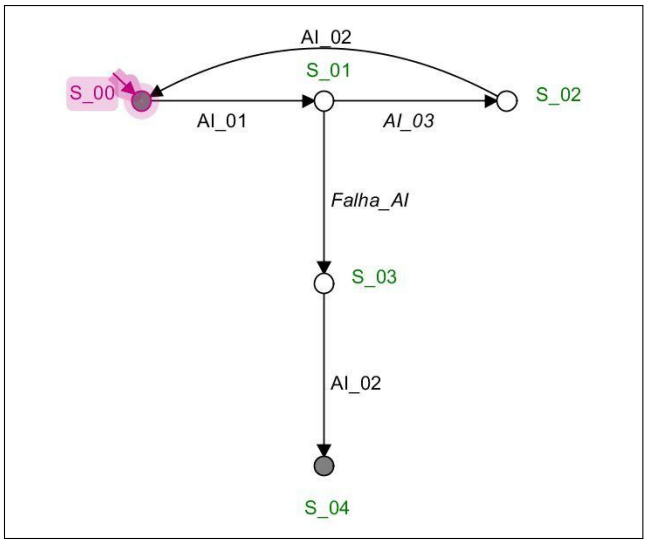

Figura 15 – Modelo do autômato AI no Supremica
Fonte: Autoria própria

Estado	Descrição
S_00	Inicial.
S_01	Adição de ingredientes.
S_02	Ingredientes Adicionados.
S_03	Falha de Adição de Ingredientes.
S_04	Final para Falha.

Quadro 5 – Estados do autômato AI
Fonte: Autoria própria

Evento	Descrição
AI_01	A máquina é ativada e prepara a forma para receber os ingredientes.
AI_02	Libera a forma após a adição dos ingredientes e desativa a máquina.
AI_03	Adiciona os ingredientes
Falha_AI	Falha na adição de Ingredientes

Quadro 6 – Eventos do autômato AI
Fonte: Autoria própria

comparado com o modelo do autômato ET, apesar de sua modelagem não contar com estado de falha. Tem como função encaminhar os ingredientes iniciais para o Misturador ou para a máquina Adiciona Nozes. A quantidade de ingredientes que segue cada caminho depende da restrição programada para o autômato EC1. Este apresenta como estado inicial e estado marcado o estado S_00 .

Figura 16 mostra o autômato EC1 modelado no *software* Supremica.

O quadro 7 contém informações sobre as estados do autômato EC1.

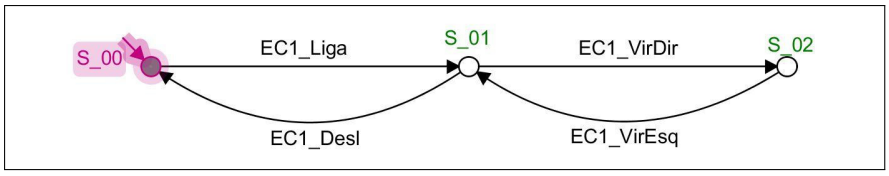

Figura 16 – Modelo do autômato EC1 no Supremica
Fonte: Autoria própria

Estado	Descrição
S_00	EC1 encontra-se desligada
S_01	EC1 encontra-se ligada
S_02	EC1 está ligada e rotacionada 90’ a direita

Quadro 7 – Estados do autômato EC1
Fonte: Autoria própria

O quadro 8 contém informações sobre os eventos do autômato EC1.

Eventos	Descrição
EC1_Liga	Liga EC1
EC1_Desl	Desliga EC1
EC1_VirDir	Desloca a esteira 90’ para direita
EC1_VirEsq	Desloca a esteira 90’ para esquerda

Quadro 8 – Eventos do autômato EC1
Fonte: Autoria própria

(D) Adiciona Nozes (AN)

A máquina Adiciona Nozes foi modelada com características semelhantes à máquina Adiciona Ingredientes, mas desta foi removida a possibilidade de falha. A figura 17 mostra o autômato AN. Como ele não conta com a possibilidade de falha, parte de sua estrutura difere do autômato AI. O autômato tem como estado inicial e final S_00. Contém um evento não controlável o AN_03

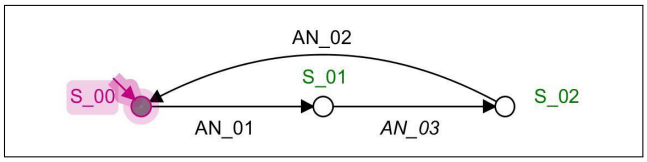

Figura 17 – Modelo do autômato AN no Supremica
Fonte: Autoria própria

O quadro 9 contém informações sobre as estados do autômato AN.

Estado	Descrição
S_00	Inicial.
S_01	Adição de nozes.
S_02	Nozes adicionadas.

Quadro 9 – Estados do autômato AN
Fonte: Autoria própria

O quadro 10 contém informações sobre os eventos do autômato AN.

Eventos	Descrição
AN_01	Ativa a máquina posiciona a forma para receber as nozes.
AN_02	Adiciona nozes.
AN_03	Libera a forma e desativa a máquina.

Quadro 10 – Eventos do autômato AN
Fonte: Autoria própria

(E) Misturador (MT)

O autômato MT apresenta as mesmas características do autômato ET: não apresenta a possibilidade de falhas, possuindo apenas eventos observáveis. A figura 18 mostra o autômato MT.

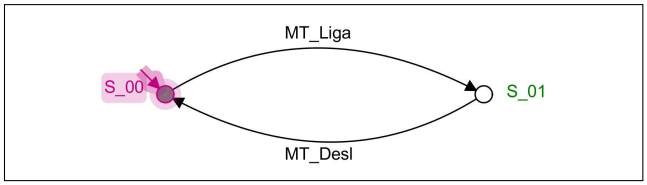

Figura 18 – Modelo do autômato MT no Supremica
Fonte: Autoria própria

O quadro 11 contém informações sobre as estados do autômato MT.

Estado	Descrição
S_00	Misturador encontra-se desligado
S_01	Misturador encontra-se ligado

Quadro 11 – Estados do autômato MT
Fonte: Autoria própria

O quadro 12 contém informações sobre os eventos do autômato MT.

Eventos	Descrição
MT_liga	Liga o misturador
MT_Desl	Desliga o misturador

Quadro 12 – Eventos do autômato MT
Fonte: Autoria própria

(F) Forno (FN)

O autômato FN, apresentado na figura 19, representa a máquina Forno e contém características similares ao autômato AI. Ele conta com a possibilidade de falha através do evento Falha_FN sendo este não-observável e não-controlável. O autômato FN contém outro evento não controlável o FN_03 e apresenta como estado inicial S_00 e estados marcados { S_00, S_04 }.

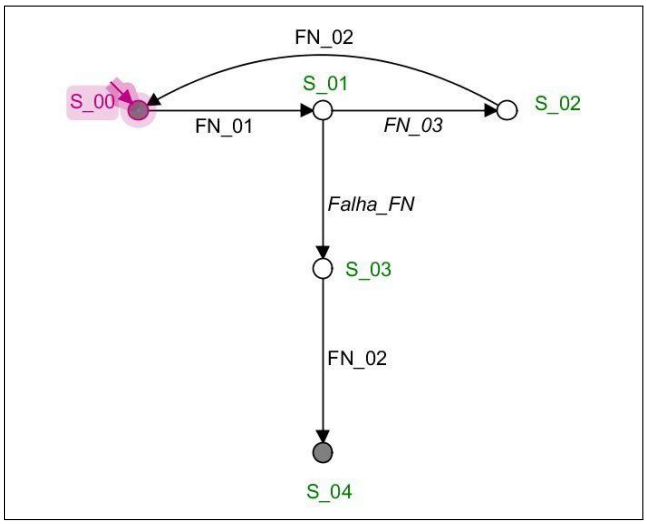

Figura 19 – Modelo do autômato FN no Supremica
Fonte: Autoria própria

O quadro 13 contém informações sobre as estados do autômato FN.

Estado	Descrição
S_00	Inicial.
S_01	Ativar forno.
S_02	Forno ativado.
S_03	Falha na ativação do forno.
S_04	Final para falha.

Quadro 13 – Estados do autômato FN
Fonte: Autoria própria

O quadro 14 contém informações sobre os eventos do autômato FN.

(G) Esteira Classificadora 2 (EC2)

O autômato EC2, mostrado na figura 20, rege o funcionamento da máquina Esteira Classificadora 2 e tem basicamente as mesmas característica do autômato EC1.

Evento	Origem
FN_01	Posiciona a forma.
FN_02	Libera a forma.
FN_03	Ativar forno e o sistema de desativamento.
Falha_FN	Falha na ativação do forno.

Quadro 14 – Eventos do autômato FN
Fonte: Autoria própria

Este é o segundo desvio condicional do processo, para este autômato também não foram modeladas falhas. Tem como função encaminhar os pães saídos do forno para a máquina Adiciona Cobertura. A quantidade de pães que segue cada caminho depende da restrição programada para o autômato EC2. Este apresenta como estado inicial e estado marcado o estado S_00 .

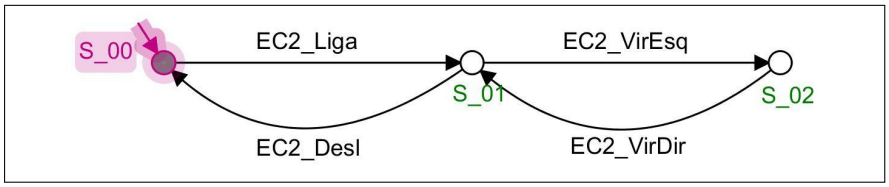

Figura 20 – Modelo do autômato EC2 no Supremica
Fonte: Autoria própria

O quadro 15 contém informações sobre as estados do autômato EC2.

Estado	Descrição
S_00	EC2 encontra-se desligada
S_01	EC2 encontra-se ligada
S_02	EC2 esta ligada e rotacionada 90º a esquerda

Quadro 15 – Estados do autômato EC2
Fonte: Autoria própria

O quadro 16 contém informações sobre os eventos do autômato EC2.

(H) Adiciona cobertura (AC)

A última máquina do processo é a Adiciona Cobertura, mostrada na figura 15, que apresenta as mesmas características mostradas no autômato AN. Este autômato não conta com a possibilidade de falha, e contém como evento não-controlável o evento AC_03. O autômato AC tem como estado inicial e estados marcado S_00.

Eventos	Descrição
EC2_Liga	Liga EC2
EC2_Desl	Desliga EC2
EC2_VirDir	Desloca a esteira 90º para esquerda
EC2_VirEsq	Desloca a esteira 90º para direita

Quadro 16 – Eventos do autômato EC2
Fonte: Autoria própria

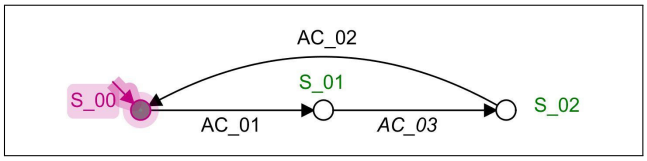

Figura 21 – Modelo do autômato AC no Supremica
Fonte: Autoria própria

O quadro 17 contém informações sobre as estados do autômato AC.

Estado	Descrição
S_00	Inicial.
S_01	Adição de cobertura.
S_02	Cobertura adicionada.

Quadro 17 – Estados do autômato AC
Fonte: Autoria própria

O quadro 18 contém informações sobre os eventos do autômato AC.

Eventos	Descrição
AC_01	Ativa a máquina e trava a forma
AC_02	Adiciona cobertura
AC_03	Libera a forma e desativa máquina

Quadro 18 – Eventos do autômato AC
Fonte: Autoria própria

4.2.2 Modelagem das restrições de controle

(A) Restrições e a relação entre os autômatos AI, ET e EC2.

A figura 22 mostra as restrições que relacionam o autômato AI, ET e EC2 vistas através do *software* Supremica.

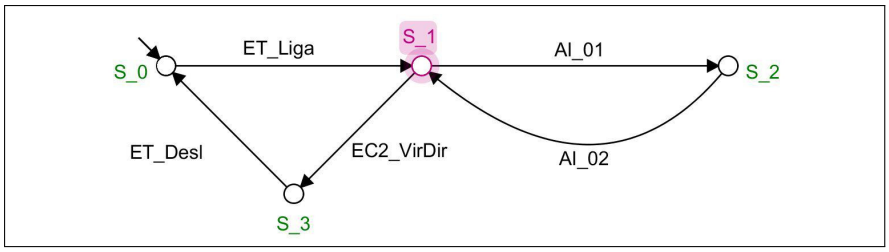

Figura 22 – Modelo das restrições autômato AI no Supremica
Fonte: Autoria própria

O arquivo que representa as restrições que relacionam aos autômato AI, ET e EC2 no Grail é visualizado no quadro 19. Esta restrição visa garantir que só ocorra o evento AI_01 após a esteira ser ativada através do evento ET_Liga e só desligue ET após a ocorrer EC2_VirDir.

(START)	|-	0
0	ET_Liga	1
1	AI_01	2
2	AI_02	1
1	EC2_VirDir	3
3	ET_Desl	0
0	-|	(FINAL)
1	-|	(FINAL)

Quadro 19 – Modelo das restrições autômato AI no Grail
Fonte: Autoria própria

(B) Restrições e a relação entre os autômatos EC1 e AI.

A figura 23 mostra as restrições do autômato EC1 com relação a outros autômatos. Para que seja possível a ocorrência do eventos EC1_VirDir que ativada a rotação da Esteira Classificadora 1, é necessário a ocorrência do evento AI_01 que libera os ingredientes por três vezes.

(C) Restrições e a relação entre os autômatos AN e EC1.

A figura 24 mostra as restrições do autômato AN e como este se relacionam com outros componentes. É possível observar que somente após a ocorrência do evento EC1_VirDir, que é realizado pela EC1, se torna possível a corência do evento AN_01.

(D) Restrições e a relação entre os autômatos MT, AI e EC1.

A figura 25 mostra as restrições do autômato MT modelado no *software* Supremica e sua relação com outros autômatos. É possível notar que o evento MT_Liga que ativa

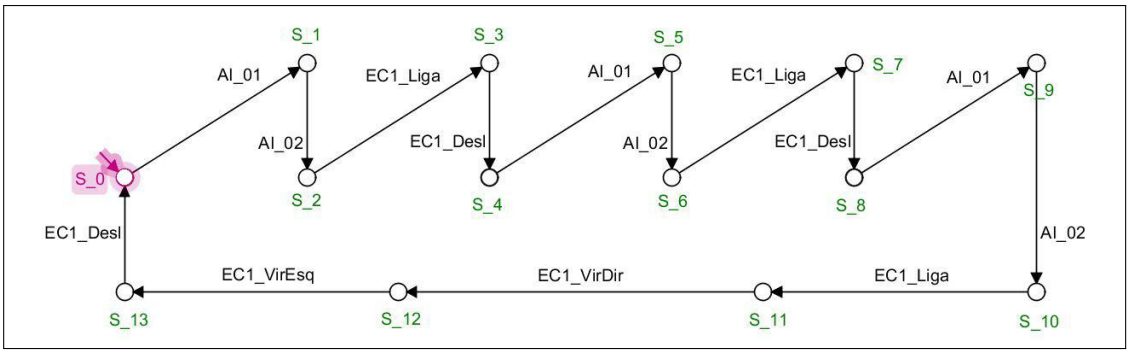

Figura 23 – Modelo das restrições do autômato EC1 no Supremica
Fonte: Autoria própria

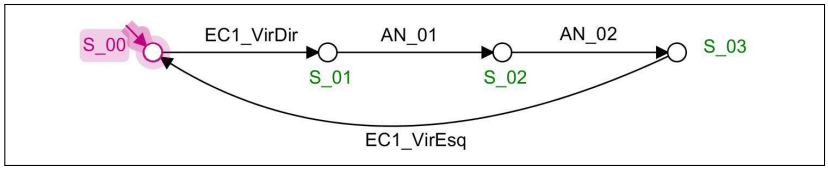

Figura 24 – Modelo das restrições do autômato AN no Supremica
Fonte: Autoria própria

o misturador só pode ocorrer após o evento EC1_Desl, assim somente depois que EC1 for desativadas que o misturador podera ser ligado. E somente após ocorrer o evento MT_Desl é que ocorra novamente o evento AI_01.

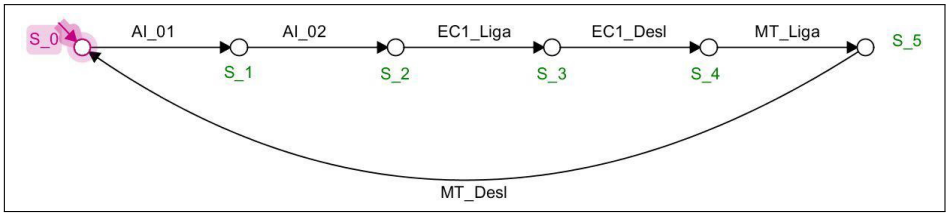

Figura 25 – Modelo das restrições do autômato MT no Supremica
Fonte: Autoria própria

(E) Restrições e a relação entre os autômatos FN e MT.

A figura 26 mostra as restrições do autômato FN modelado no software Supremica e se relacionamento com o autômato MT. É possível observar que somente após a ocorrência do MT_Desl se torna possível ocorrer o evento FN_01.

(F) Restrições e a relação entre os autômatos EC2 e FN.

A figura 27 mostra as restrições do autômato EC2 com relação ao autômato FN. Onde é possível observar que para ativar a rotação da Esteira Classificadora 2 é necessário a ocorrência do evento FN_02 que desativa o forno duas vezes. Esta

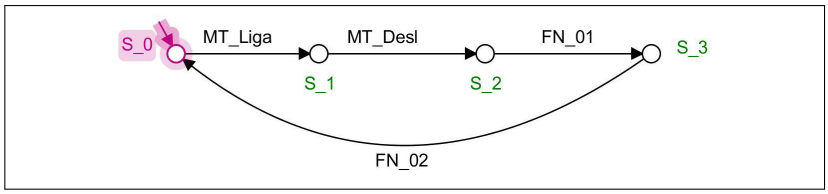

Figura 26 – Modelo das restrições do autômato FN no Supremica
Fonte: Autoria própria

restrição foi imposta para que parte dos produtos passem pela máquina que aplica cobertura.

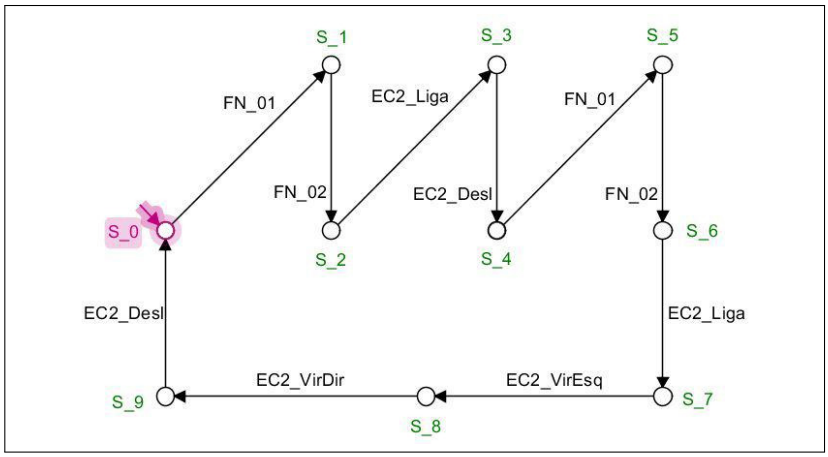

Figura 27 – Modelo das restrições do autômato EC2 no Supremica
Fonte: Autoria própria

(G) Restrições e a relação entre os autômatos AC e EC2.

A figura 28 mostra as restrições e a relação entre os autômatos AC e EC2, modeladas no *software* Supremica. É é possível observar que somente após a ocorrência do evento EC2_VirDir proveniente da esteira classificadora 2 sera possível ocorrer o evento AC_01.

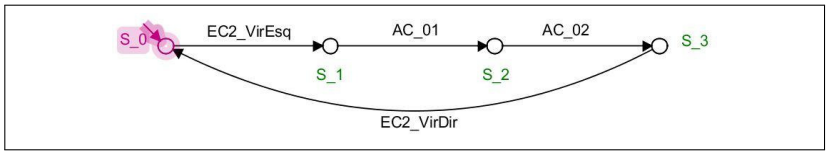

Figura 28 – Modelo das restrições autômato AC no Supremica
Fonte: Autoria própria

4.3 CARACTERÍSTICAS DO MODELO PROPOSTO

Seguindo o passo 1 da seção 2.1.4, pode ser feita a composição síncrona dos autômatos mostrados na seção 4.2. Este processo foi feito no *software* Grail para expor a explosão combinacional gerada pelo sistema. A tabela 1 apresenta o aumento da complexidade do autômato resultante da composição síncrona. Assim o autômato **Planta** é gerado pela composição síncrona dos autômatos ET, AI, EC1, AN, MT, FN, EC2 e AC, possui 8100 estados e 70200 transições.

Tabela 1 – Resultados da composição síncrona dos autômatos dos modelos

Automato								Est Inicial	Est Final	Estados	Eventos	Σ
ET	AI							1	2	10	20	6
ET	AI	EC1						1	2	30	100	10
ET	AI	EC1	AN					1	2	90	390	13
ET	AI	EC1	AN	MT				1	2	180	960	15
ET	AI	EC1	AN	MT	FN			1	4	900	5700	19
ET	AI	EC1	AN	MT	FN	EC2		1	4	2700	20700	23
ET	AI	EC1	AN	MT	FN	EC2	AC	1	4	8100	70200	26

Fonte: Autoria própria

O passo 2 também referente a seção 2.1.4 foi realizando nas etapas acima com a definição das especificações a serem respeitadas. A tabela 2 apresenta alguns detalhes da composição das especificações.

Tabela 2 – Resultados da composição síncrona das restrições dos modelos

Automato								Est Inicial	Est Final	Estados	Eventos	Σ
ET	AI	EC1	AN	MT	FN	EC2	AC	1	12	653	1280	20

Fonte: Autoria própria

Seguindo para o passo 3, obtemos síntese de uma lógica de controle não bloqueante e ótima.

5 DESENVOLVIMENTO DE UMA FERRAMENTA DE SIMULAÇÃO DE SEDS

Dentro dos objetivos propostos, para trabalhar com a ideia de falhas em sistemas de automação e facilitar a compreensão dos usuários com relação ao controle obtido através de autômatos, será apresentado nesta seção o *software* (S2DFA2) Sistema de Simulação e Diagnóstico de Falhas Aplicado à Automação, desenvolvido para permitir a simulação de um processo de automação, modelando suas máquinas e o controle que atua sobre elas.

5.1 PROJETO S2DFA2

O S2DFA2 tem como objetivo tornar-se uma ferramenta de controle e diagnóstico de falhas em uma simulação definida. Esta proposta apresenta-se como uma alternativa a *softwares* como o ITS-PLC, que não permitem alterações, ou seja, não possibilita a introdução de novas características às máquinas, como novas etapas, sensores, falhas, etc. Este fato limita os testes voltados a diagnóstico de falhas, uma vez que as falhas nesse sistema estão vinculadas ao acionar de um botão que liga ou desliga um equipamento, sem que ocorra um evento aleatório, como é a característica de uma falha.

5.2 TELAS

O S2DFA2 conta com uma tela principal mostrada na figura 29, que contém um menu de acesso a todas as partes do programa. Este menu está subdividido em:

- **Tela**
- **Opções**
- **Ajuda**
- **Sair**

O S2DFA2 foi definido desta forma para agrupar as características pertinentes a criação do autômato e da restrição que representa cada máquina dentro do menu telas. Os detalhes pertinentes ao controle da simulação como importação de controle, composição síncrona, função trim e SupC encontram-se juntos no menu Opção.

5.2.1 Menu - Tela

A opção **Tela** do menu principal apresentanda na figura 30 conta com as seguintes opções:

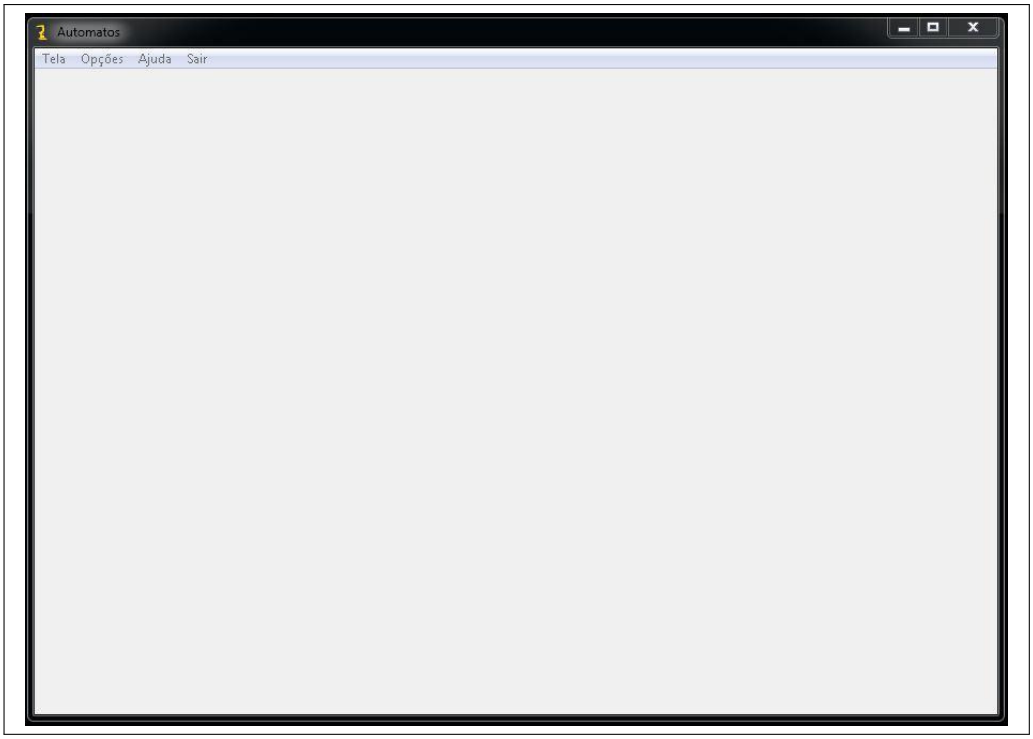

Figura 29 – Tela Principal
Fonte: Gerada pelo *software* S2DFA2

- **Máquinas**
- **Montar máquinas**
- **Simulação**

5.2.1.1 Tela - Máquinas

Através da opção **Máquinas** do menu **Tela** é possível cadastrar as máquinas definindo seu nome e a abreviação que será usada no controle. Como pode ser visto na figura 30, a tela é composta por dois campos de textos: um para o nome da máquina e o outro para a abreviação do nome, e conta ainda com 7 botões para tratamento e cadastro das informações, são eles:

Novo Limpa os campos para receber novas informações.

Alterar Altera uma máquina já cadastrada.

Salvar Salva as novas máquinas cadastradas ou as alterações realizadas.

Cancelar Cancela as alterações realizadas.

Excluir Exclui uma máquina que foi localizada.

Pesquisar Pesquisa por máquinas tanto para a alteração quanto para a exclusão.

Fechar Sai da tela Cadastro de Máquinas.

Figura 30 – Tela - Máquinas
Fonte: Gerada pelo *software* S2DFA2

5.2.1.2 Tela - Montar máquinas

Através da opção **Montar máquinas** do menu **Tela** é possível configurar o autômato e definir os eventos observáveis, não observáveis, controláveis, não controláveis, falhas e outros mais. É também nesta tela que é feito o cadastro das restrições pertinentes a cada autômato, conforme observado na figura 31.

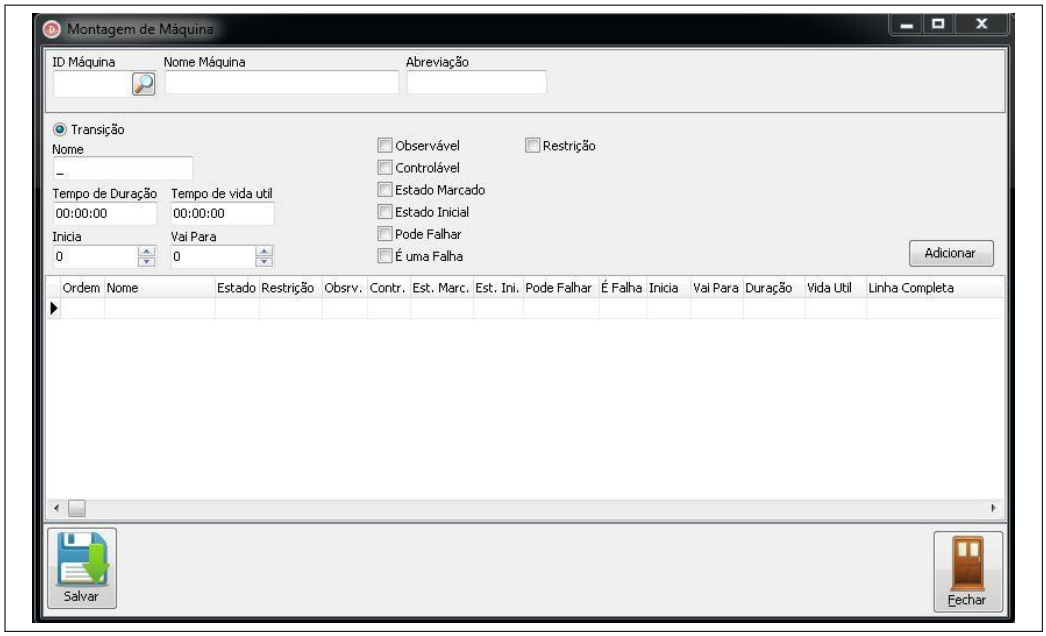

Figura 31 – Tela - Montar máquinas
Fonte: Gerada pelo *software* S2DFA2

5.2.1.3 Tela - Simulação

Através da opção **Simulação** do menu **Tela** é possível visualizar a simulação gerada. Nesta já estão operando os controles e o sistema de identificação de falhas, para o caso do controle desenvolvido contemplar a possibilidade de falhas. Assim, estas serão localizadas e sinalizadas na tela através de pequenos semáforos que sinalizam verde para máquina funcionando e vermelho para falhas, conforme representação visível na figura 32.

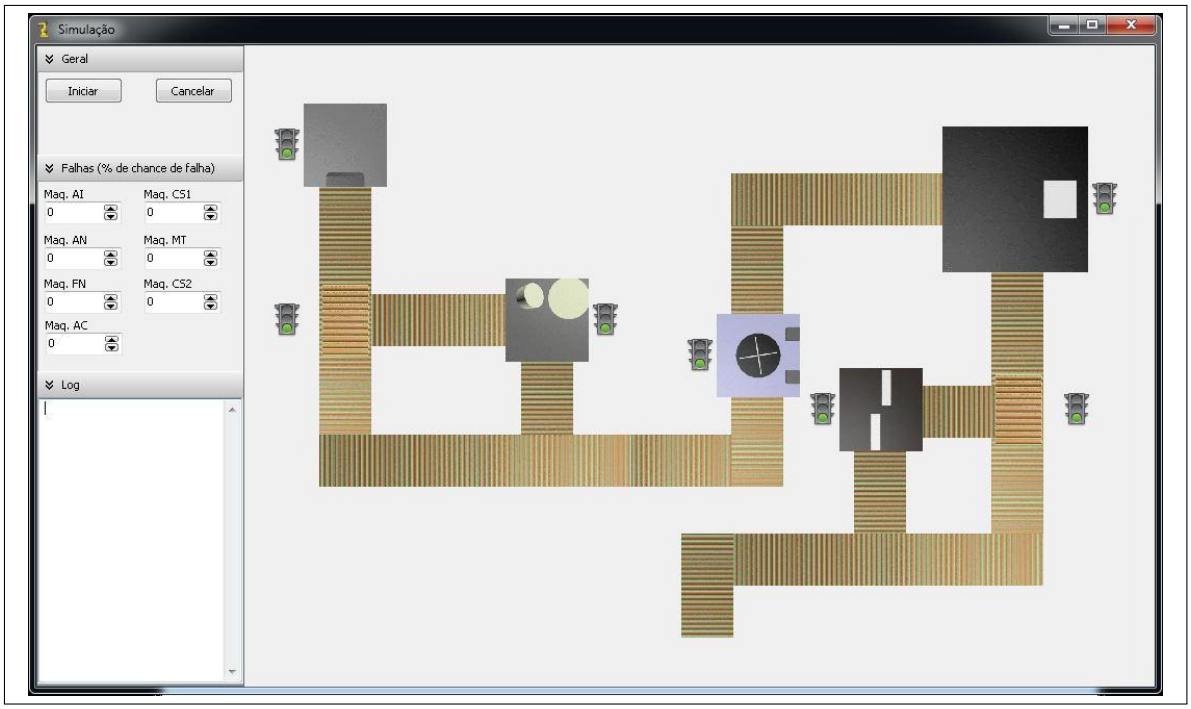

Figura 32 – Tela - Simulação
Fonte: Gerada pelo *software* S2DFA2

A tela **Simulação** conta ainda com dois botões ligados ao início e ao término da simulação, e um quadro contendo a porcentagem de chance de cada uma das máquinas falhar e por fim um quadro contendo o log com eventos ocorridos.

5.2.2 Menu - Opção

O menu **Opção** está voltado para o tratamento das informações destinadas ao controle da simulação, sejam elas geradas pelo cadastro de máquinas ou externamente seguindo o padrão do Grail, e depois importadas para o sistema para serem aplicadas na simulação. As opções do menu **Opções** são:

Importar

Exportar

Sincronizar

5.2.2.1 Opção - Importar

Através da opção **Importar** do menu **Opções** é introduzido no S2DFA2 um autômato gerado em outros softwares, desde que estes sigam o padrão do Grail.

5.2.2.2 Opção - Exportar

Através da opção **Exportar** do menu **Opções** é possível criar os arquivos do tipo texto, que serão utilizados posteriormente pelo *software* Grail para gerar o controle da simulação.

5.2.2.3 Tela - Sincronizar

Através da opção **Sincronizar** do menu **Opções** é possível indicar o local onde encontram-se os programas do *software* Grail que fazem o tratamento dos autômatos. Neste ponto é realizado o passo 3 da seção 2.1.4. Ao clicar no botão **Sincronizar** é executada a composição dos autômatos que modelam a planta e as restrição criando uma lógica de controle não bloqueante e ótima. Esta tela pode ser vista na figura 33.

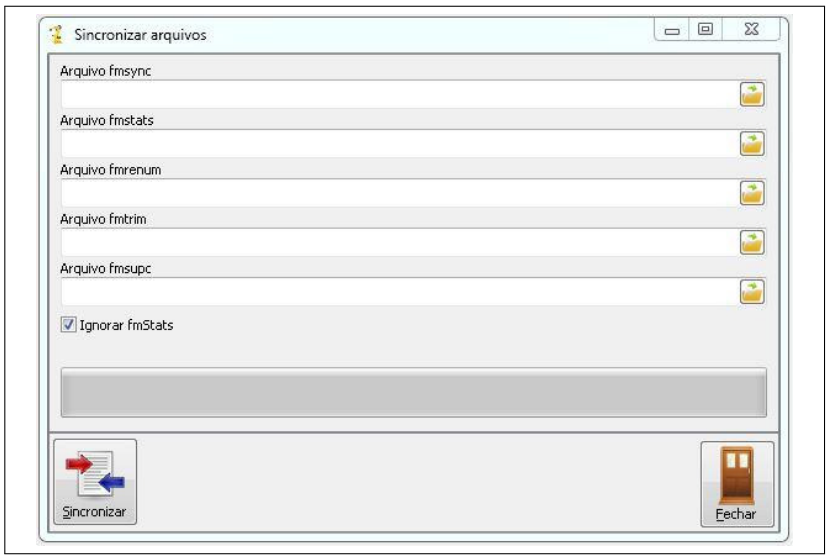

Figura 33 – Opções - Sincronizar
Fonte: Gerada pelo *software* S2DFA2

Através da opção **Importar arquivo** do menu **Opção** é possível importar de um arquivo de controle no padrão do Grail. A figura 34 mostra a tela gerada após o click na opção importar.

5.2.3 Menu - Ajuda

A opção **Ajuda** da tela principal abre o manual do S2DFA2 contendo as instruções de funcionamento, limitações e outros detalhes.

Figura 34 – Importar
Fonte: Gerada pelo *software* S2DFA2

5.2.4 Menu - Sair

A última opção do menu principal e a opção **Sair** tem como função finalizar o S2DFA2 gravando as informações já cadastradas em um banco de dados.

5.3 TESTE DO SIMDIF

Tendo como base os modelos obtidos para os subsistemas e as restrições de coordenação apresentadas na seção 4.2, nesta seção utilizaremos o S2DFA2 para validar as propóstas.

5.3.1 Cadastro das máquinas

Através do menu **Tela** na opção **Máquinas** é possível fazer o cadastro.

Deve-se digitar nos campos **Nome da máquina** o nome que deseja dar para a máquina, e no campo **Abreviação** deve ser digitado a abreviação para o nome dado para máquina. Conforme exemplo apresentando na figura 35, após o preenchimento basta clicar no botão salvar.

Figura 35 – Cadastro de máquinas
Fonte: Gerada pelo *software* S2DFA2

Este procedimento deve ser feito com todas as máquinas da tabela 3. Ao término,

pode-se clicar no botão consulta e obtera a listagem conforme observa-se figura 36

Tabela 3 – Lista de máquinas e abreviaturas

Nome da máquina	Abreviação
Esteira Transportadora	ET
Adiciona Ingredientes	AI
Esteira Classificadora 1	EC1
Adiciona Nozes	AN
Misturador	MT
Forno	FN
Esteira Classificadora 2	EC2
Adiciona Cobertura	AC

Fonte: Autoria própria

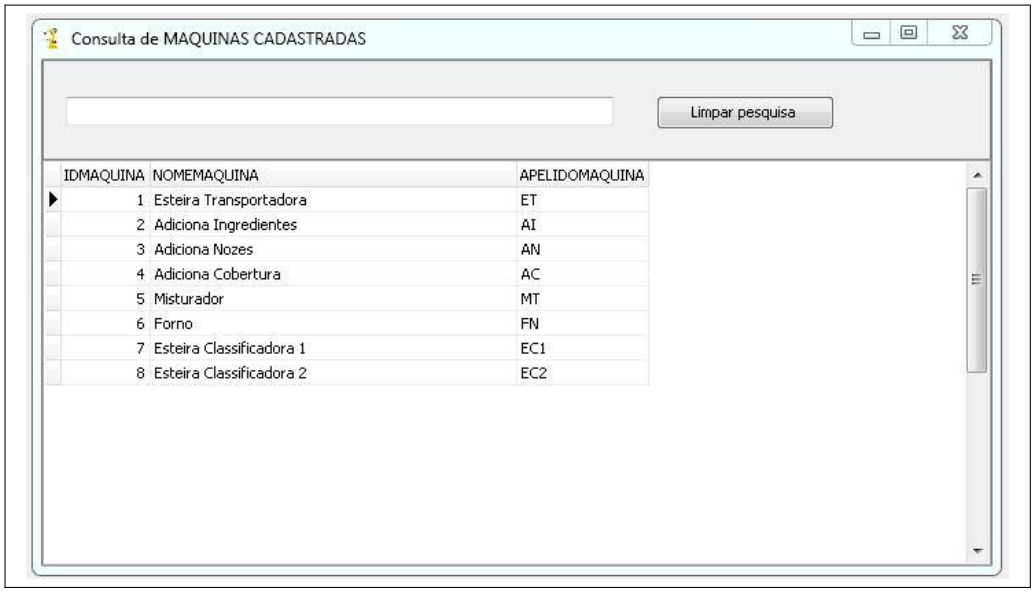

Figura 36 – Consulta de máquinas cadastradas
Fonte: Gerada pelo *software* S2DFA2

5.3.2 Montagem das máquinas

Esta parte do processo é feita através do menu **Tela** opção **Montar máquina** Observando a tela apresentada na seção 5.2 figura 31 verifica-se uma lupa próxima ao campo **ID Máquina** que ao ser clicado, abre a tela **Consulta de máquinas cadastradas** apresentada na figura 36. Basta clicar duas vezes sobre a máquina escolhida que o S2DFA2 retornará para a tela anterior com as especificações da máquina a serem preenchidas. Na figura 37 toma-se como exemplo o autômato ET. O **Nome** já contém a abreviatura da máquina selecionada, basta preencher com as informações do autômato tomado como exemplo, assim o campo nome para ao primeiro evento fica **ET_Liga** o campo **Início** deve

ser preenchido com o estado de origem do evento e o campo **Vai para** deve ser preenchido com o destino do evento, assim o campo **Início** recebe o valor 0 e o campo **Vai para** recebe o valor 1.

O segundo passo refere-se às características do evento que tem como opção os itens do quadro 20:

X	Observável
X	Controlável
	Estado marcado
X	Estado inicial
	Pode falhar
	É uma falha

Quadro 20 – Opções dos eventos.
Fonte: Autoria própria

Como o evento **ET_Liga** é observável e controlável os dois primeiro itens devem ser marcados. Como o **Estado marcado** refere-se a um estado destino esta opção deve ser marcada quando o campo **Vai para** for um estado marcado. A opção **Estado inicial** refere-se a um estado de origem de um evento logo esta deve ser marcada quando o campo **Início** for um **Estado inicial**, que é o caso deste evento. A opção **É uma falha** deve ser usada quando se está modelando um evento de falha, como não é o caso esta opção, não deve ser marcada.

Este mesmo raciocínio deve ser aplicado para todos os eventos que formam todos os autômatos, desta forma a modelagem do autômato ET fica conforme a figura 37

5.3.3 Montagem das restrições no S2DFA2

No S2DFA2 as restrições ficam vinculadas às máquinas, ou seja, elas são cadastradas junto com os eventos das máquinas. Isto se deve ao fato de uma restrição ter como característica sincronizar os eventos que podem ocorrer em uma máquina com eventos que estão ocorrendo em outra máquina, sendo possível vincular a qualquer uma das máquinas que se relacionam. A figura 38 mostra como fica o cadastro da máquina **Adiciona Ingredientes** após o cadastro da restrição mostrado na figura 22 seção 4.2.2. onde é possível observar que a coluna restrição se encontra marcada a partir do campo 6.

Este processo deve ser realizado para todas as restrições, com isso conclui-se o cadastro e a configuração das restrições.

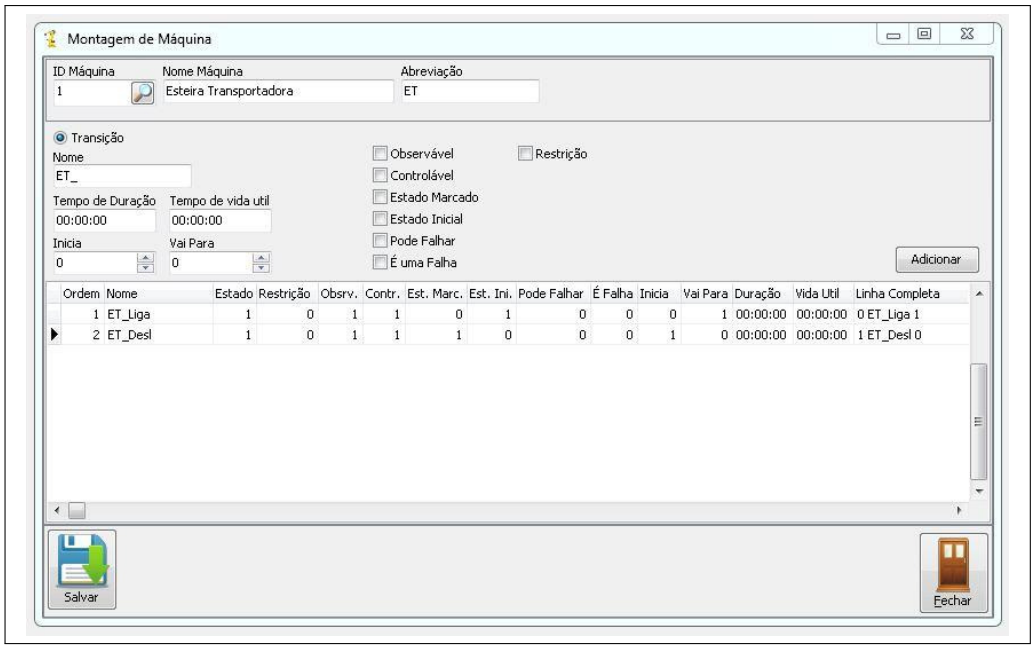

Figura 37 – Autômato ET no S2DFA2
Fonte: Gerada pelo *software* S2DFA2

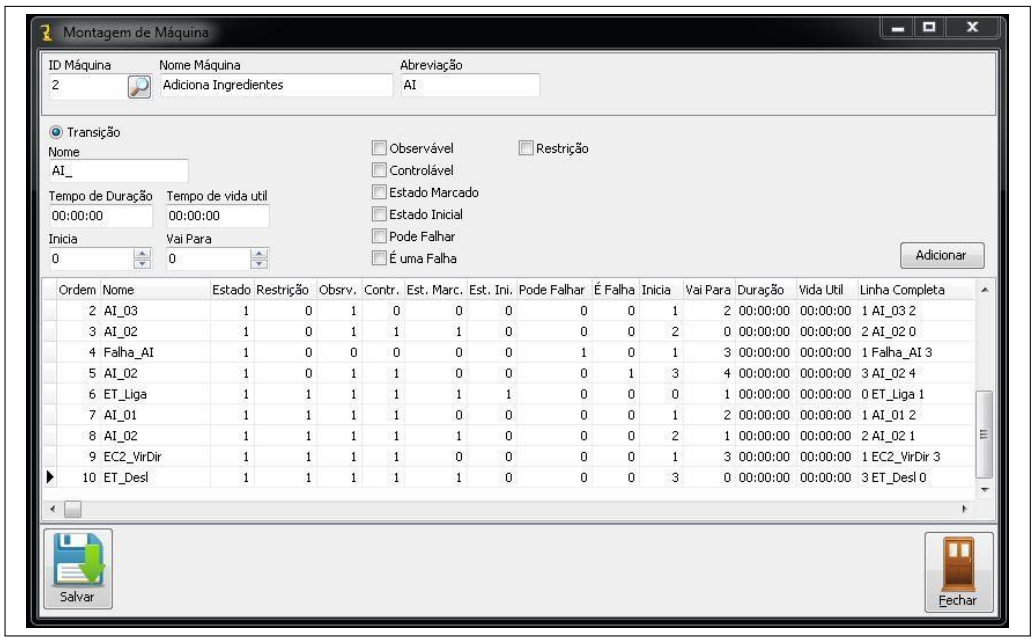

Figura 38 – Máquina Adiciona Ingredientes com cadastro de restrições
Fonte: Gerada pelo *software* S2DFA2

5.3.4 Criação de arquivos

O próximo passo é a criação dos arquivos tipo texto para uso no Grail. Esta tarefa é realizada no menu **Opções** na opção **Exportar** mostrada na seção 5.2.2.2. Este processo cria na pasta **Exp** três grupos de arquivos o primeiro mostrado na figura 39 contem um arquivo para cada autômato apresentados nas seções 4.2.1, sendo cada um referente a uma máquinas.

Figura 39 – Lista de arquivos dos autômatos
Fonte: Gerada pelo *software* S2DFA2

O segundo grupo mostrado na figura 40 refere-se os autômatos que restringe o comportamento da planta apresentados nas seções 4.2.2, como eles são cadastrados dentro de cada máquina recebem o nome da máquina mas isso não significa que ele pertence somente a essa mais pois estes arquivos mostram o relacionamento entre as máquina.

Figura 40 – Lista de arquivos das restrições
Fonte: Gerada pelo *software* S2DFA2

E o ultimo grupo mostrado na figura 41 contem os eventos não controláveis referente a cada máquina.

Figura 41 – Lista de arquivos dos eventos não controláveis
Fonte: Gerada pelo *software* S2DFA2

Estes arquivos serão usados para gerar lógica de controle não bloqueante ótima, através do processo apresentado na seção 2.1.4.

O processo dura alguns segundos, e sempre que houver alguma alteração basta repetir o processo que os arquivos anteriores são eliminados e substituídos pelos novos.

5.3.5 Sincronização dos arquivos

Segundo Cury (2001) o autômato **Strim** pode ser usado com as especificações que a planta deve obdecer. Ele e gerado através dos arquivos mostrados na subseção anterior, e pode ser criado pelo S2DFA2 ou pelo próprio Grail. No S2DFA2 basta acessar o menu **Opções** opção **Sincronizar** mostrada na subseção 5.2.2.3. Através do Grail utiliza-se a metodologia apresentada na seção 2.1.4. Quando realizado pelo S2DFA2 o processo acrescentará novos arquivos a pasta **Exp** como é mostrado na figura 42

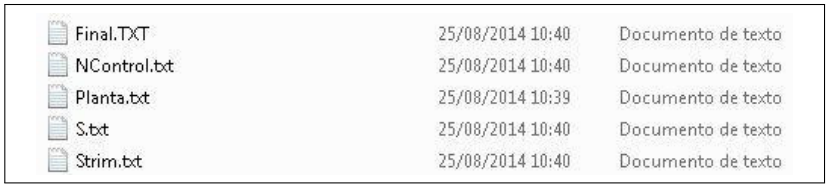

Figura 42 – Lista de arquivos de controle
Fonte: Gerada pelo *software* S2DFA2

A figura 43 apresenta parte do arquivo **Strim.txt** que contem as especificações a serem seguidas pelos sistemas.

```
(START) |- 0
0 ET_Liga 1
1 AI_01 2
2 AI_03 3
3 AI_02 4
4 EC1_Liga 5
5 EC1_Desl 6
6 MT_Liga 7
7 MT_Desl 8
8 AI_01 9
8 FN_01 10
9 AI_03 11
9 FN_01 12
9 Falha_AI 13
10 AI_01 12
10 FN_03 14
11 AI_02 15
11 FN_01 16
12 AI_03 16
12 FN_03 17
12 Falha_AI 18
13 AI_02 19
13 FN_01 18
14 AI_01 17
14 FN_02 20
15 FN_01 21
15 EC1_Liga 22
16 AI_02 21
16 FN_03 23
17 AI_03 23
17 FN_02 24
17 Falha_AI 25
18 AI_02 26
18 FN_03 25
19 FN_01 26
19 EC1_Liga 27
20 AI_01 24
20 EC2_Liga 28
```

Figura 43 – Parte do arquivo que apresenta as especificações do modelo
Fonte: Gerada pelo *software* S2DFA2

Planta.txt Contem a composição síncrona dos autômatos que representam as máquina.

NControl.txt Contem a composição síncrona de todos os eventos não controláveis da planta.

S.txt Contem a composição síncrona entre autômatos que representam as máquinas e os autômatos que representa as restrições.

Strim Contem as especificações a serem obedecidas pela planta.

Final.txt Contem o SupC que pode ser usado como especificação quando for necessário.

5.3.6 Importação dos arquivos

O arquivo gerado na subseção anterior agora deve ser importado para o sistema e será usado no controle, este procedimento é realizado através da opção **Importar arquivo** do menu **Opções**, conforme mostrado na subseção 5.2.2.1. Após esse processo o arquivo já se encontra na base de dados do programa, sendo possível inicializar a simulação.

5.3.7 Simulação

Com todas as etapas concluídas, basta clicar na opção **Tela**, escolher a o opção **Simulação** e clicar no botão **Iniciar**, imediatamente tem início a simulação que vai mostrando na aba de log os eventos que ocorreram no sistema, sendo possível definir as chances de ocorrência de falhas quando se desejar. A figura 44 mostra o início da simulação.

Figura 44 – Tela de simulação
Fonte: Gerada pelo *software* S2DFA2

Os produtos são fabricados dependendo das opções definidas nas restrições, em caso de falha, o semáfaro próximo a máquina muda da cor verde para o vermelho. Na

figura 45 mostra um LOG do S2DFA2 com os eventos que ocorreram no sistema, nesta imagem é possível verificar o que ocorre com o S2DFA2 após a ocorrência de falhas.

Toda vez que o sistema encontra um evento de falhas ele verifica a qual máquina esse evento pertence e avalia no quadro **Falhas(% de chance de falha)**, que pode ser visto na figura 45, qual a probabilidade que ocorra o evento, calcula as chances e de posse da resposta executa ou não o evento.

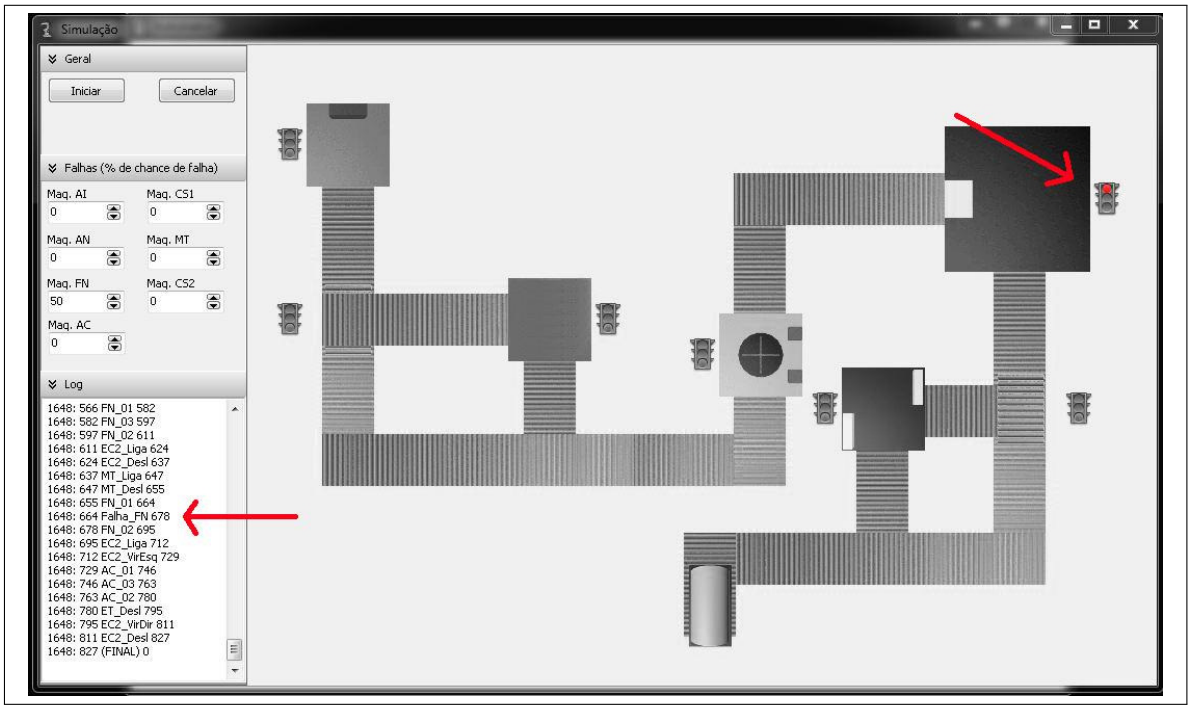

Figura 45 – Tela de simulação apresentando uma falha
Fonte: Gerada pelo *software* S2DFA2

É possível observar na listagem de log da figura 45 que o S2DFA2 encontrava-se no estado **655** quando ocorre o evento **FN_01** que muda o estado do forno saindo do estado **Inicial** e indo para o estado **Ativar forno**. Assim, o sistema sai do estado **655** para o estado **664** que apresenta três opções conforme mostra a figura 46:

A cada evento o sistema atualiza uma tabela que contem as possíveis sequências de eventos que podem ser executados por uma máquina. Se ocorrer uma evento que não faz parte do eventos esperados o sistema interpreta como a ocorrência de uma falha. Este método é baseado no modelo apresentado na seção 3. Parte do código que realiza esta função é mostrado no apêndice A.

AI_01 Ativa a máquina **Adiciona Ingredientes** e fica postergada a decisão de falha para o estado **673** e dá início a um novo produto.

FN_03 Termina o processo de ativação do **Forno** possibilitando a conclusão das funções desta máquina, e o controle segue para o estado **677**.

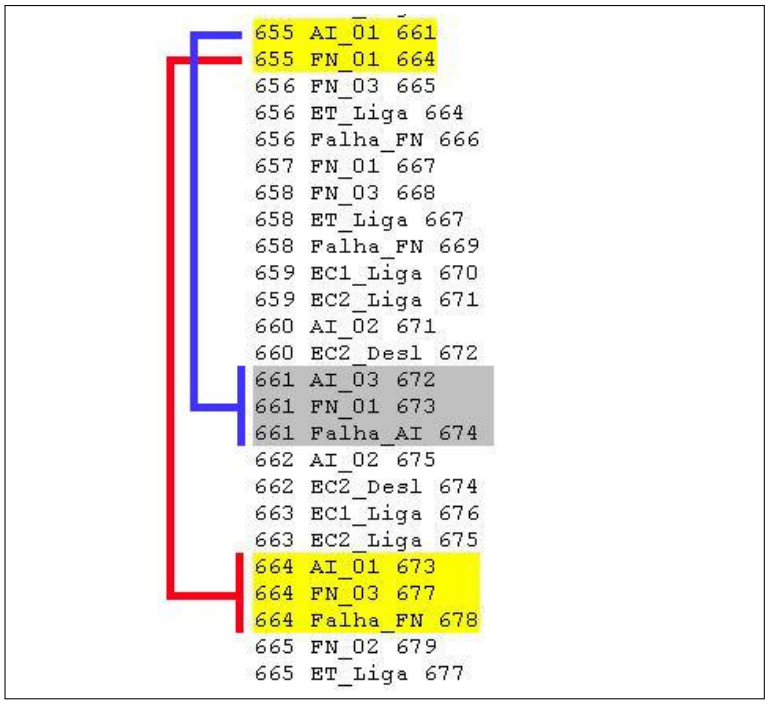

Figura 46 – Imagem parcial da linguagem de controle
Fonte: Gerada pelo *software* S2DFA2

Falha_FN Que representa uma falha na máquina **Forno** e a partir deste ponto serão concluídas as tarefas e o sistema será finalizado.

É possível observar na figura 46 que, se o evento escolhido no estado **655** fosse **AI_01** este levaria ao estado **661** com a máquina **Adiciona Ingredientes** sendo ativada, logo, esta poderia apresentar falhas através do evento **Falha_AI**.

6 RESULTADOS

O S2DFA2 consegue gerar uma simulação através dos autômatos informados, ficando suas restrições limitadas ao problema da explosão combinatória gerada pelos número de estados e eventos dos autômatos que representam cada máquina. Para mostrar o quão significativo são os problemas relacionado com a explosão combinacional foram criados alguns modelos para teste. As tabelas 7, 8 e 9 são uma representação parcial do resultado obtidos através do comando **fmstats** pertecente ao Grail. Os teste foram realizados buscando avaliar o tempo na composição síncrona dos autômatos ET, CS1, AN, MT, FN, EC2, AC; para a tabela 7, todos os autômatos têm dois estados e dois eventos; para a tabela 8, os autômatos têm três estados e três eventos; e para a tabela 9 os autômatos têm quatro estados e quatro eventos. Este teste foi realizado em um computador com as características apresentadas na figura 47.

Tabela 4 – fmsync para autômatos com dois estados e eventos PC-01

Descrição	Quant
Estado inicial	1
Estado finai	1
Estados	256
Estado bloqueados	0
Eventos	3048
Σ	16
Tempo gasto na geração da informação	2s

Fonte: Autoria própria

Tabela 5 – fmsync para autômatos com três estados e eventos PC-01

Descrição	Quant
Estado inicial	1
Estado final	1
Estados	6561
Estado bloqueados	0
Eventos	52488
Σ	24
Tempo gasto na geração da informação	27s

Fonte: Autoria própria

Os mesmos testes foram realizados em computador com características inferiores, suas informações são apresentadas na figura 48.

É possível observar que o problema causado pela explosão combinacional acaba tornando inviável modelar máquinas com muitos estados. O quadro 21 mostra um compa-

Tabela 6 – fmsync para autômatos com quatro estados e eventos PC-01

Descrição	Quant
Estado inicial	1
Estado final	1
Estados	65536
Estado bloqueados	0
Eventos	524288
Σ	32
Tempo gasto na geração da informação	49m:38s

Fonte: Autoria própria

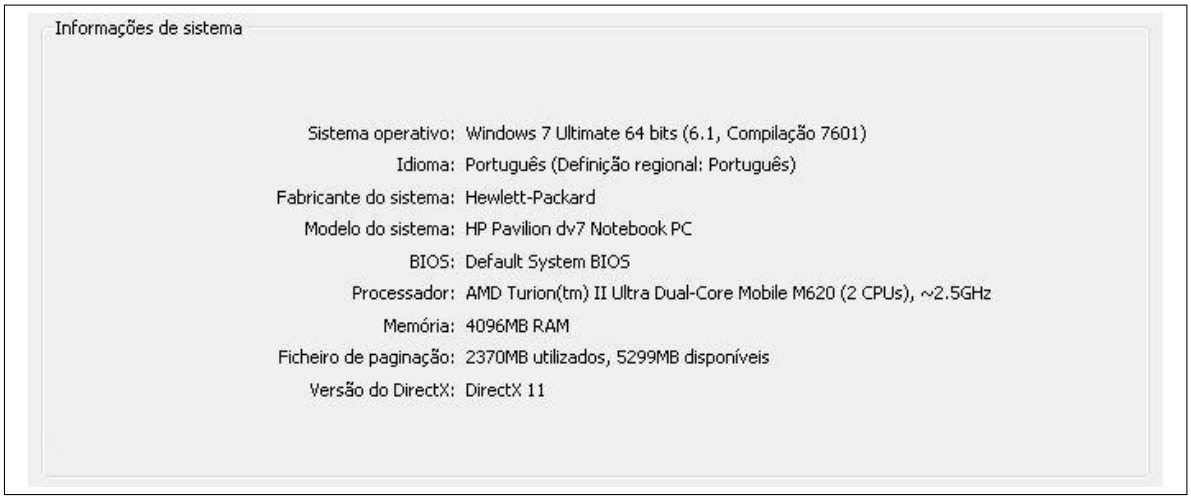

Figura 47 – Características do computador de teste PC-01.
Fonte: Gerada pelo Software DXDIAG

Tabela 7 – fmsync para autômatos com dois estados e eventos PC-02

Descrição	Quant
Estado inicial	1
Estado finai	1
Estados	256
Estado bloquiados	0
Eventos	3048
Σ	16
Tempo gasto na geração da informação	3s

Fonte: Autoria própria

rativo com relação ao tempo e o número de estados e eventos.

Tabela 8 – fmsync para autômatos com três estados e eventos PC-02

Descrição	Quant
Estado inicial	1
Estado final	1
Estados	6561
Estado bloquiados	0
Eventos	52488
Σ	24
Tempo gasto na geração da informação	39s

Fonte: Autoria própria

Tabela 9 – fmsync para autômatos com quatro estados e eventos PC-02

Descrição	Quant
Estado inicial	1
Estado final	1
Estados	65536
Estado bloquiados	0
Eventos	524288
Σ	32
Tempo gasto na geração da informação	1h:13m:20s

Fonte: Autoria própria

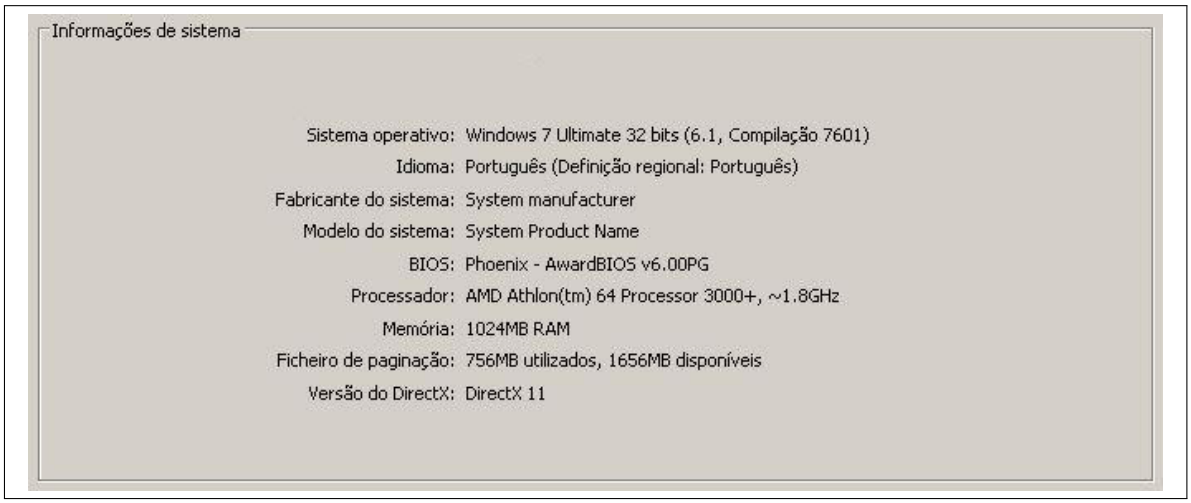

Figura 48 – Características do computador de teste PC-02.
Fonte: Gerada pelo Software DXDIAG

	Estados e Eventos		
Descrição	2	3	4
PC-01	2s	27s	49m:38s
PC-02	3s	39s	1h:13m:20s

Quadro 21 – Relação entre tempo e estados e eventos.
Fonte: Autoria própria

7 CONCLUSÃO E TRABALHOS FUTUROS

7.1 CONCLUSÃO

Comportamento do S2DFA2 :

Foram realizados dois tipos de testes:

1. Sem falhas:

   - Com autômatos contendo 4, 3, e 2 estados.

   Para este caso o S2DFA2 funcionou de forma ininterrupta.

2. Apresentando falhas, mas evitando os bloqueios através das restrições (modelo detalhado nesta dissertação):

   - Com autômatos contendo 5, 4, 3, e 2 estados.

   Para este caso as falhas foram diagnosticadas no momento em que o S2DFA2 entrou em um "caminho"de falha, conforme definição apresentada por Basilio, Carvalho e Moreira (2010) na seção 3, a partir deste ponto a máquina que apresentou o problema é sinalizada.

Estes mostraram que o autômato de controle consegue controlar a simulação, desde que as restrições eliminem os casos de bloqueio.

Flexibilidade :

A possibilidade de modelagem das máquinas e das restrições de forma livre permite criar um grande número de estratégias de controle, com variação de eficiência. Esta flexibilidade permite ainda adequar o processo às necessidades, produzindo um único produto final ou gerando dois, três ou quatro produtos na proporção desejada.

Falhas :

As falhas podem ser modeladas nos autômatos que modelam as máquinas mas isso não obriga que essa seja avaliada na execusão da simulação pois é possível usar a opção de probabilidade de falhas. Neste caso esta funcionalidade permite analisar quando uma falha ocorre em apenas uma das máquinas. Esta definição é feita no processo de modelagem e depois observada na simulação. Quando se usa os campos de probabilidade de falhas na tela **simulação** é possível alterar as chances de que ocorra uma falha em tempo real, sem que haja a necessidade de uma nova modelagem.

7.2 TRABALHOS FUTUROS

- Codificar os algorítmos de composição síncrona, trim e supC dentro do S2DFA2, visando reduzir o tempo dispensado para a obtenção da máxima linguagem controlável, através do uso dos múltiplos CORE dos processadores modernos.
- Projetar uma animação 3D, proporcionando uma melhor qualidade de imagem para a simulação.
- Implementar no S2DFA2 uma tela para modelagem de autômatos de forma visual.

REFERÊNCIAS

AKESSON, K.; FABIAN, M.; FLORDAL, H.; MALIK, R. Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems. In: IEEE. **Discrete Event Systems, 2006 8th International Workshop on**. Ann Arbor, MI, 2006. p. 384–385.

AMARAL, J. L. M. do. **Sistemas imunológicos artificiais aplicados à detecção de falhas**. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, 2006.

BANKS, J. Simulation languages and simulators. In: ACM. **Proceedings of the 24th conference on Winter simulation**. New York, NY, 1992. p. 88–96.

BASILIO, J. C.; CARVALHO, L. K.; MOREIRA, M. V. Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos. **Revista Controle & Automação**, SciELO Brasil, v. 21, n. 5, p. 510–533, 2010.

BELHOT, R. V.; FIGUEIREDO, R. S.; MALAVÉ, C. O. O uso da simulação no ensino de engenharia. In: ABENGE. **Congresso Brasileiro de Ensino de Engenharia, XXIX COBENGE**. Porto Alegre, RS, 2001. p. 445–451.

BINH, P. T. T.; TUYEN, N. D. Fault diagnosis of power system using neural petri net and fuzzy neural petri net. In: **Power India Conference, 2006 IEEE**. New Delhi: [s.n.], 2006. p. 5 pp.–.

CAO, Y.; YING, M. Observability and decentralized control of fuzzy discrete-event systems. **Fuzzy Systems, IEEE Transactions on**, v. 14, n. 2, p. 202–216, April 2006. ISSN 1063-6706.

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

CHWIF, L.; MEDINA, A. C. **Modelagem e simulação de eventos discretos**. São Paulo, SP: Afonso C. Medina, 2006.

CONTANT, O.; LAFORTUNE, S.; TENEKETZIS, D. Failure diagnosis of discrete event systems: the case of intermittent faults. In: IEEE CONTROL SYSTEMS SOCIETY. **Decision and Control, 2002, Proceedings of the 41st IEEE Conference on**. Las Vegas, NV, 2002. v. 4, p. 4006–4011 vol.4. ISSN 0191-2216.

CUNHA, A. E. C. d. **Contribuição ao Controle Hierarquico de Sistemas a Eventos Discretos**. Tese (Doutorado) — UNIVERSIDADE FEDERAL DE SANTA CATARINA, 2003.

CURY, J. E. R. Teoria de controle supervisóriode sistemas a eventos discretos. **V Simpósio Brasileiro de Automação Inteligente**, v. 82, n. 5, Nov 2001.

DUDA, R. O.; STORK, D. G.; HART, P. E. **Pattern Classification**. New York, NY: Wiley-Interscience, 2000. ISBN 0471056693 / 0-471-05669-3.

GABARDO, I.; CARNEIRO, M.; FALCÃO, L.; MENICONI, M.; BARBAND, S.; PLATTE, E. Oil spills in a tropical country-brazilian case studies. In: AMERICAN PETROLEUM INSTITUTE. **International Oil Spill Conference**. Vancouver, BC, 2003. v. 2003, n. 1, p. 1039–1049.

HAMMOURI P. KABORE, S. O. J. B. H. Failure diagnosis and nonlinear observer. application to a hydraulic process. In: ELSEVIER. **Journal of the Franklin Institute**. Villeurbanne, RH, 2002.

INMAN, D. J.; FARRAR, C. R.; LOPES, V.; STEFFEN, V. **Damage prognosis**. London: Wiley Online Library, 2005.

JR, R. G. T.; MACHADO, M. A. S.; SOUZA, R. C. Previsão de séries temporais de falhas em manutenção industrial usando redes neurais. **Engevista**, v. 7, n. 2, 2010.

JUCÁ, S. C. S. A relevância dos softwares educativos na educação profissional. **Ciências e Cognição/Science and Cognition**, v. 8, 2011.

LEAL, F.; PINHO, A. F. de; ALMEIDA, D. A. de. Análise de falhas através da aplicação do fmea e da teoria grey. **Revista Gestão Industrial**, v. 2, n. 1, 2006.

LEFEBVRE, D.; LECLERCQ, E. Stochastic petri net identification for the fault detection and isolation of discrete event systems. **Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on**, v. 41, n. 2, p. 213–225, March 2011. ISSN 1083-4427.

MAURYA, M.; RENGASWAMY, R.; VENKATASUBRAMANIAN, V. Fault diagnosis by qualitative trend analysis of the principal components. **Chemical Engineering Research and Design**, Elsevier, v. 83, n. 9, p. 1122–1132, 2005.

MELLO, B. A. de. Modelagem e simulação de sistemas. **Ciência da Computação/Sistemas de informação. Universidade Regional Integrada do Alto Uruguai e das Missões, Departamento de Engenharias e Ciência da Computação**, 2001.

MENEZES, P. B. **Linguagens Formais e Autômatos**. 6. ed. Porto Alegre, RS: Bookman, 2011.

MONTGOMERY, E. **Introdução aos Sistemas a Eventos Discretos e à Teoria de Controle Supervisório**. 1. ed. Rio de Janeiro, RJ: AltaBooks, 2004.

MORAES, C. C. d.; CASTRUCCI, P. B. d. L. **Engenharia de Automação Industrial**. 2. ed. Rio de Janeiro, RJ: LTC, 2007.

MOREIRA, M. V.; CABRAL, F. G.; DIENE, O. Diagnosticador rede de petri para um sed modelado por um autômato finito. In: SOCIEDADE BRASILEIRA DE AUTOMÁTICA (SBA). **XIX Congresso Brasileiro de Automática**. Campina Grande, PB, 2012. p. 3723–3730.

MOURELLE, L. **Redes de Petri**. disponível em http://www.eng.uerj.br/ ldmm/controle %20 de%20processos/Redes%20de%20Petri.pdf: acessado em: 25/01/2014, 2014.

NEVES, C.; DUARTE, L.; VIANA, N.; FERREIRA, V. Os dez maiores desafios da automação industrial: As perspectivas para o futuro. In: SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA (SETEC/MEC). **II Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica**. Joao Pessoa, PB, 2007.

PAOLI, A.; LAFORTUNE, S. Safe diagnosability of discrete event systems. In: IEEE CONTROL SYSTEMS SOCIETY. **Decision and Control, 2003. Proceedings. 42nd IEEE Conference on**. Maui,HI, 2003. v. 3, p. 2658–2664 Vol.3. ISSN 0191-2216.

QIU, W.; KUMAR, R. Decentralized failure diagnosis of discrete event systems. **Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on**, v. 36, n. 2, p. 384–395, 2006. ISSN 1083-4427.

RAMADGE, P.; WONHAM, W. The control of discrete event systems. **Proceedings of the IEEE**, v. 77, n. 1, p. 81–98, Jan 1989. ISSN 0018-9219.

RASMUSSEN, J. Information processing and human-machine interaction. an approach to cognitive engineering. North-Holland, 1986.

RAUSAND, M.; OIEN, K. The basic concepts of failure analysis. **Reliability Engineering & System Safety**, Elsevier, v. 53, n. 1, p. 73–83, 1996.

REISER, C. **O ambiente GRAIL para controle supervisório de sistemas a eventos discretos: reestruturação e implementação de novos algoritmos**. Tese (Doutorado) — Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica., 2005.

SAMPATH, M.; SENGUPTA, R.; LAFORTUNE, S.; SINNAMOHIDEEN, K.; TENEKETZIS, D. Diagnosability of discrete-event systems. **Automatic Control, IEEE Transactions on**, v. 40, n. 9, p. 1555–1575, 1995. ISSN 0018-9286.

SANTOS, M. P. dos. Introdução à simulação discreta. **Rio de Janeiro: UERJ**, 1999.

SAYED-MOUCHAWEH, M.; BILLAUDEL, P. Abrupt and drift-like fault diagnosis of concurrent discrete event systems. In: ASSOCIATION FOR MACHINE LEARNING AND APPLICATIONS(AMLA). **Machine Learning and Applications (ICMLA), 2012 11th International Conference on**. Boca Raton, FL, 2012. v. 2, p. 434–439.

SILVA, S. da; DIAS JÚNIOR, M.; LOPES JUNIOR, V. Técnicas de predição linear para detecção de falhas estruturais. **IV Congresso Nacional de Engenharia Mecânica(CONEM)**, Ago. 2006.

SLACK, N.; CHAMBERS, S.; JOHNSTON, R. **Administração da Produção**. 2. ed. São Paulo: Atlas, 2002.

TECNOLOGIA, P. Introdução à simulação com arena. **Material Didático, São Paulo, sem data**, 2002.

WONHAM, W. M. **Supervisory Control of Discrete-Event Systems. - Ece 1636F/1637S 2011-12**. disponível em www.control.toronto.edu/people/profs/wonham: acessado em: 04/12/2012, 2012.

APÊNDICE A – TRECHO DO ALGORITMO DE DETECÇÃO DE FALHA

Algoritmo 2

```
if lTemFalha then
    begin
    nPercFalha := 0;
    case nFalhaMq of
        AI:  nPercFalha := lForm1.edtPercAI.Value;
        AN:  nPercFalha := lForm1.edtPercAN.Value;
        AC:  nPercFalha := lForm1.edtPercAC.Value;
        MT:  nPercFalha := lForm1.edtPercMT.Value;
        FN:  nPercFalha := lForm1.edtPercFN.Value;
        CS1: nPercFalha := lForm1.edtPercCS1.Value;
        CS2: nPercFalha := lForm1.edtPercCS2.Value;
    end;

    //Cria uma lista com o nome das tabelas de controle da
    //posição atual (POSATUAL) e a tabela que contem o caminho
    //que retorna para o estado inicial caso não apresente
    //erro (CAMINHOMAQ)

    cTabelas := TStringList.Create;
    cTabelas.Add('CAMINHOMAQ');
    cTabelas.Add('POSATUAL');

    //Cria a 1ª Tabela

    CreateVirtualTable('CAMINHOMAQ',
        'SELECT * FROM COMPONENTE WHERE RESTRICAO = 0 AND
                        UPPER(C.NOME) = UPPER('+cLog+')',
        'IDMAQUINA');

    //Cria a 2ª Tabela
```

```
CreateVirtualTable('POSATUAL',
    'SELECT NULL AS IDMAQUINA, NULL AS POSATUAL FROM RDB$DATABASE',
    'IDMAQUINA');

//Vincula as duas tabelas pelo campo ID

JoinVirtualTable(cTabelas, 'IDMAQUINA');

//Abre todas as tabelas virtuais executando a pesquisa ou
//mantendo uma tabela virtual vazia para inserir dados

OpenAllTable;

//Define se a tabela pode ir somente para frente
//(N = Next) ou se ela pode ir e voltar (P = Previw)
//usar NP para ambos

SetTableDirection('CAMINHOMAQ','NP');
SetTableDirection('POSATUAL','NP');

//Antes de fazer a comparação setamos a posição inicial
//para as tabelas

SetCurrentPositions('CAMINHOMAQ', 'AUTOINC');
SetCurrentPositions('POSATUAL', cLog);

//Passa a posição atual e faz a comparação com os
//proximos evento da tabela, caso o resultado seja
//igual (caminho esperado) retorna verdadeiro, caso
//contrario (em caso de falha) retorna Falso. a DLL
//faz a comparação direto da memoria virtual (RAM)
//com os dados buscados inicialmente, e a cada
//SetCurrentPositions ele grava a posição atual e
//incrementa um next nos indices da tabela.

lRetDll := ComparePositionOfTable(cTabelas, 'IDMAQUINA',
```

```
            Falhou(nPercFalha));

//continua o caminho...

if lRetDll then
    begin
    case nFalhaMq of
        AI:  lForm1.imgFalhaAIRED.Visible  := True;
        AN:  lForm1.imgFalhaANRED.Visible  := True;
        AC:  lForm1.imgFalhaACRED.Visible  := True;
        MT:  lForm1.imgFalhaMTRED.Visible  := True;
        FN:  lForm1.imgFalhaFNRED.Visible  := True;
        CS1: lForm1.imgFalhaCS1RED.Visible := True;
        CS2: lForm1.imgFalhaCS2RED.Visible := True;
    end;

    nUltIni := nFalhaVai;

    if lForm1.chkLogFalha.Checked then
    lForm1.memLog.Lines.Add(IntToStr(self.ThreadID) + ': ' + cLog);

    Continue;
```