DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de software embarcado utilizando Uppaal para uma máquina de Selective Laser Melting

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Henrique Eduardo Felipini

Florianópolis, Fevereiro de 2018

Desenvolvimento de Software Embarcado Utilizando UPPAAL para uma Máquina de Selective Laser Melting

Henrique Eduardo Felipini

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Dr. Fabio Luis Baldissera

Banca Examinadora:

Prof. Dr. Fabio Luis Baldissera
Orientador no Curso

Fernando Silvano Gonçalves, Avaliador

Leonardo Schevz de Werk, Debatedor

Pablo Pompilio Andreus, Debatedor

Agradecimentos

À Universidade Federal de Santa Catarina, seu corpo docente, direção e administração pelas diversas oportunidades que me possibilitaram durante a graduação através das mais diversas experiências profissionais e acadêmicas. Juntamente ao programa Ciência sem Fronteiras, pela experiência única de vida de poder fazer um intercâmbio numa das melhores faculdades da Alemanha.

Aos amigos e colegas que fiz durante esse percurso, que levarei para a vida inteira.

Ao meu orientador da Universidade Federal de Santa Catarina Fabio Luis Baldissera pelo suporte, correções e incentivos para melhorar este trabalho.

Ao meu orientador do Fraunhofer e amigo Rodrigo por todo o ensinamento profissional e pessoal durante meu intercâmbio na Alemanha.

À minha família por todo amor, carinho e apoio incondicional.

Resumo

Trabalho desenvolvido no Fraunhofer - Institut für Produktionsanlagen und Konstruktionstechnik (IPK), Berlim - Alemanha. A empresa oferece soluções de sistemas orientados às aplicações industriais. Este trabalhado tem como objetivo a modelagem de uma máquina de Selective Laser Melting (SLM) em autômatos temporizados, tradução de requisitos em lógica temporal, modelagem dos sistemas e, por fim, programação em sistemas embarcados. O projeto teve início com estudos teóricos. Com o conhecimento adquirido, iniciou-se o processo de entendimento dos diversos sistemas que faziam parte da máquina. Após esse mapeamento foram verificados quais eram os principais inputs e outputs de cada um deles, entendendo a interação de cada um com os demais. Os modelos em autômatos temporizados foram gerados através do software UPPAAL. Então utilizou-se o model checker desse mesmo programa para verificar formalmente se os requisitos do sistema estavam sendo atendidos. Após a verificação formal, iniciou-se a etapa de comunicação serial com Arduino. Com o objetivo de simular com cada uma das placas eletrônicas um sistema da máquina de SLM e emular seu funcionamento. Os resultados foram satisfatórios, com todos os sistemas da máquina mapeados, modelados, verificados formalmente, e comunicando com o computador central e programação em C operando de forma satisfatória.

Palavras-chave: Modelagem em UPPAAL. Sistemas Embarcados. Model-Checker.

Abstract

Report developed at Fraunhofer - Institut für Produktionsanlagen und Konstruktionstechnik (IPK), Berlin - Germany. The company offers application-oriented system solutions covering industrial usage. This report's objective is the modelling of a Selective Laser Melting machine (SLM) using timed automaton and programming in embedded systems. The project began with theoric studies.With the knowledge required, the process to understand the several systems from the machine began.After the process mapping it was verified what were the main inputs and outputs from each one of them, understanding the interaction among them.The models with timed automaton were generated through UPPAAL software. Moreover, the Model Checker was used from the same software to formally verify if the requirements were being attended. With the properties verification concluded the project's second step began, the serial communication in Arduino.The objective was to simulate with each Arduino a system from the SLM Machine and emulate its operation. The results were satisfactory, with all the machine's system mapped, modelled, with properties verified, communication with the central computer working and modeling programmed in C operating in a satisfactory way.

Keywords: UPPAL Modelling. Embedded Systems. Model-Checker.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Computer-Aided Design - CAD

Computer Tree Logic - CTL

Fraunhofer-Institut für Produktionsanlagen und Konstruktionstechnik - Fraunhofer IPK

Graphic User Interface - GUI

Lógica Temporal Tempo-Real - TCTL

Máquina de Estados Finitos - MEF

Powder Bed Fusion - PBF

Proposição atômica - AP

Sistemas a Eventos Dinâmicos - SED

Selective Laser Melting - SLM

Selective Laser Sintering - SLS

Sistema de transição - TS

Tecnologia da Informação - TI

Sumário

1 Introdução

1.1 Motivação

O crescente emprego de componentes eletrônicos e sistemas embarcados nos mais diversos produtos facilita a realização de tarefas cotidianas (tornando-as, muitas vezes, mais seguras) e adiciona conveniências às nossas vidas. Exemplos podem ser encontrados em produtos como celulares, leitores e-books, eletrodomésticos e automóveis.

No entanto, a maior complexidade de operação naturalmente resultante desses novos produtos impõe também desafios às empresas que os projetam e manufaturam. A fim de diminuir a incidência de defeitos em produtos com mais componentes e maior interdependência entre seus subsistemas, torna-se imperativo para as empresas adotar processos de desenvolvimento adequados à nova complexidade de seus produtos.

A situação torna-se ainda mais crítica para aqueles produtos que fazem uso intensivo de software. Estudos realizados apontam que a maioria dos defeitos em sistemas com software provém do próprio software. Em Leveson [6], por exemplo, analisam-se as causas de um conjunto de falhas em missões aeroespaciais que resultaram na perda da missão. A conclusão do autor é de que o software teve alguma participação em todos os acidentes. Para mais exemplos, o leitor pode consultar o trabalho de World [7], que descreve as piores falhas de software em 2010, incluindo algumas que ganharam notoriedade do público, como o mau funcionamento do sistema de freios de carros da Toyota e BMW, ocasionadas por problemas no software dos sistemas embarcados correspondentes.

Um dos caminhos para assegurar o comportamento correto de produtos complexos passa pelo emprego de modelos formais ao longo do processo de desenvolvimento do produto, tanto na etapa de projeto quanto naquela de validação. Neste trabalho, por comportamento correto, entende-se que: a) o produto faz aquilo que é demandado pelo cliente; e b) o faz sem falhas.

Esta monografia insere-se neste contexto da adoção de modelos formais para projeto e validação de sistemas complexos. Mais especificamente, ela abordará o uso de métodos formais para desenvolvimento de uma máquina de *Selective Laser Melting* (SLM), i.e., um dispositivo de manufatura rápida, que será abordado em detalhes mais à frente.

1.2 Objetivo

Os objetivos deste trabalho são:

1. Desenvolver modelos em autômatos temporizados dos componentes de uma máquina de Selective Laser Melting;
2. Traduzir as especificações para o comportamento da máquina em uma linguagem formal;
3. Verificar formalmente se os modelos atendem às especificações do cliente;
4. Codificar em C o modelo de cada subsistema que compõe a máquina SLM;
5. Implementar o código de (4) na plataforma Arduino Uno.

Este trabalho é parte de um projeto mais amplo (passo 2 - Amadurecimento da metodologia), como pode ser visto em destaque na Figura 1. Este projeto consiste em desenvolver uma metodologia de validação e avaliação de software, utilizando modelos de estados finitos.

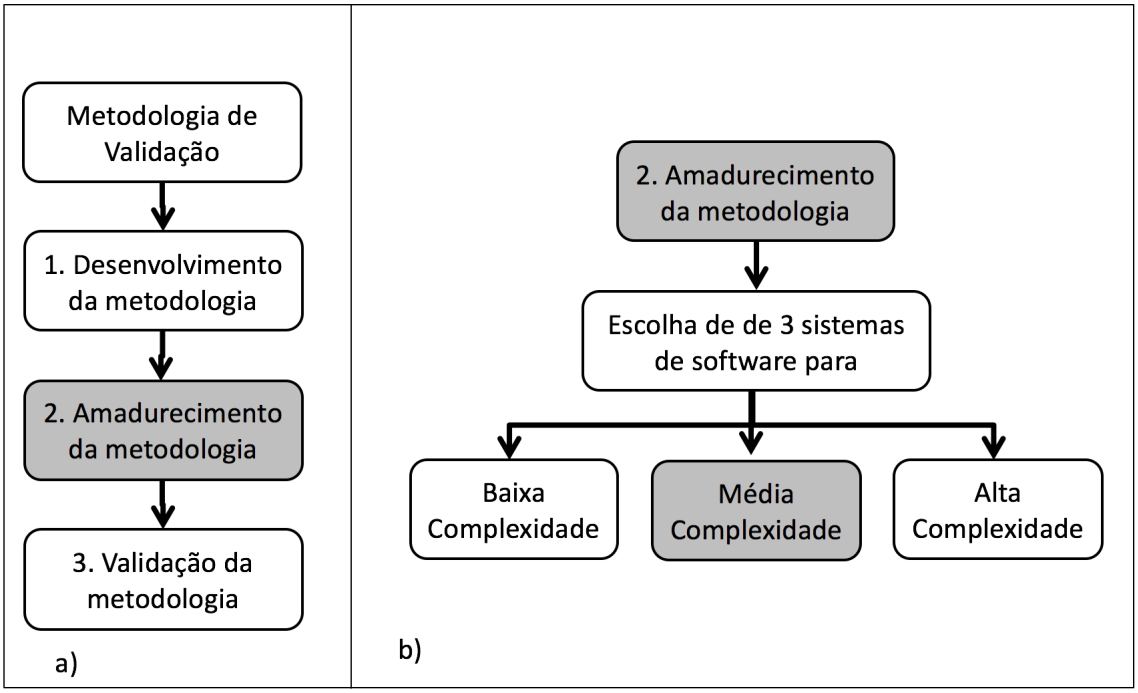

Figura 1 – Passos gerais do projeto (a). Passos específicos para o amadurecimento (b).

1.3 Metodologia do trabalho

A metodologia adotada neste trabalho está esquematicamente representada na Figura 2. Primeiramente, o sistema a ser modelado foi escolhido: a Máquina de SLM, encontrada nas dependências do *Fraunhofer-Institut für Produktionsanlagen und Konstruktionstechnik (IPK)* e objeto de estudo pelo orientador do autor. A máquina, por ser um

sistema complexo, que é um bom parâmetro para a metodologia proposta. Os requisitos e especificações foram fornecidos pelo cliente.

O trabalho foi realizado de forma individual com eventuais dúvidas e questionamentos sanados pelo orientador na empresa, mas de forma a guiar e a não fornecer a solução diretamente. Desta forma, o orientador instigou a criatividade e a capacidade de resolução de problemas do autor deste trabalho.

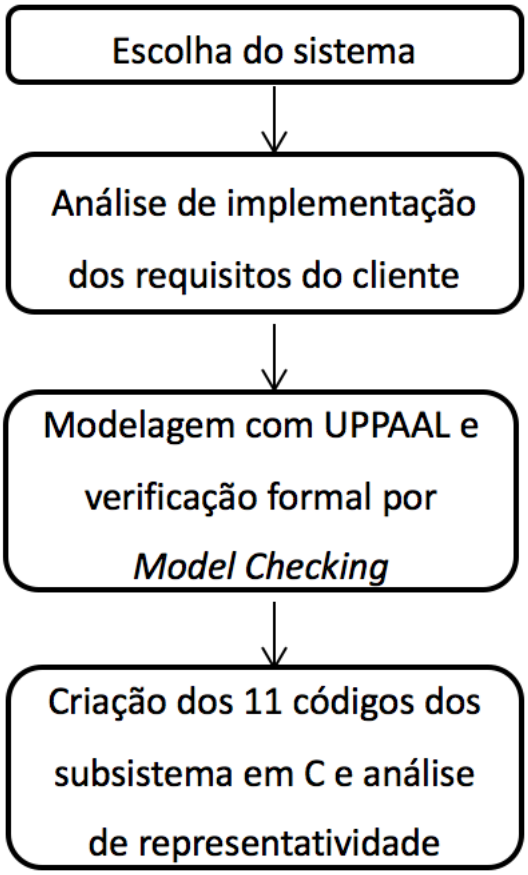

Figura 2 – Metodologia utilizada

A máquina foi então estudada e analisada, verificando todos os seus dispositivos, sensores e atuadores, para representá-los na modelagem. Com este mapeamento finalizado os requisitos do cliente foram analisados e transformados de linguagem escrita em linguagem lógica, para que o computador pudesse entender ao desenvolver o projeto.

A criação dos modelos se deu por meio de um processo iterativo. Com o objetivo de atender totalmente os requisitos do cliente, o projeto da plataforma iniciou da estrutura mais básica e retrabalhou-se até ser o modelo mais complexo desenvolvido. Com o modelo da plataforma finalizado, realizou-se a verificação formal de propriedades através do *Model Checking* do software UPPAAL, detalhado nos próximos capítulos. Com todas as modelagens prontas, os subsistemas que compõem a máquina de SLM foram codificados em C e implementados na plataforma Arduino Uno.

1.4 Contribuições do trabalho

Os principais desenvolvimentos feitos pelo autor foram o mapeamento de cada um dos sistemas que compõem a máquina de SLM (plataforma, laser, depositador de pó, etc.), modelagem de cada um dos sistemas em UPPAAL (apêndice A), tradução da linguagem escrita para TCTL validando propriedades (apêndice B) através do *Model-checker* do software, levando em conta os requisitos fornecidos pelo cliente (anexo A).

Após as etapas descritas no parágrafo anterior, o autor começou a desenvolver o sistema de comunicação e protocolo entre o computador e ARDUINO UNO. A primeira atividade foi realizar a sincronização de relógios e comunicar utilizando um protocolo de comunicação padrão que pudesse ser utilizado por todos os sistemas. Por fim, programou-se manualmente a modelagem do UPPAAL em linguagem C, gerando as funções necessárias para funcionamento e numerando cada um dos estados com seus respectivos requisitos.

1.5 Organização do trabalho

Este trabalho está organizado da seguinte forma. O capítulo 2 introduz os principais conceitos de sistemas dinâmicos, verificação formal de modelos, definindo a lógica temporal usada para traduzir os requisitos do cliente para testar em *model checker*, e explica os conceitos para modelagem e verificação com a ferramenta UPPAAL. O capítulo 3 discute os conceitos do método de fabricação de componentes por SLM juntamente com o funcionamento da máquina encontrada no instituto.

O capítulo 4 explana sobre a modelagem de cada um dos 11 subsistemas estudados neste trabalho. Nele é fornecido um resumo sobre cada componente juntamente com os requisitos fornecidos para o funcionamento do mesmo durante o processo de modelagem. São fornecidos os conceitos gerais, modelagem dos sensores, atuadores e do controle de cada subsistema e, por fim, as propriedades verificadas de cada um. O capítulo 5 discute sobre resultados e como foi realizada a programação manual de cada um dos subsistemas para o sistema embarcado ARDUINO UNO. As conclusões são apresentadas no capítulo 6.

2 Modelagem, simulação e verificação de sistemas

Este capítulo apresenta conceitos referentes à sistemas dinâmicos, o que são e em que estão sendo aqui aplicados, verificação de modelos, explicando sobre caminhos e a lógica temporal usada para traduzir os requisitos do cliente, e, por fim, como é a modelagem e verificação formal de modelos utilizando a ferramenta UPPAAL.

2.1 Sistemas Dinâmicos

Nesta seção trabalhamos o conceito de sistemas e modelos e procederemos com a seção de sistemas dinâmicos. Um sistema é um conjunto de entidades que o observador delimita do resto do mundo que interagem entre si para estudar o comportante de alguma coisa. A interpretação dos infinitos sistemas do mundo real requer habilidades de percepção, observação, análise e abstração, sendo esta uma ferramenta poderosa para o exercício de compreensão do universo.

Segundo Cassandras [1], o conceito de sistema pode ser explicado como uma conjunto de componentes que interagem entre si para realizar uma função que não seria possível pela ação isolada de cada um desses componentes. Com esta definição, pode-se notar duas caraterísticas interessantes sobre sistemas. Primeiramente, um sistema consiste da interação de componentes e, a outra, é que um sistema possui uma ou mais "funções"que podem ser realizadas.

De acordo com Sayao [8], para se compreender os diversos fenômenos existentes é necessário selecionar aqueles de maior relevância para o problema objeto de investigação e elaborar descrições adequadas. Por Cassandras [1] modelos e sistemas possuem entradas, sendo os dados coletados através de medições, e saídas, descritas como a resposta aos estímulos gerados. Chamados na literatura de *input* e *output*, respectivamente. Na Figura 3 abaixo está representado este conceito simples entre modelo e sistema.

Uma vez criado e validado, o modelo pode ser usado para investigar, predizer e analisar o funcionamento do sistema real de maneira a estudar as respostas deste às diversas entradas nele inseridas. Seu emprego é essencial para o desenvolvimento de processos ou produtos. Em Epstein [9] discute-se sobre a predição de resultados ser, erroneamente, o objetivo do processo de modelagem. Modelagem de sistemas pode ser utilizada para diversas outras aplicações, por exemplo:

- Capacidade de **Explicação** de acontecimentos e eventos, diferente de predição;

- **Sugerir analogias** como a atração eletrostática da Lei de Coulomb e a atração gravitacional da Lei de Newton possuindo a mesma forma algébrica; e,
- Possibilidade de **Gerar novas perguntas** através de resultados obtidos.

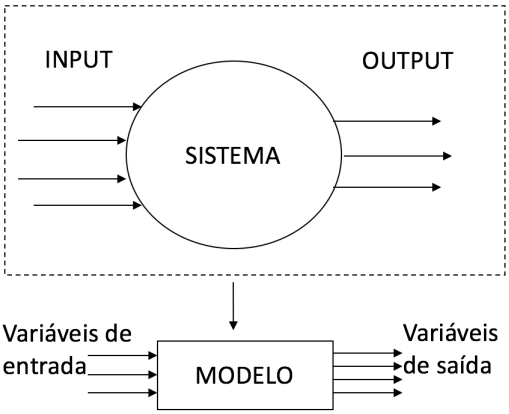

Figura 3 – Processo de modelagem simples

Um sistema dinâmico é um sistema cujo estado evolui com o tempo em um espaço de estados. Por exemplo, um motor elétrico com tensão sendo aplicada, passando corrente elétrica e tendo uma velocidade angular do eixo. É possível descrever o comportamento do motor e determinar sua posição e seu estado (posição e velocidade angular), que podem alterar com o tempo. Diferente de sistemas estáticos em que as propriedades descritivas do sistema não variam com o tempo. Por exemplo, uma viga carregada estaticamente, com cargas constantes, em que os deslocamentos de seus pontos não variam com o tempo.

Tais sistemas possuem a sub-classificação de serem sistemas contínuos ou sistemas discretos. Por Lathi [10], sistemas contínuos são sistemas cujas entradas e saídas são sinais contínuos no tempo, especificados em uma faixa de valores contínua no tempo, ressaltando que seu conjunto de estados é infinito e não enumerável. Por exemplo a câmera de vídeo, uma chamado por telefone ou o próprio conjunto dos números reais. A Figura 4 mostra a descrição matemática genérica para o sistema com variáveis contínuas.

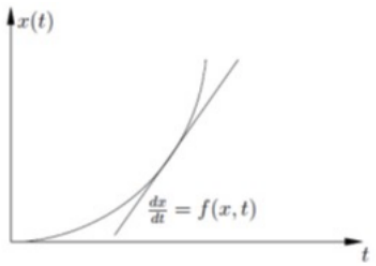

Figura 4 – Exemplo da resposta de um sistema com variáveis contínuas

Sistemas discretos, também por Lathi [10], são sistemas cujas entradas e saídas são sinais discretos no tempo, definidos apenas em instantes discretos de tempo. Por exemplo,

as médias diárias do mercado de ações. A Figura 5 representa o comportamento de um sistema discretos ao longo do tempo.

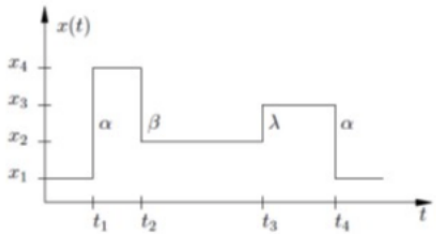

Figura 5 – Exemplo da resposta de um sistema discreto

Por Cassandras [1] Sistemas a Eventos Discretos (SEDs) são sistemas discretos. Porém, a recíproca não é verdadeira. Sistemas discretos podem ser guiado tanto por tempo quanto por eventos, sendo o último tratado na próxima seção deste trabalho. Um fácil exemplo para entender os sistemas de estado discreto guiado por tempo é o controle de temperatura de uma geladeira. No momento que o compressor estiver ligado a temperatura interna irá diminuir e quando o mesmo ciclar, a temperatura da câmara irá aumentar. Ou seja, o estado altera-se com a passagem do tempo. Na Figura 6 abaixo mostra-se a classificação de sistemas feita por Cassandras [1].

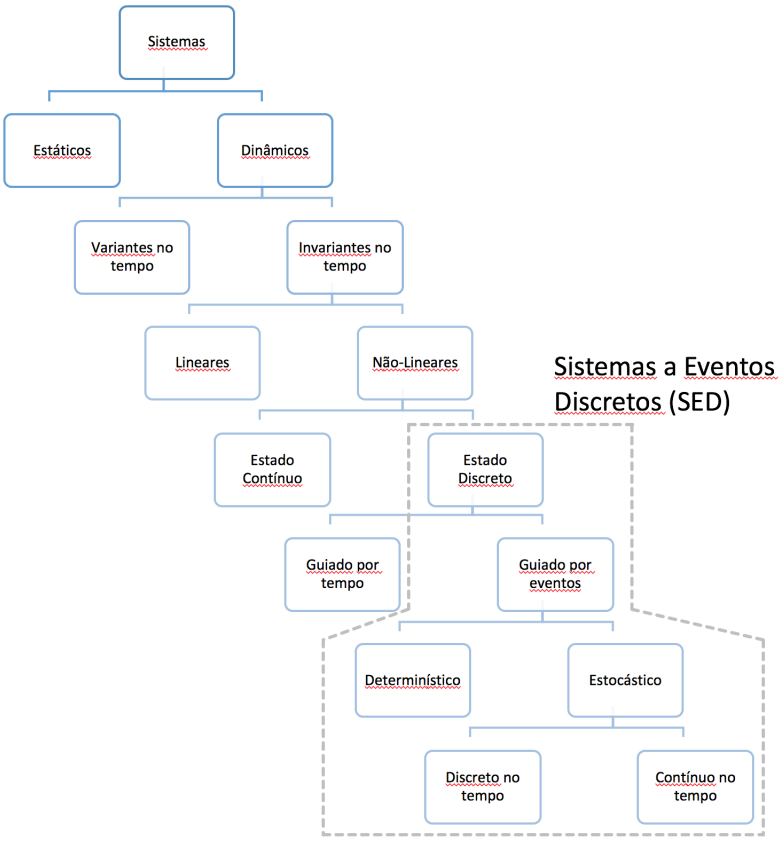

Figura 6 – Classificação dos Sistemas [1]

2.1.1 Sistemas a Eventos Discretos

Um sistema a evento discreto é um sistema dinâmico discreto e dirigido a eventos, isto é, com espaço de estados discreto e cuja evolução de estado depende inteiramente da ocorrência assíncrona de eventos discretos. Eventos são pensados como um ocorrência instantânea que causa a transição de um estado do sistema para outro. Um evento pode ser identificado como uma ação específica tomada, uma ocorrência espontânea da natureza ou ainda o resultado de diversas condições que se combinaram repentinamente.

Resumindo, as duas prioridades que caracterizam um sistema ser um SED e não um sistema dinâmico de variáveis contínuas são:

- O espaço de estados ser um conjunto discreto; e,
- O mecanismo de transição de estados ser guiado a eventos.

O avanço tecnológico tem crescido exponencialmente nos últimos anos, com a automação e a computação mais e mais presentes no cotidiano. Os aparelhos e dispositivos existentes são SEDs e funcionam guiados a eventos. Por exemplo, o estado de um robô pode ser selecionado por conjuntos como LIGADO, DESLIGADO ou OCUPADO, OCIOSO, DESLIGADO.

Segundo Cury [11], a natureza discreta dos SEDs faz com que os modelos matemáticos convencionais, baseados em equações diferenciais, não sejam adequados para tratá-los. Desta forma, foram desenvolvidos vários modelos para SEDs que refletem diferentes objetivos na análise dos sistemas em estudo como, por exemplo, Redes de Petri, Redes de Petri Controladas, Cadeias de Markov, Teoria das Filas, Processos Semi-Markovianos Generalizados e Simulação, Álgebra de Processos, Álgebra Max-Plus, Lógica Temporal e Lógica Temporal em Tempo Real, Teoria de Linguagens e Autômatos.

Neste trabalho, o foco é dado para os métodos da Teoria de Linguagens e Autômatos, já que estes são utilizados para a elaboração e estudo dos modelos.

2.1.2 Modelagem com Autômatos de Estados Finitos

Autômato finito é um formalismo matemático que pode ser empregado para modelar o comportamento de SEDs. Um autômato é definido como uma quíntupla (X, E, f, x0, Xm) em que é a forma de representação do comportamento dinâmico de um sistema dirigido a eventos discretos, onde:

- X - Conjunto finito de estados;
- E - Conjunto finito de eventos;

- f $\subseteq$ X $\times$ E $\times$ X é uma relação de transição;
- x0 - Estado inicial; e,
- Xm - Conjunto de estados finais.

Em cada estado existe no máximo uma transição de saída associada a cada evento. Podendo ou não possuir condições para que a transição ocorra, denominadas guardas, que serão detalhadas nos próximos tópicos.

A Figura 7 mostra um exemplo de um autômato finito simples. Este modelo representa o acionamento de uma lâmpada. Seus possíveis estados são: 'Desligado' e 'Ligado', sendo o estado inicial representado por uma seta sem origem e o final por duas circunferências concêntricas. Este sistema pode ser acionado por dois botões 'Liga' e 'Desliga'. Quando o primeiro é pressionado a lâmpada é acesa e quando o segundo é acionado ela se apaga. Se a lâmpada se encontrar no estado 'Desligado' e o botão 'Desliga' for apertado, nada acontece. O mesmo serve para o estado 'Ligado' e o botão 'Liga'. Os botões no modelo representam os eventos associados às transições do autômato.

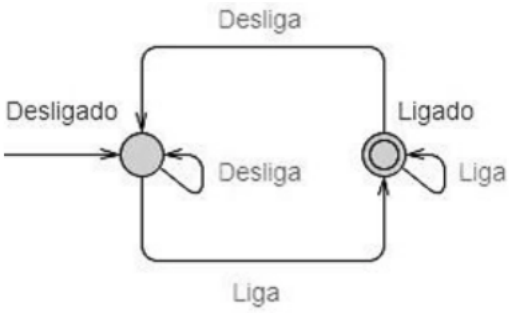

Figura 7 – Exemplo de Autômato do acionamento de uma lâmpada

2.1.3 Simulação de modelos

Simulação é a computação de uma possível trajetória de um modelo matemático de um sistema. De acordo com Hartmann [12], simulações estão proximamente relacionadas com modelos dinâmicos, sendo estes responsáveis pelo estudo da evolução do sistema de acordo com eventos. É ressaltado que a simulação pode ser útil não somente para modelos de solução complexa como pode ser de ajuda na solução de sistemas mais simples por métodos analíticos, já que a visualização dos resultados de simulação com auxílio de um computador é sempre uma vantagem para o estudo.

Além disso, em [12], é citada também a importância da determinação da simulação como contínua ou discreta. Sendo naquela, a estrutura espaço-tempo e o conjunto de estados possíveis assumidos como contínuos; e nesta, a estrutura espaço-tempo discreta desde o início. Para simulações contínuas, a linguagem mais conveniente para estudo do

modelo é por equações diferenciais enquanto que para as simulações discretas é a linguagem de autômatos.

Dentro dos diversos possíveis empregos da simulação, o seguinte trabalho procura explorar tal ferramenta como substituta a um experimento. Sendo assim, a simulação aplicada neste projeto busca a abordagem teórica do funcionamento de um sistema que seria inacessível de outra forma, já que não há acesso ao software do sistema escolhido: a máquina de SLM.

2.2 Verificação formal de modelos

O correto funcionamento do hardware e software de sistemas é crítico para sua aplicação e confiabilidade. A partir do desenvolvimento de novas tecnologias e dispositivos, a garantia do desempenho de um sistema depende do funcionamento de cada componente que o integra, muitas vezes, sendo necessário que estes trabalhem mesmo sobre falhas. A importância do desenvolvimento de hardwares e softwares com alta confiabilidade exige que, nas etapas de projeto, todos os sistemas sejam verificados e testados em todos os prováveis cenários de funcionamento do sistema, e que suas especificações sejam atendidas.

O *model checking* é, segundo Clarke [13], um método automático para verificação de sistemas de estados finitos. Conforme citado em [14], *"dado um sistema de transição (TS) de estados e uma propriedade, o algoritmo de model checking explora exaustivamente o espaço de estados para determinar se o sistema satisfaz a propriedade verificada"*.

A maior dificuldade na verificação de modelos é lidar com a "explosão"de estados, que ocorre em sistemas com muitos componentes, interações e valores, exigindo uma grande capacidade de processamento do computador utilizado. Abaixo serão definidos tópicos necessários para o entendimento de modelagem e verificação na ferramenta UPPAAL.

2.2.1 Caminhos e Lógica Temporal

O termo lógica temporal abrange todas as abordagens de representação e raciocínio sobre tempo e informações temporais dentro de um *framework* lógico. Com grande utilização para declarar requerimentos de sistemas de *hardware* e *software*. Este trabalho focará na explicação sobre sequência de estados, ou caminhos, e lógica temporal *Computation Tree Logic (CTL)*, por ser a utilizada durante o desenvolvimento do projeto.

Para se definir sequência de estados, é necessário primeiramente entender cada parte dos fragmentos que o compõem. Segundo Cassandras [1], fragmentos de caminhos podem ser tanto finitos quanto infinitos. Fragmentos de caminho finitos são sequências de estados que possuem um estado posterior à eles, tendo um estado inicial e um estado final obrigatoriamente. Enquanto os infinitos também possuem um estado inicial e um

estado posterior, mas não possuem um estado final. Dessa forma, sendo um fragmento de caminho infinito.

Com esta definição, é possível detalhar também fragmento de caminho máximo e fragmento de caminho inicial. Fragmento de caminho máximo é ou fragmento de caminho finito que acaba em um estado final ou um fragmento de caminho infinito. Fragmento de caminho máximo é um fragmento de caminho que não pode ser prolongado, ou seja, ou é um fragmento de caminho infinito ou um finito em que, em dado estado, não se pode realizar transições. Um fragmento de caminho é chamado inicial se ele começa em um estado inicial.

Após ser definido o que são fragmentos de caminho, é possível definir o que é um caminho. Caminho de sistema de transição é uma sequencia finita/infinita de estados permitidos por este sistema de transição que parte do estado inicial. Facilmente entendível pela figura 8 abaixo. Temos que [S_0; S_1] é um caminho, assim como [S_0; S_2]. Porém, [S_1; S_2] não é um caminho, pois não possui o estado inicial do sistema, que é S_0.

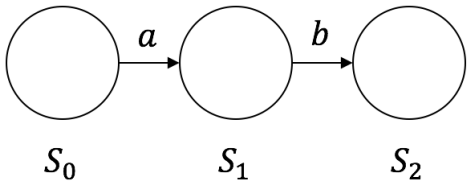

Figura 8 – Exemplo de sistema de transição para explicar caminhos

A lógica temporal CTL é uma ramificação temporal lógica para especificar propriedades de sistemas. Em CTL os caminhos, sequências de estados, são obtidos de um sistema de transição que pode ser ramificado. Ou seja, um estado pode ter diversos e distintos estados sucessores diretos, obtendo diferentes futuros possíveis.

Segundo Cassandras [1], a sintaxe de CTL é uma fórmula de estados dado um conjunto PA de proposições atômicas, "x igual à 0"ou "x é maior que 200"(para dada variável inteira x) por exemplo, e formado de acordo à gramática da figura 9 abaixo, onde $p \in PA$.

$$\begin{aligned}\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\phi \Leftrightarrow \phi) \\ \mid \quad \mathbf{AX}\,\phi \mid \mathbf{EX}\,\phi \mid \mathbf{AF}\,\phi \mid \mathbf{EF}\,\phi \mid \mathbf{AG}\,\phi \mid \mathbf{EG}\,\phi \mid \mathbf{A}\,[\phi \mathbf{U}\ \phi] \mid \mathbf{E}\,[\phi \mathbf{U}\ \phi]\end{aligned}$$

Figura 9 – Gramática em CTL

Definindo a sintaxe em CTL, A significa *Por Todos os Caminhos - 'Along All Paths'*, inevitável. E significa *Pelo Menos (existe) um Caminho - 'Along at least (there Exists) one Path'*, possível. A e E são operadores temporais quantificadores através de caminhos, que serão melhor definidos nos parágrafos abaixo.

Para realizar a verificação formal utilizando qualquer lógica temporal, é necessário que haja operador lógico para obter comparações e resultados que estão sendo buscados. Os operados lógicos em CTL são os mais comuns, como mostrados anteriormente na gramática, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, também fazendo uso dos booleanos *true* e *false*.

Segundo Cassandras [1], além dos operadores lógicos também é necessário o emprego de operadores temporais em lógica temporal. Em CTL são definidos quantificadores através de caminhos, A e E, e quantificadores de caminhos-específicos, X, G, F, U e W explicados a seguir:

- $A\phi$ - All: ϕ tem que dizer para manter todos os caminhos começando do estado atual (ou $\forall$ em notação usual);
- $E\phi$ - Exists: existe ao menos um caminho começando do estado atual onde ϕ é mantido (ou $\exists$ em notação usual);
- $X\phi$ - Next: ϕ precisa para manter o próximo estado (ou $\bigcirc$ em notação usual);
- $G\phi$ – Globally: ϕ tem que manter todo o caminho subsequente (ou $\Box$ em notação usual);
- $F\phi$ - Finally: ϕ eventualmente tem que manter (ou $\Diamond$ em notação usual);
- ϕ U ψ – Until: ϕ precisa manter ao menos antes de alguma posição ψ manter; e,
- ϕ W ψ – Weak até: ϕ tiver que manter ψ mantém.

Cassandras [1] define que semântica em CTL é interpretada como caminhos e estados de um sistema de transição. Com tal sistema, a semântica é definida por duas relações de satisfação, representadas por $\models$. Sendo uma relação de caminho e outra de estados. Com isso, deixa $a \in AP$ ser uma proposição atômica, e TS a quíntupla definida anteriormente, $s \in S$, Φ, Ψ sendo relações de estado em CTL e φ a relação de caminho em CTL. A relação de satisfação $\models$ é definida por relação de estado como:

$s \models \neg\Phi$ sse não $s \models \Phi$

$s \models \Phi \wedge \Psi$ sse $s \models \Phi$ e $s \models \Psi$

$s \models \exists\varphi$ sse $\pi \models \varphi$ para algum $\pi \in Caminhos(s)$

$s \models \forall\varphi$ sse $\pi \models \varphi$ para todos $\pi \in Caminhos(s)$

Para o caminho π a relação de satisfação $\models$ é definida como abaixo, onde para o caminho $\pi = s_0 s_1 s_2 ...$ e inteiro $i \geq 0$, $\pi[i]$ representa o (i+1)-ésimo estado de π, ou seja, $\pi[i] = s_{\mathrm{i}}$.

$\pi \models X\Phi$ sse $\pi[1] \models \Phi$

$\pi \models \Phi \cup \Psi$ sse $\exists j \geq 0.\ (\pi[j] \models \Psi \wedge (\forall 0 \leq k < j.\pi[k] \models \Phi))$

Para melhor entendimento do que foi discutido acima, abaixo há um exemplo de grafo de alcançabilidade retirado do livro de Cassandras [1] na figura 10 para melhor entendimento sobre o que é um sistema de transição (*TS* na figura), com os estados [S_0; S_1; S_2; S_3] e transições $[a, b]$. Grafo de alcançabilidade é um gráfico que exibe todos os estados possíveis alcançáveis, dada tal proposição atômica.

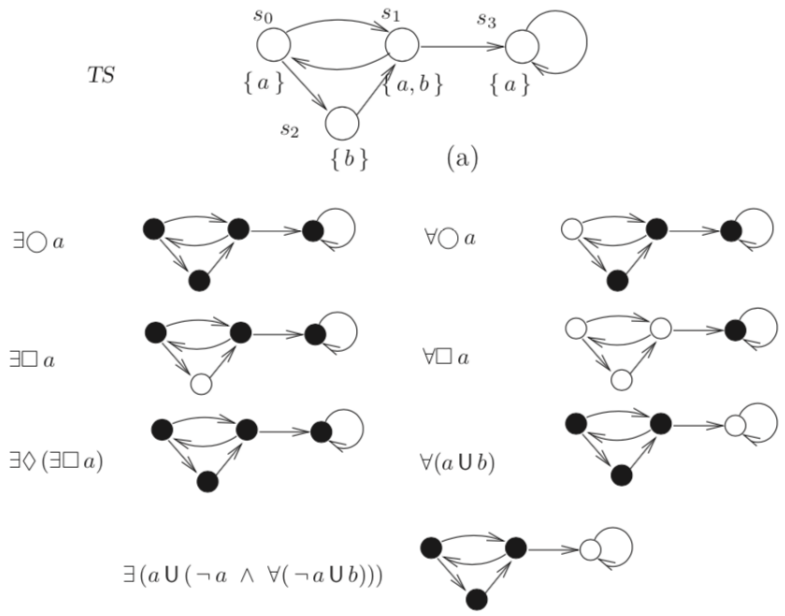

Figura 10 – Exemplo de lógica temporal em um sistema de transição [1]

2.3 Modelagem e verificação de modelos com UPPAAL

Uma das ferramentas de verificação de modelos por *model checking* é o UPPAAL, que é baseada na modelagem de sistemas em autômatos temporizados e que utiliza a lógica CTL (*Computation Tree Logic*) para especificar as propriedades a serem verificadas. Como explicado anteriormente, linguagens temporais são sistemas de regras e símbolos utilizados para representar e explicar requisitos.

O UPPAAL se baseia em uma arquitetura cliente-servidor, sendo dividido em uma interface gráfica do usuário e a ferramenta de *model checking*. A interface do usuário, ou cliente, é implementada em Java e a ferramenta, ou servidor, é compilado por diferentes plataformas (Linux, Windows, Solaris) [15].

No processo de modelagem no UPPAAL, o sistema é definido como uma rede de autômatos temporizados, chamados processos, que ocorrem paralelamente. Cada processo é instanciado em um *template* com parâmetros específicos. Como ainda visto em [15], o

mecanismo de instanciação é similar ao utilizado para classes em linguagens orientadas a objetos.

A interface gráfica (GUI - *Graphic User Interface*) do Java mostra a modelagem de um sistema baseado em autômatos temporizados por meio de um editor gráfico, a sua simulação para validação de seu comportamento e a verificação da correspondência do modelo com as propriedades especificadas. Ela é dividida em três abas: editor, simulador e verificador. No editor, pode-se modelar o sistema como autômatos temporizados. No simulador, o comportamento do autômato pode ser validado e, no verificador, verifica-se se o modelo atende às propriedades desejadas e traduzidas em uma lógica temporal.

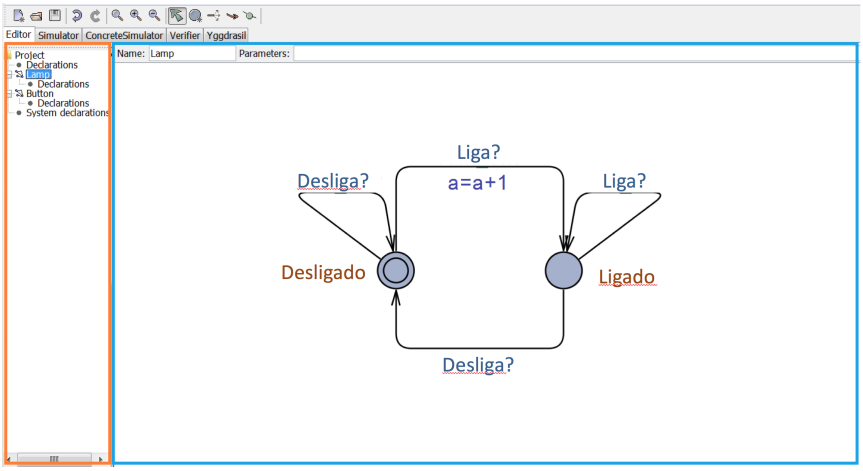

Figura 11 – Interface do UPPAAL

O editor se divide em duas partes, conforme pode ser visualizados na Figura 11. Na janela em laranja, encontra-se a árvore para acesso aos diferentes *templates* e declarações e, na janela azul fica a área para desenho do modelo. Por meio da árvore em *'Declarations'*, podem ser acessadas as variáveis, as funções e os canais de sincronização que são compartilhados entre os *templates*.

Já em *'System Declarations'* é apresentada a definição do sistema como um conjunto dos *templates*. A Figura 12 mostra a declaração das variáveis e funções globais, as aplicáveis ao *template* 'Lâmpada' e a definição do sistema em *'System Declarations'*, respectivamente.

O simulador pode ser usado de três formas: o usuário pode acionar o sistema manualmente e escolher quais transições serão realizadas; o modo *random* pode ser acionado fazendo o sistema rodar por conta própria; ou o usuário pode seguir um caminho (salvo ou importado do verificador) para visualizar como determinados estados são atingidos.

Na Figura 13, pode-se visualizar a aparência do Simulador. Na área em vermelho (*'Enabled Transitions'* e *'Simulation Trace'*) são exibidas as transições conforme essas ocorrem e se pode controlar a simulação aleatória pelos botões *'Prev'*, *'Replay'*, e os demais. No quadro em verde, são exibidas as variáveis e o valor que essas assumem conforme o

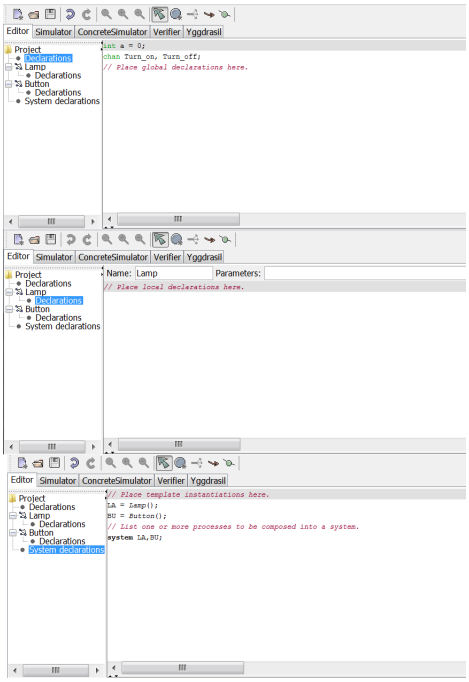

Figura 12 – Exemplo de declaração das variáveis e instanciação dos *templates*

decorrer da simulação. Nos quadros da direita, são exibidos, no azul, os *templates*, onde é possível visualizar a ocorrência das transições e mudança de estados; e, no quadro roxo, o diagrama de sequência, que permite a visualização da sincronização dos processos e sua ativação a cada interação.

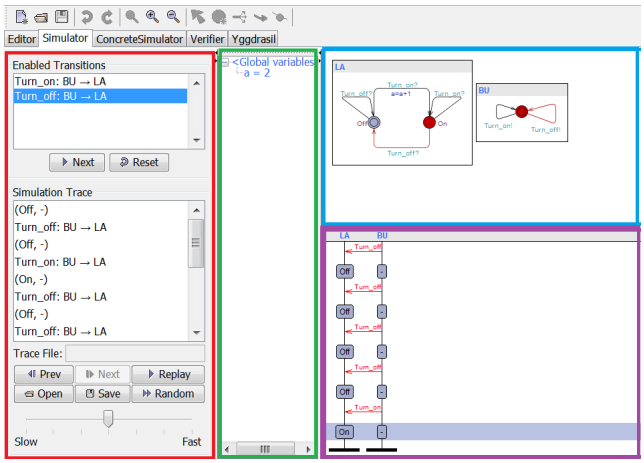

Figura 13 – Exemplo de autômato sendo simulado no UPPAAL

No verificador, as propriedades do sistema podem ser definidas, editadas e comentadas. Na Figura 14, as áreas em destaque demonstram o estado das propriedades verificadas. Quando estas são satisfeitas, o marcador sinaliza em verde, e quando não, em vermelho. No caso de ocorrer uma propriedade falsa, o UPPAAL pode fornecer o caminho que resultou na violação da propriedade. Outra possível resposta é a falha de memória, que ocorre quando o o número de interações é muito grande para ser calculado. O *model checker* não

consegue, então, obter nenhuma conclusão a respeito da propriedade, como visto em [14]. A especificação do *model checker* do UPPAAL emprega a lógica CTL da seguinte maneira:

A - Todos os caminhos (A em UPPAAL);

E - Existe um caminho (E em UPPAAL);

G - Todos os estados em um caminho ([] em UPPAAL);

F - Algum estado em um caminho (<> em UPPAAL).

As propriedades de acessibilidade (ou *Reachability*) são aquelas que permitem especificar que uma situação pode ser atingida. A propriedade de alcançabilidade é aquela que permite verificar se existe uma trajetória do modelo na qual um dado estado pode ser alcançado. No UPPAAL, elas são expressas no formato $E<>p$.

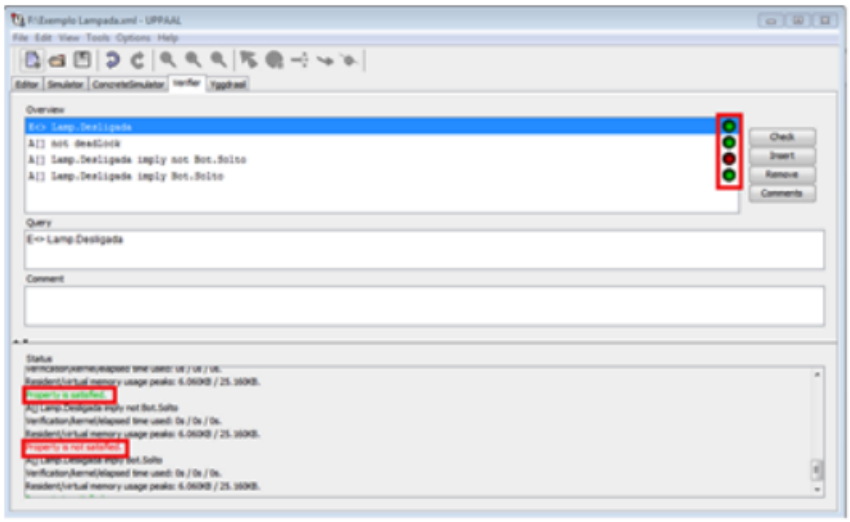

Figura 14 – Exemplo de verificação no UPPAAL

As propriedades de segurança (ou *Safety*) são aquelas que permitem especificar que algo negativo nunca vá acontecer, podendo se limitar a uma trajetória ou abranger todas as possíveis. Este tipo de propriedade é fundamental para garantir situações críticas em um modelo, por exemplo, validando que duas portas nunca sejam abertas simultaneamente em um sistema de controle de acessos. No UPPAAL, elas são expressas de forma positiva: *A[]p*, quando deve ser verdade para todos os caminhos, ou *E[]p*, quando deve ser verdade em todos os estados de algum caminho [16].

Para a modelagem de qualquer sistema, é importante o entendimento de guardas, sincronizações e invariantes para autômatos temporizados. Uma guarda é uma expressão aplicada a uma transição que define as condições para que essa ocorra. Sincronização é um rótulo da forma *'Expressão!', 'Expressão?'* ou vazio. Os sinais '!' e '?' identificam quem envia ou recebe a mensagem '*Expressão*', respectivamente. A sincronização entre dois, e somente dois, autômatos ocorre por meio das variáveis do tipo *channel*, como no exemplo

dado acima. Quando for necessário que a sincronização ocorra de um emissor para um número maior de receptores, utiliza-se a variável *broadcast channel*. Entretanto, deve-se ter cuidado, pois, neste caso, a transição ocorre mesmo quando não existir nenhum receptor.

A representação dos estados, como vista nas figuras anteriores, é diferente da modelagem com autômatos tradicional, apresentada no tópico 2.2.1. Para cada estado no UPPAAL, mostrado na Figura 15, é possível definir sua condição como:

- Iniciais (*initial locations*): estado em que o modelo se encontra antes de ocorrer qualquer transição;
- Urgentes (*urgent locations*): estados em que o tempo não avança;
- Comprometidos (*committed locations*): estados urgentes que possuem prioridade sobre qualquer outro estado;
- Normais (*normal locations*): todos outros estados que não se enquadram como iniciais, urgentes ou comprometidos.

Figura 15 – a) Inicial b) Urgente c) Comprometido d) Normal

Os invariantes, por fim, definem em quais condições se pode permanecer no estado atual, forçando a ocorrência de um evento, como visto em [14]. A Figura 16 apresenta dois autômatos genéricos construídos no UPPAAL, representando suas definições de guarda, troca de mensagens e invariantes.

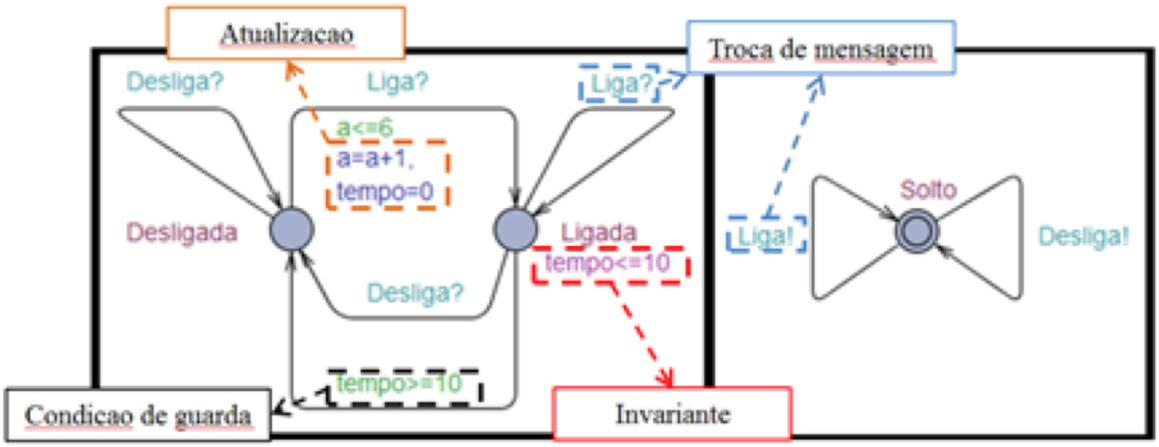

Figura 16 – Exemplo de dois autômatos na modelagem em UPPAAL

3 Máquina de SLM - Selective Laser Melting

Nos tópicos seguintes serão descritos os conceitos da máquina de SLM, seu funcionamento e os principais subsistemas que a constituem.

3.1 Conceitos

Segundo Mani [17], *Powder Bed Fusion*(PBF) é uma das sete categorias de processo de manufatura aditiva definida em ASTM F2792 [18], que também é conhecida como prototipagem rápida, fabricação aditiva, fabricação de forma livre, impressão 3D, manufatura rápida e utilizada como tecnologia avançada para fabricar componentes por agregar e montar material camada por camada. Processos de PBF usam energia térmica para fundir seletivamente áreas de uma camada de pó usando laser ou raio de elétrons como fonte de energia [18].

Existem diversos tipos de sistemas comerciais de PBF que conseguem manufaturar componentes tanto com polímeros, quanto com metais. Esse trabalho é focado na classe de processos de PBF que utiliza feixes de energia de alta potência para derreter completamente as partículas de pó metálicas, as quais se fundem juntamente à camada anterior quando o material resfria. Este processo de manufatura de componentes é intitulado de SLM. Repetindo-o, camada por camada, o resultado é um componente com quase 100% de densidade [17].

Hoje em dia, tecnologias de fabricação utilizando SLM são usadas vastamente na indústria e pesquisa, oferecendo uma vasta gama de vantagens se comparadas às técnicas convencionais de manufatura: menor tempo para chegar ao mercado, uso de materiais mais baratos, versatilidade, habilidade para produzir mais funcionalidades no componente e características projetadas únicas [19]. A fabricação de componentes através da consolidação de pó por derretimento à laser está se tornando uma técnica de manufatura promissora, devido ao fácil controle sobre o laser e sobre a deposição do pó [20]. Segundo Over [21], tecnologias de SLM tornam possível criar componentes totalmente funcionais diretamente de metais, cerâmicos ou plásticos sem usar qualquer ligante intermediário ou qualquer processo adicional após a queima com o laser.

O que torna o processo de fabricação por SLM desafiador e complexo são as grandes quantidade de variáveis que podem influenciar no sistemas [22], por exemplo:

- Pó: Composição, distribuição do tamanho, forma, propriedades óticas e de transferência de calor e espessura da camada depositada para cada ciclo do processo de manufatura;

- Laser: Potência, tamanho do ponto, distribuição espacial do feixe, velocidade de escaneamento e aplicação da atmosfera do gás de proteção; e,
- Estratégia de manufatura: Definição da orientação e distância entre elas e definição das posições relativas de cada camada em dois planos consecutivos.

3.2 Funcionamento da máquina

Segundo [22], o processo de fabricação utilizando SLM inicia com um modelo em Computer-Aided Design (CAD) completo do componente que será feito. Então divido em camadas, transformando-o em arquivo de StereoLitography, por um software, o modelo é então diretamente envolvido no processo de fabricação.

Com a máquina de SLM podem ser manufaturados componentes funcionais, camada por camada, de um material metálico utilizando pó. Com isso, a energia de um feixe de laser é absorvida pelo pó metálico e leva a uma fusão limitada localmente das partículas. Virtualmente, qualquer forma pode ser produzida diretamente de dados de projeto. Na figura 17 pode ser visto um rascunho dos principais subsistemas da máquina de SLM.

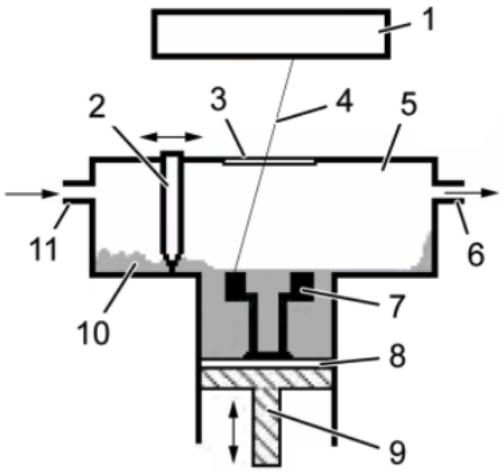

Figura 17 – Princípio de funcionamento da máquina SLM

1. Sistema de laser (Banco óptico);
2. Dispositivo de deposição de pó;
3. Vidro de proteção F-Theta;
4. Feixe de laser;
5. Câmara da máquina;
6. Saída de gás;

7. Componente;
8. Base de montagem;
9. Plataforma;
10. Pó de metal; e,
11. Entrada de gás.

Conforme observado, nota-se que o processo de fabricação utilizando a máquina de SLM é um processo cíclico, que se repete até o fim dos passos do processo de manufatura. Em primeiro lugar, o dispositivo de deposição de pó (item 2 na figura) aplica uma camada de pó de metal (item 10 na figura) com uma espessura de camada de 20 até 100 μm. O próximo passo, a exposição, consiste na compactação local do pó através de um feixe de laser (item 4 na figura). A absorção da radiação do laser causa um aquecimento do pó de metal maior que a temperatura de derretimento do pó. Isso causa uma ligação por fusão das áreas expostas da camada atual e com as regiões já solidificadas da camada subjacente.

Enquanto o feixe de laser se move sobre a camada de pó e sobre o material derretido, um processo de recozimento brilhante é visível. Através do processo de interação térmica as partículas de metal são lançadas para fora da parte derretida. As informações sobre a superfície a ser exposta da respectiva camada são transferidas do computador para o escâner de controle.

Como passo final é realizado a descida da plataforma (item 9 na figura) na respectiva espessura da camada. Deste modo, é possível produzir componentes tridimensionais (item 7 na figura). O ciclo inteiro do processo então se repete. Por exemplo, para fazer um componente de 60 mm com uma espessura de pó 30 μm será preciso repetir o ciclo 2000 vezes.

Para evitar a oxidação do metal, encontra-se no processo um gás atmosférico inerte, gás argônio. Além disso, a base de montagem é aquecida a 200 °C para que as tensões residuais no componente sejam reduzidas, maiores detalhes podem ser observados na seção 3.3.5.

Após a finalização da fabricação do componente, ele é retirado da máquina para realizar o pós-processamento. Segundo [23], apesar da grande variedade de vantagens do processo de fabricação por SLM, a qualidade superficial do componente produzido pela máquina de SLM é geralmente inferior. Sendo que [24] testemunhou que a qualidade superficial das paredes verticais é menor do que das superfícies horizontais, devido às partículas parcialmente derretidas. Para isso, em ordem de melhorar a qualidade superficial do componente fabricado pela máquina de SLM, várias técnicas de pós-processamento

incluindo jateamento de areia, maquinaria, ataque químico, eletro-polimento e jateamento a plasma são empregadas.

3.3 Principais componentes e seus princípios de funcionamento

Nesta seção serão abordados quais são os principais componentes que fazem parte de todo o sistema da máquina de SLM, com breves explicações sobre seus funcionamentos.

3.3.1 Plataforma

A plataforma é responsável por dar espessura à camada de pó que será depositada pelo dispositivo de deposição de pó. Pode ser vista no item 9 da figura 17. Seu esquema de ligação pode ser visto na figura 18 abaixo. As cores verdes representam as entradas para o controle da plataforma e as cores azuis e laranjas representam as saídas do controlador. As setas azuis são enviadas para a interface para verificação de status e as setas laranjas são comandos enviados ao motor para parar ou rotacionar no sentido horário ou anti-horário, alterando a altura da plataforma.

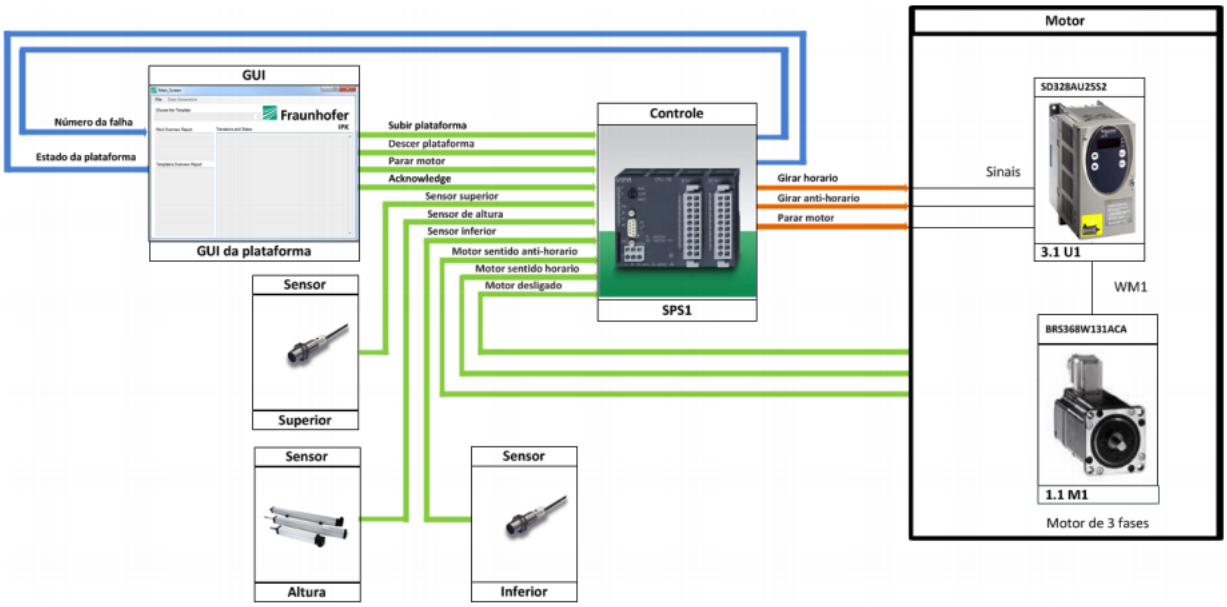

Figura 18 – Esquemático da ligação da plataforma

A plataforma contém um motor de passo de 3 fases da empresa Berger Lahr de referência BRS368W131ACA. Sua alimentação limite é de 325V DC - 230 VAC com corrente nominal de operação de 0.9 A. Além disso, sua resolução é de 200 a 10000 passos (1.8°C até 0.036°C por passo).

O driver para acionamento do motor é o SD328AU25S2, fabricado pela empresa *Schneider Electric*. Sua alimentação limite varia de 100 VAC a 240 VAC, com potência nominal de 180W em 115V a 270W em 230V.

Os sensores de fim de curso da plataforma são dois sensores indutivos, capazes de medir a proximidade de objetos metálicos que entra em seu campo magnético. Para o sensor de altura é utilizada uma régua potenciométrica, transdutor potenciométrico capaz de monitorar, medir e controlar movimentos mecânicos.

3.3.2 Dispositivo de deposição de pó

O dispositivo de deposição de pó é responsável por depositar e espalhar a camada de pó sobre o bloco de montagem do componente. O esquemático do sistema pode ser visto na figura 19 abaixo. A descrição das cores é igual ao descrito na plataforma da figura 18.

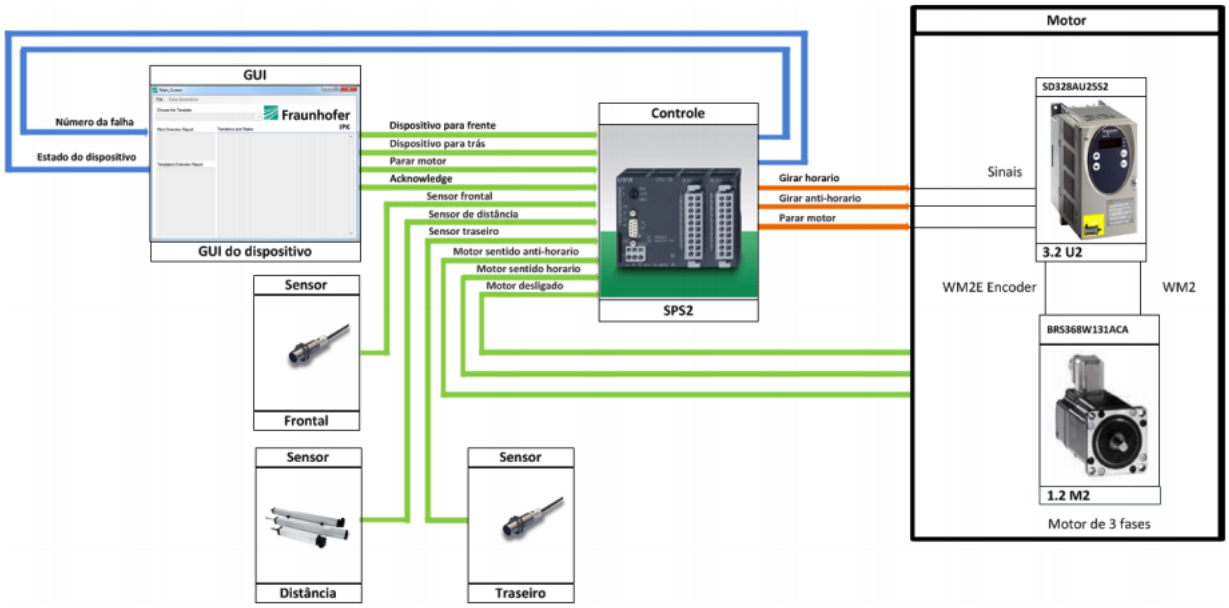

Figura 19 – Esquemático da ligação do dispositivo de deposição de pó

Os sensores, o motor e o driver de acionamento para o dispositivo de deposição de pó são os mesmos modelos dos utilizados para a plataforma, porém nesse caso o motor utiliza *encoder*. "*Encoder* é um dispositivo eletro-mecânico que converte a posição angular, ou movimentação de um eixo para um sinal analógico ou digital. Um monitoramento que rotaciona a medida que o motor gira, compara uma posição de referência com a posição atual e informa como resposta um erro se a diferença exceder um limite específico. [25]"

3.3.3 Sistemas de carregamento 1, 2 e 3

Segundo observado na máquina, o sistema de carregamento 1 se encontra sobre o tanque principal do dispositivo de deposição de pó e tem um eixo com depressões, assim chamadas câmaras, que transportam uma quantidade definida de pó de metal para dentro do tanque, que pode ser visto na parte a) da figura 20. Uma unidade aciona o eixo no

carregador, de modo que o pó de metal cai para dentro do dispositivo de deposição de pó, quando o mesmo estiver na posição traseira, como pode ser visto na figura 21 abaixo. Nessa figura é possível visualizar o dispositivo de deposição de pó na parte traseira da máquina, onde será realizado o carregamento do pó.

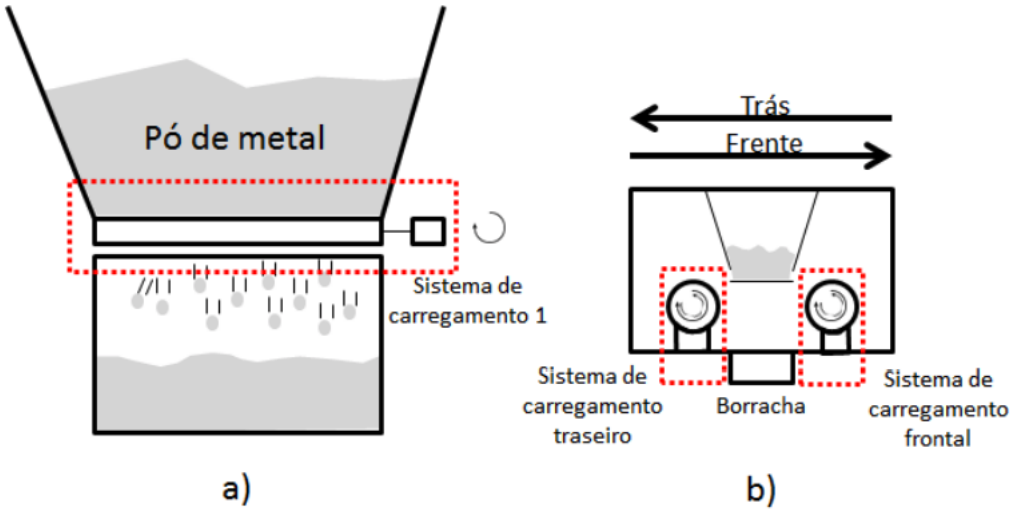

Figura 20 – Visão frontal e lateral do dispositivo de deposição de pó

Os sistemas de carregamento 2 e 3 são responsáveis por adicionar o pó de metal que será usado para fora do tanque de armazenamento, sendo depositado sobre a camada anterior. Os carregadores 2 e 3 estão localizados um na parte frontal e o outro na parte traseira do dispositivo de deposição de pó, que pode ser visto na figura 20 b).

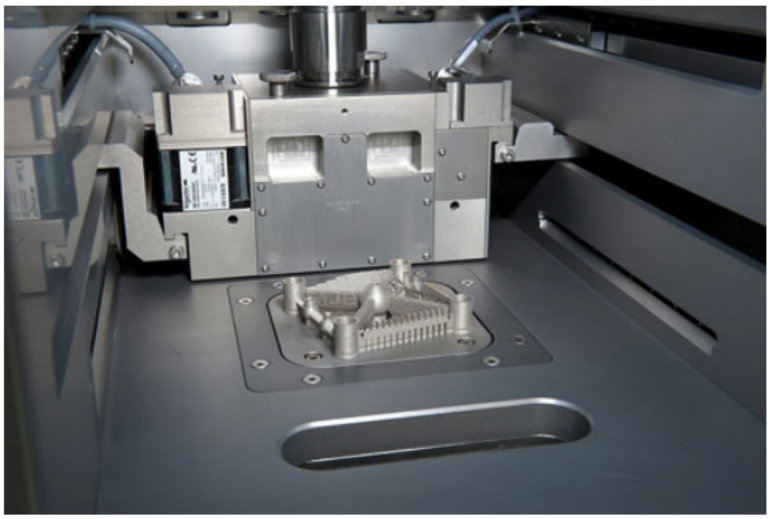

Figura 21 – Interior da câmara de construção [2]

3.3.4 Laser

Segundo [26], lasers são dispositivos que geram ou amplificam radiação em frequências de infravermelho, visíveis, ultravioleta ou regiões do espectro eletromagnético. Lasers operam ao utilizar um principio que foi originalmente inventado em frequências de micro-ondas, onde era chamado de amplificação de micro-onda por emissão ou radiação estimuladas.

Na máquina de SLM existe um laser na parte superior da câmara de onde o processo é realizado, item 1 da figura 17. O sistema de laser do processo é resfriado por canal de ar, retirando as excessivas temperaturas que o sistema de laser pode chegar. Evitando defeitos na máquina e que falhas no componente aconteçam.

3.3.5 Sistema de aquecimento

Segundo [3], o aumento da eficiência do processo de fabricação SLM ou a melhoria da qualidade dos componentes manufaturados pode ser atingido através de um sistema de aquecimento. Isso pode ser feito ao aquecer a base de construção em ordem de mitigar o gradiente de temperatura entre a mesma e a camada mais superior do componente a ser montado.

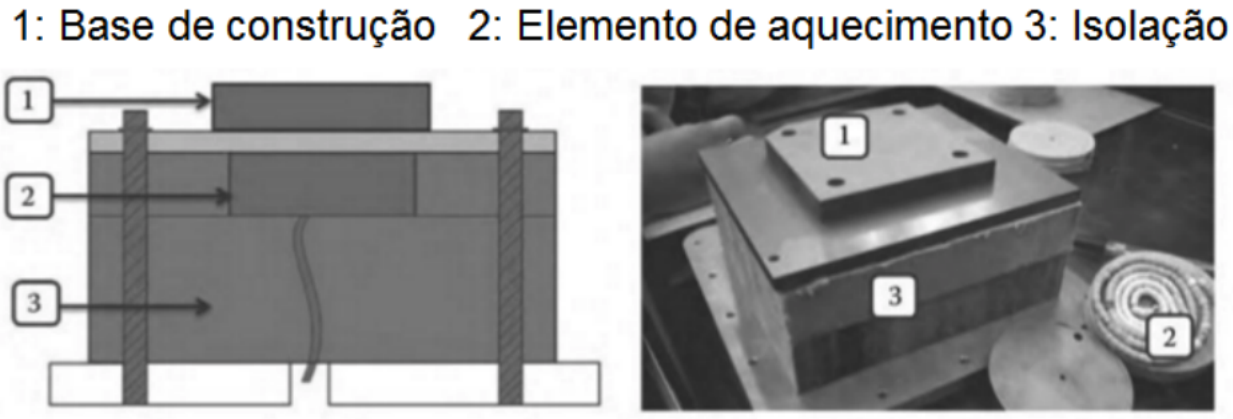

Figura 22 – Sistema de aquecimento [3]

A figura 22 mostra uma visão geral de como o módulo de aquecimento funciona. O elemento de aquecimento (rotulado como 2 na figura 22) é instalado abaixo da plataforma e anexado pelo material isolante. A temperatura da base de montagem pode ser monitorada por termopar. Para a máquina instalada nas dependências do Fraunhofer essa temperatura a ser mantida e controlada é de 200°C.

3.3.6 Sistema de refrigeração

De acordo com [4], o sistema de refrigeração resfria a água aquecida dentro do equipamento, que pode ser visto na figura 23 abaixo, e refrigera os sistemas da máquina,

por exemplo o laser, com ar refrigerado. Através do processo de convecção, os principais componentes da máquina são refrigerados para evitar que superaqueçam. Dessa forma, evitam-se defeitos na máquina, em seus equipamentos internos e no componente que está sendo manufaturado.

Figura 23 – Sistema de refrigeração a ar [4]

A unidade do sistema de refrigeração da máquina observada possui 2 mangueiras de refrigeração: abastecimento e retorno de água refrigerada. A água refrigerada circula em um circuito fechado dentro do sistema de refrigeração, para sempre estar a baixas temperaturas e refrigerar os outros subsistemas da máquina de SLM.

3.3.7 Sistema de filtragem

Segundo [27], numa máquina adjacente à máquina de SLM, encontra-se o sistema de filtragem. O sistema também é protegido com gás argônio, retirando o oxigênio para evitar que as partículas de pó do metal filtrado oxidem. O operador deve retirar o máximo de pó da máquina, colocando-o dentro do cilindro, fechando-o e o transferindo para o sistema de filtragem.

Com o cilindro de pó instalado no sistema, o pó cai e, através de vibração, as partículas que possuem tamanho suficientemente pequeno passam pelo filtro, sendo depositadas em outro cilindro, como pode ser visto na figura 24 abaixo. Ao final, o cilindro é retirado do sistema de filtragem e armazenado novamente, para que o pó de metal que ainda se encontra utilizável após a fabricação seja usado em futuros projetos.

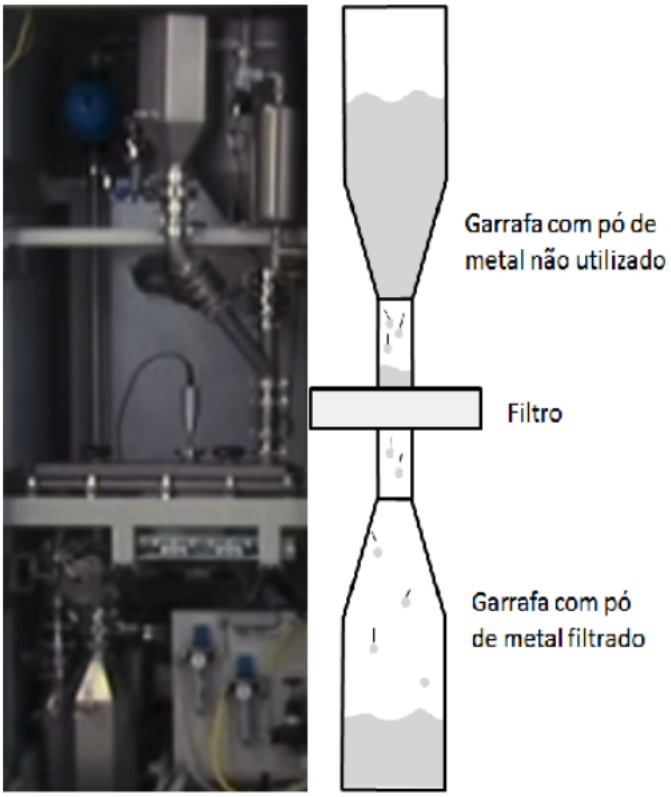

Figura 24 – Máquina de filtragem [5]

3.3.8 Sistema de vácuo

Conforme observado na máquina, antes de iniciar a depositar o pó por sobre a base de construção, a máquina retira oxigênio da atmosfera da câmara, para evitar que o pó de metal oxide. Para tal, a máquina de SLM abre uma sequência de válvulas e aciona o motor do sistema de vácuo para retirar o oxigênio.

Como pode ser visto na figura 25 abaixo, o motor retira o oxigênio pela saída de gás, item 6 da figura, que passa pelo sistema de válvulas automáticas do sistema e fica retido dentro do cilindro de gás oxigênio. Quando a concentração de gás oxigênio atinge seu nível desejado, o sistema fecha as válvulas, desliga o motor e permite que o gás de proteção seja adicionado.

Ao final da fabricação do componente, somente é possível abrir a porta da câmara da máquina caso a concentração de gás oxigênio dentro seja maior ou igual a 15%. Enquanto a máquina estiver em operação e o nível de gás de oxigênio atingir um nível indesejado, o motor será acionado e as válvulas abertas para diminuir a concentração do gás oxigênio de dentro da câmara.

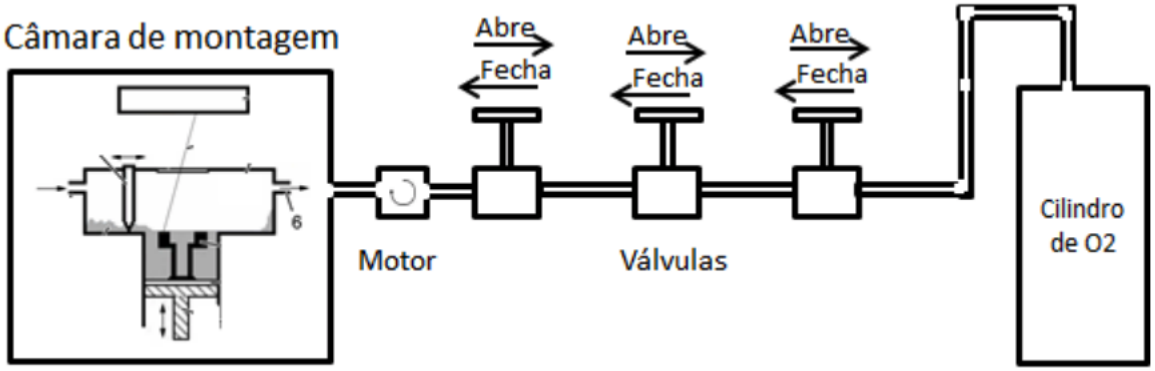

Figura 25 – Funcionamento do sistema de vácuo

3.3.9 Sistema de proteção a gás

Segundo observado em máquina, durante o processo de montagem a câmara de processo será continuamente preenchida com gás argônio, como pode ser visto na figura 26 abaixo, sendo o item 11 a entrada de gás do sistema. O gás argônio é introduzido no sistema de circulação de gás por um motor de circulação.

Segundo [3], o argônio é adicionado durante o processo SLM por ser um gás neutro, não reage com o sistema e evita que o pó do componente oxide. Ao final do processo, o gás de argônio é filtrado e adicionado novamente ao cilindro de gás argônio.

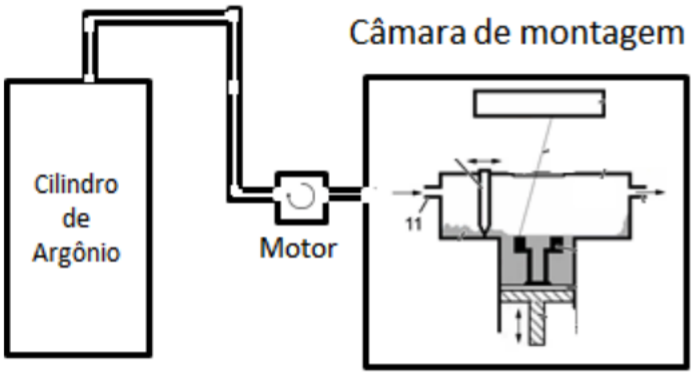

Figura 26 – Funcionamento do sistema de proteção a gás

O sistema é acionado quando o sensor de argônio, instalado dentro da câmara de construção do componente, indicar que a concentração de argônio está abaixo do nível desejado. Ao receber o sinal, o motor do sistema de proteção a gás será acionado, adicionando gás argônio na câmara, aumentando sua concentração dentro da mesma.

3.4 Trabalhos relacionados

Em [28] os autores abordam uma introdução sobre SLM, aplicações atuais e como é possível melhorar a produtividade do processo de SLM. A pesquisa cita um estudo

ilustrando que o processo de SLM oferece grande potencial para atender requisitos de clientes individuais com alta qualidade assim a baixo custo de produção. Entretanto, o atual estado do processo de SLM ainda não está adequado para produção em série, devido ao seu alto custo e produtividade baixa. Além disso, em [29] é descrito um estudo detalhado das mais importantes tecnologias e sistemas de manufatura aditiva por PBF. Neste estudo, o estado da arte de pesquisa de manufatura aditiva de componentes de metais focados em sistemas de manufatura aditiva por PBF foi realizado, contendo as mais importantes tecnologias e seus parâmetros.

Em [30] são descritos alguns estudos que examinam diferentes processos de SLM e Selective Laser Sintering (SLS) com relação às condições que são muitos importantes para a manufatura, tais como precisão, material, propriedades mecânicas, velocidade e confiabilidade. Características geométricas, propriedades mecânicas e aspectos econômicos foram testados para cada *benchmark* realizado. O estudo não tem como objetivo na comparação de resultados entre as técnicas de SLS/SLM, pois os processos diferiram em material do pó, equipamento e foco da pesquisa; precisão, velocidade ou propriedades mecânicas. O estudo foi realizado para entender as limitações de SLS/SLM e encontrar potenciais aplicações de manufatura.

Em [31] é estudado uma modelagem de uma furadeira utilizando as ferramentas de verificação CADP, SPIN e UPPAAL. Essa máquina foi escolhida por não ser de alta complexidade, mas ser complexa o suficiente por conter características interessantes para modelar e por já conter modelos de estudo, conforme em [32]. O modelo da máquina é classificado nas três ferramentas, mas nesse trabalho será descrito somente como foi realizada a modelagem pelo UPPAAL. Ainda em [31], o modelo é divido em 5 processos principais, eles são: *Processo da mesa giratória, Processo de prender, Processo da broca, Processo do testador e o Processo do controle principal.* Cada processo foi modelado em um *template* separado dos demais, como o realizado neste trabalho. Após, são verificadas as propriedades do sistema e fornecidas comparações entre as ferramentas, para verificar qual delas mostrou-se mais fácil ou mais trabalhosa de traduzir o modelo da ferramenta. Por fim, concluiu-se que o UPPAAL é a ferramenta mais simples para modelagem e verificação do que as demais estudadas.

Em [33], é descrito um estudo de caso de um *scanner* de água de uma indústria de semicondutores. É mostrado como as técnicas de *model checking* podem ser utilizadas para computar uma simples política para evitar *deadlock* e uma programação com rendimento otimizado, na ausência de erros. Prevenção de *deadlock* é a base do estudo de um modelo de estado finito utilizado, e então um modelo de autômato foi construído e analisado utilizando a ferramenta UPPAAL.

4 Modelagem da máquina de SLM

Para a modelagem da máquina de SLM e seus subsistemas no UPPAAL foram necessárias três etapas. Primeiro, analisou-se a implementação dos requisitos fornecidos pelo cliente, no Anexo A deste trabalho, e traduzindo-os para lógica temporal CTL. Segundo, foram identificados cada um dos sistemas e sua interação com os demais. Então, foram desenvolvidos os modelos iniciais e houve uma simulação preliminar. Por último, foram verificadas as propriedades formais dos modelos, verificando a aderência com a lista de requisitos da plataforma.

O processo de modelagem da plataforma se consistiu num aprendizado iterativo. Obtendo-se o modelo e o submetendo aos requisitos, para verificar a aderência do modelo à planta. Primeiramente foi criado o modelo mais simples, contendo somente estados de parado, subindo e descendo. Então, estados foram sendo adicionados para atender aos requisitos fornecidos pelo cliente. Por exemplo, em uma parte do processo o modelo funcionava respeitando os limites totais de 0 mm à 400 mm, porém não atendia à restrição dos limites úteis de 50 mm à 350 mm. A plataforma estava entre 0 e 50 mm ou entre 350 mm e 400 mm e o sistema não apontava falha. Assim, a modelagem do sistema se aperfeiçoava à medida que novas iterações foram feitas e os problemas eram corrigidos.

Na primeira seção deste capitulo é apresentada a identificação de todos os subsistemas que compõem a máquina de SLM e como eles interagem com o sistema de controle e a interface gráfica com o usuário. Em seguida, em cada nova seção, é descrito cada um dos sistemas. Nestas, são discutidos o estudo do sistema, explicando seu princípio de funcionamento, os requisitos usados como base para modelagem, principais entradas e saídas, a concepção do modelo em autômatos temporizados e as propriedades verificadas.

4.1 Identificação geral da interação entre os modelos

Para desenvolver os modelos, é necessário primeiramente entender como os componentes de uma máquina SLM e o usuário podem se comunicar, identificando as possíveis mensagens trocadas e comandos enviados, assim como as variáveis que podem ser necessárias para a descrição do modelo. A Figura 27 apresenta como é feita essa interação descrevendo o emissor, receptor e nome das mensagens trocadas.

Foram identificados 11 subsistemas associados a atuadores da máquina SLM. Cada subsistema possui um 'Controle' que interage com ao menos um componente. A figura 27 abaixo possui mais detalhes de todas as interações entre modelos.

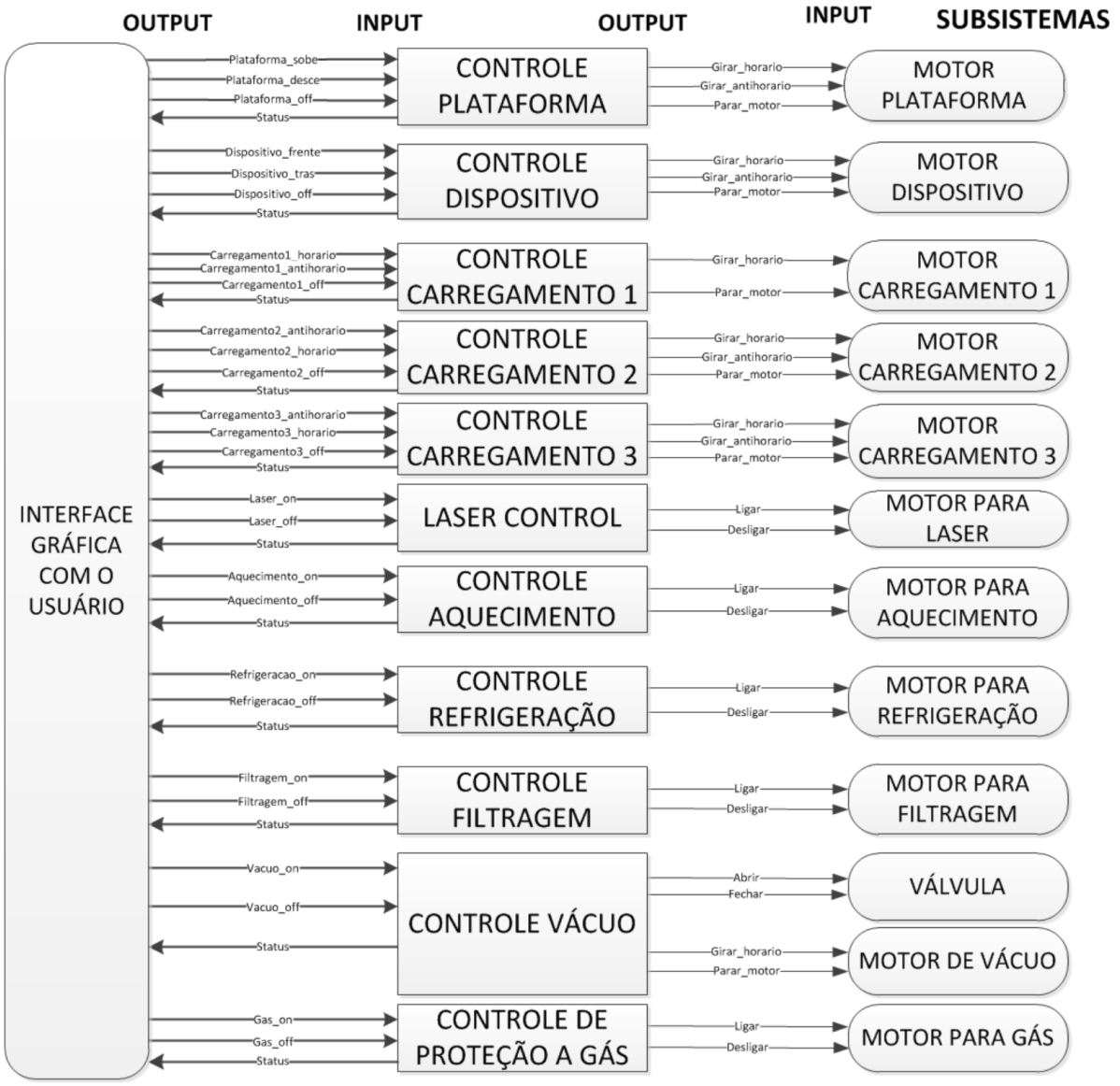

Figura 27 – Visão geral da interação dos modelos da máquina

4.2 Plataforma

O primeiro modelo de subsistema desenvolvido é relacionado à plataforma. Modelada para elevar e abaixar a altura da peça a ser feita, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada. A plataforma é utilizada, neste trabalho, como o modelo padrão para o dispositivo de deposição de pó, por terem funcionamento básico similar.

4.2.1 Conceitos gerais e requisitos do cliente

Conforme já descrito na seção 3.3.1, a plataforma é responsável por dar espessura à camada de pó que será depositada pelo dispositivo de deposição de pó. Ela se move somente na vertical, variando entre uma altura mínima de 0 mm (topo) à 400 mm (fundo),

e de altura de trabalho sendo entre 50 mm à 350 mm, de acordo com o requisito R6 do anexo A.

Para movimentar a plataforma um motor é utilizado, girando tanto no sentido horário, para subir a plataforma, quanto para o sentido anti-horário, para descer a plataforma. A mesma possui um sensor de altura analógico e dois sensores de fim-de-curso digitais, máximo e mínimo, como descrito no capítulo 3. Através destes sensores, o controlador tem conhecimento da altura da plataforma se encontra em todos os momentos. Os dois sensores digitais informam ao controlador se a plataforma atingiu as distâncias úteis trabalho entre 50 mm e 350 mm, enquanto estiver em operação, e o software garante que estes limites não serão ultrapassados, parando a plataforma quando tais situações ocorrerem.

Para a realização deste trabalho o cliente, orientador na instituição, forneceu os principais requisitos necessários para a modelagem e simulação da plataforma. No documento de requisitos estão contidos como a plataforma deve funcionar, quais são os estados, limites de funcionamento, suas falhas, severidades e como devem ser tratadas. Abaixo segue um subconjunto do documento de requisitos, o documento de requisitos completo para a plataforma, em inglês, pode ser encontrado no Anexo A - Lista de Requisitos.

- R1 - O computador deve processar cada comando;
- R2 - O tempo de processamento deve ser menor que 1 segundo;
- R3 - O computador deve mandar uma resposta para cada comando recebido;
- R4 - A altura mínima da plataforma deve ser 400 mm;
- R5 - A altura máxima da plataforma deve ser 0 mm;
- R6 - A distância de trabalho deve ser de 300 mm; e,
- R7 - O comando 'PARAR MOTOR' deve possuir a maior prioridade.

Assim como descrito no estudo da plataforma, os sensores são os *inputs* do subsistema, além dos botões para mover e parar a plataforma. Para a GUI, interface gráfica com o usuário, existem os seguintes botões:

- ***'Subir Plataforma'*** responsável por fazer a plataforma subir, se estiver nas condições corretas;
- ***'Descer Plataforma'*** responsável por fazer a plataforma descer, se estiver nas condições corretas;

- ***'Parar Motor'*** responsável por fazer a plataforma parar de se movimentar; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para a plataforma são as seguintes:

- ***'Ligar motor horário'*** sinal digital que manda o motor girar no sentido para a plataforma subir;
- ***'Ligar motor anti-horário'*** sinal digital que manda o motor girar no sentido para a plataforma descer;
- ***'Desligar motor'*** sinal digital que manda o motor parar de girar, fazendo com que a plataforma pare de se mover;
- ***'Alarme'*** variável que indica se houve alguma falha durante o sistema; e,
- ***'Estado da plataforma'*** variável que indica qual é o atual estado da plataforma.

Estas informações podem ser facilmente visualizadas na figura 28 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

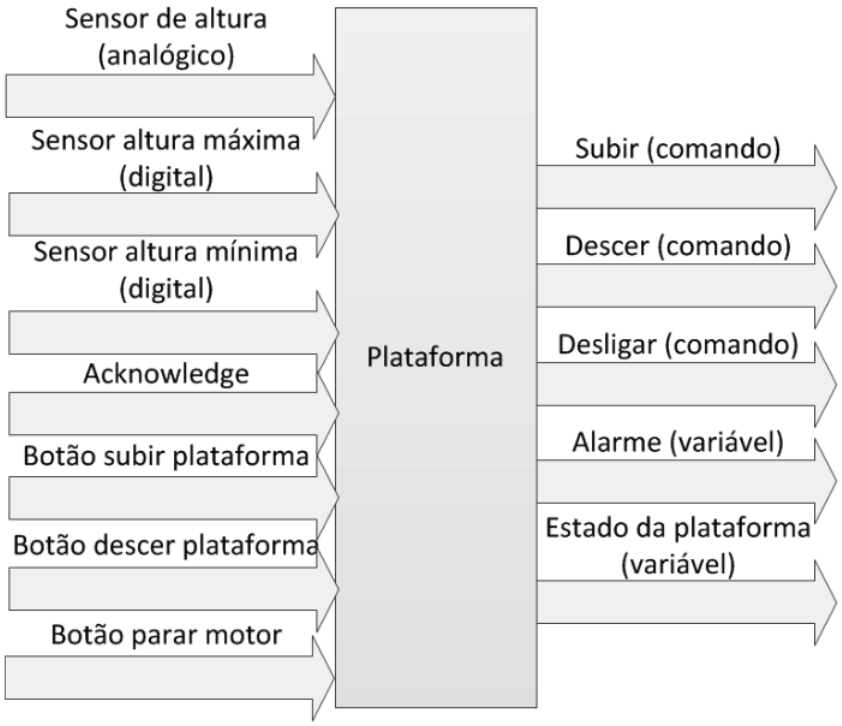

Figura 28 – Entradas e saídas do subsistema Plataforma

4.2.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo da plataforma, as entradas e saídas da figura 28 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em autômatos temporizados. A partir destes, foram estabelecidas os eventos que comandam a movimentação da plataforma. Os quatro *'templates'* foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'SENSORES' representa os sensores digitais, indicando se a plataforma ultrapassou os limites úteis de trabalho;
- 'MOTOR' recebe os comandos do *'template - COMMAND'* e gira no sentido horário ou anti-horário; e
- 'COMMAND' representa o controlador da plataforma.

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para a plataforma e o UPPAAL simula o modelo para verificar se o modelo da plataforma possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 29 abaixo. Nele, estão todos os comandos que o usuário consegue mandar e enviar no subsistema da plataforma, que são:

- `BS_M1_UP!` - Comando para a plataforma subir;
- `BS_M1_DOWN!` - Comando para a plataforma descer;
- `BS_M1_OFF!` - Comando para a plataforma parar; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

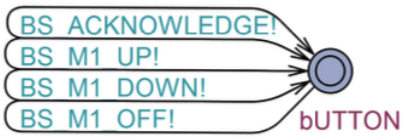

Figura 29 – *'Template GUI' - Plataforma*

Os sensores informam ao sistema se a plataforma atingiu o limite máximo ou mínimo de altura, referenciados como uP_LIMIT_SIGNAL e lOW_LIMIT_SIGNAL, respectivamente. Indicando ao sistema que a plataforma ultrapassou os limites úteis de trabalho entre 50 mm e 350 mm. Caso a plataforma tenha atingido as alturas máxima

ou mínima de uso, de 0 mm a 400 mm, então os autômatos *uPPER_LIMIT_SIGNAL* e *lOWER_LIMIT_SIGNAL* serão acionados.

As variáveis *VB_UPPER, VB_UPPER0, VB_LOWER* e *VB_LOWER0* são acionadas quando as alturas citadas anteriormente são atingidas, gerando o *flag* na variável. Informando que uma falha aconteceu no sistema. Após o usuário ser informado, a variável retorna ao seu valor inicial de 0. Os comandos que os sensores enviam podem ser vistos na figura 30 abaixo.

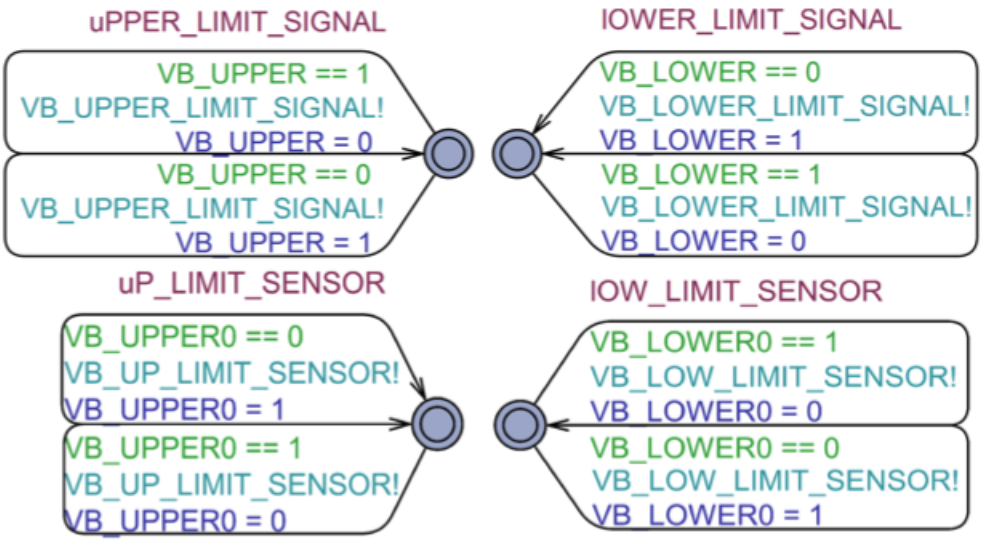

Figura 30 – *'Template Sensores' - Plataforma*

O modelo do motor, que pode ser visto na figura 31 e recebe os comandos do *'template - COMMAND'* e rotaciona no sentido horário, caso o comando for para subir a plataforma, ou no sentido anti-horário, caso o comando for para descer a plataforma.

Nesse *'template'* o motor inicialmente está no estado *'sTOPPED'*, ou seja, parado. A altura é atualizada sempre que o estado *'cLOCKWISE'* ou *'aNTI_CLOCKWISE'* é alcançado, utilizando as funções *'countUp()'* e *'countDown()'* caso a plataforma esteja subindo ou descendo, respectivamente. O motor deve sempre, no mínimo, incrementar um decrementar uma unidade de distância toda vez que é adicionado. Para garantir esse requisito os estados *'mOTOR_CLOCKWISE'* e *'mOTOR_ANTI_CLOCKWISE'* foram criados. Desta forma, o caminho *VB_START_M1_UP; VB_STOP_M1* não irá acionar o motor e logo em seguida desligá-lo, sem incrementar sua altura. Para o caso de decrementar é análogo.

Caso o *'template - COMMAND'* envie o comando `VB_START_M1_UP!` o motor irá para o estado *'cLOCKWISE'*, rotacionando o motor no sentido horário, diminuindo o valor da variável da altura da plataforma. A variável que indica qual o sentido de rotação do motor, `VB_MOTOR_STATE`, é atualizada para 1, indicando que o motor está girando no sentido horário. O motor para de rotacionar somente quando receber o comando

`BS_STOP_M1!` do *'template - COMMAND'*, atualizando a variável `VB_MOTOR_STATE` para 0, indicando que o motor desligou. O caso de rotacionar para o sentido anti-horário é análogo. Porém, a mensagem do *'template - COMMAND'* muda para `VB_START_M1_DOWN!`, aumentando o valor da variável da altura da plataforma e a variável `VB_MOTOR_STATE` é atualizada para 2, indicando que o motor está girando no sentido anti-horário, ou seja, descendo a plataforma.

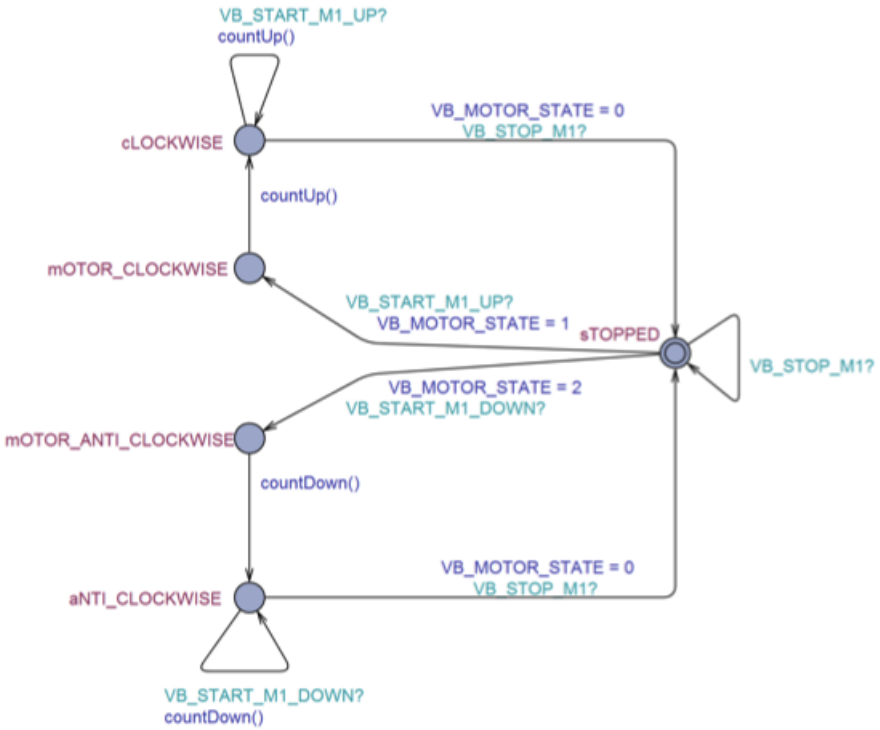

Figura 31 – *'Template Motor' - Plataforma*

4.2.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato que pode ser observado no material suplementar dentro da pasta *'Modelos'* e em seguida em *'Modelo Plataforma UPPAAL'*. Seu estado inicial é nomeado *'sTOPPED'*. A partir deste, a máquina verifica se variável da altura da plataforma, `AS_P_X`, está dentro dos limites úteis de trabalho (entre 50 mm e 350 mm). Se estiver, o sistema vai para o estado *'pLATFORM_ MIDDLE'* e atualiza a variável `VB_PLATFORM_STATE` para 0, indicando que está dentro dos limites úteis de trabalho. Nesse estado o sistema aguarda o comando `BS_M1_UP?` ou `BS_M1_DOWN?`, oriundos do *'Template GUI'*, para começar a movimentação da plataforma. Na figura 32 é apresentada uma versão hierárquica resumida do funcionamento do controle da plataforma, que será discutido nos próximos parágrafos.

Caso a variável de altura da plataforma não esteja dentro dos limites úteis de trabalho, ou seja, a variável `AS_P_X` seja maior que 350 mm ou menor que 50 mm, o

sistema irá para os estados *'pLATFORM_ BOTTOM'* ou *'pLATFORM_ TOP'*, respectivamente. O sistema no estado *'pLATFORM_ TOP'* significa que o valor da variável `AS_P_X` é menor do que 50 mm, ultrapassando os limites de segurança de funcionamento da máquina. A variável `VB_PLATFORM_STATE` é atualizada para 1 e a variável de falha, nomeada `VB_FAILURE_NUMBER`, é atualizada para 2, indicando que a falha do sistema número 2 ocorreu. Nesse estado o sistema não consegue subir a plataforma, caso o sistema receba o comando `BS_M1_UP?`, nada acontecerá. O sistema só poderá mover a plataforma quando o comando `BS_M1_DOWN?` for recebido.

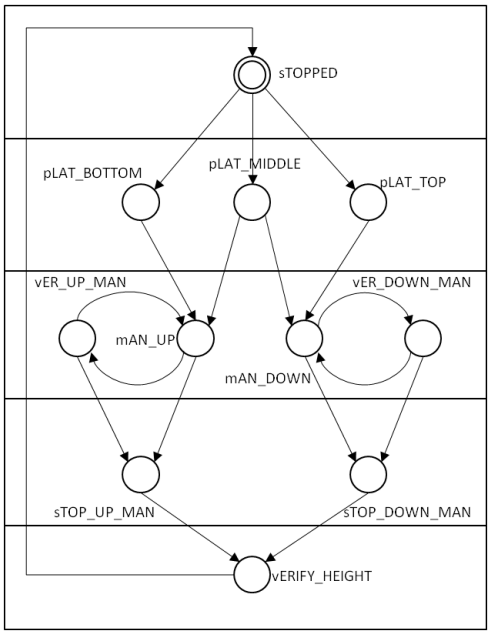

Figura 32 – *'Template Command'* - *Plataforma*

Para o estado *'pLATFORM_ BOTTOM'* é análogo, as mudanças são que a variável de falha, `VB_PLATFORM_STATE`, é atualizada para 2 e a variável `VB_FAILURE_NUMBER` para 1. Nesse estado, o sistema não consegue descer a plataforma. Caso o sistema receba o comando `BS_M1_DOWN?`, nada acontecerá. O sistema só poderá mover a plataforma quando o comando `BS_M1_UP?` for recebido.

Em modo normal, o sistema estará no estado *'pLATFORM_ MIDDLE'* aguardando os comandos `BS_M1_UP?` ou `BS_M1_DOWN?` da GUI. Ao receber o comando `BS_M1_UP?`, o sistema mudará para o estado *'mAN_ UP'* e atualizará uma variável de controle que salva a altura no último momento do estado, `AS_P_X_ANT`. Nesse estado o sistema enviará o comando `VB_START_M1_UP!`, caso `AS_P_X` for maior que o limite superior, o motor iniciará o seu processo de rotacionar no sentido horário, diminuindo a variável `AS_P_X`, como já foi explicado no tópico sobre o *'Template Motor'*.

Ao enviar o comando `VB_START_M1_UP!`, a variável `AS_P_X_ANT` salva o valor da altura e o sistema vai para o estado *'vERIFY_ MOTOR_ UP_ MAN'*. Nesse estado é verificado se a altura foi modificada e se a variável do motor foi atualizada para indicar

que está rotacionando no sentido horário. Se ambas as condições forem preenchidas, o motor retorna ao estado *'mAN_ UP'* e continua esse ciclo até que o limite de altura seja atingido ou o comando BS_M1_OFF? seja enviado.

Com o recebimento do comando BS_M1_OFF? o sistema muda do estado *'mAN_ UP'* para *'sTOP_ MAN_ UP'*. Nessa transição o *'template - COMMAND'* envia o comando `BS_M1_OFF!` fazendo com que o motor pare de rotacionar. O sistema irá automaticamente para o próximo estado, *'vERIFY_ HEIGHT'*, para verificar se a variável `AS_P_X` está dentro dos limites úteis de trabalho. Se estiver, é verificado se a variável `VB_MOTOR_STATE` é igual a 0, indicando que o motor está parado, e retorna ao estado inicial *'sTOPPED'*, recomeçando todo o processo.

O sistema de funcionamento caso receba a mensagem `BS_M1_DOWN?` é análogo. Se, por ventura, ocorrer algo no sistema que não esteja previsto no que foi anteriormente descrito, por exemplo, o sistema receber o comando `BS_M1_DOWN?` durante o estado *'mAN_ UP'* ou o valor de `AS_P_X` for maior ou menor que os limites úteis de trabalho após o estado *'sTOP_ MAN_ UP'*, serão indicadas diferentes falhas no sistema, fazendo com que o sistema aponte qual falha ocorreu e qual a sua severidade. Ao ocorrer a falha, é necessário o envio do comando `ACKNOWLEDGE`, para que o sistema retorne ao seu estado inicial *'sTOPPED'*.

4.2.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis, se os limites úteis de trabalho são respeitados e quais os possíveis estados do motor para cada um. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades da plataforma verificadas pode ser visto no Apêndice B.

A primeira verificação de propriedades a ser feita é conferir se o sistema não sofrerá algum tipo de bloqueio em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira, significando que o sistema não ficará bloqueado em um estado para sempre. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar, conhecido como lifelock.

Uma propriedade verificada bastante importante é se todos os estados do sistema são alcançáveis, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 33 abaixo. Aqui, cada estado do controle

deverá conter uma propriedade, como a descrita abaixo. Checker do UPPAAL:

E<> 'estado do Controlador'

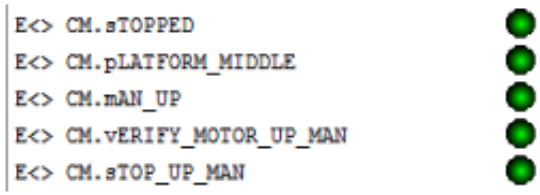

Figura 33 – Propriedades dos estados alcançáveis - Plataforma

Outra propriedade importante é verificar se em todos os estados os limites úteis de trabalho são respeitados, ou seja, se a variável `AS_P_X` está entre os valores 50 mm e 350 mm durante todo o processo. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz 'verde', conforme a figura 34 abaixo. Aqui, cada estado do controle deverá conter duas propriedade. Checker do UPPAAL:

A[] 'estado do Controlador' imply VB_UPPER_LIMIT <= AS_P_X

A[] 'estado do Controlador' imply AS_P_X <= VB_LOWER_LIMIT

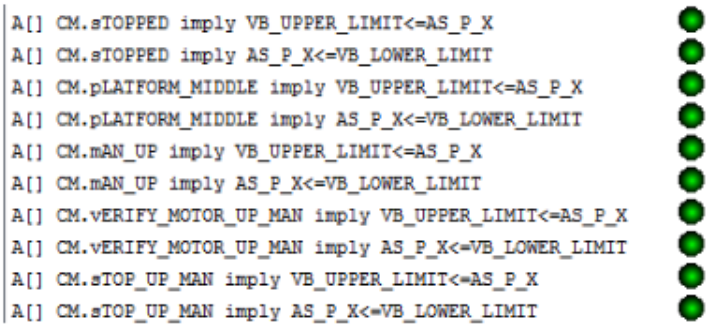

Figura 34 – Propriedades das verificações de altura - Plataforma

O último tipo de propriedade verificada é observar quais os possíveis estados do motor para cada estado do sistema. Essas propriedades são dadas abaixo, para as quais os resultados serão tanto a luz 'verde' quanto a luz 'vermelha', conforme a figura 35. Cada estado do controle deverá conter três propriedades, como as descritas abaixo. Checker do UPPAAL:

E<> 'estado do Controlador' && FE.sTOPPED

E<> 'estado do Controlador' && FE.cLOCKWISE

E<> 'estado do Controlador' && FE.aNTI_CLOCKWISE

Analisando a figura 35 abaixo, há várias luzes 'vermelhas' e várias luzes 'verdes'. Isso acontece, por que, por exemplo no estado *CM.sTOPPED* o motor deve estar, obrigatoriamente, parado. O motor não pode estar se movendo quando o sistema atingir esse estado,

caso contrário seria falha do sistema. Devido a esse fato é possível ver na imagem que no estado *'CM.sTOPPED'* que o estado do *'template - Motor'* o estado *'FE.sTOPPED'* está verde e os estados *'FE.cLOCKWISE'* e *'FE.aNTI_ CLOCKWISE'* estão vermelhos. No Apêndice B é apresentado todas as propriedades verificadas de todos os subsistemas desenvolvidos neste trabalho.

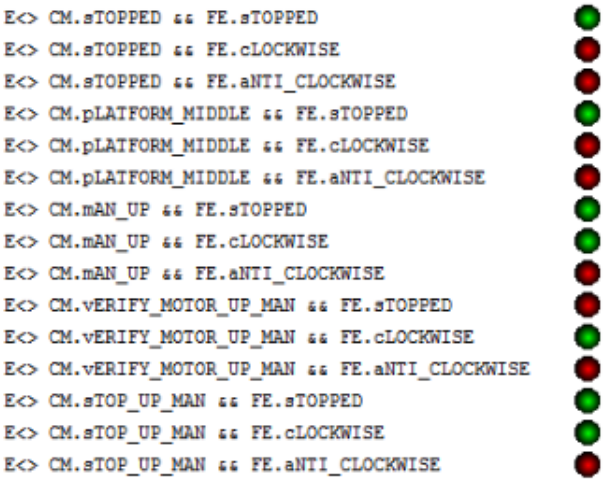

Figura 35 – Propriedades das verificações do motor - Plataforma

4.3 Dispositivo de deposição de pó

O dispositivo de deposição de pó foi modelado para ir para frente e para trás, conforme os comandos do usuário, e indicar falhas quando alguma situação indesejada ocorre-se.

4.3.1 Conceitos gerais

Conforme já descrito na seção 3.3.2, o dispositivo de deposição de pó é responsável por depositar a camada de pó. Ele se move somente na horizontal, de trás para frente e vice-versa, variando entre uma distância mínima de 0 mm (trás) até 600 mm (frente), e de distância de trabalho sendo entre 50 mm à 500 mm.

Para realizar o movimento do dispositivo de deposição de pó um motor é utilizado, girando tanto no sentido horário, para levar o dispositivo de deposição de pó para trás, quanto para o sentido anti-horário, para levar o dispositivo de deposição de pó para frente.

O dispositivo de deposição de pó possui um sensor de distância analógico e dois sensor de fim-de-curso, máximo e mínimo, digitais. Através desses sensores, o controlador tem conhecimento de onde o dispositivo de deposição de pó se encontra em todos os momentos. Os dois sensores digitais informam ao controlador se o dispositivo de deposição de pó atingiu as distâncias de trabalho entre 50 mm e 500 mm enquanto estiver em operação, e o software garante que esses limites não serão ultrapassados, parando o dispositivo de deposição de pó quando tais situações ocorrerem. Outro sensor é o contador, que realiza a contagem do *encoder*, quantidade de passos que o motor deve realizar. Esses sensores são os *inputs* do subsistema, além dos botões para mover e parar o dispositivo de deposição de pó.

O cliente não forneceu os requisitos necessários para a modelagem e simulações do sistema.

Assim como descrito no estudo do dispositivo de deposição de pó, os sensores são os inputs do subsistema, além dos botões para mover e parar a plataforma. Para a GUI existem os seguintes botões:

- ***'Para frente Dispositivo'*** responsável por fazer o dispositivo de deposição de pó ir para frente, se estiver nas condições corretas;
- ***'Para trás Dispositivo'*** responsável por fazer o dispositivo de deposição de pó ir para trás, se estiver nas condições corretas;
- ***'Parar Motor'*** responsável por fazer o dispositivo de deposição de pó parar de se movimentar; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o dispositivo de deposição de pó são as seguintes:

- ***'Ligar motor horário'*** sinal digital que manda o motor girar no sentido para o dispositivo de deposição de pó ir para trás;
- ***'Ligar motor anti-horário'*** sinal digital que manda o motor girar no sentido para o dispositivo de deposição de pó para ir para frente;
- ***'Desligar motor'*** que é um sinal digital que manda o motor parar de girar, fazendo com que o dispositivo de deposição de pó pare de se mover;
- ***'Alarme'*** variável que indica se houve alguma falha durante o sistema; e,
- ***'Estado do dispositivo'*** variável que indica qual é o atual estado do dispositivo de deposição de pó.

Essas informações podem ser facilmente visualizadas na figura 36 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

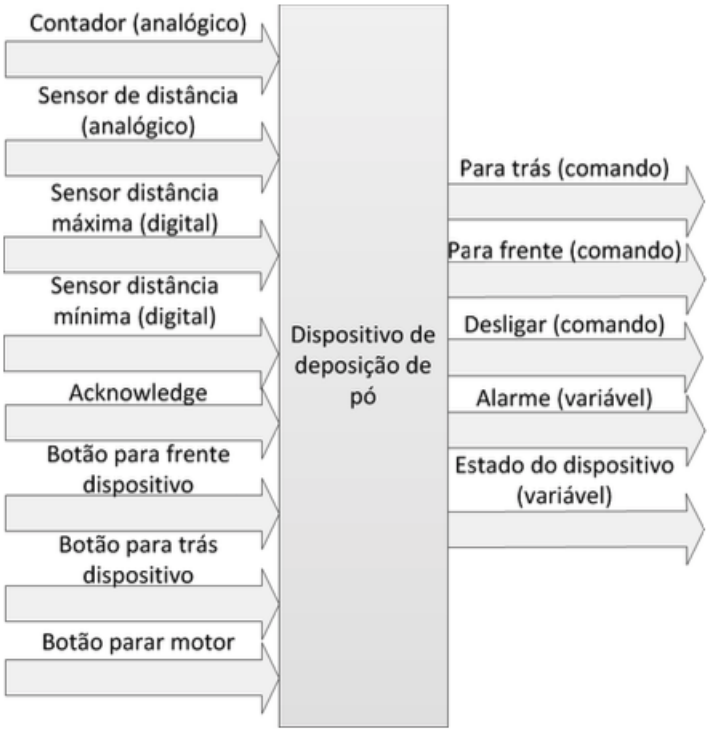

Figura 36 – Entradas e saídas do subsistema Dispositivo de deposição de pó

4.3.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do dispositivo de deposição de pó, as entradas e saídas da figura 36 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a

modelagem em autômatos temporizados. A partir destes, foram estabelecidas os eventos que comandam a movimentação do dispositivo de deposição de pó. Os cinco *'templates'* foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'SENSORES' representa os sensores digitais, indicando se o dispositivo de deposição de pó ultrapassou os limites úteis de trabalho;
- 'ENCODER' representa o *encoder* do sistema;
- 'MOTOR' recebe os comandos do *'template - COMMAND'* e gira no sentido horário ou anti-horário; e
- 'COMMAND' representa o controlador do dispositivo de deposição de pó.

Os componentes de saída do primeiro *'template'*, a GUI, são os botões digitais. Ele é o responsável por enviar os comandos para o dispositivo de deposição de pó e o UPPAAL simula o modelo, para verificar se o modelo possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 65, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema do dispositivo de deposição de pó, que são:

- `BS_M1_BACK!` - Comando para o dispositivo de deposição de pó ir para trás;
- `BS_M1_FRONT!` - Comando para o dispositivo de deposição de pó ir para frente;
- `BS_M1_OFF!` - Comando para o dispositivo de deposição de pó parar; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O *encoder* realiza a quantidade de passos dados pelo motor de passo. O *template* do *encoder* pode ser visto na figura 66, no apêndice B.

O *template* dos sensores informa ao sistema se o dispositivo de deposição de pó atingiu o limite máximo ou mínimo de distância. Indicando ao sistema que o dispositivo de deposição de pó ultrapassou os limites úteis de trabalho. Os comandos que os sensores enviam podem ser vistos na figura 67, no apêndice B.

O modelo do motor, que pode ser visto na figura 68, no Apêndice B, recebe os comandos do *'template - COMMAND'* e rotaciona no sentido horário, caso o comando for para mandar o dispositivo de deposição de pó para trás, ou no sentido anti-horário, caso o comando for para mandar o dispositivo de deposição de pó para frente.

O modelo do motor é análogo ao motor da plataforma. Neste caso, ao motor girar no sentido horário o dispositivo de deposição de pó irá para trás e variável de distância será decrementada. Caso o motor gire no sentido anti-horário o dispositivo irá para frente e a variável de distância será incrementada.

4.3.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato que pode ser observado no material suplementar dentro da pasta *'Modelos'* e em seguida em *'Modelo Dispositivo de Deposição de Pó UPPAAL'*. Seu estado inicial é nomeado *'sTOPPED'*. A partir deste, a máquina verifica se variável da distância do dispositivo de deposição de pó, `AS_P_X`, está dentro dos limites úteis de trabalho (entre 50 mm e 500 mm). Se estiver, o sistema vai para o estado *'rECOATER_MIDDLE'* e atualiza a variável `VB_RECOATER_STATE` para 0, indicando que está dentro dos limites úteis de trabalho. Nesse estado o sistema aguarda o comando `BS_M1_BACK?` ou `BS_M1_FRONT?` para começar a etapa de movimentar o dispositivo de deposição de pó.

Caso a variável de distância do dispositivo de deposição de pó não esteja dentro dos limites úteis de trabalho, ou seja, a variável `AS_P_X` seja maior que 500 mm ou menor que 50 mm, o sistema irá para os estados *'rECOATER_FRONT'* ou *'rECOATER_BACK'*, respectivamente. O sistema no estado *'rECOATER_BACK'* significa que o valor da variável `AS_P_X` é menor do que 50 mm, ultrapassando os limites de segurança de funcionamento da máquina. A variável `VB_RECOATER_STATE` é atualizada para 1 e a variável de falha, `VB_FAILURE_NUMBER`, é atualizada para 2, indicando que a falha do sistema número 2 ocorreu. Nesse estado o sistema não consegue enviar o dispositivo de deposição de pó para trás, caso o sistema receba o comando `BS_M1_BACK?`, nada acontecerá. O sistema só poderá mover o dispositivo de deposição de pó quando o comando `BS_M1_FRONT?` for recebido.

Para o estado *'rECOATER_FRONT'* é análogo, as mudanças são que a variável `VB_RECOATER_STATE` é atualizada para 2 e a variável da falha, `VB_FAILURE_NUMBER`, para 1. Nesse estado, o sistema não consegue levar o o dispositivo de deposição de pó para frente. Caso o sistema receba o comando `BS_M1_FRONT?`, nada acontecerá. O sistema só poderá mover o dispositivo de deposição de pó quando o comando `BS_M1_BACK?` for recebido.

Em modo normal de operação, o sistema estará no estado *'rECOATER_MIDDLE'* aguardando os comandos `BS_M1_BACK?` ou `BS_M1_FRONT?` da GUI. Ao receber o comando `BS_M1_BACK?`, o sistema mudará para o estado *'mAN_BACK'* e atualizará uma variável de controle que salva a distância no último momento do estado, `AS_P_X_ANT`. Nesse estado o sistema enviará o comando `VB_START_M1_BACK!`, caso `AS_P_X` for maior que o limite superior, o motor iniciará o seu processo de rotacionar no sentido horário, diminuindo a variável `AS_P_X`, como já foi explicado no tópico sobre o *'Template Motor'*.

Ao enviar o comando `VB_START_M1_BACK!`, a variável da altura anterior, `AS_P_X_ANT`, salva o valor da distância e o sistema vai para o estado *'vERIFY_MOTOR_BACK_MAN'*. Nesse estado é verificado se a distância foi modificada e se a variável do motor foi atualizada para indicar que está rotacionando no sentido horário. Se ambas as condições forem preenchidas, o motor retorna ao estado *'mAN_BACK'* e atualiza a variável `VB_ENCODER`, responsável por representar a quantidade de passos que o motor deve realizar, e continua esse ciclo até que o limite de distância seja atingido ou o comando `BS_M1_OFF?` seja enviado.

Com o recebimento do comando `BS_M1_OFF?` o sistema muda do estado *'mAN_BACK'* para *'sTOP_BACK_MAN'*. Nessa transição o *'template - COMMAND'* envia o comando `BS_M1_OFF!` fazendo com que o motor pare de rotacionar. O sistema irá automaticamente para o próximo estado, *'vERIFY_DISTANCE'*, para verificar se a variável `AS_P_X` está dentro dos limites úteis de trabalho. Se estiver, é verificado se a variável `VB_MOTOR_STATE` é igual a 0, indicando que o motor está parado, e retorna ao estado inicial *'sTOPPED'*, recomeçando todo o processo.

O sistema de funcionamento caso receba a mensagem `BS_M1_FRONT?` é análogo. Se, por ventura, ocorrer algo no sistema que não esteja previsto no que foi anteriormente descrito, por exemplo, o sistema receber o comando `BS_M1_FRONT?` durante o estado *'mAN_BACK'* ou o valor de `AS_P_X` for maior ou menor que os limites úteis de trabalho definidos após o estado *'sTOP_MAN_BACK'*, serão indicadas diferentes falhas no sistema, fazendo com que o sistema aponte qual falha ocorreu e qual a sua severidade. Ao ocorrer a falha, é necessário o envio do comando `ACKNOWLEDGE!`, para que o sistema retorno ao seu estado inicial *'sTOPPED'*.

4.3.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis, se os limites úteis de trabalho são respeitados e quais os possíveis estados do motor para cada um. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do dispositivo de deposição de pó verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação de propriedades a ser feita é conferir se o sistema não sofrerá algum tipo de bloqueio em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira, significando que o sistema não ficará bloqueado em um estado para sempre. Porém, essa propriedade não representa o não travamento do sistema em um

ciclo de dois ou mais estados que não é possível escapar, conhecido como lifelock.

Uma relevante verificação de propriedades é da alcançabilidade de todos os estados do sistema, ou seja, se em algum momento do funcionamento da máquina aquele estado será atingido. Estas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 37. Aqui, cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes. Checker do UPPAAL:

E<> 'estado do Controlador'

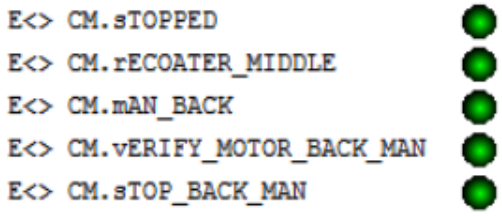

Figura 37 – Propriedades dos estados alcançáveis - Dispositivo de deposição de pó

Outra propriedade importante é verificar se em todos os estados os limites úteis de trabalho são respeitados, ou seja, se a variável `AS_P_X` está entre os valores 50 mm e 500 mm durante todo o processo. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz 'verde', conforme a figura 38. Aqui, cada estado do controle deverá conter duas propriedades, como as descritas abaixo. Checker do UPPAAL:

A[] 'estado do Controlador' imply `VB_BACK_LIMIT` $<=$ `AS_P_X`

A[] 'estado do Controlador' imply `AS_P_X` $<=$ `VB_FRONT_LIMIT`

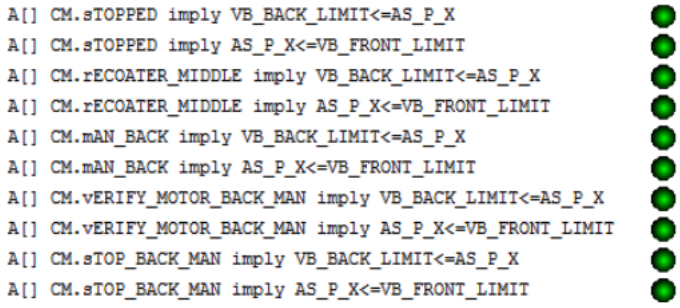

Figura 38 – Propriedades das verificações de distância - Dispositivo de deposição de pó

O último tipo de propriedade verificada é observar quais os possíveis estados do motor para cada estado do sistema. Essas propriedades são dadas abaixo, para as quais os resultados serão tanto a luz 'verde' quanto a luz 'vermelha', conforme a figura 39. Cada estado do controle deverá conter três propriedades, como as descritas abaixo. Checker do UPPAAL:

E<> 'estado do Controlador' && FE.sTOPPED

E<> 'estado do Controlador' && FE.cLOCKWISE

E<> 'estado do Controlador' && FE.aNTI_CLOCKWISE

Analisando a figura 39, há várias luzes 'vermelhas' e várias luzes 'verdes'. Isso acontece, por que, por exemplo no estado *CM.sTOPPED* o motor deve estar, obrigatoriamente, parado. O motor não pode estar se movendo quando o sistema atingir esse estado, caso contrário seria falha do sistema. Devido a esse fato é possível ver na imagem que no estado *'CM.sTOPPED'* que o estado do *'template - Motor'* o estado *'FE.sTOPPED'* está verde e os estados *'FE.cLOCKWISE'* e *'FE.aNTI_CLOCKWISE'* estão vermelhos. No Apêndice B é apresentado todas as propriedades verificadas de todos os subsistemas desenvolvidos nesse trabalho.

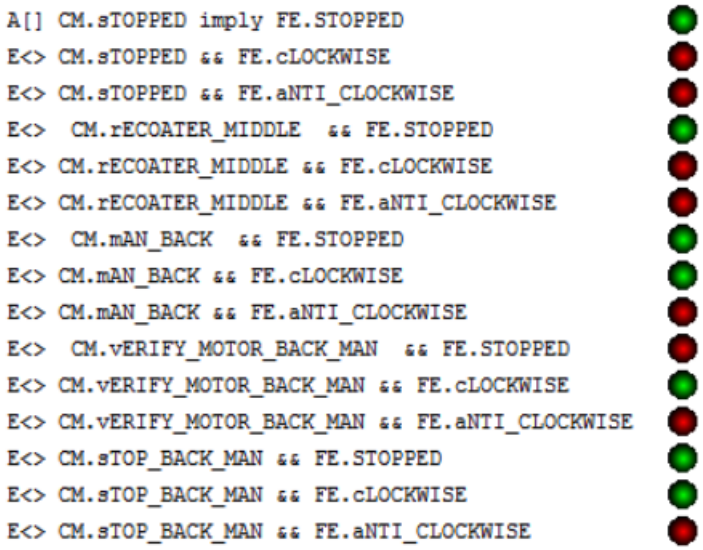

Figura 39 – Propriedades das verificações do motor - Dispositivo de deposição de pó

4.4 Sistema de carregamento 1

O sistema de carregamento 1 foi modelado para rotacionar o motor no sentido horário, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

4.4.1 Conceitos gerais

Conforme já foi descrito na seção 3.3.3, o sistema de carregamento 1 é responsável por carregar o tanque principal do dispositivo de deposição de pó com pó de metal. Ao adicionar o frasco no topo da máquina, o sistema de carregamento 1 gira seu motor para facilitar a queda do pó para dentro do tanque da máquina.

O sistema de carregamento 1 possui um sensor 'contador', que realiza a contagem do *encoder*, quantidade de passos que o motor deve realizar. Esse sensor é o *input* do subsistema, além dos botões para rotacionar e parar o sistema de carregamento 1.

O cliente não forneceu os requisitos necessários para a modelagem e simulações do sistema. Assim como descrito na seção 4.4.1, os sensores são inputs do subsistema. Para a GUI existem os seguintes botões:

- ***'Rotacionar horário'*** responsável por fazer o sistema de carregamento 1 rotacionar o motor no sentido horário;
- ***'Rotacionar anti-horário'*** responsável por fazer o sistema de carregamento 1 rotacionar o motor no sentido anti-horário;
- ***'Parar Motor'*** responsável por fazer o sistema de carregamento 1 parar o motor; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de carregamento 1 são as seguintes:

- ***'Ligar motor horário'*** sinal digital que manda o motor girar no sentido horário;
- ***'Ligar motor anti-horário'*** sinal digital que manda o motor girar no sentido anti-horário;
- ***'Desligar motor'*** que é um sinal digital que manda o motor parar de girar; e,
- ***'Alarme'*** variável que indica se houve alguma falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 40 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

4.4.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do sistema de carregamento 1, as entradas e saídas da figura 40 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o sistema de carregamento 1 transições. Os quatro *'template'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'ENCODER' representa o *encoder* do sistema;
- 'MOTOR' recebe os comandos do *'template - COMMAND'* e gira no sentido horário ou anti-horário; e
- 'COMMAND' representa o controlador do sistema de carregamento 1.

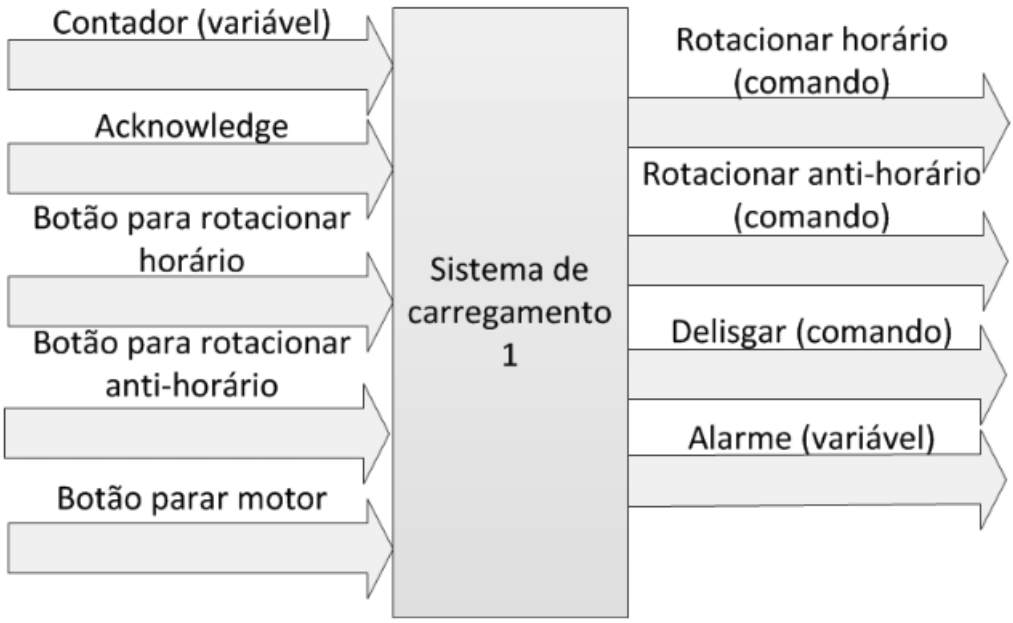

Figura 40 – Entradas e saídas do subsistema Sistema de carregamento 1

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema de carregamento 1 e o UPPAAL simula o modelo, para verificar se o modelo possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 70, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema de carregamento 1, que são:

- `BS_M1_CLOCKWISE!` - Comando para o sistema de carregamento 1 rotacionar o motor no sentido horário;
- `BS_M1_ANTICLOCKWISE!` - Comando para o sistema de carregamento 1 rotacionar o motor no sentido anti-horário;
- `BS_M1_OFF!` - Comando para o sistema de carregamento 1 parar; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo do *encoder* realiza a quantidade de passos que o motor de passo deve realizar. O *template* do *encoder* pode ser visto na figura 71, no apêndice B.

O modelo do motor, que pode ser visto na figura 72, no Apêndice B, recebe os comandos do *'template - COMMAND'* e rotaciona no sentido horário, caso o comando for para mandar o sistema de carregamento 1 rotacionar para o sentido horário, mas não para o sentido anti-horário. O funcionamento é análogo ao motor da plataforma.

4.4.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 73, no Apêndice B. Seu estado inicial é nomeado *'sTOPPED'*. Nesse estado o sistema aguarda o comando `BS_M1_CLOCKWISE?` para começar a etapa de rotacionar o motor do sistema de carregamento 1.

Em modo normal de operação, o sistema estará no estado *'sTOPPED'* aguardando o comando `BS_M1_CLOCKWISE?` da GUI. Ao receber o comando `BS_M1_CLOCKWISE?`, o sistema mudará para o estado *'mAN_CLOCKWISE'*. Nesse estado o sistema enviará o comando `VB_START_M1_CLOCKWISE!`, caso `VB_COUNTER` seja maior do que 0, o motor iniciará o seu processo de rotacionar no sentido horário, como já foi explicado no tópico sobre o *'Template Motor'*.

Ao enviar o comando `VB_START_M1_CLOCKWISE!` o sistema muda para o estado *'vERIFY_MOTOR_CLOCKWISE_MAN'*. Nesse estado é verificado se a a variável do motor foi atualizada para indicar que está rotacionando no sentido horário. Se a condição for preenchida, o sistema retorna ao estado *'mAN_CLOCKWISE'* e atualiza a variável `VB_COUNTER`, diminuindo o valor dela em 1, responsável por representar a quantidade de passos que o motor deve realizar, e continua esse ciclo até que a variável `VB_COUNTER` seja 0 ou o comando `BS_M1_OFF?` seja enviado.

Com o recebimento do comando `BS_M1_OFF?` o sistema muda do estado *'mAN_CLOCKWISE'* para *'sTOP_BACK_CLOCKWISE'*. Nessa transição o *'template - COMMAND'* envia o comando `BS_M1_OFF!` fazendo com que o motor pare de rotacionar. O sistema irá automaticamente para o próximo estado, *'sTOP_NORMAL'*. Aqui é verificado se a variável do

motor, `VB_MOTOR_STATE`, é igual a 0, indicando que o motor está parado. Se for, o sistema retorna ao estado inicial *'sTOPPED'*, recomeçando todo o processo.

O sistema de funcionamento caso receba `BS_M1_ANTICLOCKWISE?` não existe. Caso essa mensagem seja enviada, o sistema irá ignorar a mensagem.

4.4.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis e quais os possíveis estados do motor para cada um. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do sistema de carregamento 1 verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação de propriedades a ser feita é conferir se o sistema não sofrerá algum tipo de bloqueio em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira, significando que o sistema não ficará bloqueado em um estado para sempre. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar, conhecido como lifelock.

Uma relevante verificação de propriedades é da alcançabilidade de todos os estados do sistema, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 41. Aqui, cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

Checker do UPPAAL:

E<> 'estado do Controlador'

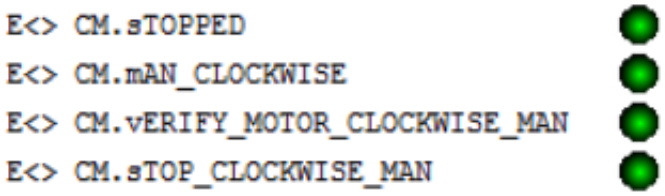

Figura 41 – Propriedades dos estados alcançáveis - Sistema de carregamento 1

Outra verificação é observar quais os possíveis estados do motor para cada estado do sistema. Essas propriedades são dadas abaixo, para as quais os resultados serão tanto a luz 'verde' quanto a luz 'vermelha', conforme a figura 42. Cada estado do controle deverá

conter três propriedades, como as descritas abaixo. Nesse trabalho foi notado que, se o controlador possuir n estados, logo deverão existir 3*n propriedades semelhantes.

E<> 'estado do Controlador' && FE.sTOPPED

E<> 'estado do Controlador' && FE.cLOCKWISE

E<> 'estado do Controlador' && FE.aNTI_CLOCKWISE

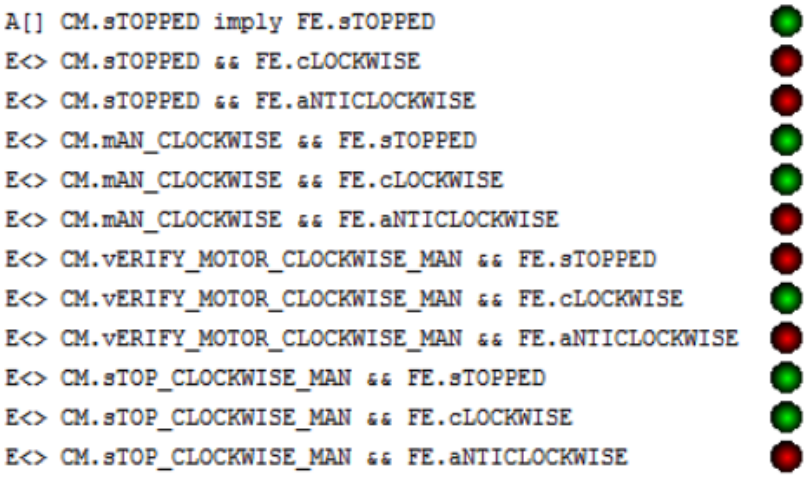

Figura 42 – Propriedades das verificações do motor - Sistema de carregamento 1

Analisando a figura 42, há várias luzes 'vermelhas' e várias luzes 'verdes'. Isso acontece, por que, por exemplo no estado *'CM.sTOPPED'* o motor deve estar, obrigatoriamente, parado. O motor não pode estar se movendo quando o sistema atingir esse estado, caso contrário seria falha do sistema. Devido a esse fato é possível ver na imagem que no estado *'CM.sTOPPED'* que o estado do *'template - Motor'* o estado *'FE.sTOPPED'* está verde e os estados *'FE.cLOCKWISE'* e *'FE.aNTI_CLOCKWISE'* estão vermelhos. No Apêndice B é apresentado todas as propriedades verificadas de todos os subsistemas desenvolvidos nesse trabalho.

4.5 Sistema de carregamento 2

O sistema de carregamento 2 foi modelado para rotacionar o motor no sentido horário ou anti-horário, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

4.5.1 Conceitos gerais

Conforme já foi descrito na seção 3.3.3, o sistema de carregamento 2 é responsável por carregar o dispositivo de deposição de pó sobre a camada de pó anterior. Ele rotaciona o motor no sentido horário e anti-horário. Através do *encoder* ele adiciona a quantidade de pó desejada para que uma nova camada do componente seja produzida.

O sistema de carregamento 2 possui um sensor 'contador', que realiza a contagem do *encoder*, quantidade de passos que o motor deve realizar. Esse sensor é o *input* do subsistema, além dos botões para rotacionar e parar o sistema de carregamento 2. Para a GUI existem os seguintes botões:

- ***'Rotacionar horário'*** responsável por fazer o sistema de carregamento 2 rotacionar o motor no sentido horário;
- ***'Rotacionar anti-horário'*** responsável por fazer o sistema de carregamento 2 rotacionar o motor no sentido anti-horário;
- ***'Parar Motor'*** responsável por fazer o sistema de carregamento 2 parar o motor; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de carregamento 2 são as seguintes:

- ***'Ligar motor horário'*** sinal digital que manda o motor girar no sentido horário;
- ***'Ligar motor anti-horário'*** sinal digital que manda o motor girar no sentido anti-horário;
- ***'Desligar motor'*** que é um sinal digital que manda o motor parar de girar; e,
- ***'Alarme'*** variável que indica se houve alguma falha durante o sistema.

Estas informações podem ser facilmente visualizadas na figura 43 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

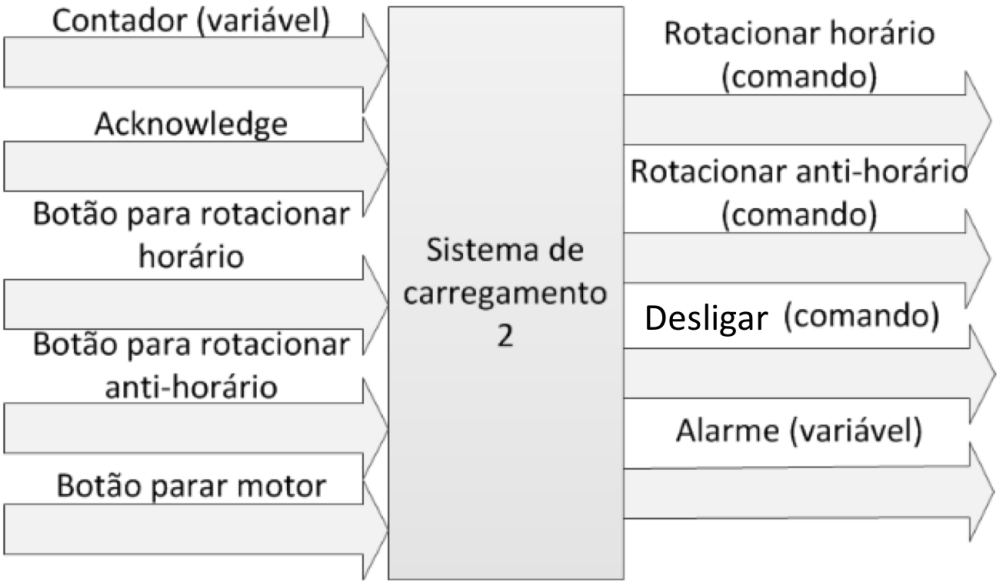

Figura 43 – Entradas e saídas do subsistema Sistema de carregamento 2

4.5.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do sistema de carregamento 2, as entradas e saídas da figura 43 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o sistema de carregamento 1 transições. Os quatro *'template'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'ENCODER' representa o *encoder* do sistema;
- 'MOTOR' recebe os comandos do *'template - COMMAND'* e gira no sentido horário ou anti-horário; e
- 'COMMAND' representa o controlador do sistema de carregamento 2.

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema de carregamento 2 e o UPPAAL simula o modelo, para verificar se o modelo possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 74, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema de carregamento 2, que são:

- `BS_M1_CLOCKWISE!` - Comando para o sistema de carregamento 2 rotacionar o motor no sentido horário;
- `BS_M1_ANTICLOCKWISE!` - Comando para o sistema de carregamento 2 rotacionar o motor no sentido anti-horário;
- `BS_M1_OFF!` - Comando para o sistema de carregamento 2 parar; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O *encoder* realiza a quantidade de passos que o motor de passo deve realizar. O *template* do *encoder* pode ser visto na figura 75, no apêndice B.

O modelo do motor, que pode ser visto na figura 76, no Apêndice B, recebe os comandos do *'template - COMMAND'* e rotaciona no sentido horário, caso o comando for para mandar o sistema de carregamento 2 rotacionar para o sentido horário, ou no sentido anti-horário, caso o comando for para mandar o sistema de carregamento 2 rotacionar para o sentido anti-horário. O funcionamento é análogo ao motor da plataforma.

4.5.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 77, no Apêndice B. Seu estado inicial é nomeado *'sTOPPED'*. A partir deste, nesse estado o sistema aguarda o comando `BS_M1_CLOCKWISE?` ou `BS_M1_ANTICLOCKWISE?` para começar a etapa de rotacionar os motores do sistema de carregamento 2.

Em modo normal de operação, o sistema estará no estado *'sTOPPED'* aguardando os comandos `BS_M1_CLOCKWISE?` ou `BS_M1_ANTICLOCKWISE?` da GUI. Ao receber o comando `BS_M1_CLOCKWISE?`, o sistema mudará para o estado *'mAN_CLOCKWISE'*. Nesse estado o sistema enviará o comando `VB_START_M1_CLOCKWISE!`, caso `VB_COUNTER` for maior do que 0, o motor iniciará o seu processo de rotacionar no sentido horário, como já foi explicado no tópico sobre o *'Template Motor'*.

Ao enviar o comando `VB_START_M1_CLOCKWISE!` o sistema muda para o estado *'vERIFY_MOTOR_CLOCKWISE_MAN'*. Nesse estado é verificado se a variável do motor foi atualizada para indicar que está rotacionando no sentido horário. Se a condição for preenchida, o sistema retorna ao estado *'mAN_CLOCKWISE'* e atualiza a variável `VB_COUNTER`, diminuindo o valor dela em 1, responsável por representar a quantidade de passos que o motor deve realizar, e continua esse ciclo até que a variável `VB_COUNTER` seja 0 ou o comando `BS_M1_OFF?` seja enviado.

Com o recebimento do comando `BS_M1_OFF?` o sistema muda do estado *'mAN_CLOCKWISE'* para *'sTOP_BACK_CLOCKWISE'*. Nessa transição o *'template - COMMAND'* envia

o comando `BS_M1_OFF!` fazendo com que o motor pare de rotacionar. O sistema irá automaticamente para o próximo estado, *'sTOP_NORMAL'*. É verificado se a variável `VB_MOTOR_STATE` é igual a 0, indicando que o motor está parado. Se for, o sistema retorna ao estado inicial *'sTOPPED'*, recomeçando todo o processo.

O sistema de funcionamento caso receba `BS_M1_ANTICLOCKWISE?` é análogo. Se, por ventura, ocorrer algo no sistema que não esteja previsto no que foi anteriormente descrito. Por exemplo, se o sistema receber o comando `BS_M1_ANTICLOCKWISE?` durante o estado *'mAN_CLOCKWISE'*, serão indicadas diferentes falhas no sistema, fazendo com que o sistema aponte qual falha ocorreu e qual a sua severidade. Ao ocorrer a falha, é necessário o envio do comando `ACKNOWLEDGE`, para que o sistema retorne ao seu estado inicial *'sTOPPED'*.

4.5.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis e quais os possíveis estados do motor para cada um. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do sistema de carregamento 2 verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação de propriedades a ser feita é conferir se o sistema não sofrerá algum tipo de bloqueio em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira, significando que o sistema não ficará bloqueado em um estado para sempre. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar, conhecido como lifelock.

Uma relevante verificação de propriedades é da alcançabilidade de todos os estados do sistema, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 44. Aqui, cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

Checker do UPPAAL:

E<> 'estado do Controlador'

Outra verificação é observar quais os possíveis estados do motor para cada estado do sistema. Essas propriedades são dadas abaixo, para as quais os resultados serão tanto a luz 'verde' quanto a luz 'vermelha', conforme a figura 45. Cada estado do controle deverá

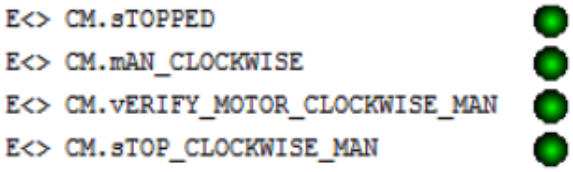

Figura 44 – Propriedades dos estados alcançáveis - Sistema de carregamento 2

conter três propriedades, como as descritas abaixo. Nesse trabalho foi notado que, se o controlador possuir n estados, logo deverão existir 3*n propriedades semelhantes.

E<> 'estado do Controlador' && FE.sTOPPED

E<> 'estado do Controlador' && FE.cLOCKWISE

E<> 'estado do Controlador' && FE.aNTI_ CLOCKWISE

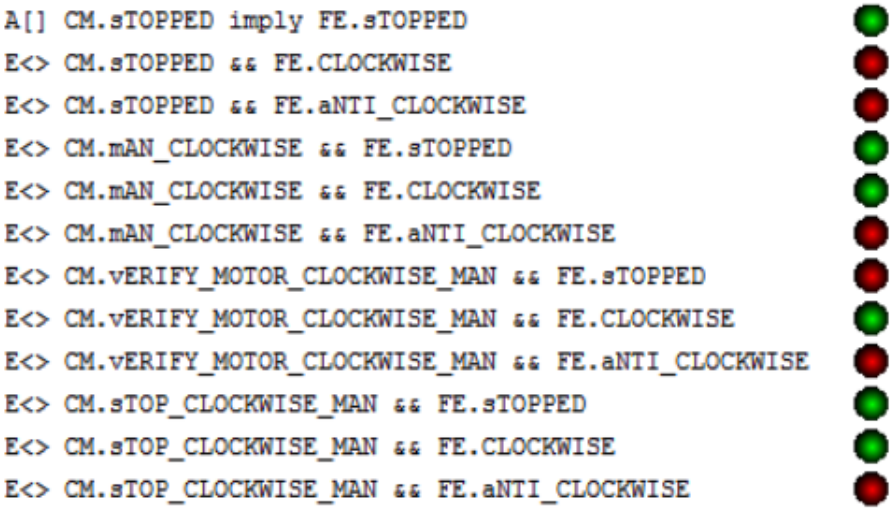

Figura 45 – Propriedades das verificações do motor - Sistema de carregamento 2

Analisando a figura 45, há várias luzes 'vermelhas' e várias luzes 'verdes'. Isso acontece, por que, por exemplo no estado *CM.sTOPPED* o motor deve estar, obrigatoriamente, parado. O motor não pode estar se movendo quando o sistema atingir esse estado, caso contrário seria falha do sistema. Devido a esse fato é possível ver na imagem que no estado *'CM.sTOPPED'* que o estado do *'template - Motor'* o estado *'FE.sTOPPED'* está verde e os estados *'FE.cLOCKWISE'* e *'FE.aNTI_ CLOCKWISE'* estão vermelhos. No Apêndice B é apresentado todas as propriedades verificadas de todos os subsistemas desenvolvidos nesse trabalho.

4.6 Sistema de carregamento 3

O sistema de carregamento 3 possui funcionamento idêntico ao do sistema de carregamento 2. A diferença entre os dois subsistemas é o número do sistema. No sistema de carregamento 3 o número do sistema é 6.

4.7 Laser

O laser foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada. O laser é utilizado, nesse trabalho, como o modelo padrão para o sistema de aquecimento, de refrigeração, de filtragem, de vácuo e de proteção a gás, por terem funcionamento básico similar.

4.7.1 Conceitos gerais

Conforme já descrito na seção 3.3.4, o laser é responsável por derreter o pó e, desta forma, formar o componente camada a camada. Ele se mantém estático, simplesmente ligando para derreter a camada de pó e depois desligando. Ele aguarda até que a nova camada de pó seja depositada para ligar novamente.

O laser possui um funcionamento muito simples, o sistema possui três botões como entrada, que para a GUI são os seguintes botões:

- ***'Ligar laser'*** responsável por ligar o laser, se estiver nas condições corretas;
- ***'Desligar laser'*** responsável por desligar o laser; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o laser são as seguintes:

- ***'Ligar'*** sinal digital que manda o laser ligar;
- ***'Desligar'*** sinal digital que manda o laser desligar; e,
- ***'Alarme'*** variável que indica se houve falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 46 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

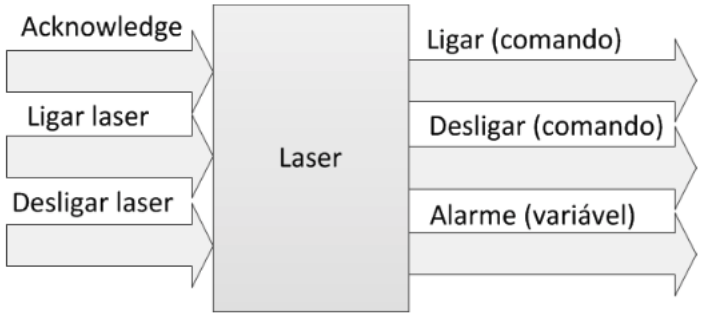

Figura 46 – Entradas e saídas do subsistema laser

4.7.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do laser, as entradas e saídas da figura 46 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o funcionamento do laser. Os três *'templates'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'LASER' recebe os comandos do *'template - COMMAND'* e liga ou desliga o laser; e
- 'COMMAND' representa o controlador do sistema.

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema e o UPPAAL simula o modelo, para verificar se o modelo do laser possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 82, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema, que são:

- `BS_ON!` - Comando para ligar o laser;
- `BS_OFF!` - Comando para desligar o laser; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo do componente do laser, que pode ser visto na figura 83, no Apêndice B, recebe os comandos do *'template - COMMAND'* e liga o laser, caso o comando for para ligar, ou desliga o laser, caso o comando for para desligar.

Nesse *'template'* é atualizada a variável `VB_LASER_ON` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_TURN_ON!` o componente do laser irá para o estado *'oN'*. A variável que indica se o componente do laser está ligado, `VB_LASER_ON`, é atualizada para 1, indicando para o sistema que o componente do laser está ligado e operando. O componente do laser desliga quando receber o comando `VB_TURN_OFF!` do *'template - COMMAND'*, atualizando a variável `VB_LASER_ON` para 0, indicando que o componente do laser desligou.

4.7.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 84, que pode ser vista no Apêndice B. Seu estado inicial é nomeado *'oFF'*. Nesse estado, caso o sistema receba o comando `BS_OFF?` nada acontecerá. O sistema só irá começar o processo de ligar o componente do laser ao receber o comando `BS_ON?` da GUI.

Ao receber o comando `BS_ON?`, o sistema mudará para o estado *'mESSAGE_ON_MAN'* e enviará a sincronização `VB_TURN_ON!` para o *'Template Laser'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_MAN'*, em que aguardará que a variável `VB_LASER_ON` seja atualizada para 1, indicando dessa forma que o componente do laser está ligado e funcionando. Mudando-se assim o estado do sistema para *'mAN_ON'*. O sistema continua com o componente do laser ligado enquanto o sistema não receber o comando `BS_OFF?` da GUI.

Com o recebimento do comando `BS_OFF?`, o sistema muda do estado *'mAN_ON'* para o estado *'sTOP_MAN'*. Nesse estado o sistema envia o comando `VB_TURN_OFF!` para o *'Template Laser'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_LASER_ON` de 1 para 0, indicando que o componente do laser foi desligado. Após enviar a sincronização ao *'Template Laser'*, o sistema muda para o estado *'fEED_OFF_MAN'* e aguarda que a variável `VB_LASER_ON` fique igual a 0, indicando que o componente do laser está desligado e podendo mudar seu estado para o estado inicial *'oFF'* e, assim, recomeçar o processo.

4.7.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis e qual o estado do componente do laser para cada um. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do laser verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação a ser feita é conferir se o sistema será bloqueado em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar.

Uma relevante verificação de propriedades é da alcançabilidade de todos os estados do sistema, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 47.Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

Checker do UPPAAL:

E<> 'estado do Controlador'

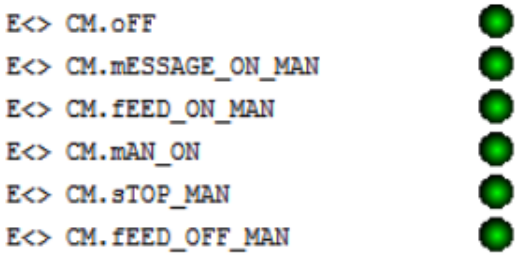

Figura 47 – Propriedades dos estados alcançáveis - Laser

Outra propriedade importante é verificar se em todos os estados a variável `VB_LASER_ON` está configurada da maneira correta, estando desligada no estado *'oFF'* e ligada no estado *'oN'*. Essa propriedade é dada abaixo e pode também ser vista na figura 48. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

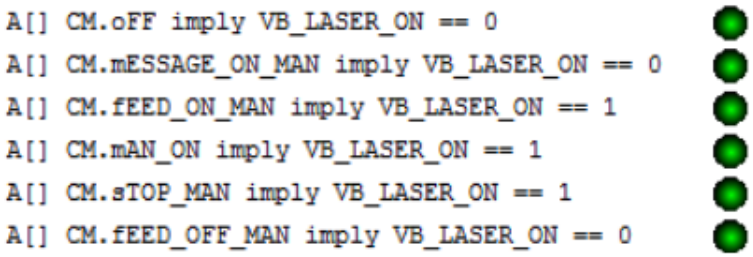

Figura 48 – Propriedades de verificação da variável `VB_LASER_ON` - Laser

A[] 'estado do Controlador' imply `VB_LASER_ON` $==$ *1*, ou

A[] 'estado do Controlador' imply `VB_LASER_ON` $==$ *0*

Com essas propriedades observa-se que o estado *'oFF'* está invariavelmente com a variável `VB_LASER_ON` igual a 0, ou seja, com o componente do laser sempre desligado. Enquanto o estado *'oN'* está invariavelmente com a variável `VB_LASER_ON` igual a 1, ou seja, com o componente do laser sempre ligado.

4.8 Sistema de aquecimento

O sistema de aquecimento foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

4.8.1 Conceitos gerais

Conforme já descrito na seção 3.3.5, o sistema de aquecimento é responsável por aquecer a câmara e a base de construção da máquina SLM antes e durante o processo de produção do componente.

O sistema de aquecimento possui um sensor de temperatura analógico, para verificar a temperatura atual da câmara, e um sensor de temperatura máxima digital. Através desses sensores, o controlador tem conhecimento da temperatura da câmara em todos os momentos. O sensor digital informa ao controlador se a câmara atingiu a temperatura máxima enquanto estiver em operação, e o software garante que este limite não será ultrapassado, desligando o sistema de aquecimento quando tal situação ocorrer. Esses sensores são os *inputs* do subsistema, além dos botões para ligar, desligar e *acknowledge*.

O sistema de aquecimento possui um funcionamento muito simples, possuindo três botões como entrada, que para a GUI são os seguintes botões:

- ***'Ligar aquecimento'*** responsável por ligar o sistema de aquecimento, se estiver nas condições corretas;
- ***'Desligar aquecimento'*** responsável por desligar o sistema de aquecimento; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de aquecimento são as seguintes:

- ***'Ligar'*** sinal digital que manda o sistema de aquecimento ligar;
- ***'Desligar'*** sinal digital que manda o sistema de aquecimento desligar; e,
- ***'Alarme'*** variável que indica se houve falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 49 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

4.8.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do sistema de aquecimento, as entradas e saídas da figura 49 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em

autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o funcionamento do sistema de aquecimento. Os três *'templates'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'AQUECIMENTO' recebe os comandos do *'template - COMMAND'* e liga ou desliga o sistema de aquecimento; e
- 'COMMAND' representa o controlador do sistema.

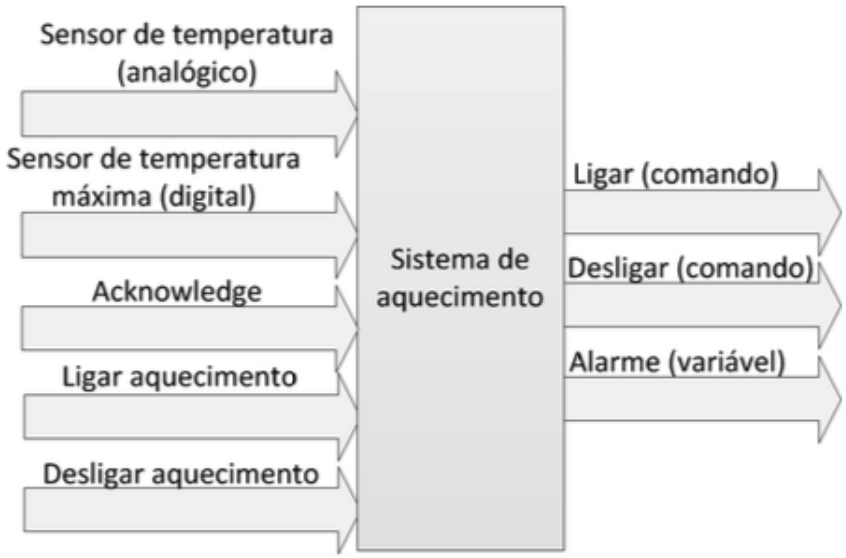

Figura 49 – Entradas e saídas do sistema de aquecimento

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema e o UPPAAL simula o modelo, para verificar se o modelo do sistema de aquecimento possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 85, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema, que são:

- `BS_ON!` - Comando para ligar o sistema de aquecimento;
- `BS_OFF!` - Comando para desligar o sistema de aquecimento; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo do motor de aquecimento, que pode ser visto na figura 86, no Apêndice B, recebe os comandos do *'template - COMMAND'* e liga o sistema de aquecimento, caso o comando for para ligar e a temperatura da câmara for menor que 200 °C, ou desliga o

sistema de aquecimento, caso o comando for para desligar ou se a temperatura da câmara for maior que 200 °C.

Nesse *'template'* é atualizada a variável `VB_HEATER_ON` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_TURN_ON!` o motor de aquecimento irá para o estado *'oN'*. A variável que indica se o motor de aquecimento está ligado, `VB_HEATER_ON`, é atualizada para 1, indicando para o sistema que o motor de aquecimento está ligado e operando. Além disso, a variável que representa a temperatura da câmara, `VB_TEMPERATURE`, é aumentada a cada ciclo do processo. O motor de aquecimento desliga quando receber o comando `VB_TURN_OFF!` do *'template - COMMAND'*, atualizando a variável `VB_HEATER_ON!` para 0, indicando que o motor de aquecimento desligou.

4.8.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 87, que pode ser vista no Apêndice B. Seu estado inicial é nomeado *'oFF'*. Nesse estado, caso o sistema receba o comando `BS_OFF?` nada acontecerá. O sistema só irá começar o processo de ligar o motor de aquecimento ao receber o comando `BS_ON?` da GUI.

Ao receber o comando `BS_ON?` e a temperatura da câmara for menor que 200 °C, o sistema mudará para o estado *'mESSAGE_ON_MAN'* e enviará a sincronização `VB_TURN_ON!` para o *'Template Aquecimento'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_MAN'*, em que aguardará que a variável `VB_HEATER_ON` seja atualizada para 1, indicando dessa forma que o motor de aquecimento está ligado e funcionando. Mudando-se assim o estado do sistema para *'mAN_ON'*. O sistema continua com o motor de aquecimento ligado enquanto não receber o comando `BS_OFF?` da GUI ou a temperatura da câmara for menor que 200 °C.

Com o recebimento do comando `BS_OFF?`, o sistema muda do estado *'mAN_ON'* para o estado *'sTOP_MAN'*. Nesse estado o sistema envia o comando `VB_TURN_OFF!` para o *'Template Aquecimento'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_HEATER_ON` de 1 para 0, indicando que o motor de aquecimento foi desligado. Após enviar a sincronização ao *'Template Aquecimento'*, o sistema muda para o estado *'fEED_OFF_MAN'* e aguarda que a variável `VB_HEATER_ON` fique igual a 0, indicando que o motor de aquecimento está desligado e podendo mudar seu estado para o estado inicial *'oFF'* e, assim, recomeçar o processo.

4.8.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis, qual o estado do motor de aquecimento para cada um e se os limites de temperatura são respeitados. A verificação de propriedades foi realizada utilizando a

ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do sistema de aquecimento verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação a ser feita é conferir se o sistema será bloqueado em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar.

Uma propriedade bastante importante é verificar se todos os estados do sistema são alcançáveis, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 50. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

Checker do UPPAAL:

E<> 'estado do Controlador'

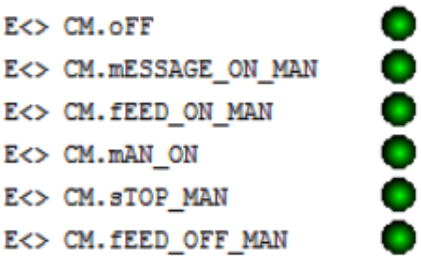

Figura 50 – Propriedades dos estados alcançáveis - Sistema de aquecimento

Outra propriedade importante é verificar se em todos os estados a variável `VB_HEATER_ON` está configurada da maneira correta, estando desligada no estado *'oFF'* e ligada no estado *'oN'*. Essa propriedade é dada abaixo e pode também ser vista na figura 51. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

A[] 'estado do Controlador' imply `VB_HEATER_ON` $== 1$, ou

A[] 'estado do Controlador' imply `VB_HEATER_ON` $== 0$

Com essas propriedades observa-se que o estado *'oFF'* está invariavelmente com a variável `VB_HEATER_ON` igual a 0, ou seja, com o motor de aquecimento sempre desligado. Enquanto o estado *'oN'* está invariavelmente com a variável `VB_HEATER_ON` igual a 1, ou seja, com o motor de aquecimento sempre ligado.

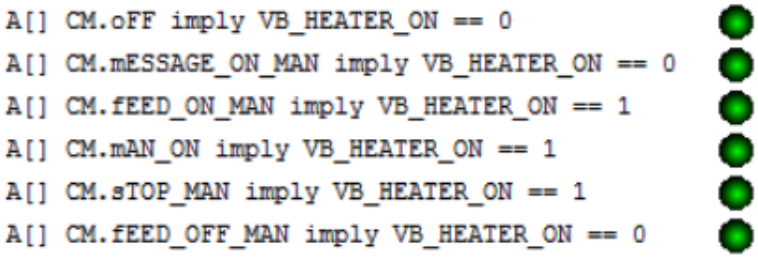

Figura 51 – Propriedades de verificação da variável `VB_HEATER_ON` - Sistema de aquecimento

O último tipo de propriedade é verificar se em todos os estados a variável `VB_TEMPERATURE` está sempre abaixo do limite máximo, ou seja, ser sempre menor que 200 °C, independente do estado que estiver. Essa propriedade é dada abaixo e pode também ser vista na figura 52. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

A[] 'estado do Controlador' imply `VB_TEMPERATURE` <= 200

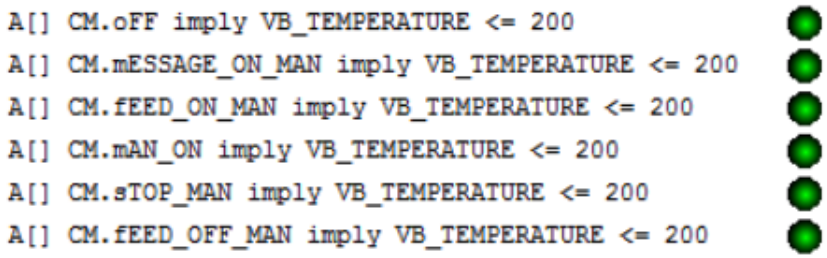

Figura 52 – Propriedades de verificação da variável `VB_TEMPERATURE` - Sistema de aquecimento

4.9 Sistema de refrigeração

O sistema de refrigeração foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

4.9.1 Conceitos gerais

Conforme já descrito na seção 3.3.6, o sistema de refrigeração é responsável por resfriar os principais componentes da máquina SLM. Por exemplo, o sistema de laser, evitando fadiga na máquina e falhas no componente a ser manufaturado.

O sistema de refrigeração possui um sensor de temperatura analógico, para verificar a temperatura atual da câmara, e um sensor de temperatura mínima digital. Através desses sensores, o controlador tem conhecimento da temperatura da câmara em todos os momentos. O sensor digital informa ao controlador se a câmara atingiu a temperatura mínima enquanto estiver em operação, e o software garante que esse limite não será ultrapassado, desligando o sistema de refrigeração quando tal situação ocorrer. Esses sensores são os *inputs* do subsistema, além dos botões para ligar, desligar e *acknowledge*.

O sistema de refrigeração possui um funcionamento muito simples, possuindo três botões como entrada, que para a GUI são os seguintes botões:

- ***'Ligar refrigeração'*** responsável por ligar o sistema de refrigeração, se estiver nas condições corretas;
- ***'Desligar refrigeração'*** responsável por desligar o sistema de refrigeração; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de refrigeração são as seguintes:

- ***'Ligar'*** sinal digital que manda o sistema de refrigeração ligar;
- ***'Desligar'*** sinal digital que manda o sistema de refrigeração desligar; e,
- ***'Alarme'*** variável que indica se houve falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 53 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

4.9.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do sistema de refrigeração, as entradas e saídas da figura 53 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em

autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o funcionamento do sistema de refrigeração. Os três *'templates'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'REFRIGERAÇÃO' recebe os comandos do *'template - COMMAND'* e liga ou desliga o sistema de refrigeração; e
- 'COMMAND' representa o controlador do sistema.

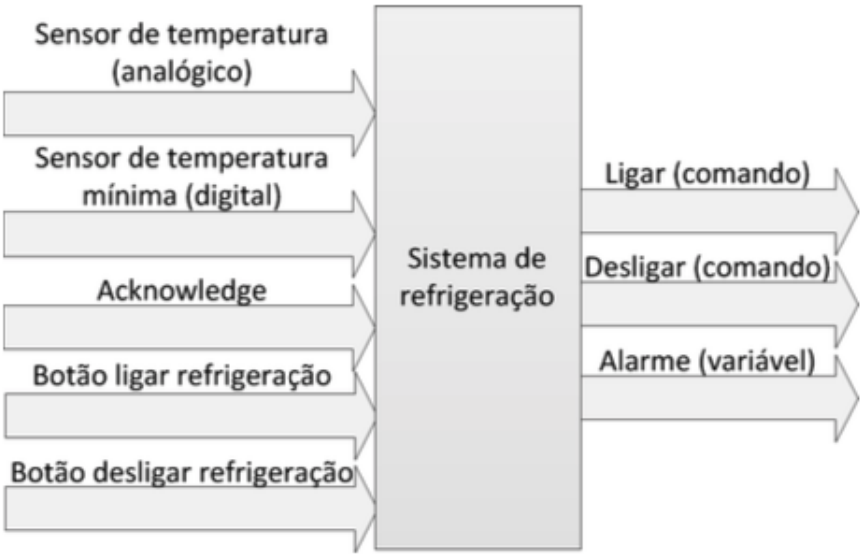

Figura 53 – Entradas e saídas do subsistema sistema de refrigeração

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema e o UPPAAL simula o modelo, para verificar se o modelo do sistema de refrigeração possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 88, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema, que são:

- `BS_ON!` - Comando para ligar o sistema de refrigeração;
- `BS_OFF!` - Comando para desligar o sistema de refrigeração; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo do motor de refrigeração, que pode ser visto na figura 89, no Apêndice B, recebe os comandos do *'template - COMMAND'* e liga o sistema de refrigeração, caso o comando for para ligar e a temperatura da câmara for maior que 20 °C, ou desliga o

sistema de refrigeração, caso o comando for para desligar ou se a temperatura da câmara for menor que 20 °C.

Nesse *'template'* é atualizada a variável `VB_REFRIGERATION_ON` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_TURN_ON!` o motor de refrigeração irá para o estado *'oN'*. A variável que indica se o motor de refrigeração está ligado, `VB_REFRIGERATION_ON`, é atualizada para 1, indicando para o sistema que o motor de refrigeração está ligado e operando. Além disso, a variável que representa a temperatura da câmara, `VB_TEMPERATURE`, é diminuída a cada ciclo do processo. O motor de refrigeração desliga quando receber o comando `VB_TURN_OFF!` do *'template - COMMAND'*, atualizando a variável `VB_REFRIGERATION_ON` para 0, indicando que o motor de refrigeração desligou.

4.9.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 90, que pode ser vista no Apêndice B. Seu estado inicial é nomeado *'oFF'*. Nesse estado, caso o sistema receba o comando `BS_OFF?` nada acontecerá. O sistema só irá começar o processo de ligar o motor de refrigeração ao receber o comando `BS_ON?` da GUI.

Ao receber o comando `BS_ON?` e a temperatura da câmara for maior que 20 °C, o sistema mudará para o estado *'mESSAGE_ON_MAN'* e enviará a sincronização `VB_TURN_ON!` para o *'Template Refrigração'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_MAN'*, em que aguardará que a variável `VB_REFRIGERATION_ON` seja atualizada para 1, indicando dessa forma que o motor de refrigeração está ligado e funcionando. Mudando-se assim o estado do sistema para *'mAN_ON'*. O sistema continua com o motor de refrigeração ligado enquanto o sistema não receber o comando `BS_OFF?` da GUI ou a temperatura da câmara for menor que 20 °C.

Com o recebimento do comando `BS_OFF?`, o sistema muda do estado *'mAN_ON'* para o estado *'sTOP_MAN'*. Nesse estado o sistema envia o comando `VB_TURN_OFF!` para o *'Template Refrigração'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_REFRIGERATION_ON` de 1 para 0, indicando que o motor de refrigeração foi desligado. Após enviar a sincronização ao *'Template Refrigração'*, o sistema muda para o estado *'fEED_OFF_MAN'* e aguarda que a variável `VB_REFRIGERATION_ON` fique igual a 0, indicando que o motor de refrigeração está desligado e podendo mudar seu estado para o estado inicial *'oFF'* e, assim, recomeçar o processo.

4.9.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis, qual o estado do motor de refrigeração para cada um e se os limites de

temperatura são respeitados. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação a ser feita é conferir se o sistema será bloqueado em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar.

As propriedades para verificar se os estados do sistema são alcançáveis e qual o estado do motor de refrigeração em cada estado do *template* - *'Refrigeration'* são análogos ao do sistema de aquecimento, com os mesmos resultados, somente mudando a variável de `VB_HEATER_ON` para `VB_REFRIGERATION_ON`.

A propriedade diferente é verificar se em todos os estados a variável `VB_TEMPERATURE` está sempre acima do limite mínimo, ou seja, ser sempre maior que 20 °C, independente do estado que estiver. Essa propriedade é dada abaixo e pode também ser vista na figura 54. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

A[] 'estado do Controlador' imply `VB_TEMPERATURE` $>= 20$

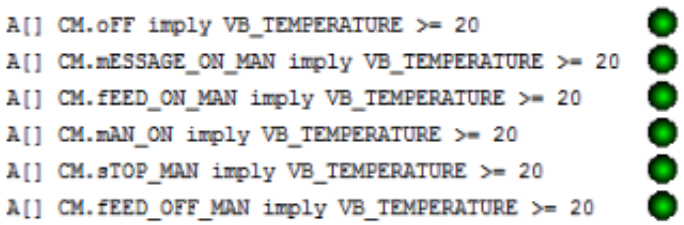

Figura 54 – Propriedades de verificação da variável `VB_TEMPERATURE` - Sistema de refrigeração

4.10 Sistema de filtragem

O sistema de filtragem foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

4.10.1 Conceitos gerais

Conforme já descrito na seção 3.3.7, o sistema de filtragem é responsável por filtrar o pó não utilizado que sai da máquina SLM para ser devidamente armazenado e utilizado em projetos futuros, caso o pó se encontre em boas condições.

O sistema de filtragem possui um sensor de pó analógico, para verificar a quantidade atual de pó para ser filtrado, e um sensor de quantidade mínima de pó digital. Através desses sensores, o controlador tem conhecimento da quantidade de pó para ser filtrada em todos os momentos. O sensor digital informa ao controlador se a câmara atingiu a quantidade de pó mínima enquanto estiver em operação, e o software garante que esse limite não será ultrapassado, ligando o sistema de filtragem quando tal situação ocorrer. Esses sensores são os *inputs* do subsistema, além dos botões para ligar, desligar e *acknowledge*.

O sistema de filtragem possui um funcionamento muito simples, possuindo três botões como entrada, que para a GUI são os seguintes botões:

- ***'Ligar filtragem'*** responsável por ligar o sistema de filtragem, se estiver nas condições corretas;
- ***'Desligar filtragem'*** responsável por desligar o sistema de filtragem; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de filtragem são as seguintes:

- ***'Ligar'*** sinal digital que manda o sistema de filtragem ligar;
- ***'Desligar'*** sinal digital que manda o sistema de filtragem desligar; e,
- ***'Alarme'*** variável que indica se houve falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 55 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

4.10.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do sistema de filtragem, as entradas e saídas da figura 55 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em

autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o funcionamento do sistema de filtragem. Os três *'templates'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'FILTRAGEM' recebe os comandos do *'template - COMMAND'* e liga ou desliga o sistema de filtragem; e
- 'COMMAND' representa o controlador do sistema.

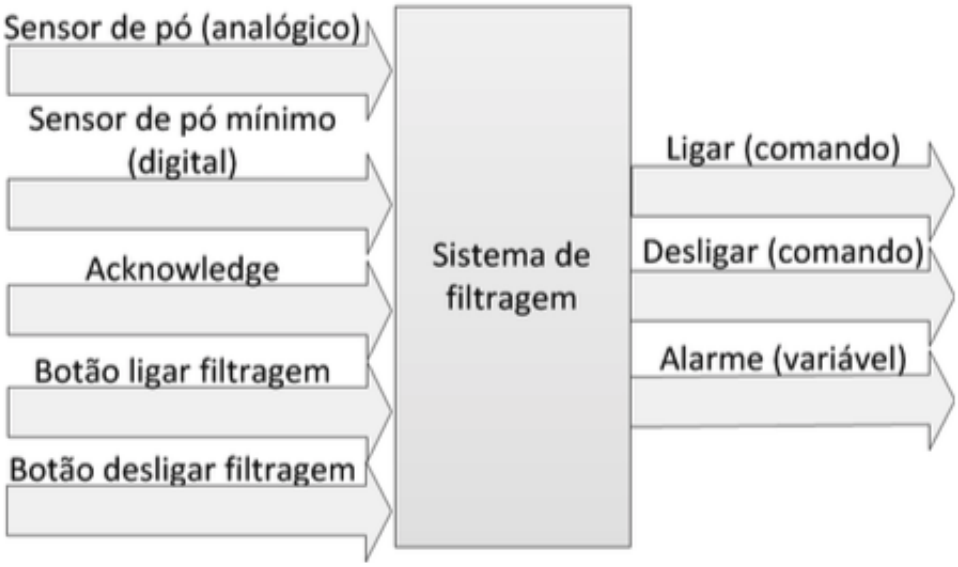

Figura 55 – Entradas e saídas do subsistema sistema de filtragem

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema e o UPPAAL simula o modelo, para verificar se o modelo do sistema de filtragem possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 91, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema, que são:

- `BS_ON!` - Comando para ligar o sistema de filtragem;
- `BS_OFF!` - Comando para desligar o sistema de filtragem; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo do motor de filtragem, que pode ser visto na figura 92, no Apêndice B, recebe os comandos do *'template - COMMAND'* e liga o sistema de filtragem, caso o comando for para ligar e a quantidade de pó para ser filtrado for maior que 30% e o

comando de ligar for enviado, ou desliga o sistema de filtragem, caso o comando for para desligar ou se a quantidade de pó para ser filtrada seja menos que 30%.

Nesse *'template'* é atualizada a variável `VB_FILTER_ON` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_TURN_ON!` o motor de filtragem irá para o estado *'oN'*. A variável que indica se o motor de filtragem está ligado, `VB_FILTER_ON`, é atualizada para 1, indicando para o sistema que o motor de filtragem está ligado e operando. Além disso, a variável que representa a quantidade de pó para ser filtrada, `VB_CAPACITY`, é diminuída a cada ciclo do processo. O motor de filtragem desliga quando receber o comando `VB_TURN_OFF!` do *'template - COMMAND'*, atualizando a variável `VB_FILTER_ON` para 0, indicando que o motor de filtragem desligou.

4.10.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 93, que pode ser vista no Apêndice B. Seu estado inicial é nomeado *'oFF'*. Nesse estado, caso o sistema receba o comando `BS_OFF?` nada acontecerá. O sistema só irá começar o processo de ligar o motor de filtragem ao receber o comando `BS_ON?` da GUI ou caso a quantidade de pó seja maior ou igual a 60%.

Ao receber o comando `BS_ON?`, ou caso a quantidade de pó seja maior ou igual a 60%, e a quantidade de pó for maior que 30%, o sistema mudará para o estado *'mESSAGE_ON_MAN'* e enviará a sincronização `VB_TURN_ON!` para o *'Template Filtragem'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_MAN'*, em que aguardará que a variável `VB_FILTER_ON` seja atualizada para 1, indicando dessa forma que o motor de filtragem está ligado e funcionando. Mudando-se assim o estado do sistema para *'mAN_ON'*. O sistema continua com o motor de filtragem ligado enquanto o sistema não receber o comando `BS_OFF?` da GUI ou a quantidade de pó for maior que 30%.

Com o recebimento do comando `BS_OFF?`, o sistema muda do estado *'mAN_ON'* para o estado *'sTOP_MAN'*. Nesse estado o sistema envia o comando `VB_TURN_OFF!` para o *'Template Filtragem'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_FILTER_ON` de 1 para 0, indicando que o motor de filtragem foi desligado. Após enviar a sincronização ao *'Template Filtragem'*, o sistema muda para o estado *'fEED_OFF_MAN'* e aguarda que a variável `VB_FILTER_ON` fique igual a 0, indicando que o motor de filtragem está desligado e podendo mudar seu estado para o estado inicial *'oFF'* e, assim, recomeçar o processo.

4.10.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis e qual o estado do motor de filtragem para cada um. A verificação de propriedades

foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do sistema de filtragem verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação a ser feita é conferir se o sistema será bloqueado em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira. Porém, essa propriedade não representa não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar.

As propriedades para verificar se os estados do sistema são alcançáveis e qual o estado do motor de filtragem em cada estado do *template* - *'Controle'* são análogos ao do sistema de aquecimento, com os mesmos resultados, somente mudando o nome da variável `VB_HEATER_ON` para `VB_FILTER_ON`.

4.11 Sistema de vácuo

O sistema de vácuo foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

4.11.1 Conceitos gerais

Conforme já descrito na seção 3.3.8, o sistema de vácuo é responsável por retirar o oxigênio da máquina, realizando vácuo dentro da câmara do processo.

O sistema de vácuo possui um sensor de oxigênio analógico, para verificar a concentração atual de oxigênio dentro da câmara, e um sensor de concentração mínima de oxigênio digital. Através desses sensores, o controlador tem conhecimento da concentração atual de oxigênio dentro da câmara em todos os momentos. O sensor digital informa ao controlador se a câmara atingiu a concentração de oxigênio mínima enquanto estiver em operação, e o software garante que esse limite não será ultrapassado, ligando o sistema de vácuo quando tal situação ocorrer. Esses sensores são os *inputs* do subsistema, além dos botões para ligar, desligar e *acknowledge*.

O sistema de vácuo possui um funcionamento muito simples, possuindo três botões como entrada, que para a GUI são os seguintes botões:

- ***'Ligar vácuo'*** responsável por ligar o sistema de vácuo, se estiver nas condições corretas;
- ***'Desligar vácuo'*** responsável por desligar o sistema de vácuo; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de vácuo são as seguintes:

- ***'Ligar motor'*** sinal digital que manda o sistema de vácuo ligar o motor;
- ***'Desligar motor'*** sinal digital que manda o sistema de vácuo desligar o motor;
- ***'Abrir válvula'*** sinal digital que manda o sistema de vácuo abrir a válvula;
- ***'Fechar válvula'*** sinal digital que manda o sistema de vácuo a fechar a válvula; e,
- ***'Alarme'*** variável que indica se houve falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 56 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

4.11.2 Concepção do modelo e simulação preliminar

Para elaborar o modelo do sistema de vácuo, as entradas e saídas da figura 56 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o funcionamento do sistema de vácuo. Os quatro *'templates'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'VÁLVULA' recebe os comandos do *'template - COMMAND'* e abre ou fecha a válvula do sistema de vácuo; e
- 'MOTOR' recebe os comandos do *'template - COMMAND'* e liga ou desliga o motor sistema de vácuo; e
- 'COMMAND' representa o controlador do sistema.

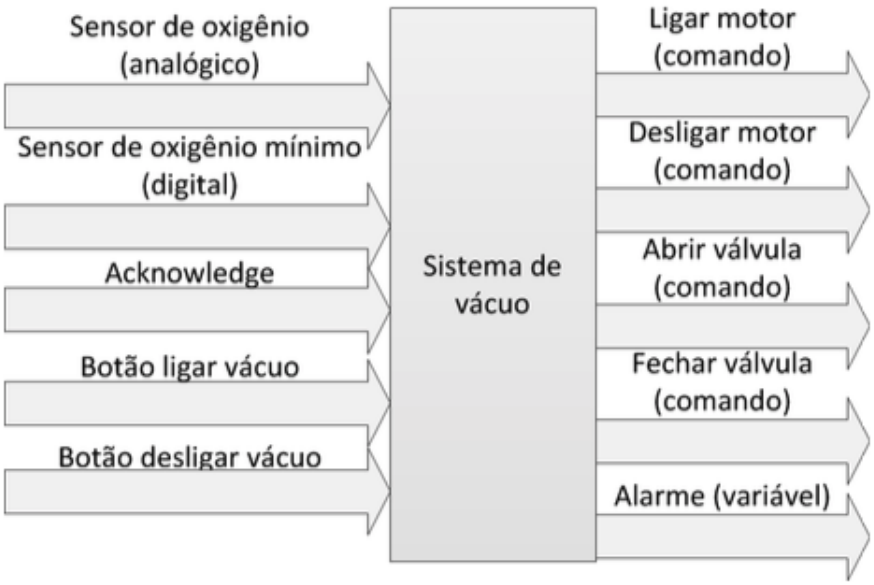

Figura 56 – Entradas e saídas do subsistema sistema de vácuo

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema e o UPPAAL simula o modelo, para verificar se o modelo do sistema de vácuo possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 94, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema, que são:

- `BS_ON!` - Comando para ligar o sistema de vácuo;

- `BS_OFF!` - Comando para desligar o sistema de vácuo; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo da válvula, que pode ser visto na figura 96, no Apêndice B, recebe os comandos do *'template - COMMAND'* e abre a válvula do sistema de vácuo, caso o comando for para ligar e concentração de oxigênio estiverem corretas, ou fecha a válvula do sistema de vácuo, caso o comando for para desligar ou se a concentração de oxigênio atingir seu limite.

Nesse *'template'* é atualizada a variável `VB_VALVE_OPEN` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_OPEN_VALVE!` a válvula irá para o estado *'oN'*. A variável que indica se a válvula está aberta, `VB_VALVE_OPEN`, é atualizada para 1, indicando para o sistema que a válvula está aberta e operando. A válvula fecha quando receber o comando `VB_CLOSE_VALVE!` do *'template - COMMAND'*, atualizando a variável `VB_VALVE_OPEN` para 0, indicando que a válvula fechou.

O modelo do motor de vácuo, que pode ser visto na figura 95, no Apêndice B, recebe os comandos do *'template - COMMAND'* e liga o sistema de vácuo, caso o comando for para ligar e a válvula esteja aberta, ou desliga o sistema de vácuo, caso receba a mensagem para desligar o motor e a válvula esteja fechada.

Nesse *'template'* é atualizada a variável `VB_MOTOR_ON` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_TURN_ON_MOTOR!` o motor de vácuo irá para o estado *'oN'*. A variável que indica se o motor de vácuo está ligado, `VB_MOTOR_ON`, é atualizada para 1, indicando para o sistema que o motor de vácuo está ligado e operando. Além disso, a variável que representa a pressão dentro da câmara, `VB_PRESSURE`, é diminuída a cada ciclo do processo. O motor de vácuo desliga quando receber o comando `VB_TURN_OFF_MOTOR!` do *'template - COMMAND'*, atualizando a variável `VB_MOTOR_ON` para 0, indicando que o motor de vácuo desligou.

4.11.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 97, que pode ser vista no Apêndice B. Seu estado inicial é nomeado *'oFF'*. Nesse estado, caso o sistema receba o comando `BS_OFF?` nada acontecerá. O sistema só irá começar o processo de abrir a válvula e ligar o motor de vácuo ao receber o comando `BS_ON?` da GUI.

Ao receber o comando `BS_ON?` e a pressão for maior que 5 mbar o sistema mudará para o estado *'mESSAGE_ON_VALVE_MAN'* e enviará a sincronização `VB_OPEN_VALVE!` para o *'Template - Válvula'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_VALVE_MAN'*, em que aguardará que a variável `VB_VALVE_OPEN` seja

atualizada para 1, indicando dessa forma que a válvula do sistema de vácuo está aberta funcionando. Mudando-se assim o estado para *'mESSAGE_ON_MOTOR_MAN'*.

Nesse estado o *'template - Command'* envia `VB_TURN_ON_MOTOR!` para o *'template - Motor'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_MOTOR_MAN'*, em que aguardará que a variável `VB_MOTOR_ON` seja atualizada para 1, indicando dessa forma que o motor do sistema de vácuo está ligado e funcionando. Mudando-se assim o estado do sistema para *'mAN_ON'*.

O sistema continua com a válvula aberta e com o motor de vácuo funcionando enquanto não receber o comando `BS_OFF?` da GUI ou a pressão dentro da câmara for maior que 5 mbar.

Com o recebimento do comando `BS_OFF?`, o sistema muda do estado *'mAN_ON'* para o estado *'sTOP_VALVE_MAN'*. Nesse estado o sistema envia o comando `VB_CLOSE_VALVE!` para o *'template - Válvula'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_VALVE_OPEN` de 1 para 0, indicando que a válvula do sistema de vácuo foi fechada. Após enviar a sincronização ao *'template - válvula'*, o sistema muda para o estado *'fEED_OFF_VALVE_MAN'* e aguarda que a variável `VB_VALVE_OPEN` fique igual 0, indicando que a válvula está fechada e pode mudar para o estado *'sTOP_MOTOR_MAN'*.

Nesse estado o sistema envia o comando `VB_TURN_OFF_MOTOR!` para o *'template - Motor'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_MOTOR_ON` de 1 para 0, indicando que o motor de vácuo foi desligado. Após enviar a sincronização ao *'template - Motor'*, o sistema muda para o estado *'fEED_OFF_MOTOR_MAN'* e aguarda que a variável `VB_MOTOR_ON` fique igual a 0, indicando ao sistema que o motor de vácuo está desligado e podendo mudar seu estado para o estado inicial *'oFF'* e, assim, recomeçar o processo.

4.11.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis, qual o estado da válvula e do motor de vácuo para cada um e se a pressão mínima é respeitada em todos os estados. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do sistema de vácuo verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação a ser feita é conferir se o sistema será bloqueado em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira. Porém, essa propriedade não representa não travamento do sistema em um ciclo de dois ou mais

estados que não é possível escapar.

Uma relevante verificação de propriedades é da alcançabilidade de todos os estados do sistema, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 57. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir *n* estados, logo deverão existir n propriedades semelhantes.

Checker do UPPAAL:

E<> 'estado do Controlador'

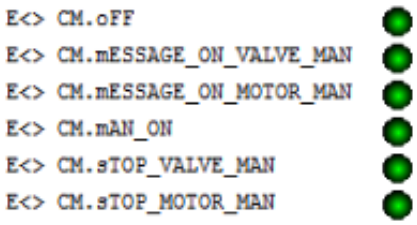

Figura 57 – Propriedades dos estados alcançáveis - Sistema de vácuo

Outra propriedade importante é verificar se em todos os estados as variáveis `VB_VALVE_OPEN` e `VB_MOTOR_ON` estão configuradas da maneira correta, estando fechada e desligado no estado *'oFF'* e aberto e ligado no estado *'oN'*, respectivamente. Essa propriedade é dada abaixo e pode também ser vista na figura 58. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir *n* estados, logo deverão existir 2*n propriedades semelhantes.

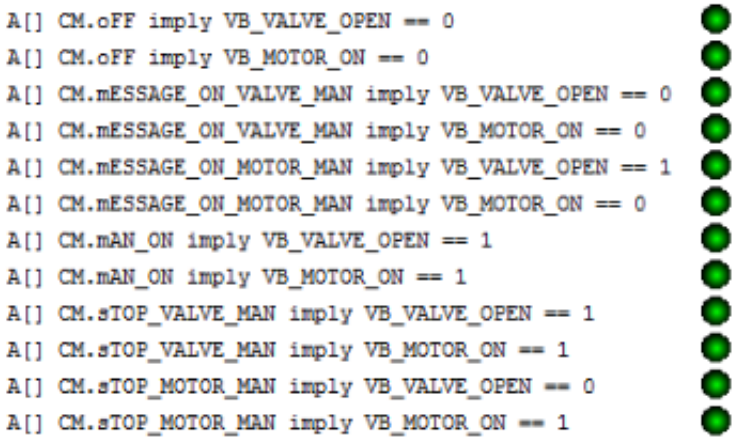

Figura 58 – Propriedades de verificação das variáveis `VB_VALVE_OPEN` e `VB_MOTOR_ON` - Sistema de vácuo

A[] 'estado do Controlador' imply `VB_VALVE_OPEN` *== 1*, ou

A[] 'estado do Controlador' imply `VB_VALVE_OPEN` *== 0*; e

A[] 'estado do Controlador' imply VB_MOTOR_ON == 1, ou

A[] 'estado do Controlador' imply VB_MOTOR_ON == 0

Com essas propriedades observa-se que o estado *'oFF'* está invariavelmente com as variáveis `VB_VALVE_OPEN` e `VB_MOTOR_ON` iguais a 0, ou seja, com a válvula sempre fechada e com o motor de vácuo sempre desligado. Enquanto o estado *'oN'* está invariavelmente com as variáveis `VB_VALVE_OPEN` e `VB_MOTOR_ON` iguais a 1, ou seja, com a válvula sempre aberta e com o motor de vácuo sempre ligado.

O último tipo de propriedade é verificar se em todos os estados a variável `VB_PRESSURE` está sempre acima do limite mínimo, ou seja, ser sempre maior que 5 mbar, independente do estado que estiver. Essa propriedade é dada abaixo e pode também ser vista na figura 59. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

A[] 'estado do Controlador' imply VB_PRESSURE >= 5

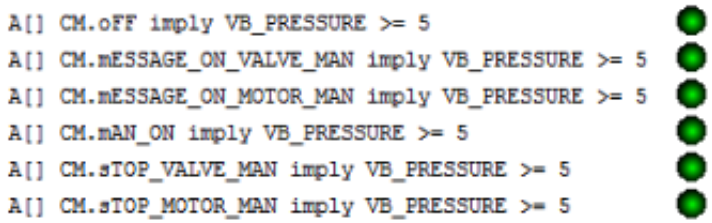

Figura 59 – Propriedades de verificação da variável `VB_PRESSURE` - Sistema de vácuo

4.12 Sistema de proteção a gás

O sistema de proteção a gás foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorrer alguma situação indesejada ocorre-se.

4.12.1 Conceitos gerais

Conforme já descrito na seção 3.3.9, o sistema de proteção a gás é responsável por proteger o ambiente da câmara com um gás inerte não explosivo. Protegendo a câmara, o usuário e a peça.

O sistema de proteção a gás possui um sensor de argônio analógico, para verificar a quantidade de gás de proteção atual dentro da câmara, e um sensor de sensor máximo de argônio digital. Através desses sensores, o controlador tem conhecimento da quantidade de gás de proteração dentro da câmara em todos os momentos. O sensor digital informa ao controlador se a câmara atingiu a quantidade máxima enquanto estiver em operação, e o software garante que esse limite será mantido, para garantir que não há oxigênio o suficiente dentro da câmara e para não haver acidentes, desligando o sistema de proteção a gás quando tal situação ocorrer. Esses sensores são os *inputs* do subsistema, além dos botões para ligar, desligar e *acknowledge*.

O sistema de proteção a gás possui um funcionamento muito simples, possuindo três botões como entrada, que para a GUI são os seguintes botões:

- ***'Ligar gás'*** responsável por ligar o sistema de proteção a gás, se estiver nas condições corretas;
- ***'Desligar gás'*** responsável por desligar o sistema de proteção a gás; e,
- ***'Acknowledge'*** Envia o comando de reconhecimento de falha pelo usuário.

As saídas consideradas para o sistema de proteção a gás são as seguintes:

- ***'Ligar'*** sinal digital que manda o sistema de proteção a gás ligar;
- ***'Desligar'*** sinal digital que manda o sistema de proteção a gás desligar; e,
- ***'Alarme'*** variável que indica se houve falha durante o sistema.

Essas informações podem ser facilmente visualizadas na figura 60 abaixo e são utilizadas para o desenvolvimento do modelo, das propriedades e do código.

4.12.2 Modelo da planta - Sensores e atuadores

Para elaborar o modelo do sistema de proteção a gás, as entradas e saídas da figura 60 foram agrupadas e descritas em *'templates'*. Em cada *'template'*, criou-se a modelagem em autômatos temporizados. A partir destes, foram estabelecidos os eventos que comandam o funcionamento do sistema de proteção a gás. Os três *'templates'* criados foram:

- 'GUI' representa a interface gráfica com o usuário, enviando os comandos;
- 'FILTRAGEM' recebe os comandos do *'template - COMMAND'* e liga ou desliga o sistema de proteção a gás; e
- 'COMMAND' representa o controlador do sistema.

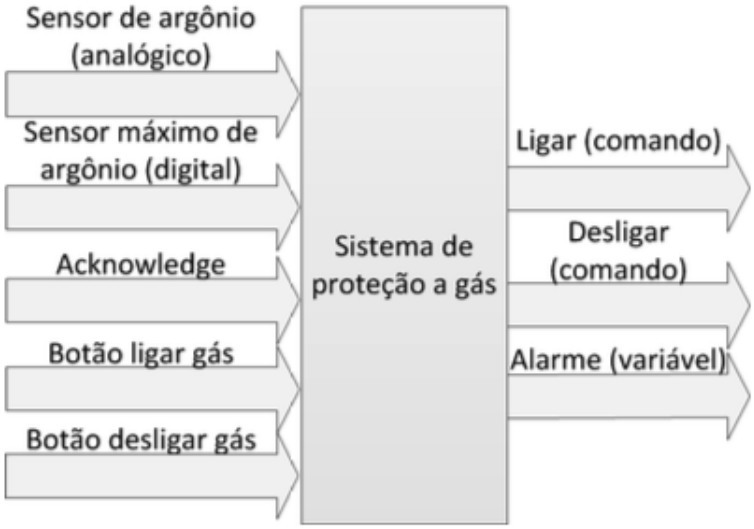

Figura 60 – Entradas e saídas do subsistema sistema de proteção a gás

Os componentes de saída do primeiro *'template'* são os botões digitais. Ele é o responsável por enviar os comandos para o sistema e o UPPAAL simula o modelo, para verificar se o modelo do sistema de proteção a gás possui aderência com a máquina disponível no instituto.

Os comandos que a GUI envia podem ser vistos na figura 98, no Apêndice B. Nele estão todos os comandos que o usuário consegue mandar e enviar no subsistema, que são:

- `BS_ON!` - Comando para ligar o sistema de proteção a gás;
- `BS_OFF!` - Comando para desligar o sistema de proteção a gás; e,
- `ACKNOWLEDGE!` - Comando de reconhecimento da falha.

O modelo do motor de proteção a gás, que pode ser visto na figura 99, no Apêndice B, recebe os comandos do *'template - COMMAND'* e liga o sistema de proteção a gás, caso o comando for para ligar e concentração de gás dentro da câmara for menor que 95%, ou desliga o sistema de proteção a gás, caso o comando for para desligar ou se a concentração de gás dentro da câmara for maior que 95%.

Nesse *'template'* é atualizada a variável `VB_ARGON_ON` cada vez que o estado *'oN'* é alcançado. Caso o *'template - COMMAND'* envie o comando `VB_TURN_ON!` o motor de proteção a gás irá para o estado *'oN'*. A variável que indica se o motor de proteção a gás está ligado, `VB_ARGON_ON`, é atualizada para 1, indicando para o sistema que o motor de proteção a gás está ligado e operando. Além disso, a variável que representa a concentração de gás dentro da câmara, `VB_ARGON`, é aumentada a cada ciclo do processo. O motor de proteção a gás desliga quando receber o comando `VB_TURN_OFF!` do *'template - COMMAND'*, atualizando a variável `VB_ARGON_ON` para 0, indicando que o motor de proteção a gás desligou.

4.12.3 Modelo do controlador

Para o controle, foi desenvolvido o autômato da figura 100, que pode ser vista no Apêndice B. Seu estado inicial é nomeado *'oFF'*. Nesse estado, caso o sistema receba o comando `BS_OFF?` nada acontecerá. O sistema só irá começar o processo de ligar o motor de proteção a gás ao receber o comando `BS_ON?` da GUI.

Ao receber o comando `BS_ON?` e a concentração de gás dentro da câmara for menor que 95%, o sistema mudará para o estado *'mESSAGE_ON_MAN'* e enviará a sincronização `VB_TURN_ON!` para o *'Template Filtragem'*. Após enviar a sincronização, o sistema mudará para o estado *'fEED_ON_MAN'*, em que aguardará que a variável `VB_ARGON_ON` seja atualizada para 1, indicando dessa forma que o motor de proteção a gás está ligado e funcionando. Mudando-se assim o estado do sistema para *'mAN_ON'*. O sistema continua com o motor de proteção a gás ligado enquanto o sistema não receber o comando `BS_OFF?` da GUI ou a concentração de gás dentro da câmara for maior que 95%.

Com o recebimento do comando `BS_OFF?`, o sistema muda do estado *'mAN_ON'* para o estado *'sTOP_MAN'*. Nesse estado o sistema envia o comando `VB_TURN_OFF!` para o *'Template Filtragem'*. Que irá passar do estado *'oN'* para *'oFF'* e atualizar a variável `VB_ARGON_ON` de 1 para 0, indicando que o motor de proteção a gás foi desligado. Após enviar a sincronização ao *'Template Filtragem'*, o sistema muda para o estado *'fEED_OFF_MAN'* e aguarda que a variável `VB_ARGON_ON` fique igual a 0, indicando que o motor de proteção a gás está desligado e podendo mudar seu estado para o estado inicial *'oFF'* e, assim, recomeçar o processo.

4.12.4 Propriedades do sistema

Para a etapa de verificação todos os estados foram testados para verificar se são atingíveis, qual o estado do motor de proteção a gás para cada um e se os limites de concentração de gás são respeitados. A verificação de propriedades foi realizada utilizando a ferramenta *'Verifier'* do UPPAAL.

Para demonstrar como foram verificadas as propriedades, nos próximos parágrafos são descritos alguns exemplos. O documento com todas as propriedades do sistema de proteção a gás verificadas e explicadas pode ser visto no Apêndice B.

A primeira verificação a ser feita é conferir se o sistema será bloqueado em algum momento. Isso é testado através do comando *'A[] not deadlock'* e deve resultar em uma luz 'verde' no verificador do UPPAAL, mostrando que a propriedade é verdadeira. Porém, essa propriedade não representa o não travamento do sistema em um ciclo de dois ou mais estados que não é possível escapar.

Uma relevante verificação de propriedades é da alcançabilidade de todos os estados do sistema, ou seja, se em algum momento do funcionamento da máquina, aquele estado será atingido. Essas propriedades são dadas abaixo, para as quais o resultado também deverá ser a luz verde, conforme a figura 61. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

Checker do UPPAAL:

E<> 'estado do Controlador'

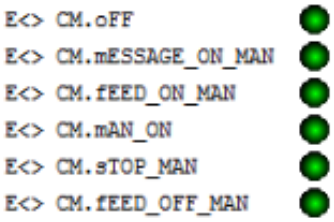

Figura 61 – Propriedades dos estados alcançáveis - Sistema de de proteção a gás

Outra propriedade importante é verificar se em todos os estados a variável `VB_ARGON_ON` está configurada da maneira correta, estando desligada no estado *'oFF'* e ligada no estado *'oN'*. Essa propriedade é dada abaixo e pode também ser vista na figura 62. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

A[] 'estado do Controlador' imply `VB_ARGON_ON` $== 1$, ou

A[] 'estado do Controlador' imply `VB_ARGON_ON` $== 0$

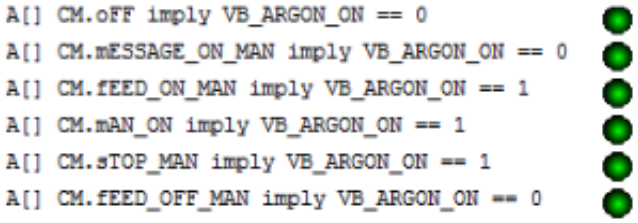

Figura 62 – Propriedades de verificação da variável `VB_ARGON_ON` - Sistema de proteção a gás

Com essas propriedades observa-se que o estado *'oFF'* está invariavelmente com a variável `VB_ARGON_ON` igual a 0, ou seja, com o motor de proteção a gás sempre desligado. Enquanto o estado *'oN'* está invariavelmente com a variável `VB_ARGON_ON` igual a 1, ou seja, com o motor de proteção a gás sempre ligado.

O último tipo de propriedade é verificar se em todos os estados a variável `VB_ARGON` está sempre abaixo do limite máximo, ou seja, ser sempre menor que 95%, independente do estado que estiver. Essa propriedade é dada abaixo e pode também ser vista na figura 63. Nesse trabalho foi observado que cada estado do controle deverá conter uma propriedade, como a descrita abaixo. Se o controlador possuir n estados, logo deverão existir n propriedades semelhantes.

A[] 'estado do Controlador' imply `VB_ARGON` $<= 95\%$

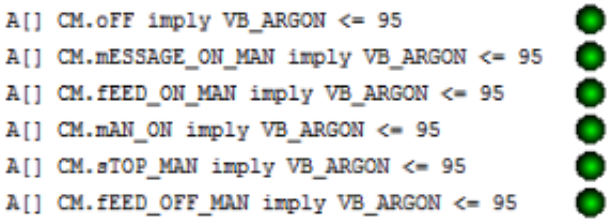

Figura 63 – Propriedades de verificação da variável `VB_TEMPERATURE` - Sistema de proteção a gás

5 Implementação e programação dos sistemas

Neste capítulo é discutido a implementação e programação do sistema. Neste último é explicado o protocolo de comunicação utilizado por cada subsistema e a programação manual do UPPAAL, mapeando os estados, referenciando as variáveis entre UPPAAL e programação em C e explicando as principais funções utilizadas.

5.1 Plataforma

O primeiro modelo de subsistema desenvolvido é relacionado à plataforma. Modelada para elevar e abaixar a altura da peça a ser feita, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada. A plataforma é utilizada, neste trabalho, como o modelo padrão para o dispositivo de deposição de pó, por terem funcionamento básico similar.

Com o objetivo de implementar o software da plataforma, utilizou-se um sistema embarcado de baixo custo, aqui escolhido o ARDUINO UNO. Esta é uma plataforma eletrônica aberta baseada em uma abordagem de fácil utilização software-hardware, permitindo que qualquer pessoa possa utilizá-lo para o desenvolvimento de projetos interativos [34].

Para mostrar o desenvolvimento do software, o código foi elaborado a partir do modelo do UPPAAL da plataforma. Com as entradas é possível simular o funcionamento da plataforma. Os comandos serão recebidos através de comunicação serial com o computador, que contém um programa capaz de se conectar com o ARDUINO UNO, receber mensagens de status e enviar mensagens de comando. Para garantir o sucesso da comunicação entre o computador e o ARDUINO UNO um protocolo de comunicação foi desenvolvido e implementado.

Neste capítulo, será apresentada a construção do subsistema da plataforma, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.1.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são os comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, altura, entre outros. Criando um sistema de comunicação e protocolo de mensagens entre o ARDUINO UNO e o computador.

Nesse tópico serão discutidos como os problemas de comunicação e sincronização de relógios foram resolvidos, quais suas principais funções e objetivos. Primeiramente, definiu-se que o ARDUINO UNO deveria conter a data e hora corretas para que seu funcionamento fosse aplicado e os dados serem computados de forma correta, ou seja, sincronizar o relógio do ARDUINO UNO com o relógio do computador.

O ARDUINO UNO possui uma hora e data de início padrão, que é o dia 01/01/1970 e hora 00:00:00, data conhecida como *Unix epoch* [35], isso se deve ao fato de esta data, dia 1 de janeiro de 1970 às 00:00:00 do Tempo Universal Coordenado (UTC) ser o marco zero do sistema de calendário usado pelo sistema operacional UNIX. Quando o ARDUINO UNO é ligado, ele começa a contar o tempo a partir deste padrão. Para este trabalho, foi desenvolvida uma pequena função para receber a mensagem do computador, contendo a data e hora atual do computador, e sincronizar internamente. Essa função pode ser vista na figura 64 abaixo.

```
#define TIME_MSG_LEN  11
#define TIME_HEADER  'T'
#define TIME_REQUEST  7
void processSyncMessage()
{
  if ( T_SYNC[0] == TIME_HEADER )
  {
    time_t pctime = 0;
    for (int i = 1; i <= TIME_MSG_LEN ; i++)
    {
      if ( T_SYNC[i] >= '0' && T_SYNC[i] <= '9')
      {
        pctime = (10 * pctime) + (T_SYNC[i] - '0') ;
      }
    }
    setTime(pctime);
  }
}
```

Figura 64 – Função de sincronização do relógio

Para realizar essa sincronização o computador deve enviar uma mensagem contendo 'T + Epoch', por exemplo, T1454523714. Ao enviar essa mensagem ao ARDUINO UNO, a função da figura 64 irá calcular o dia e hora atual e sincronizar em sua memória. Caso o ARDUINO UNO recebe essa mensagem, ele irá ajustar sua data e hora com o dia 03 de Fevereiro de 2016 18:21:54 GMT.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhor maneira possível, foi desenvolvido um protocolo de comunicação serial para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

Na tabela 1, é apresentado o protocolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último Byte, 0xFE e 0xFF, respectivamente, são os caracteres que são verificados para checar se a mensagem recebida

começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto.

Tabela 1 – Protocolo de comunicação da plataforma

Protocolo - Plataforma					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Altura da Plataforma	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do motor	Numero da falha	Severidade da falha	Estado do sensor 1	Estado do sensor 2	0xFF

A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;
- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Representam a altura da plataforma;
- Bytes 20 & 21 - Informam o número do sistema, que no caso da plataforma o número é 1;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do motor;

- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32 & 33 - Informam o estado do sensor 1;
- Bytes 34 & 35 - Informam o estado do sensor 2; e,
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Para enviar a mensagem do computador para a plataforma é utilizado outro protocolo, conforme as tabelas 2 e 3. Ao iniciar o ARDUINO UNO pela primeira vez, é necessário sincronizar seu relógio. Para isso, a mensagem seguindo o protocolo da figura 2 é enviado primeiramente. A função de cada parte da mensagem está descrita abaixo:

Tabela 2 – Protocolo de comunicação do computador antes da sincronização

Protocolo - Computador - Antes da sincronização do relógio				
14 Bytes				
1. Byte	2. Byte	3. Byte	4., 5., ..., 12. & 13. Bytes	14. Byte
1 Byte	1 Byte	1 Byte	10 Bytes	1 Byte
0xFF	Comando	T	Epoch	0xFE

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do usuário para o ARDUINO UNO;
- Byte 2 - Comando do computador para iniciar a sincronização do relógio do ARDUINO UNO;
- Byte 3 - 'T' representa o início da mensagem de sincronização;
- Bytes 4 até 13 - Mensagem contendo o número Epoch; e,
- Byte 14 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Após o ARDUINO UNO estar com o relógio sincronizado com o computador, o protocolo da tabela 3 entrará em ação. Ao enviar a mensagem com o comando para o ARDUINO UNO, o computador tem que se certificar que a mensagem chegará da maneira correta. Com isso, é utilizado também o mesmo sistema de protocolo no início e final da mensagem. Desta maneira o ARDUINO UNO lê a mensagem e verifica se o primeiro e último correspondem ao protocolo. Em caso afirmativo, ele lê a mensagem do Byte 2 e realiza o comando que foi enviado. A função de cada parte da mensagem e os comandos possíveis enviados pelo computador estão descritos abaixo:

Tabela 3 – Protocolo de comunicação do computador após a sincronização

Protocolo - Computador - Após sincronização do relógio		
3 Bytes		
1. Byte	2. Byte	3. Byte
1 Byte	1 Byte	1 Byte
0xFF	Comando	0xFE

- Byte 1 - '0xFF' (ou letra 'F' em ASCII) representa o início da mensagem do usuário para o ARDUINO UNO;
- Byte 2 - O comando enviado pelo computador para o ARDUINO UNO; e,
- Byte 3 - '0xFE' (ou letra 'E' em ASCII) representa o final da mensagem.

- 10000001 - Comando que pede ao ARDUINO UNO informar quais são os status atuais do programa;
- 10011001 - Comando para sincronizar o relógio do ARDUINO UNO;
- 10000000 - Comando para enviar o *acknowledge*;
- 00000100 - Comando para fazer o motor parar;
- 00000010 - Comando para fazer a plataforma descer; e,
- 00000001 - Comando para fazer a plataforma subir.

5.1.2 Programação manual do UPPAAL para C

Outra tarefa deste trabalho foi a implementação do código a partir do modelo UPPAAL em linguagem C. Cada um dos estados do modelo tornou-se um valor de uma variável no código C. A numeração de todos os estados pode ser vista na tabela 4 abaixo. O valor da esquerda indica qual a sua numeração e do lado direito qual é o estado, respectivamente. A partir disso, em cada estado existem transições, podendo existir também condições de guardas, sincronização ou atualização de variáveis. O código pode ser analisado no Material complementar dentro da pasta Plataforma em Modelos e Códigos. Nela estão os códigos para controle manual e controle automático do sistema.

Para a questão das guardas, usou-se o mesmo padrão do modelo no UPPAAL, pois as variáveis declaradas no programa em C são idênticas às do modelo. As condições de guarda são representadas pelos laços 'if-else' e as transições e atualizações de variáveis ocorrem dentro dos laços satisfeitos.

Tabela 4 – Mapeamento dos estados da plataforma do UPPAAL para C

Numeração dos Estados - Plataforma					
0	sTOPPED	21	gENERAL_FAILURE_STATE	42	f19
1	vERIFY_HEIGHT	22	wAIT_FEEDBACK_3136	43	f20
2	sTOP_NORMAL	23	aCKNOWLEDGE_3738	44	f21
3	pLATFORM_MIDDLE	24	f1	45	f22
4	pLATFORM_BOTTOM	25	f2	46	f23
5	pLATFORM_TOP	26	f3	47	f24
6	wAITS_CYCLE_MAN	27	f4	48	f25
7	mAN_UP	28	f5	49	f26
8	vERIFY_MOTOR_UP_MAN	29	f6	50	f27
9	sTOP_UP_MAN	30	f7	51	f28
10	mAN_DOWN	31	f8	52	f29
11	vERIFY_MOTOR_DOWN_MAN	32	f9	53	f30
12	sTOP_DOWN_MAN	33	f10	54	f31
13	mAN_UP_UPPER_SIGNAL	34	f11	55	f32
14	fAILURE_STATE_GOING_UP	35	f12	56	f33
15	wAIT_FEEDBACK_2335	36	f13	57	f34
16	wAIT_FEEDBACK_1718	37	f14	58	f35
17	mAN_DOWN_LOWER_SIGNAL	38	f15	59	f36
18	fAILURE_STATE_GOING_DOWN	39	f16	60	f37
19	wAIT_FEEDBACK_2230	40	f17	61	f38
20	wAIT_FEEDBACK_1516	41	f18		

Para a questão de sincronização foi necessário outro método. No código em C foram criadas variáveis para representarem as sincronizações, que são os *channels* dentro do modelo UPPAAL. Por exemplo, no modelo UPPAAL, para sair do estado *'pLATFORM_MIDDLE'* e ir ao estado *'mAN_UP'* é necessário que o GUI envie a sincronização `BS_M1_UP!` e que a plataforma esteja dentro dos limites de trabalho.

No caso do código C, isso funciona de maneira diferente. Para sair do estado *'pLATFORM_MIDDLE'* e ir ao estado *'mAN_UP'* é necessário que a plataforma receba o comando '00000001' do 'GUI'. Ao receber este comando, a variável `BS_M1_UP` vai ser atualizada em 1 e a condição dentro do estado *'pLATFORM_MIDDLE'* será cumprida. Desta maneira, a variável de estado dentro da condição do estado *'pLATFORM_MIDDLE'* será alterada para que, no próximo 'loop' do programa, o próximo estado a ser realizado seja *'mAN_UP'* e não *'pLATFORM_MIDDLE'*. Isso acontece com todos as outras sincronizações do programa.

Tabela 5 – Tabela de equivalência de variáveis entre o modelo UPPAAL e a codificação em C

Tipo UPPAAL	Nome no UPPAAL	Tipo C	Nome em C	Função
int	VB_MOTOR_STATE	int	VB_MOTOR_STATE	Indica o estado do motor
int	AS_P_X	int	AS_P_X	Altura da plataforma
int	AS_P_X_ANT	int	AS_P_X_ANT	Última altura da plataforma
int	VB_FAILURE_NUMBER	int	VB_FAILURE_NUMBER	Número da falha
int	VB_SEVERITY	int	VB_SEVERITY	Severidade da falha
int	VB_PLATFORM_STATE	int	VB_PLATFORM_STATE	Indica o estado da plataforma
int	VB_UPPER_LIMIT_SIGNA	int	VB_UPPER_LIMIT_SIGNA	Variável para o sensor de limite superior
int	VB_LOWER_LIMIT_SIGNA	int	VB_LOWER_LIMIT_SIGN	Variável para o sensor de limite inferior
channel	BS_M1_UP!	bool	BS_M1_UP	Comando para subir
channel	BS_M1_DOWN!	bool	BS_M1_DOWN	Comando para descer
channel	BS_M1_OFF!	bool	BS_M1_OFF	Comando para desligar
channel	VB_ACKNOWLEDGE	bool	VB_ACKNOWLEDGE	Acknowledge dos erros

Todas as sincronizações que se encontravam no modelo UPPAAL foram transformadas em variáveis no código em C. A tabela contendo a equivalência da sincronização no modelo UPPAAL e variáveis no código C pode ser facilmente vista na tabela 5 acima.

5.1.3 Funções desenvolvidas

No total, foram desenvolvidas quatro funções, além das padrões para arduino de *'setup()'* e *'loop()'*. As principais funções desenvolvidas para que o código funcione, além da troca de estados, são responsáveis por receber/enviar as mensagens de/para o computador, limpar variáveis e sincronização do relógio. Estas estão apresentadas na tabela 6.

Tabela 6 – Funções para o código embarcado

Função em C	Descrição
void setup()	Responsável pelas configurações iniciais do sistema, seta a velocidade de ler e escrever a porta serial, setar os pinos que serão usados como inputs ou outputs e ajusta variáveis
void loop ()	Essa é a função que contém todos os estados e suas trocas dentro do ARDUINO, nela está contida toda a programação do modelo UPPAAL em código C. É onde ocorrerá a repetição a cada vez que um ciclo for concluido
void Command()	Função incubida de ler as portas digitais e analógicas do sistema, de receber a mensagem do computador e verificar se está dentro do protocolo. Caso esteja, verifica qual foi o comando recebido e altera as variáveis do sistema
void CleanVariable ()	Realiza a limpeza das variáveis quando o sistema retorna ao seu estado inicial
void processSyncMessage()	Recebe a mensagem de sincronização do computador (T+Epoch) e sincroniza a hora e data do ARDUINO com a hora e data da mensagem enviada
void showTime(int fail_1)	Escreve em um vetor de char todas as informações relevantes atuais do sistema, como data, hora, altura, número do sistema, estado atual do sistema, estado do motor, número da falha e sua severidade e os estados dos sensores. Por fim colocando os Bytes de início e fim e escreve na USB para enviar ao computador

A função 'Command()', que realiza a leitura do sinal analógico, escreve nas portas digitais, aguarda mensagem do computador, verifica se a mensagem recebida está dentro do protocolo, e qual foi o comando recebido, atualizando as variáveis. Outra função é a CleanVariable(), responsável por limpar as variáveis do sistema.

A função para sincronizar relógio (processSyncMessage()) é responsável por processar a mensagem de sincronização do computador. Por último, tem-se a função showTime() responsável por pegar todas as variáveis do sistema e informações importantes e alocar num vetor de caracteres no padrão do protocolo definido. Por fim, enviar pela porta serial para que a mensagem seja lida e interpretada pelo computador.

Cada um dos subsistemas possui um número de identificação, que é a variável `'SystN'`, declarada como inteiro dentro desta função. Para a plataforma, o número de identificação é 1. A tabela completa contendo o número de cada subsistema pode ser vista na tabela 7 abaixo.

Tabela 7 – Número de sistema de cada subsistema

Número	Subsistema
1	Plataforma
2	Dispositivo de deposição de pó
3	Laser
4	Sistema de carregamento de pó 1
5	Sistema de carregamento de pó 2
6	Sistema de carregamento de pó 3
7	Sistema de aquecimento
8	Sistema de refrigeração
9	Sistema de filtragem
10	Sistema de vácuo
11	Sistema de proteção a gás

5.2 Dispositivo de deposição de pó

O dispositivo de deposição de pó foi modelado para ir para frente e para trás, conforme os comandos do usuário, e indicar falhas quando alguma situação indesejada ocorre-se.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Para mostrar o desenvolvimento do software, o código foi elaborado a partir do modelo do UPPAAL do dispositivo de deposição de pó. Com as entradas é possível simular o funcionamento do dispositivo de deposição de pó. Os comandos serão recebidos através de comunicação serial com o computador, que contém um programa capaz de se conectar com o ARDUINO UNO, receber mensagens de status e enviar mensagens de comando. Para garantir o sucesso da comunicação entre o computador e o ARDUINO UNO um protocolo de comunicação para o dispositivo de deposição de pó foi desenvolvido e implementado.

Nesta seção, será apresentada a construção do subsistema do dispositivo de deposição de pó, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.2.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, distância, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhor maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso. Na tabela 8, é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;

Tabela 8 – Protocolo de comunicação - Dispositivo de deposição de pó

Protocolo - Dispositivo de deposição de pó					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Distância do dispositivo	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do motor	Numero da falha	Severidade da falha	Estado do sensor 1	Estado do sensor 2	0xFF

- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;
- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Representam a distância do dispositivo de deposição de pó;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do dispositivo de deposição de pó o número é 2;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do motor;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32 & 33 - Informam o estado do sensor 1;

- Bytes 34 & 35 - Informam o estado do sensor 2; e,
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Para enviar a mensagem do computador para o dispositivo de deposição de pó é utilizado outro protocolo, ele é análogo ao caso da plataforma. Com exceção de que alguns comandos mudam quando a data e hora do sistema do dispositivo de deposição de pó já estiver sincronizado, eles são:

- 10000001 - Comando que pede ao ARDUINO UNO informar quais são os status atuais do programa;
- 10011001 - Comando para sincronizar o relógio do ARDUINO UNO;
- 10000000 - Comando para enviar o *acknowledge*;
- 00000100 - Comando para fazer o motor parar;
- 00000010 - Comando para fazer o dispositivo de deposição de pó ir para frente; e,
- 00000001 - Comando para fazer o dispositivo de deposição de pó ir para trás.

5.2.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo da plataforma. As mudanças ocorreram no momento da numeração dos estados, que pode ser vista na tabela 9.

Tabela 9 – Numeração dos estados do Dispositivo de deposição de pó

Numeração dos Estados - Dispositivo de deposição de pó					
0	sTOPPED	21	wAIT_FEEDBACK_3136	42	f20
1	vERIFY_DISTANCE	22	aCKNOWLEDGE_3738	43	f21
2	sTOP_NORMAL	23	f1	44	f22
3	rECOATER_MIDDLE	24	f2	45	f23
4	rECOATER_FRONT	25	f3	46	f24
5	rECOATER_BACK	26	f4	47	f25
6	mAN_BACK	27	f5	48	f26
7	vERIFY_MOTOR_BACK_MAN	28	f6	49	f27
8	sTOP_BACK_MAN	29	f7	50	f28
9	mAN_FRONT	30	f8	51	f29
10	vERIFY_MOTOR_FRONT_MAN	31	f9	52	f30
11	sTOP_FRONT_MAN	32	f10	53	f31
12	mAN_UP_BACK_SIGNAL	33	f11	54	f32
13	fAILURE_STATE_GOING_BACK	34	f12	55	f33
14	wAIT_FEEDBACK_2335	35	f13	56	f34
15	wAIT_FEEDBACK_1718	36	f14	57	f35
16	mAN_DOWN_FRONT_SIGNAL	37	f15	58	f36
17	fAILURE_STATE_GOING_FRONT	38	f16	59	f37
18	wAIT_FEEDBACK_2230	39	f17	60	f38
19	wAIT_FEEDBACK_1516	40	f18		
20	gENERAL_FAILURE_STATE	41	f19		

A equivalência da sincronização no modelo UPPAAL e variáveis no código C pode ser vista na tabela 10 abaixo. As funções utilizadas para o dispositivo de deposição de pó são análogas às explicadas na plataforma.

Tabela 10 – Tabela de equivalência de variáveis entre o modelo UPPAAL e a codificação em C do Dispositivo de deposição de pó

DISPOSITIVO DE DEPOSIÇÃO DE PÓ				
Tipo UPPAAL	**Nome no UPPAAL**	**Tipo C**	**Nome em C**	**Função**
int	VB_MOTOR_STATE	int	VB_MOTOR_STATE	Indica o estado do motor
int	AS_P_X	int	AS_P_X	Distância do dispositivo
int	AS_P_X_ANT	int	AS_P_X_ANT	Ultima distância do dispositivo
int	VB_FAILURE_NUMBER	int	VB_FAILURE_NUMBER	Número da falha
int	VB_SEVERITY	int	VB_SEVERITY	Severidade da falha
int	VB_RECOATER_STATE	int	VB_RECOATER_STATE	Indica o estado do dispositivo
int	VB_BACK_LIMIT_SIGNAL	int	VB_BACK_LIMIT_SIGNAL	Variável para o sensor de limite frontal
int	VB_FRONT_LIMIT_SIGNAL	int	VB_FRONT_LIMIT_SIGNAL	Variável para o sensor de limite traseiro
int	VB_ENCODER	int	VB_ENCODER	Quantidade de passos que o motor deve dar
channel	BS_M1_BACK!	bool	BS_M1_BACK	Comando para ir para trás
channel	BS_M1_FRONT!	bool	BS_M1_FRONT	Comando para ir para frente
channel	BS_M1_OFF!	bool	BS_M1_OFF	Comando para desligar
channel	VB_ACKNOWLEDGE	bool	VB_ACKNOWLEDGE	Acknowledge dos erros

5.3 Sistema de carregamento 1

O sistema de carregamento 1 foi modelado para rotacionar o motor no sentido horário, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema de carregamento 1, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.3.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhor maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

Na tabela 11 abaixo, é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;

- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Representam o valor do contador do *encoder* do sistema de carregamento 1;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do sistema de carregamento 1 o número é 4;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do motor;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32, 33, 34 & 35 - Vazio representado por 0;
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Tabela 11 – Protocolo de comunicação - Sistema de carregamento 1

Protocolo - Sistema de carregamento 1					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Contador	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do motor	Numero da falha	Severidade da falha	0	0	0xFF

Para enviar a mensagem do computador para o sistema de carregamento 1 é utilizado outro protocolo, ele é análogo ao caso da plataforma. Com exceção de que alguns comandos mudam quando a data e hora do sistema de carregamento 1 já estiver sincronizado, eles são:

- 10000001 - Comando que pede ao ARDUINO UNO informar quais são os status atuais do programa;
- 10011001 - Comando para sincronizar o relógio do ARDUINO UNO;
- 10000000 - Comando para enviar o *acknowledge*;
- 00000100 - Comando para fazer o motor parar;
- 00000010 - Comando para fazer o motor do sistema de carregamento 1 rotacionar no sentido anti-horário; e,
- 00000001 - Comando para fazer o motor do sistema de carregamento 1 rotacionar no sentido horário.

5.3.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo da plataforma. As mudanças ocorreram no momento da numeração dos estados, que pode ser vista na tabela 12.

Tabela 12 – Numeração dos estados do Sistema de carregamento 1

Numeração dos Estados - Sistema de carregamento 1			
0	sTOPPED	4	sTOP_CLOCKWISE_MAN
1	sTOP_NORMAL	5	wAIT_FEEDBACK_2335
2	mAN_CLOCKWISE	6	f23
3	vERIFY_MOTOR_CLOCKWISE_MAN	7	f24

A equivalência da sincronização no modelo UPPAAL e variáveis no código C pode ser vista na tabela 13. As funções utilizadas para o sistema de carregamento 1 são análogas às explicadas na plataforma.

Tabela 13 – Tabela de equivalência de variáveis entre o modelo UPPAAL e a codificação em C do Sistema de carregamento 1

SISTEMA DE CARREGAMENTO 1				
Tipo UPPAAL	**Nome no UPPAAL**	**Tipo C**	**Nome em C**	**Função**
int	VB_MOTOR_STATE	int	VB_MOTOR_STATE	Indica o estado do motor
int	VB_FAILURE_NUMBER	int	VB_FAILURE_NUMBER	Número da falha
int	VB_SEVERITY	int	VB_SEVERITY	Severidade da falha
int	VB_COUNTER	int	VB_COUNTER	Quantidade de passos que o motor deve realizar
channel	BS_M1_ANTICLOCKWISE	bool	BS_M1_ANTICLOCKWIS	Comando para o motor girar no sentido anti-horário
channel	BS_M1_CLOCKWISE!	bool	BS_M1_CLOCKWISE	Comando para o motor girar no sentido horário
channel	BS_M1_OFF!	bool	BS_M1_OFF	Comando para desligar
channel	VB_ACKNOWLEDGE!	bool	VB_ACKNOWLEDGE	Acknowledge dos erros

5.4 Sistema de carregamento 2

O sistema de carregamento 2 foi modelado para rotacionar o motor no sentido horário ou anti-horário, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1. Neste capítulo, será apresentada a construção do subsistema de carregamento 2, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C.

5.4.1 Sistema e protocolo de comunicação

O protocolo de comunicação para o sistema de carregamento 2 é análogo ao desenvolvido para o sistema de carregamento 1. A única diferença é o número do sistema. O número do sistema do sistema de carregamento 2 é 5.

5.4.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo da plataforma. As mudanças ocorreram no momento da numeração dos estados, que pode ser vista na tabela 14.

Tabela 14 – Numeração dos estados do Sistema de carregamento 2

Numeração dos Estados - Sistema de carregamento 2			
0	sTOPPED	9	wAIT_FEEDBACK_2230
1	sTOP_NORMAL	10	wAIT_FEEDBACK_3136
2	mAN_CLOCKWISE	11	f19
3	vERIFY_MOTOR_CLOCKWISE_MAN	12	f22
4	sTOP_CLOCKWISE_MAN	13	f23
5	mAN_ANTICLOCKWISE	14	f24
6	vERIFY_MOTOR_ANTICLOCKWISE_MAN	15	f31
7	sTOP_ANTICLOCKWISE_MAN	16	f36
8	wAIT_FEEDBACK_2335		

A equivalência da sincronização no modelo UPPAAL e variáveis no código C é análoga ao sistema de carregamento 1 e as funções utilizadas para o sistema de carregamento 2 são análogas às explicadas na plataforma.

5.5 Sistema de carregamento 3

O sistema de carregamento 3 possui funcionamento idêntico ao do sistema de carregamento 2. A diferença entre os dois subsistemas é o número do sistema. No sistema de carregamento 3 o número do sistema é 6.

5.6 Laser

O laser foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada. O laser é utilizado, nesse trabalho, como o modelo padrão para o sistema de aquecimento, de refrigeração, de filtragem, de vácuo e de proteção a gás, por terem funcionamento básico similar.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema do laser, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.6.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhora maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso. Na tabela 15 é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;

- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Vazio representado por 0;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do laser o número é 3;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do componente do laser;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32, 33, 34 & 35 - Vazio representado por 0;
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Tabela 15 – Protocolo de comunicação - Laser

Protocolo - Laser					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	0	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do laser	Numero da falha	Severidade da falha	0	0	0xFF

Para enviar a mensagem do computador para o laser é utilizado outro protocolo, ele é análogo ao caso da plataforma. Com exceção de que alguns comandos mudam quando a data e hora do sistema do laser já estiver sincronizado, eles são:

- 10000001 - Comando que pede ao ARDUINO UNO informar quais são os status atuais do programa;

- 10011001 - Comando para sincronizar o relógio do ARDUINO UNO;
- 10000000 - Comando para enviar o *acknowledge*;
- 00000010 - Comando para desligar o laser; e,
- 00000001 - Comando para ligar o laser.

5.6.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo da plataforma. As mudanças ocorreram no momento da numeração dos estados, que pode ser vista na tabela 16.

Tabela 16 – Numeração dos estados do Laser

Numeração dos Estados - Laser			
0	oFF	4	sTOP_MAN
1	mESSAGE_ON_MAN	5	fEED_OFF_MAN
2	fEED_ON_MAN	6	f1
3	mAN_ON	7	f2

A equivalência da sincronização no modelo UPPAAL e variáveis no código C pode ser vista na tabela 17. As funções utilizadas para o laser são análogas às explicadas na plataforma.

Tabela 17 – Tabela de equivalência de variáveis entre o modelo UPPAAL e a codificação em C do Laser

LASER				
Tipo UPPAAL	Nome no UPPAAL	Tipo C	Nome em C	Função
bool	VB_LASER_ON	int	VB_LASER_ON	Indica o estado do componente do laser
int	VB_FAILURE_NUMBER	int	VB_FAILURE_NUMBER	Número da falha
int	VB_SEVERITY	int	VB_SEVERITY	Severidade da falha
channel	BS_ON	bool	BS_ON	Comando para ligar
channel	BS_OFF	bool	BS_OFF	Comando para desligar
channel	VB_ACKNOWLEDGE	bool	VB_ACKNOWLEDGE	Acknowledge dos erros

5.7 Sistema de aquecimento

O sistema de aquecimento foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema de aquecimento, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.7.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, temperatura, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhora maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

Na tabela 18 abaixo, é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. Para enviar a mensagem do computador para o sistema de aquecimento é utilizado outro protocolo, ele é análogo ao caso do laser. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;

- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Informam a temperatura da câmara;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do sistema de aquecimento o número é 7;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do motor de aquecimento;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32, 33, 34 & 35 - Vazio representado por 0; e,
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Tabela 18 – Protocolo de comunicação - Sistema de aquecimento

Protocolo - Sistema de aquecimento					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Temperatura da câmara	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do motor	Numero da falha	Severidade da falha	0	0	0xFF

5.7.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo do laser. Os estados utilizados no sistema de aquecimento são os mesmos que os utilizados no laser. A equivalência da sincronização no modelo UPPAAL e variáveis no código C é diferente e pode ser vista na tabela 19. As funções utilizadas para o sistema de aquecimento são análogas às explicadas na plataforma.

Tabela 19 – Tabela de equivalência de variáveis entre o modelo UPPAAL e a codificação em C do sistema de aquecimento

SISTEMA DE AQUECIMENTO				
Tipo UPPAAL	**Nome no UPPAAL**	**Tipo C**	**Nome em C**	**Função**
bool	VB_HEATER_ON	int	VB_HEATER_ON	Indica o estado do motor de aquecimento
int	VB_FAILURE_NUMBER	int	VB_FAILURE_NUMBER	Número da falha
int	VB_SEVERITY	int	VB_SEVERITY	Severidade da falha
int	VB_TEMPERATURE	int	VB_TEMPERATURE	Indica a temperatura da câmara
channel	BS_ON	bool	BS_ON	Comando para ligar
channel	BS_OFF	bool	BS_OFF	Comando para desligar
channel	VB_ACKNOWLEDGE	bool	VB_ACKNOWLEDGE	Acknowledge dos erros

5.8 Sistema de refrigeração

O sistema de refrigeração foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema de refrigeração, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.8.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, temperatura, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhora maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

O protocolo do sistema de refrigeração é análogo ao protocolo do sistema de aquecimento. A diferença é que o número do sistema do sistema de refrigeração é 8. Para enviar a mensagem do computador para o sistema de refrigeração é utilizado outro protocolo, ele é análogo ao caso do laser.

5.8.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo do laser. Os estados utilizados no sistema de refrigeração são os mesmos que os utilizados no laser. A equivalência da sincronização no modelo UPPAAL e variáveis no código C é análoga ao sistema de aquecimento. Mudando-se a variável `VB_HEATER_ON` para `VB_REFRIGERATION_ON`. As funções utilizadas para o sistema de refrigeração são análogas às explicadas na plataforma.

5.9 Sistema de filtragem

O sistema de filtragem foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema de filtragem, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.9.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhora maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

Na tabela 20 abaixo, é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. Para enviar a mensagem do computador para o sistema de filtragem é utilizado outro protocolo, ele é análogo ao caso do laser. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;

- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Informam a quantidade de pó para ser filtrado, de 0% até 100%;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do sistema de filtragem o número é 9;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do motor de filtragem;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32, 33, 34 & 35 - Vazio representado por 0; e,
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Tabela 20 – Protocolo de comunicação - Sistema de filtragem

Protocolo - Sistema de filtragem					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Quantidade de pó para ser filtrado	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do motor	Numero da falha	Severidade da falha	0	0	0xFF

5.9.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo do laser. Os estados utilizados no sistema de filtragem são os mesmos que os utilizados no laser. A equivalência da sincronização no modelo UPPAAL e variáveis no código C é análoga ao sistema de aquecimento. Mudando-se as variáveis `VB_HEATER_ON` para `VB_FILTER_ON` e `VB_TEMPERATURE` para `VB_CAPACITY`. As funções utilizadas para o sistema de filtragem são análogas às explicadas na plataforma.

5.10 Sistema de vácuo

O sistema de vácuo foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorre alguma situação indesejada.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema de vácuo, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.10.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhora maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

Na tabela 21 abaixo, é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. Para enviar a mensagem do computador para o sistema de vácuo é utilizado outro protocolo, ele é análogo ao caso do laser. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;

- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Informam a pressão dentro da câmara;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do sistema de vácuo o número é 10;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Vazio representado por 0 ;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32 & 33 - Informam o estado atual da válvula;
- Bytes 34 & 35 - Informam o estado atual do motor de vácuo; e,
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

Tabela 21 – Protocolo de comunicação - Sistema de vácuo

Protocolo - Sistema de vácuo					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Pressão na câmara	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
0	Numero da falha	Severidade da falha	Estado da válvula	Estado do motor	0xFF

5.10.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo da plataforma. As mudanças ocorreram no momento da numeração dos estados, que pode ser vista na tabela 22 abaixo.

A equivalência da sincronização no modelo UPPAAL e variáveis no código C pode ser vista na tabela 23. As funções utilizadas para o sistema de vácuo são análogas às explicadas na plataforma.

Tabela 22 – Numeração dos estados do Sistema de vácuo

Numeração dos Estados - Sistema de vácuo			
0	oFF	6	sTOP_VALVE_MAN
1	mESSAGE_ON_VALVE_MAN	7	fEED_OFF_VALVE_MAN
2	fEED_ON_VALVE_MAN	8	sTOP_MOTOR_MAN
3	mESSAGE_ON_MOTOR_MAN	9	fEED_OFF_MOTOR_MAN
4	fEED_ON_MOTOR_MAN	10	f1
5	mAN_ON	11	f2

Tabela 23 – Tabela de equivalência de variáveis entre o modelo UPPAAL e a codificação em C do Sistema de vácuo

SISTEMA DE VÁCUO				
Tipo UPPAAL	**Nome no UPPAAL**	**Tipo C**	**Nome em C**	**Função**
bool	VB_VALVE_OPEN	bool	VB_VALVE_OPEN	Indica o estado da válvula
bool	VB_MOTOR_ON	bool	VB_MOTOR_ON	Indica o estado do motor de vácuo
int	VB_FAILURE_NUMBER	int	VB_FAILURE_NUMBER	Número da falha
int	VB_SEVERITY	int	VB_SEVERITY	Severidade da falha
int	VB_PRESSURE	int	VB_PRESSURE	Indica a pressão dentro da câmara
channel	BS_ON	bool	BS_ON	Comando para ligar
channel	BS_OFF	bool	BS_OFF	Comando para desligar
channel	VB_ACKNOWLEDGE	bool	VB_ACKNOWLEDGE	Acknowledge dos erros

5.11 Sistema de proteção a gás

O sistema de proteção a gás foi modelado para ligar e desligar, conforme os comandos do usuário, e indicar falhas quando ocorrer alguma situação indesejada ocorre-se.

De forma análoga à plataforma, utilizou-se também para este caso um sistema embarcado de baixo custo, o ARDUINO UNO. Seu desenvolvimento foi análogo, também sendo criado o sistema de comunicação e a programação do sistema. Para quaisquer dúvidas de desenvolvimento desta parte, vide a seção 5.1.1.

Nesta seção, será apresentada a construção do subsistema de proteção a gás, seu sistema de comunicação e como foi realizada a programação do UPPAAL para a linguagem C, linguagem do ARDUINO UNO.

5.11.1 Sistema e protocolo de comunicação

Para que este trabalho fosse realizado o ARDUINO UNO deveria não somente se conectar ao computador, mas também ser capaz tanto de receber mensagens, que são comandos recebidos pelo computador, quanto enviar mensagens, informando o status de funcionamento, qual a data e hora, estado, altura, entre outros. Para o problema de sincronização de relógio a solução é análoga ao da plataforma.

Para que a comunicação entre o computador e o ARDUINO UNO ocorra da melhora maneira possível, foi desenvolvido um protocolo de comunicação para que ambos se comuniquem de maneira correta. Isso evita que a mensagem recebida seja lida de forma errada pelo sistema e mostrando algo falso ao usuário ou realizando um comando falso.

Tabela 24 – Protocolo de comunicação - Sistema de proteção a gás

Protocolo - Sistema de proteção a gás					
36 Bytes					
1. Byte	2. Byte	3. Byte	4. & 5. Byte	6. & 7. Bytes	8. & 9. Bytes
1 Byte	1 Byte	1 Byte	2 Bytes	2 Bytes	2 Bytes
0xFE	Erro de Protocolo	Mensagem Sinc.	Dia	Mes	Ano
10. & 11. Bytes	12. & 13. Bytes	14. & 15. Bytes	16., 17., 18. & 19. Bytes	20. & 21. Bytes	22. & 23. Bytes
2 Bytes	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes
Hora	Minuto	Segundo	Concentração de gás	Numero do sistema	Estado do sistema
24. & 25. Bytes	26., 27., 28. & 29. Bytes	30. & 31. Bytes	32. & 33. Bytes	34. & 35. Bytes	36. Byte
2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
Estado do motor	Numero da falha	Severidade da falha	0	0	0xFF

Na tabela 24 acima, é apresentado o procolo de comunicação do ARDUINO UNO para o computador contendo 36 Bytes de tamanho. O primeiro e último, '0xFE' e '0xFF', respectivamente, são os caracteres que são verificados para checar que a mensagem recebida

começa e termina de forma correta e que nenhuma informação foi perdida durante o trajeto. Para enviar a mensagem do computador para o sistema de proteção a gás é utilizado outro protocolo, ele é análogo ao caso do laser. A função de cada parte da mensagem está descrita abaixo:

- Byte 1 - '0xFE' (ou letra 'E' em ASCII) representa o início da mensagem do ARDUINO UNO para o usuário;
- Byte 2 - Informa se houve erro de protocolo;
- Byte 3 - Informa se o ARDUINO UNO foi sincronizado;
- Bytes 4 & 5 - Informam o Dia salvo no ARDUINO UNO;
- Bytes 6 & 7 - Informam o Mês salvo no ARDUINO UNO;
- Bytes 8 & 9 - Informam o Ano salvo no ARDUINO UNO;
- Bytes 10 & 11 - Informam a Hora salva no ARDUINO UNO;
- Bytes 12 & 13 - Informam os Minutos salvos no ARDUINO UNO;
- Bytes 14 & 15 - Informam os Segundos salvos no ARDUINO UNO;
- Bytes 16, 17, 18 & 19 - Informam a concentração de gás dentro da câmara;
- Bytes 20 & 21 - Informam o número do sistema, que no caso do sistema de proteção a gás o número é 11;
- Bytes 22 & 23 - Informam o estado atual do sistema;
- Bytes 24 & 25 - Informam o estado atual do motor de proteção a gás;
- Bytes 26, 27, 28 & 29 - Informam o número da falha que ocorreu. Caso nenhuma falha tenha ocorrido o valor mostrado será 0000;
- Bytes 30 & 31 - Informam a severidade da falha ocorrida, caso nenhuma falha tenha ocorrido o valor mostrado será 00;
- Bytes 32, 33, 34 & 35 - Vazio representado por 0; e,
- Byte 36 - '0xFF' (ou letra 'F' em ASCII) representa o final da mensagem.

5.11.2 Programação manual do UPPAAL para C

Para a implementação do código a partir do modelo UPPAAL em linguagem C a estratégia foi análoga à realizada no modelo do laser. Os estados utilizados no sistema de proteção a gás são os mesmos que os utilizados no laser. A equivalência da sincronização no modelo UPPAAL e variáveis no código C é análoga ao sistema de aquecimento. Mudando-se as variáveis `VB_HEATER_ON` para `VB_ARGON_ON` e `VB_TEMPERATURE` para `VB_ARGON`. As funções utilizadas para o sistema de proteção a gás são análogas às explicadas na plataforma.

6 Conclusões e Perspectivas

Neste trabalho foi abordada uma parte do ciclo de desenvolvimento de software, como parte de um projeto mais amplo, para amadurecimento de uma metodologia de validação e verificação de software, em desenvolvimento. O maior desafio e dificuldade encontrado pelo autor deste trabalho foi na elaboração e design de cada um dos subsistemas, tendo de re-avalair e retrabalhar nos modelos diversas vezes e verificar todas as propriedades se estavam, ainda, sendo executadas de forma correta, gastando assim muito tempo.

Utilizando esta metodologia, buscou-se analisar a redução do tempo de desenvolvimento e implementação, o aumento da confiabilidade, obtenção de modelos padrão para as funcionalidades e a rastreabilidade de possíveis defeitos não previstos, durante o ciclo de desenvolvimento de software. Obteve-se êxito ao utilizar a metodologia proposta, escolhendo o sistema, recebendo os requisitos e traduzindo-os para linguagem CTL, modelando e verificando com o Model Checker e, por fim, gerando os códigos em C, obtendo resultados muito similares com os de Rodrigo Pastl, orientador no instituto. Estes resultados não serão apresentador por serem condifenciais. Validando assim que, mesmo fazendo de modos diferentes e sem conversar entre partes sobre o desenvolvimento, chegou-se no mesmo resultado.

Partindo da máquina de SLM, foram recebidas as especificações do cliente. Posteriormente, foram desenvolvidos modelos de cada subsistema da máquina, verificação das principais propriedades de cada modelo e implementados os códigos para os onze subsistemas. Sendo cada subsistema implementado em sistema embarcado de baixo custo, utilizando ARDUINO UNO.

Segundo Pontes [14], o desenvolvimento de software utilizando a ferramenta *model checking* do UPPAAL permitiu concluir que esta técnica é uma boa solução para software embarcado, detectando erros e falhas no modelo no ciclo inicial de desenvolvimento. Evitando futuramente altos custos de retrabalho ao projeto e encontrando de forma rápida, prática e segura possíveis problemas e falhas do modelo. Sem o uso da ferramenta, somente em etapas mais avançadas do desenvolvimento estes problemas seriam detectados, promovendo altos custos de projeto para fixar os problemas e uma grande quantidade de horas de retrabalho.

Esse projeto foi de grande crescimento e desenvolvimento pessoal e profissional ao autor. Tendo a chance de se adaptar, trabalhar e crescer num dos mais renomeados institutos da Alemanha, Fraunhofer - Institut für Produktionsanlagen und Konstruktionstechnik. Além da evolução pessoal, o resultado atingiu a proposta inicial e é usado, ainda hoje, como referência no instituto para os projetos que utilizam a máquina SLM como foco.

6.1 Sugestões para trabalhos futuros

Para apresentar o trabalho desenvolvido de forma mais visual e de fácil entendimento para um leigo, é sugerida a montagem de um protótipo funcional. Por exemplo, montar um protótipo individual para cada sistema, apresentando um ao lado do outro, com todos funcionando ao mesmo tempo e conversando com a central. Juntamente com uma tela ou monitor apresentando todas as variáveis e o que está acontecendo na máquina.

Para facilitar o processo de modelagem e codificação em C do modelo, o software Matlab possui uma ferramenta capaz de modelar o sistema e, automaticamente, criar o código em C desta modelagem. Facilitando e otimizando a geração de código e, desta forma, evitando que haja erro humano ao traduzir do modelo para a codificação. Porém, foi uma alternativa descoberta somente no final do projeto, não sendo utilizada.

Uma oportunidade é de realizar a modelagem contendo todos os sistemas em um único modelo, verificando todas as propriedades e as simulações. Assim observando de estado por estado de sistema por sistema toda a evolução do processo de fabricação de um componente pela máquina de SLM.

Estudo mais aprofundado de cada um dos sistemas para fazer modelos mais otimizados, custando menos poder de processamento, tanto ao verificar suas propriedades, quanto ao traduzir o modelo para código em C no ARDUINO UNO. Resultando num processo mais rápido e com possibilidade menor de erros.

Por fim, uma oportunidade analisada é de transformar todos os requisitos em linguagem matemática para o UPPAAL, gerando o mínimo de requisitos obrigatórios para a verificação de propriedades da ferramenta.

Referências

1 CASSANDRAS C. G., L. S. *Introduction to Discrete Event Systems*. 2nd ed. ed. [S.l.]: New York, Springer, 2008. Citado 7 vezes nas páginas 9, 27, 29, 32, 33, 34 e 35.

2 IMAGEM fornecida por: SLM Solutions. [Online; Acessado em 18-Fevereiro-2016]. Disponível em: <http://www.slm-solutions.com/cms/upload/slideshow/SLM125_Bauraum.jpg>. Citado 2 vezes nas páginas 9 e 46.

3 SRIVATSAN T.S; SUDARSHAN, T. Additive manufacturing: Innovations, advances, and applications. *CRC Press*, 2016. Citado 3 vezes nas páginas 9, 47 e 50.

4 RIEDEL Process in Cooling. [Online; Acessado em 16-Fevereiro-2016]. Disponível em: <http://www.riedel-cooling.de/produkte/standard-baureihen/pc-baureihe-chiller-fuer-die-industriekuehlung-1-226-kw>. Citado 3 vezes nas páginas 9, 47 e 48.

5 YOUTUBE - SLM Solutions GmbH und System SLM 250HL (em alemão). [Online; Acessado em 16-Fevereiro-2016]. Disponível em: <https://www.youtube.com/watch?v=4I_18xp7LT0>. Citado 2 vezes nas páginas 9 e 49.

6 LEVESON, N. Role of software in spacecraft accidents. *Journal of Spacecrafts and Rockets*, v. 41, no. 4, p. 564–575, 2005. Citado na página 23.

7 WORLD, C. *Piores falhas de Software em 2010*. 2010. [Acessado em 03-Fevereiro-2016]. Disponível em: <http://www.computerworld.com.pt/2010/12/23/piores-falhas-de-software-em-2010/>. Citado na página 23.

8 SAYAO, L. *Modelos teóricos em ciência da informação - abstração e método científico*. 2001. [Online; Acessado em 03-Fevereiro-2016]. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-19652001000100010&lang=pt>. Citado na página 27.

9 EPSTEIN, J. M. Why model? *Journal of Artificial Societies and Social Simulation*, v. 11, no. 4 12, 2008. Citado na página 27.

10 LATHI, B. *Sinais e Sistemas Lineares*. 1a ed. ed. [S.l.]: Bookman, 2006. Citado na página 28.

11 CURY, J. *Teoria de Controle Supervisório de Sistemas a Eventos Discretos*. [S.l.]: Em Proceedings of the V Simpósio Brasileiro de Automação Inteligente, Canela, RS - Brasil, 2001. Citado na página 30.

12 HARTMANN, S. *The World as a Process: Simulations in the Natural and Social Sciences em R. Hegselmann et al. (eds.), Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View, Theory and Decision Library*. [S.l.]: Dordrecht, 1996. Citado na página 31.

13 CLARKE, E. *Model Checking*. 1st ed. ed. [S.l.]: Michigan, 1999. Citado na página 32.

14 PONTES, R. *Contribuições do Model Checking e da Metodologia COFI para o Software Embarcado Espacial*. [S.l.]: São José dos Campos, 2011. Citado 4 vezes nas páginas 32, 38, 39 e 145.

15 BEHRMANN, G. *A Tutorial on UPPAAL em Proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. [S.l.: s.n.], 2004. Citado na página 35.

16 CARVALHO, J. *Tutorial de UPPAAL*. [S.l.: s.n.], 2009. Citado na página 38.

17 MANI M.; LANE, B. D. A. F. S. M. S. F.-R. Nistir 8036, measurement science needs for real-time control of additive manufacturing powder bed fusion processes. *NIST - National Institute of Standards and Technology*, Fevereiro 2015. Citado na página 41.

18 INTERNATIONAL, A. Astm standard 2792. standard terminology for additive manufacturing technologies. *West Conchocken, PA*, Fevereiro 2012. Citado na página 41.

19 WOHLERS, T. Wohlers report, rapid prototyping, tooling & manufacturing state of the industry. *Annual Worldwide Progress Report*, 2005. Citado na página 41.

20 MEINERS W.; WISSENBACK, K. P. R. Direct selective laser sintering of steel powder. *Proceedings of the LANE'97*, p. 615–622, 1997. Citado na página 41.

21 OVER, C. Selective laser melting: a new approach for the direct manufacturing of metal parts and tools. *Proceedings of the International Conferences on LANE*, p. 391–398, 2001. Citado na página 41.

22 YADROITSEV I.; BERTRAND, P. S. I. Parametric analysis of the selective laser melting process. *Applied Surface Science*, v. 253, p. 8064–8069, 2007. Citado 2 vezes nas páginas 41 e 42.

23 VAITHILINGAM J.; GOODRIDGE, R. H. R. C. S. E. S. The effect of laser remelting on the surface chemistry of ti_6al4v componentes fabricated by selective laser melting. *Journal of Materials Processing Technology*, v. 232, p. 1–8, 2016. Citado na página 43.

24 MEIER H; HABERLAND, C. Experimental studies on selective laser melting of metallic parts. *Materialwissenschaft und Werkstofftechnik*, v. 39, Issue 9, p. 665–670, Setembro 2008. Citado na página 43.

25 LIVRE, W. a enciclopédia. *Rotary Encoder*. 2016. [Online; Acessado em 18-Fevereiro-2016]. Disponível em: <https://en.wikipedia.org/wiki/Rotary_encoder>. Citado na página 45.

26 SIEGMAN, A. *Lasers*. [S.l.]: University Science Books, 1986. Citado na página 47.

27 TODAY, P. *SLM Solutions Announces New PSA 500 Automatic Powder Sieving Station*. [Online; Acessado em 16-Fevereiro-2016]. Disponível em: <http://www.prototypetoday.com/slm-solutions/slm-solutions-announces-new-psa-500-automatic-powder-sieving-station>. Citado na página 48.

28 BREMEN S.; MEINERS, W. D. A. Selective laser melting - a manufacturing technology for the future. *Laser Journal - LTJ*, v. 2, p. 33–38, Abril 2002. Citado na página 50.

29 UDROIU, R. Powder bed additive manufacturing systems and its applications. *Academic Journal of Manufacturing Engineering*, v. 10, p. 122–129, Outubro 2012. Citado na página 51.

30 KRUTH J.P.; VANDENBROUCKE, B. V. V. J. M. P. Benchmarking of different sls/slm processes as rapid manufacturing techniques. *Int. Conf. Polymers und Moulds Innovations (PMI)*, Abril 2005. Citado na página 51.

31 BORTNIK E.; TRCKA, N. W. A. L. B. V. D. M.-F. J. B. J. F. W. R. J. Analyzing a χ model of a turntable system using spin, cadp and uppaal. *The Journal of Logic and Algebraic Programming*, v. 65, p. 51–104, 2005. Citado na página 51.

32 BOS V.; KLEIN, J. *AFormal specification and analysis of industrial systems*. Tese (Doutorado) — Eindhoven University of Technology, 2002. Citado na página 51.

33 HENDRIKS M; NIEUWELLAR, B. V. F. Model checker aided design of a controller for a water scanner. *International Journal on Software Tools for Technology Transfer - Springer Journals*, v. 8, p. 633–637, 2006. Citado na página 51.

34 ARDUINO. [Online; Acessado em 16-Fevereiro-2016]. Disponível em: <https://www.arduino.cc>. Citado na página 111.

35 UNIX Epoch. [Online; Acessado em 03-Fevereiro-2016]. Disponível em: <https://en.wikipedia.org/wiki/Unix_time>. Citado na página 112.

Apêndices

APÊNDICE A – Modelagem UPPAAL

A.1 Dispositivo de deposição de pó

Abaixo estão apresentados os *'template'* desenvolvidos para o subsistema do dispositivo de deposição de pó.

A.1.1 GUI

A figura 65 apresenta o GUI desenvolvido para o modelo UPPAAL do dispositivo de deposição de pó.

Figura 65 – *'Template GUI'* - Dispositivo de deposição de pó

A.1.2 Encoder

A figura 66 apresenta o encoder desenvolvido no UPPAAL do dispositivo de deposição de pó.

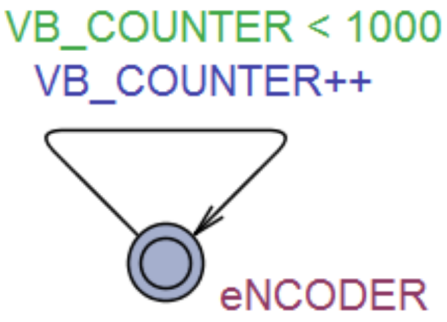

Figura 66 – *'Template Encoder'* - Dispositivo de deposição de pó

A.1.3 Sensores

A figura 67 apresenta os sensores desenvolvido para o modelo UPPAAL do dispositivo de deposição de pó.

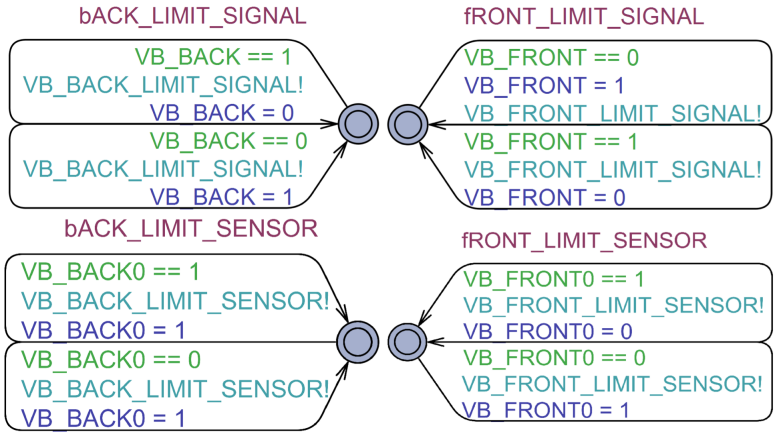

Figura 67 – *'Template Sensores'* - *Dispositivo de deposição de pó*

A.1.4 Motor

A figura 68 apresenta o motor do dispositivo de deposição de pó desenvolvido no UPPAAL.

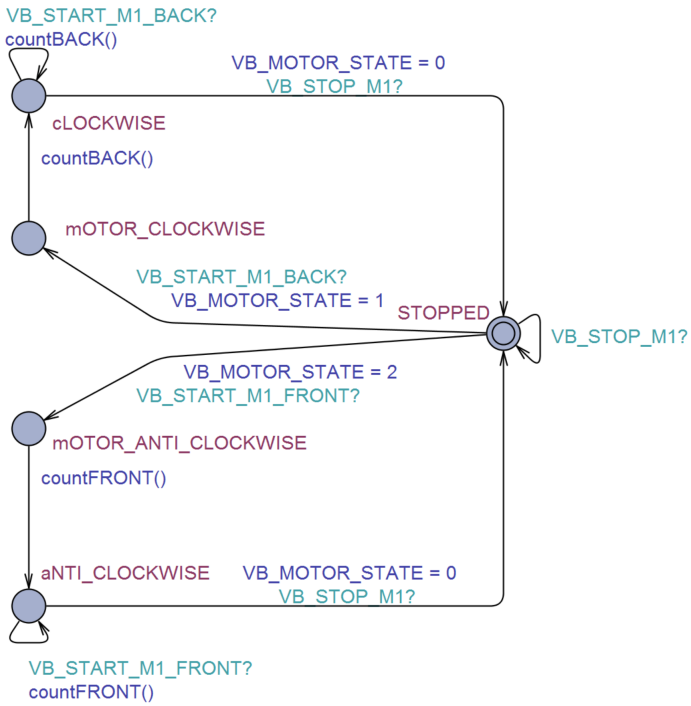

Figura 68 – *'Template Motor'* - Dispositivo de deposição de pó

A.1.5 Comando

A figura 69 apresenta o modelo para o controlador do dispositivo de deposição de pó desenvolvido no UPPAAL.

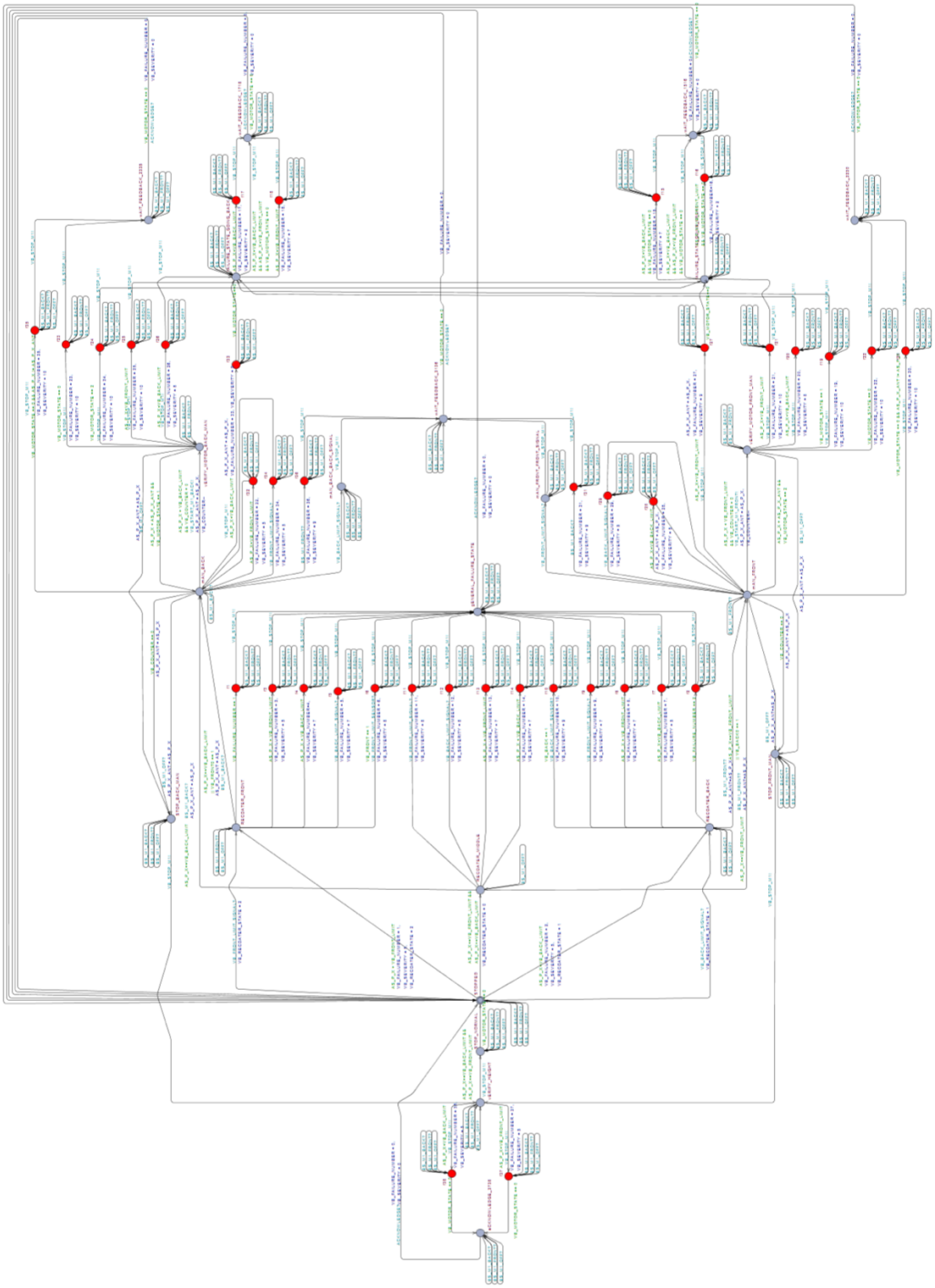

Figura 69 – *'Template Command'* - Dispositivo de deposição de pó

A.2 Sistema de carregamento 1

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de carregamento 1.

A.2.1 GUI

A figura 70 apresenta o GUI desenvolvido no UPPAAL do sistema de carregamento 1.

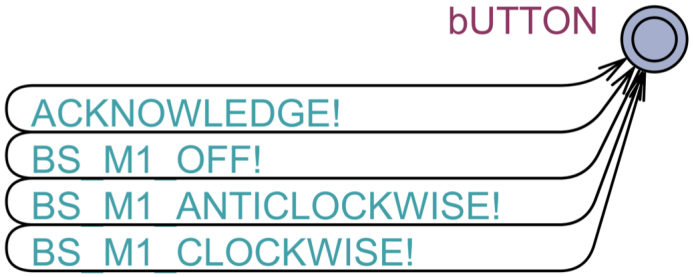

Figura 70 – *'Template GUI'* - Sistema de carregamento 1

A.2.2 Encoder

A figura 71 apresenta o encoder desenvolvido no UPPAAL do sistema de carregamento 1.

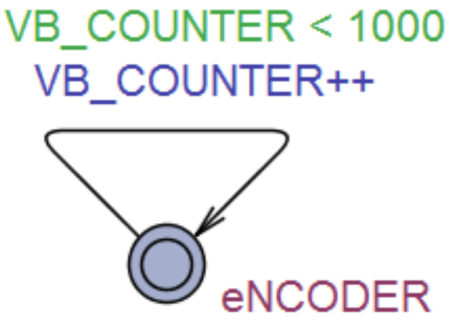

Figura 71 – *'Template Encoder'* - Sistema de carregamento 1

A.2.3 Motor

A figura 72 apresenta o motor do sistema de carregamento 1 desenvolvido no UPPAAL.

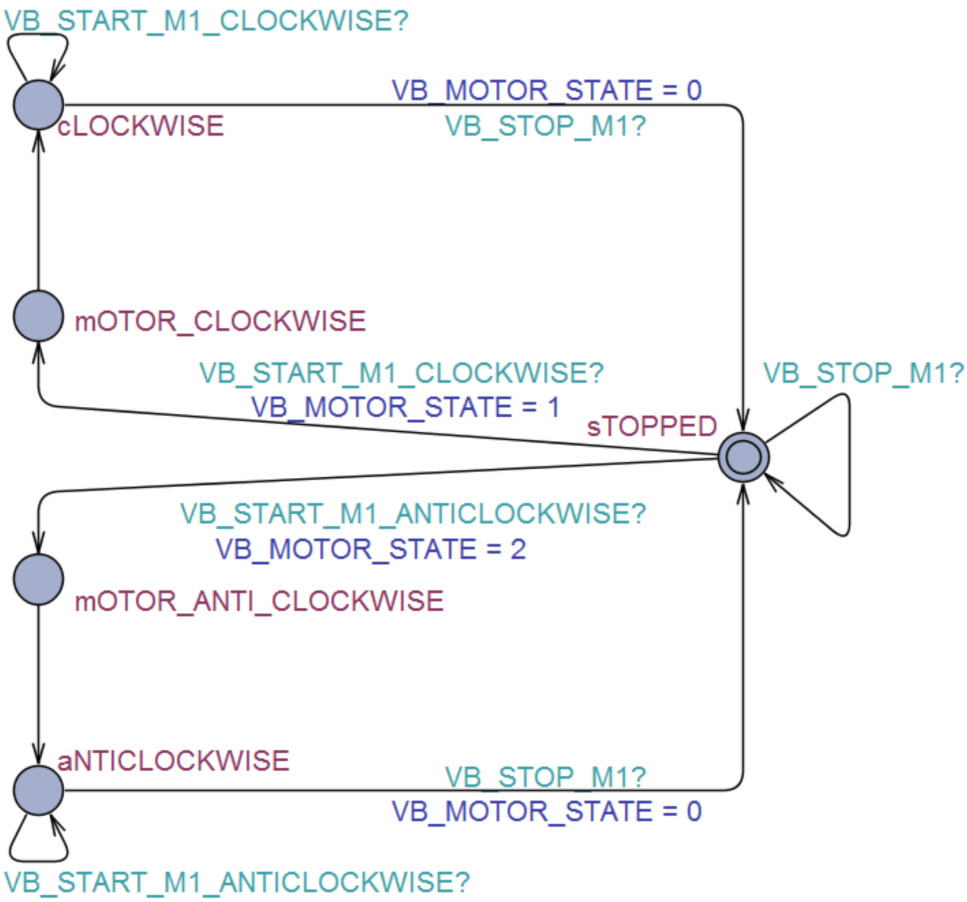

Figura 72 – *'Template Motor'* - Sistema de carregamento 1

A.2.4 Comando

A figura 73 apresenta o modelo para o controlador do sistema de carregamento 1 desenvolvido no UPPAAL.

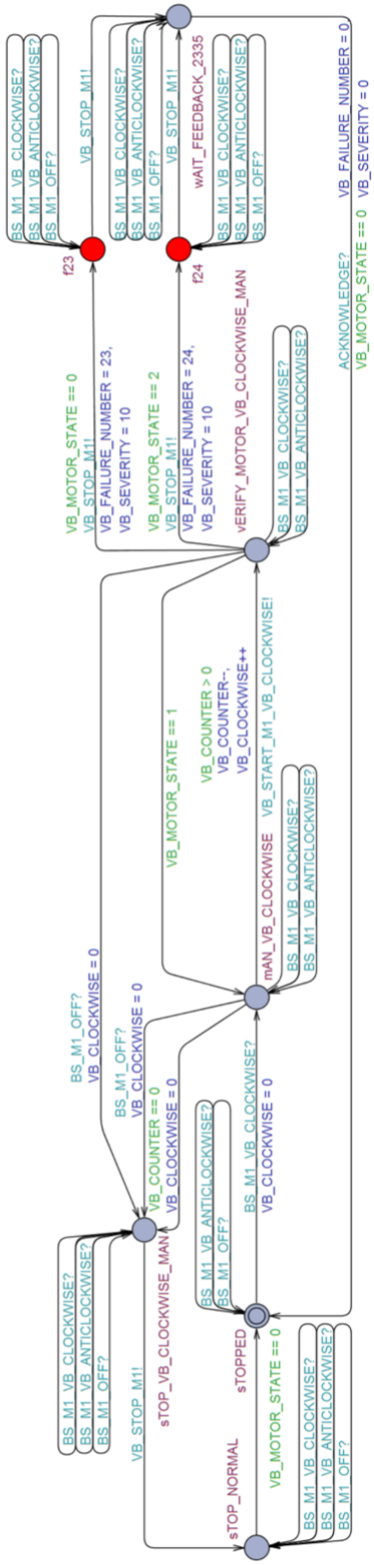

Figura 73 – *'Template Command'* - Sistema de carregamento 1

A.3 Sistema de carregamento 2

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de carregamento 2.

A.3.1 GUI

A figura 74 apresenta o GUI desenvolvido no UPPAAL do sistema de carregamento 2.

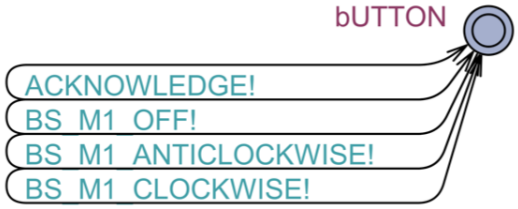

Figura 74 – *'Template GUI'* - Sistema de carregamento 2

A.3.2 Encoder

A figura 75 apresenta o encoder desenvolvido no UPPAAL do sistema de carregamento 2.

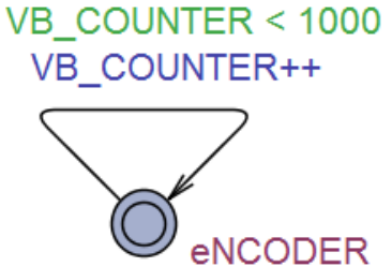

Figura 75 – *'Template Encoder'* - Sistema de carregamento 2

A.3.3 Motor

A figura 76 apresenta o motor do sistema de carregamento 2 desenvolvido no UPPAAL.

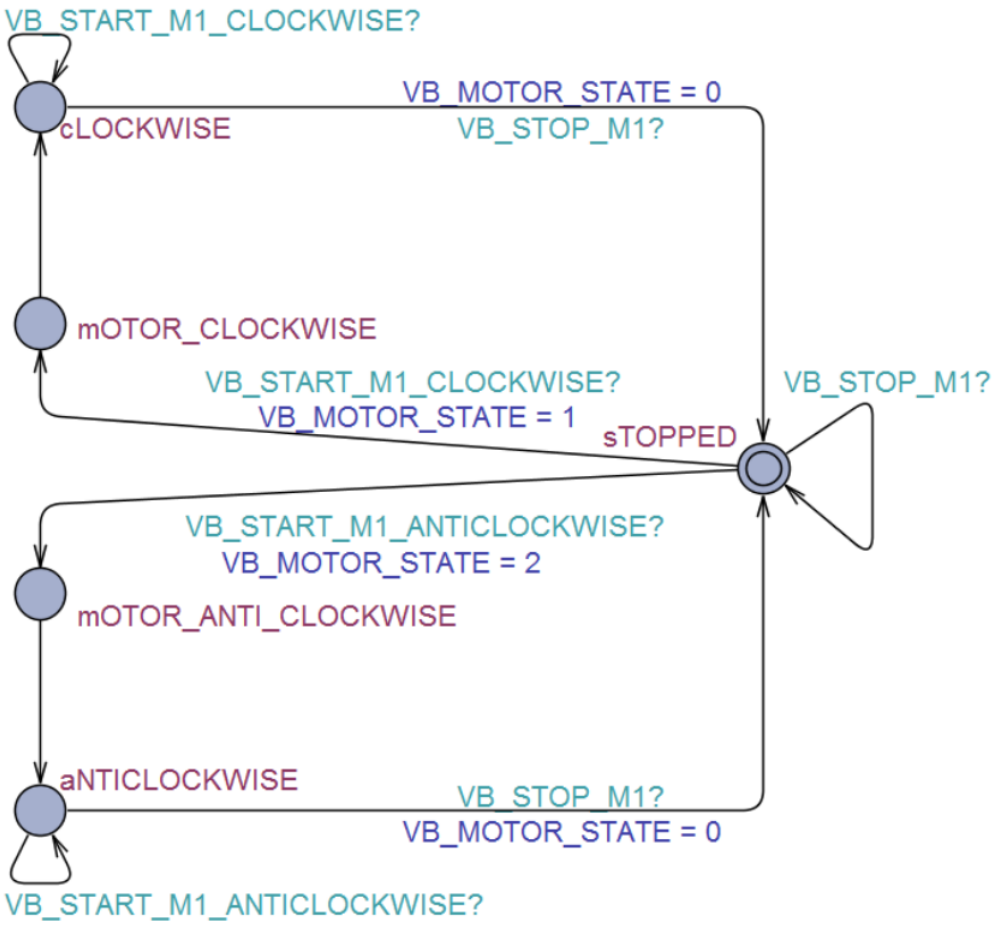

Figura 76 – *'Template Motor'* - Sistema de carregamento 2

A.3.4 Comando

A figura 77 apresenta o modelo para o controlador do sistema de carregamento 2 desenvolvido no UPPAAL.

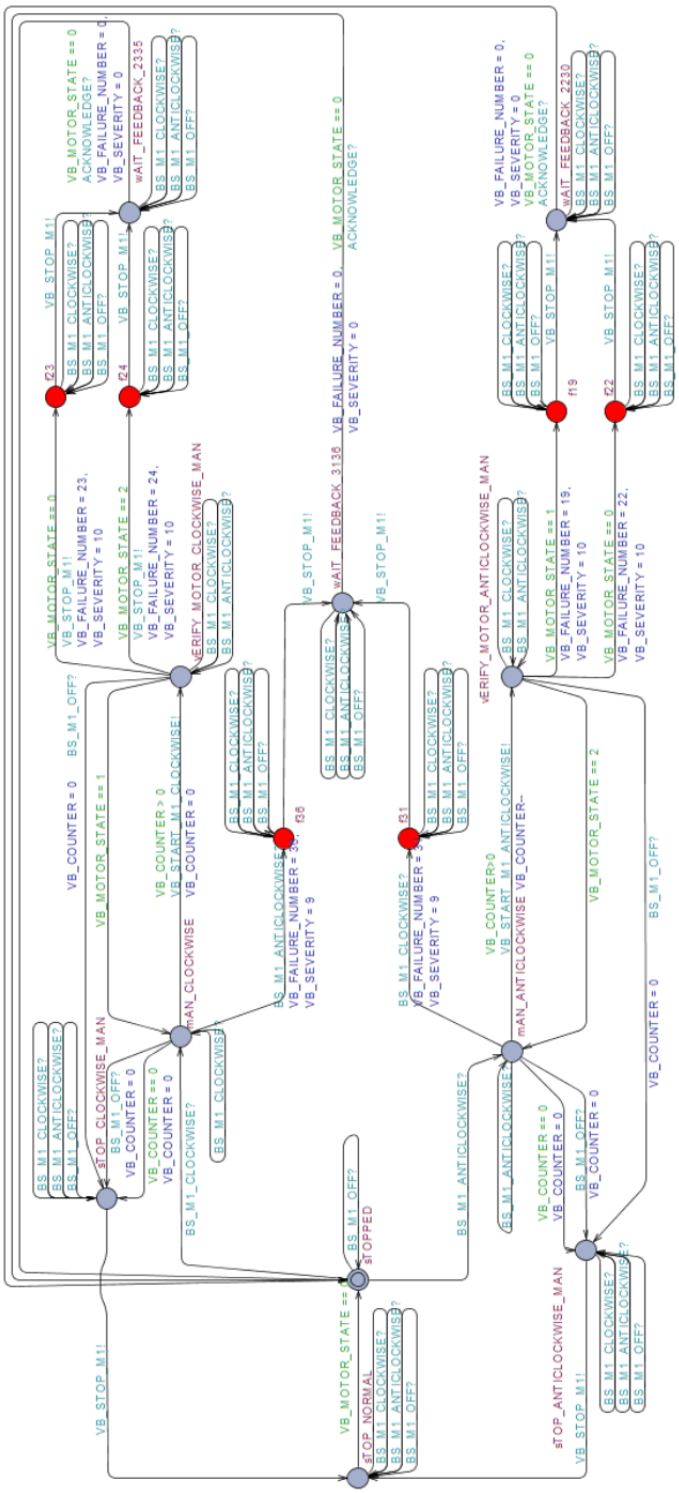

Figura 77 – *'Template Command'* - Sistema de carregamento 2

A.4 Sistema de carregamento 3

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de carregamento 3.

A.4.1 GUI

A figura 78 apresenta o GUI desenvolvido no UPPAAL do sistema de carregamento 3.

Figura 78 – *'Template GUI'* - Sistema de carregamento 3

A.4.2 Encoder

A figura 79 apresenta o encoder desenvolvido no UPPAAL do sistema de carregamento 3.

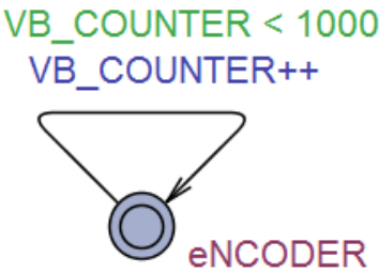

Figura 79 – *'Template Encoder'* - Sistema de carregamento 3

A.4.3 Motor

A figura 80 apresenta o motor do sistema de carregamento 3 desenvolvido no UPPAAL.

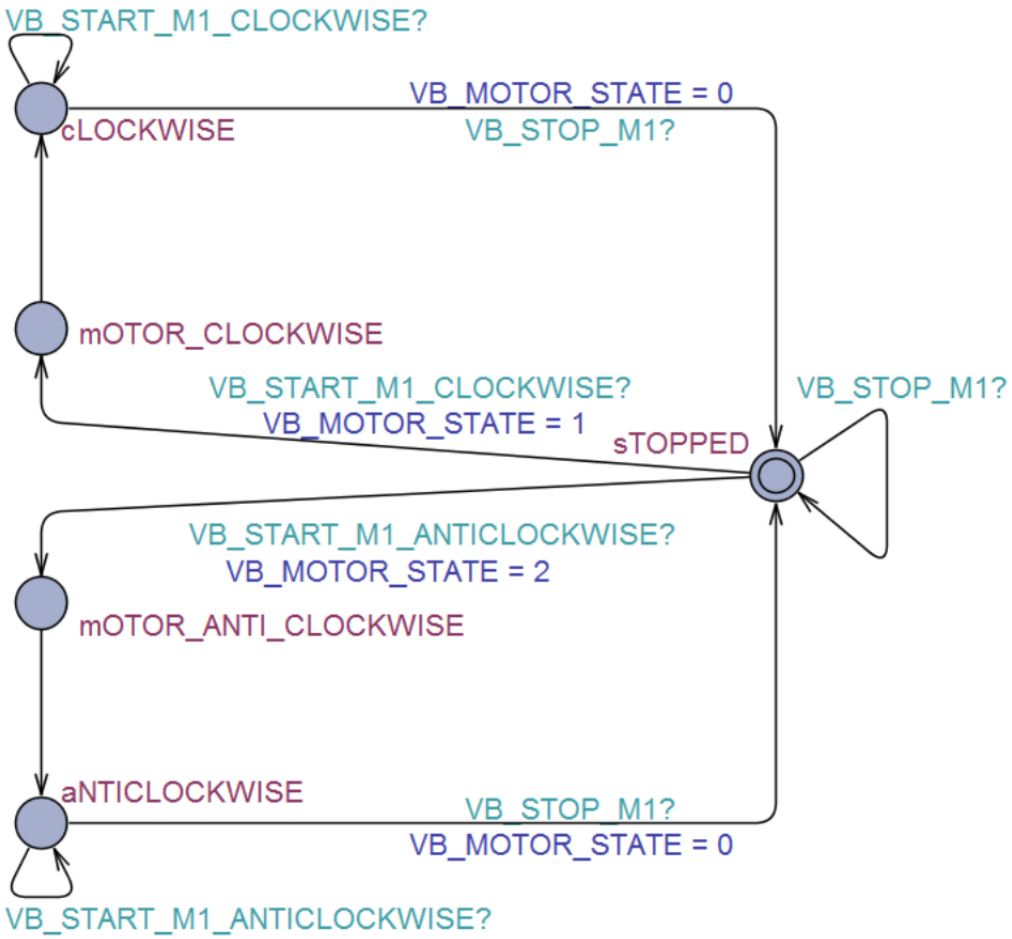

Figura 80 – *'Template Motor'* - Sistema de carregamento 3

A.4.4 Comando

A figura 81 apresenta o modelo para o controlador do sistema de carregamento 3 desenvolvido no UPPAAL.

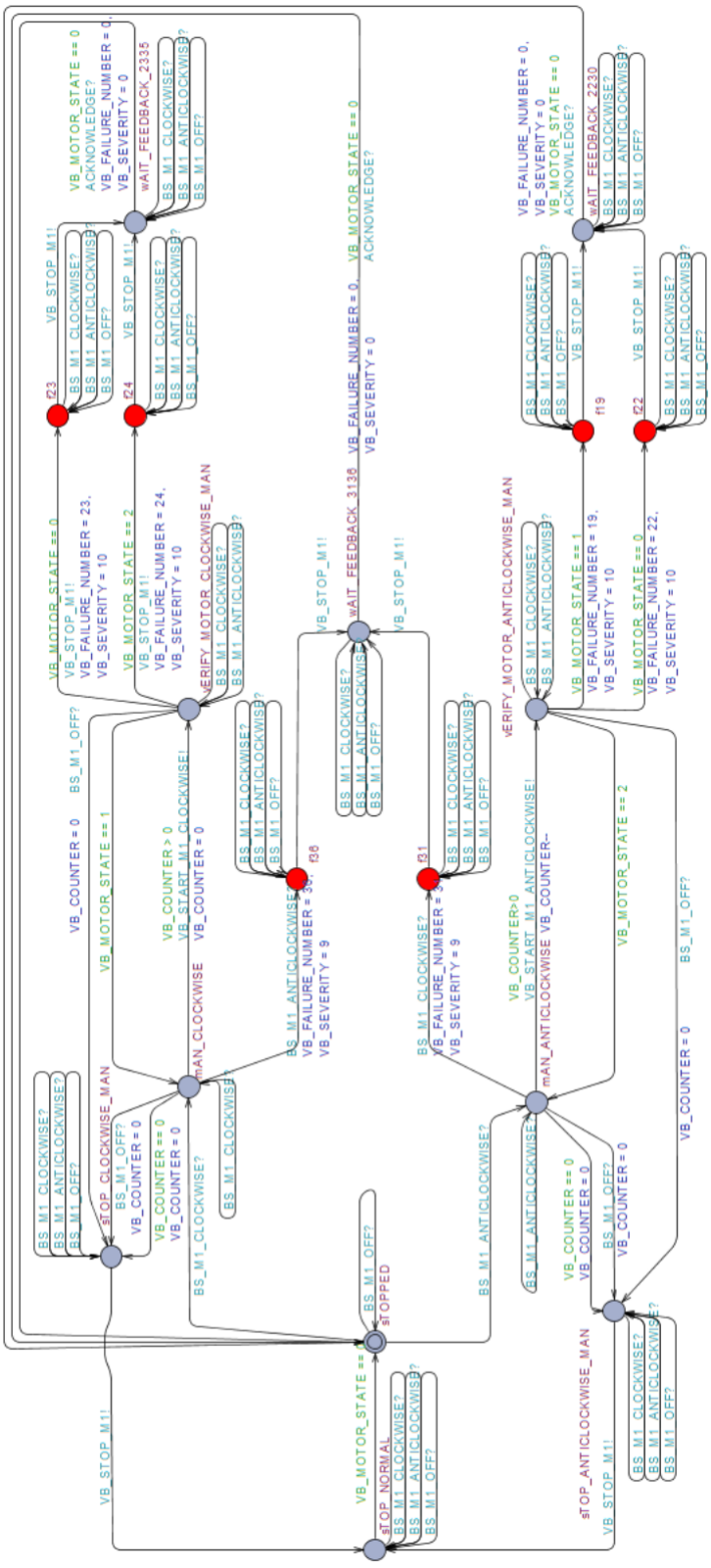

Figura 81 – *'Template Command'* - Sistema de carregamento 3

A.5 Laser

Abaixo estão apresentados os *'template'* desenvolvidos para o subsistema do laser.

A.5.1 GUI

A figura 82 apresenta o GUI desenvolvido no UPPAAL do laser.

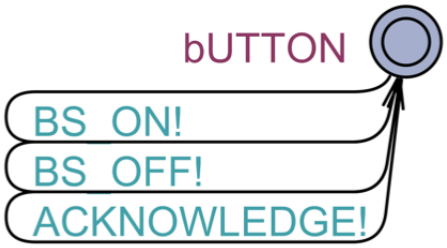

Figura 82 – *'Template GUI'* - Laser

A.5.2 Componente do laser

A figura 83 apresenta o componente laser desenvolvido no UPPAAL.

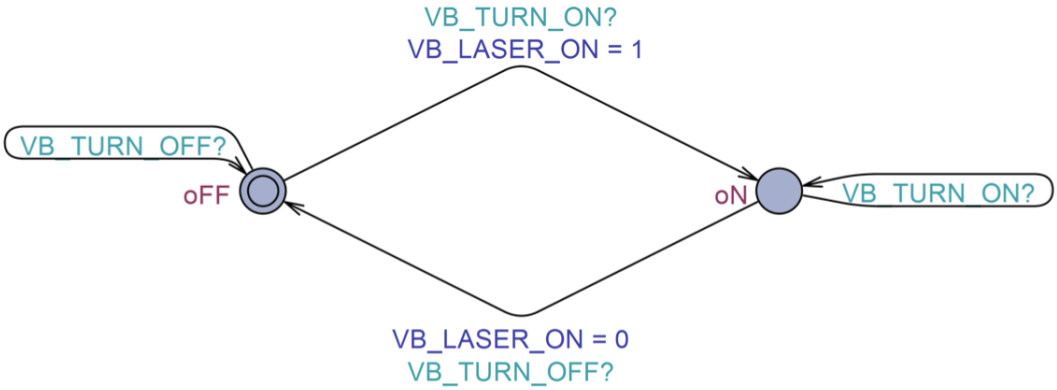

Figura 83 – *'Template Laser'* - Laser

A.5.3 Comando

A figura 84 apresenta o modelo para o controlador do laser desenvolvido no UPPAAL.

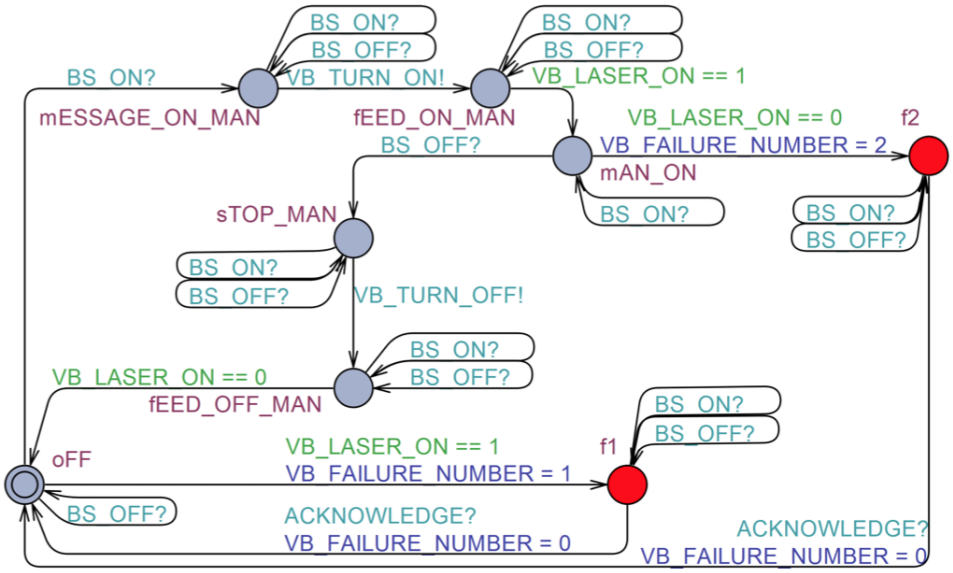

Figura 84 – *'Template Command'* - Laser

A.6 Sistema de aquecimento

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de aquecimento.

A.6.1 GUI

A figura 85 apresenta o GUI desenvolvido no UPPAAL do sistema de aquecimento.

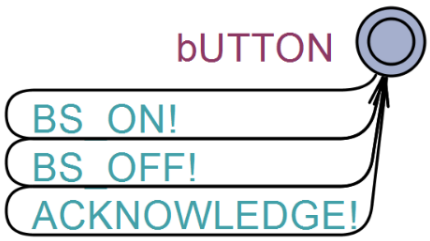

Figura 85 – *'Template GUI'* - Sistema de aquecimento

A.6.2 Componente do sistema de aquecimento

A figura 86 apresenta o componente do sistema de aquecimento desenvolvido no UPPAAL.

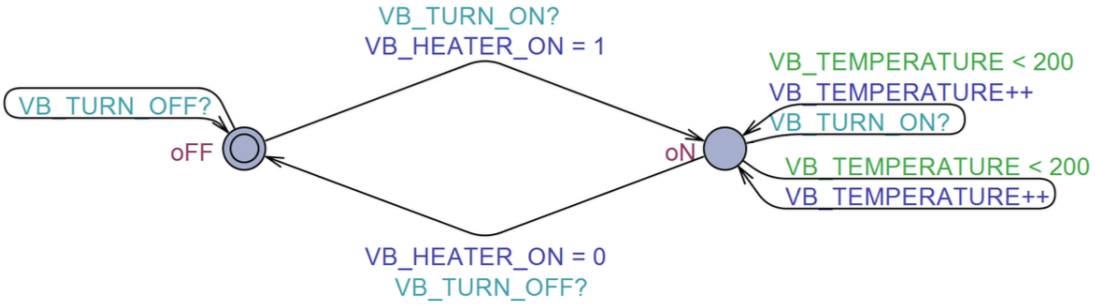

Figura 86 – *'Template Heater'* - Sistema de aquecimento

A.6.3 Comando

A figura 87 apresenta o modelo para o controlador do sistema de aquecimento desenvolvido no UPPAAL.

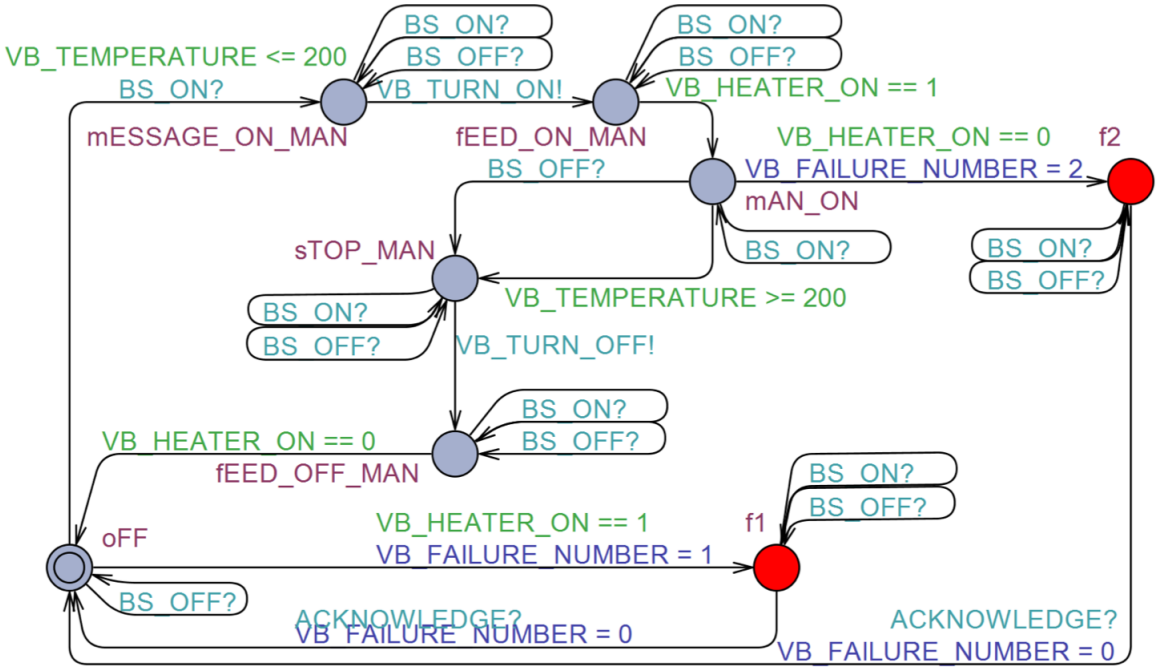

Figura 87 – *'Template Command'* - Sistema de aquecimento

A.7 Sistema de refrigeração

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de refrigeração.

A.7.1 GUI

A figura 88 apresenta o GUI desenvolvido no UPPAAL do sistema de refrigeração.

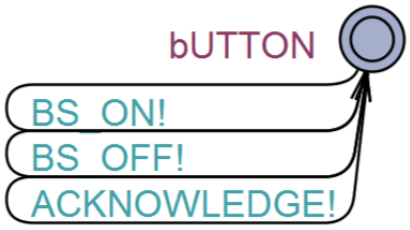

Figura 88 – *'Template GUI'* - Sistema de refrigeração

A.7.2 Componente do sistema de refrigeração

A figura 89 apresenta o componente do sistema de refrigeração desenvolvido no UPPAAL.

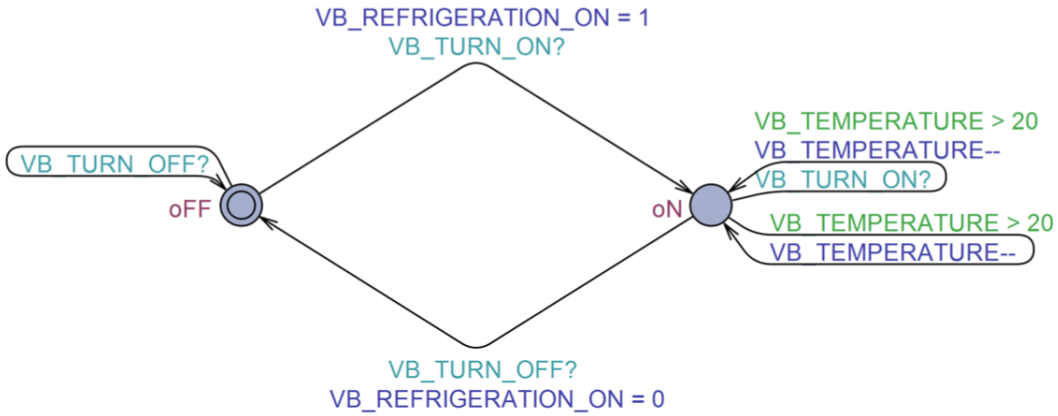

Figura 89 – *'Template Refrigeration'* - Sistema de refrigeração

A.7.3 Comando

A figura 90 apresenta o modelo para o controlador do sistema de refrigeração desenvolvido no UPPAAL.

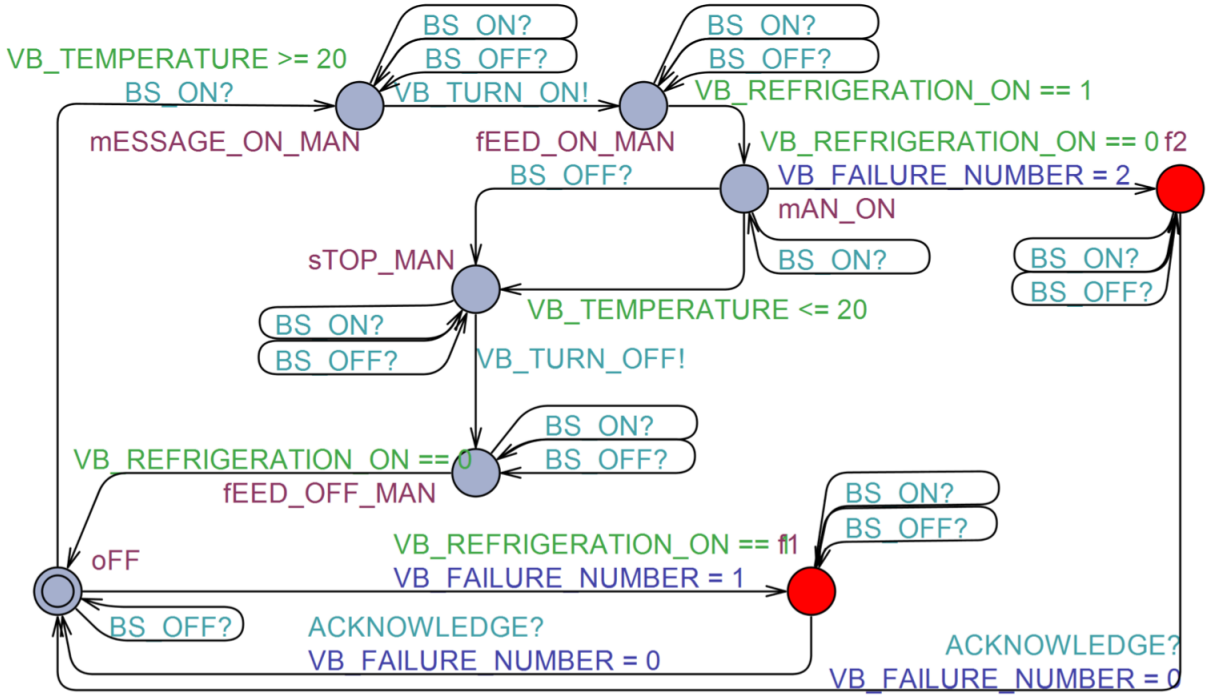

Figura 90 – *'Template Command'* - Sistema de refrigeração

A.8 Sistema de filtragem

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de filtragem.

A.8.1 GUI

A figura 91 apresenta o GUI desenvolvido no UPPAAL do sistema de filtragem.

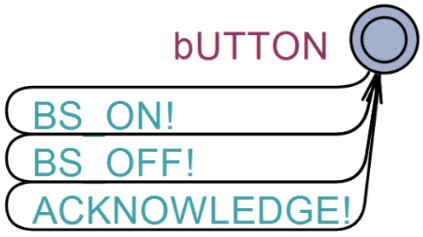

Figura 91 – *'Template GUI'* - Sistema de filtragem

A.8.2 Componente do sistema de filtragem

A figura 92 apresenta o componente do sistema de filtragem desenvolvido no UPPAAL.

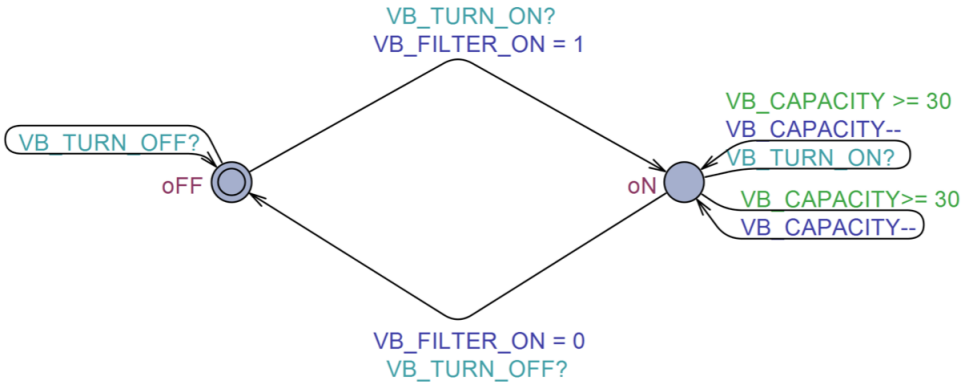

Figura 92 – *'Template Filter'* - Sistema de filtragem

A.8.3 Comando

A figura 93 apresenta o modelo para o controlador do sistema de filtragem desenvolvido no UPPAAL.

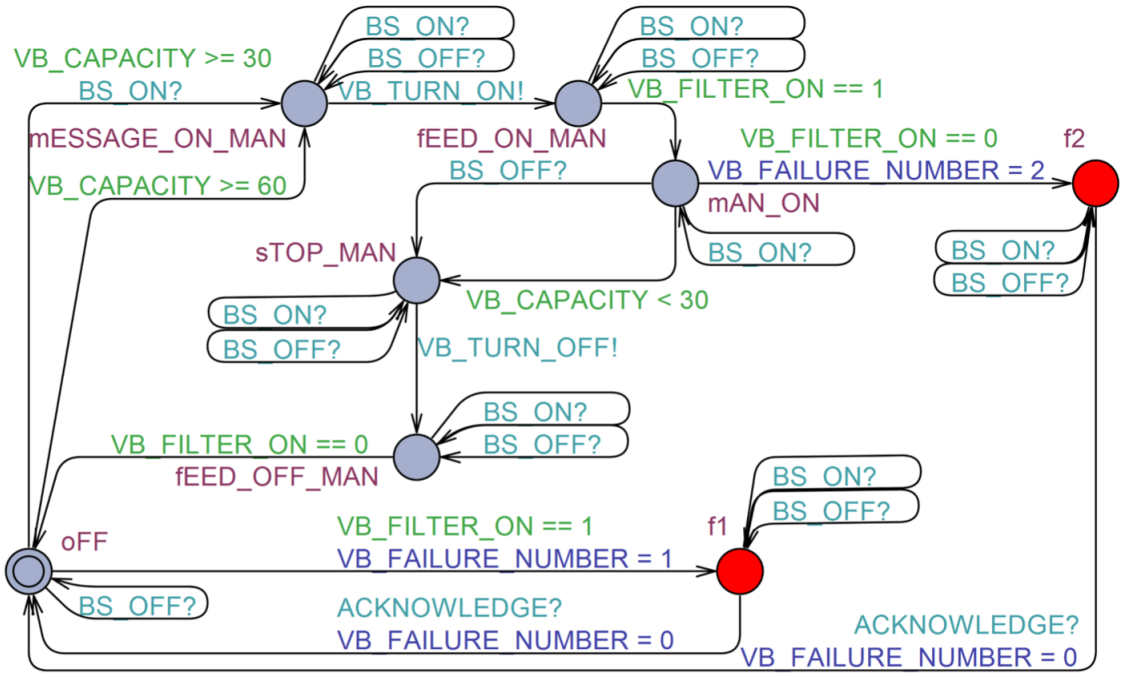

Figura 93 – *'Template Command'* - Sistema de filtragem

A.9 Sistema de vácuo

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de vácuo.

A.9.1 GUI

A figura 94 apresenta o GUI desenvolvido no UPPAAL do sistema de vácuo.

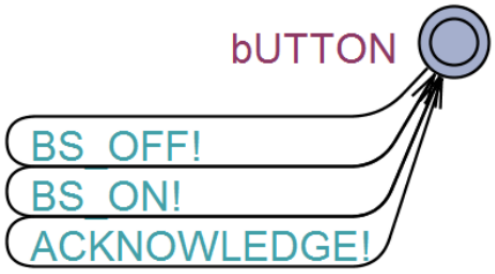

Figura 94 – *'Template GUI'* - Sistema de vácuo

A.9.2 Motor do sistema de vácuo

A figura 95 apresenta o motor do sistema de vácuo desenvolvido no UPPAAL.

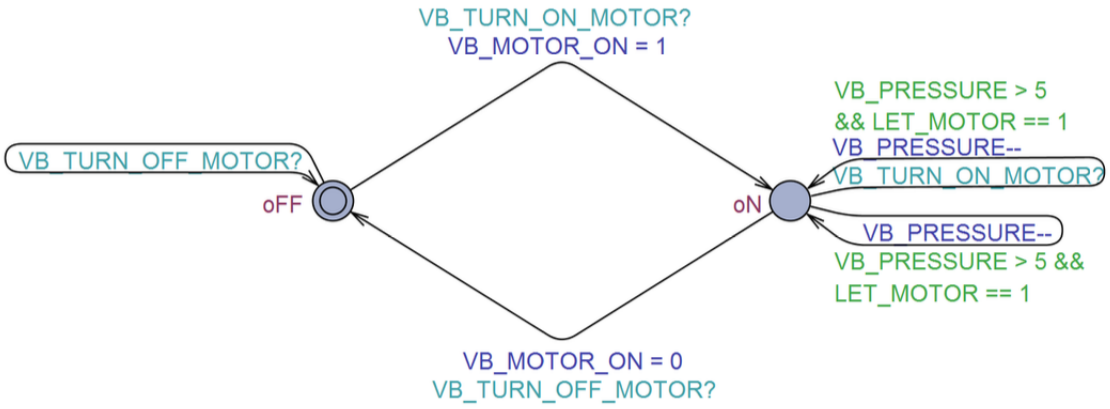

Figura 95 – *'Template Motor'* - Sistema de vácuo

A.9.3 Válvula do sistema de vácuo

A figura 96 apresenta oa válvula do sistema de vácuo desenvolvido no UPPAAL.

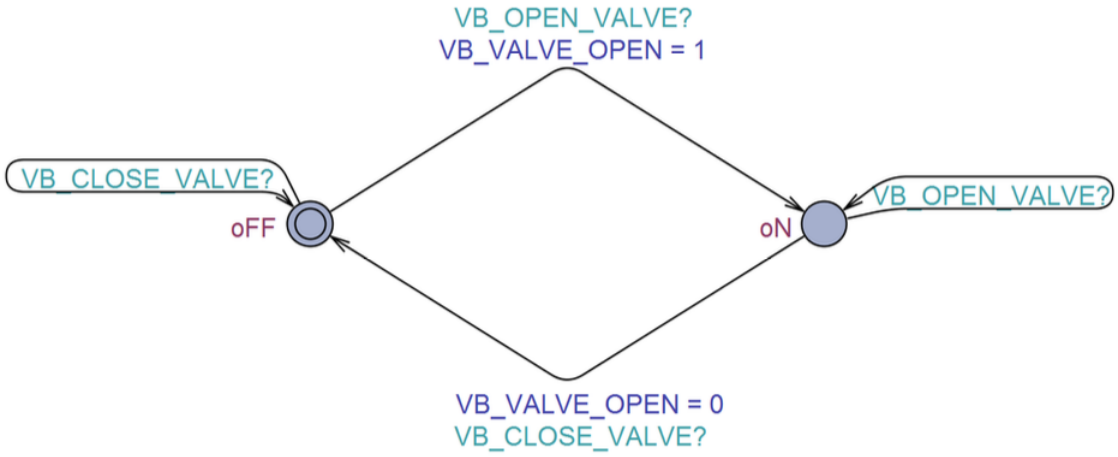

Figura 96 – *'Template Valve'* - Sistema de vácuo

A.9.4 Comando

A figura 97 apresenta o modelo para o controlador do sistema de vácuo desenvolvido no UPPAAL.

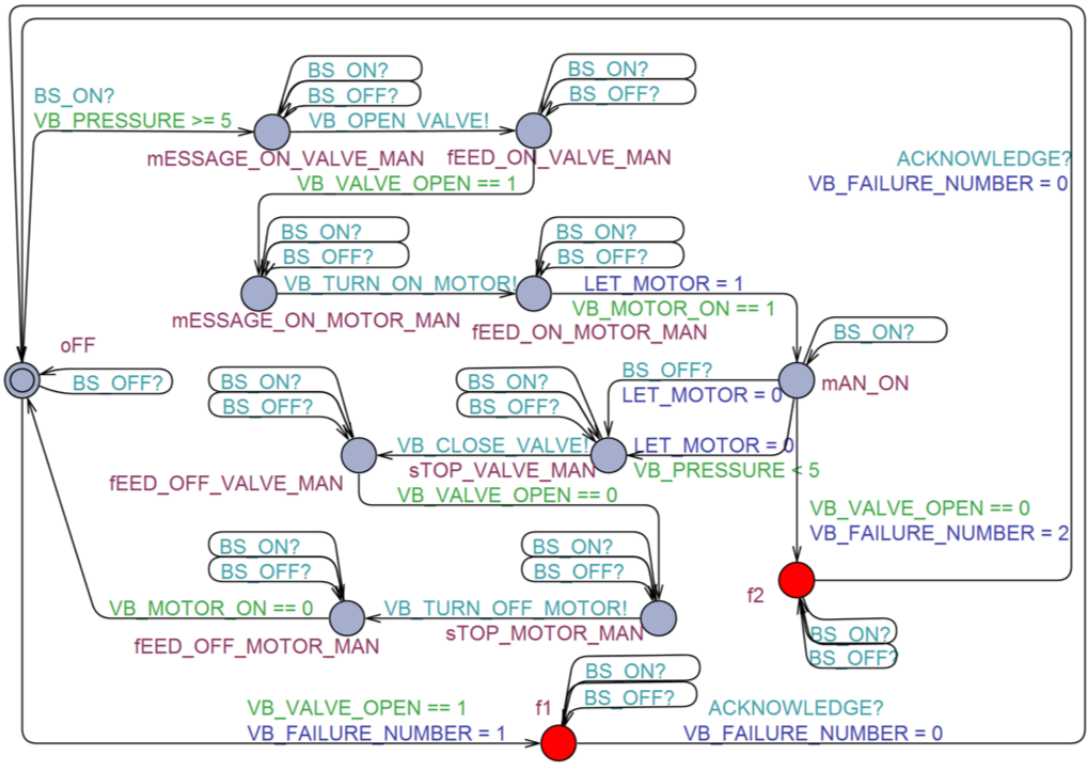

Figura 97 – *'Template Command'* - Sistema de vácuo

A.10 Sistema de proteção a gás

Abaixo estão apresentados os *'template'* desenvolvidos para o sistema de proteção a gás.

A.10.1 GUI

A figura 98 apresenta o GUI desenvolvido no UPPAAL do sistema de proteção a gás.

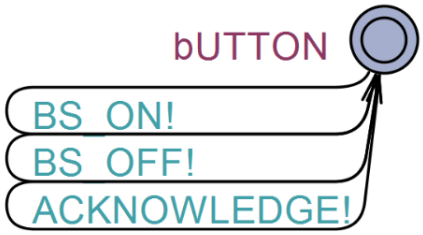

Figura 98 – *'Template GUI'* - Sistema de proteção a gás

A.10.2 Componente do sistema de proteção a gás

A figura 99 apresenta o componente do sistema de proteção a gás desenvolvido no UPPAAL.

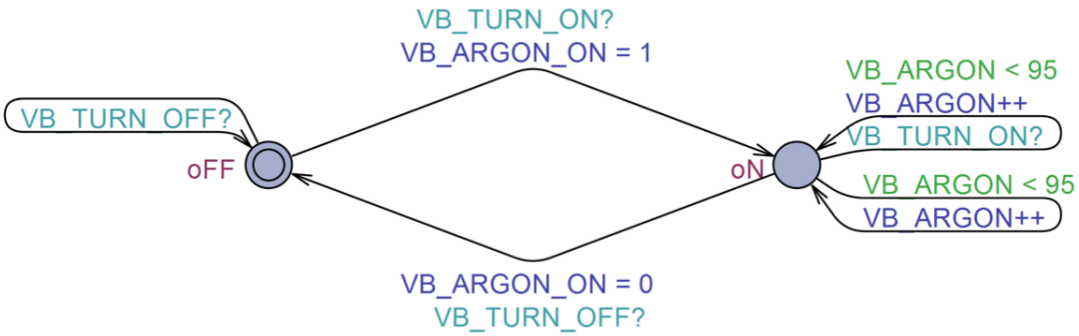

Figura 99 – *'Template Argon'* - Sistema de proteção a gás

A.10.3 Comando

A figura 100 apresenta o modelo para o controlador do sistema de proteção a gás desenvolvido no UPPAAL.

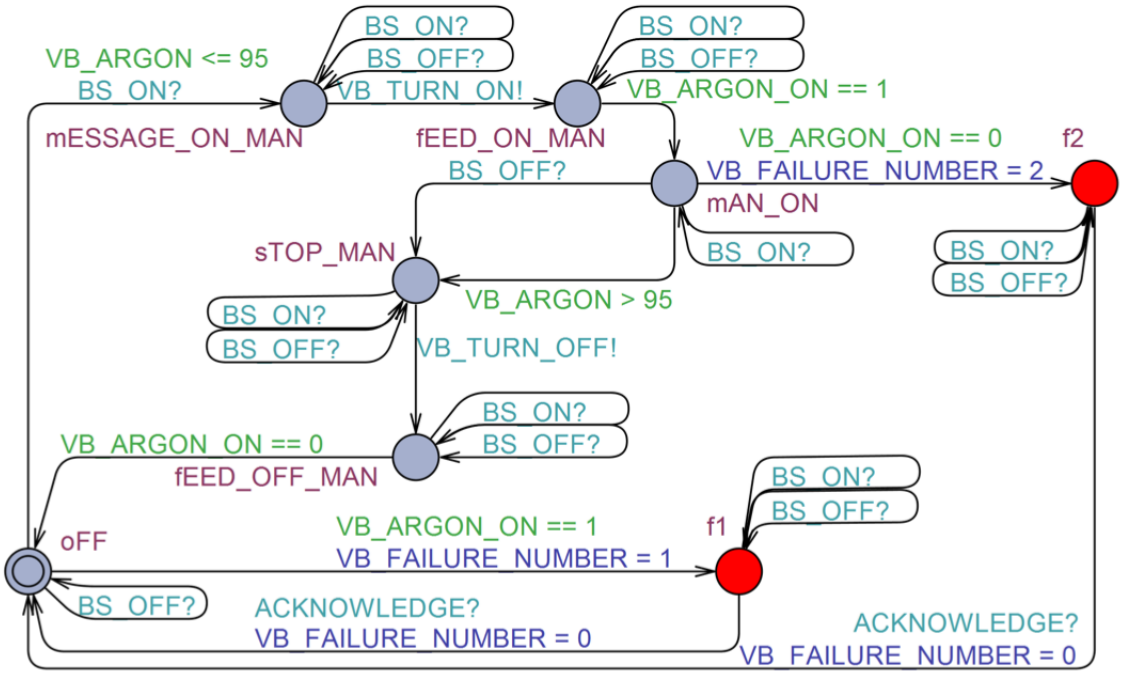

Figura 100 – *'Template Command'* - Sistema de proteção a gás

APÊNDICE B – Lista de Propriedades verificadas nos modelos

Abaixo estão apresentadas as propriedades de cada um dos subsistemas, contendo a escrita da propriedade no UPPAAL à esquerda, o resultado da propriedade ao centro, com 'verde' caso seja verdadeiro e 'vermelho' caso seja falso, e a explicação da propriedade à direita, em inglês.

B.1 Plataforma

Tabela 25 – Propriedades da Plataforma - Parte 1 de 3

	Plataform	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.sTOPPED		Verifies if the state sTOPPED at CM is reached
3	A[] CM.sTOPPED imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state sTOPPED is always higher than the VB_UPPER_LIMIT
4	A[] CM.sTOPPED imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state sTOPPED is always lower than the VB_LOWER_LIMIT
5	E<> CM.sTOPPED && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOPPED
6	E<> CM.sTOPPED && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOPPED
7	A[] CM.sTOPPED imply FE.sTOPPED		Verifies if the moto is always stopped at the state sTOPPED
8	E<> CM.vERIFY_HEIGHT		Verifies if the state vERIFY_HEIGHT at CM is reached
9	A[] CM.vERIFY_HEIGHT imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state vERIFY_HEIGHT is always higher than the VB_UPPER_LIMIT
10	A[] CM.vERIFY_HEIGHT imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state vERIFY_HEIGHT is always lower than the VB_LOWER_LIMIT
11	E<> CM.vERIFY_HEIGHT && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_HEIGHT
12	E<> CM.vERIFY_HEIGHT && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_HEIGHT
13	A[] CM.vERIFY_HEIGHT imply FE.sTOPPED		Verifies if the motor is always stopped at the state vERIFY_HEIGHT
14	E<> CM.sTOP_NORMAL		Verifies if the state sTOP_NORMAL at CM is reached
15	A[] CM.sTOP_NORMAL imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state sTOP_NORMAL is always higher than the VB_UPPER_LIMIT
16	A[] CM.sTOP_NORMAL imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state sTOP_NORMAL is always lower than the VB_LOWER_LIMIT
17	E<> CM.sTOP_NORMAL && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_NORMAL
18	E<> CM.sTOP_NORMAL && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_NORMAL
19	A[] CM.sTOP_NORMAL imply FE.sTOPPED		Verifies if the motor is always stopped at the state sTOP_NORMAL
20	E<> CM.pLATFORM_MIDDLE		Verifies if the state pLATFORM_MIDDLE at CM is reached
21	A[] CM.pLATFORM_MIDDLE imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state pLATFORM_MIDDLE is always higher than the VB_UPPER_LIMIT
22	A[] CM.pLATFORM_MIDDLE imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state pLATFORM_MIDDLE is always lower than the VB_LOWER_LIMIT
23	E<> CM.pLATFORM_MIDDLE && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state pLATFORM_MIDDLE
24	E<> CM.pLATFORM_MIDDLE && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state pLATFORM_MIDDLE
25	A[] CM.pLATFORM_MIDDLE imply FE.sTOPPED		Verifies if the motor is always stopped at the state pLATFORM_MIDDLE
26	E<> CM.pLATFORM_BOTTOM		Verifies if the state pLATFORM_BOTTOM at CM is reached
27	A[] CM.pLATFORM_BOTTOM imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state pLATFORM_BOTTOM is always higher than the VB_UPPER_LIMIT
28	A[] CM.pLATFORM_BOTTOM imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state pLATFORM_BOTTOM is always lower than the VB_LOWER_LIMIT
29	E<> CM.pLATFORM_BOTTOM && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state pLATFORM_BOTTOM
30	E<> CM.pLATFORM_BOTTOM && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state pLATFORM_BOTTOM
31	A[] CM.pLATFORM_BOTTOM imply FE.sTOPPED		Verifies if the motor is always stopped at the state pLATFORM_BOTTOM
32	E<> CM.pLATFORM_TOP		Verifies if the state pLATFORM_TOP at CM is reached
33	A[] CM.pLATFORM_TOP imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state pLATFORM_TOP is always higher than the VB_UPPER_LIMIT
34	A[] CM.pLATFORM_TOP imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state pLATFORM_TOP is always lower than the VB_LOWER_LIMIT
35	E<> CM.pLATFORM_TOP && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state pLATFORM_TOP
36	E<> CM.pLATFORM_TOP && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state pLATFORM_TOP
37	A[] CM.pLATFORM_TOP imply FE.sTOPPED		Verifies if the motor is always stopped at the state pLATFORM_TOP
38	E<> CM.mAN_UP		Verifies if the state mAN_UP at CM is reached
39	A[] CM.mAN_UP imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state mAN_UP is always higher than the VB_UPPER_LIMIT
40	A[] CM.mAN_UP imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state mAN_UP is always lower than the VB_LOWER_LIMIT
41	E<> CM.mAN_UP && FE.sTOPPED		Verifies if the motor can be stopped at the state mAN_UP
42	E<> CM.mAN_UP && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_UP
43	E<> CM.mAN_UP && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_UP
44	E<> CM.vERIFY_MOTOR_UP_MAN		Verifies if the state vERIFY_MOTOR_UP_MAN at CM is reached
45	A[] CM.vERIFY_MOTOR_UP_MAN imply VB_UPPER_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state vERIFY_MOTOR_UP_MAN is always higher than the VB_UPPER_LIMIT
46	A[] CM.vERIFY_MOTOR_UP_MAN imply AS_P_X<=VB_LOWER_LIMIT		Verifies if the AS_P_X at the state vERIFY_MOTOR_UP_MAN is always lower than the VB_LOWER_LIMIT
47	E<> CM.vERIFY_MOTOR_UP_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_UP_MAN
48	E<> CM.vERIFY_MOTOR_UP_MAN && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_UP_MAN
49	E<> CM.vERIFY_MOTOR_UP_MAN && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_UP_MAN
50	E<> CM.sTOP_UP_MAN		Verifies if the state sTOP_UP_MAN at CM is reached

Tabela 26 – Propriedades da Plataforma - Parte 2 de 3

51	A[] CM.sTOP_UP_MAN imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state sTOP_UP_MAN is always higher than the VB_UPPER_LIMIT
52	A[] CM.sTOP_UP_MAN imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state sTOP_UP_MAN is always lower than the VB_LOWER_LIMIT
53	E<> CM.sTOP_UP_MAN && FE.sTOPPED	Verifies if the motor can be stopped at the state sTOP_UP_MAN
54	E<> CM.sTOP_UP_MAN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state sTOP_UP_MAN
55	E<> CM.sTOP_UP_MAN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state sTOP_UP_MAN
56	E<> CM.mAN_DOWN	Verifies if the state mAN_DOWN at CM is reached
57	A[] CM.mAN_DOWN imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state mAN_DOWN is always higher than the VB_UPPER_LIMIT
58	A[] CM.mAN_DOWN imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state mAN_DOWN is always lower than the VB_LOWER_LIMIT
59	E<> CM.mAN_DOWN && FE.sTOPPED	Verifies if the motor can be stopped at the state mAN_DOWN
60	E<> CM.mAN_DOWN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state mAN_DOWN
61	E<> CM.mAN_DOWN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state mAN_DOWN
62	E<> CM.vERIFY_MOTOR_DOWN_MAN	Verifies if the state vERIFY_MOTOR_DOWN_MAN at CM is reached
63	A[] CM.vERIFY_MOTOR_DOWN_MAN imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state vERIFY_MOTOR_DOWN_MAN is always higher than the VB_UPPER_LIMIT
64	A[] CM.vERIFY_MOTOR_DOWN_MAN imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state vERIFY_MOTOR_DOWN_MAN is always lower than the VB_LOWER_LIMIT
65	E<> CM.vERIFY_MOTOR_DOWN_MAN && FE.sTOPPED	Verifies if the motor can be stopped at the state vERIFY_MOTOR_DOWN_MAN
66	E<> CM.vERIFY_MOTOR_DOWN_MAN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_DOWN_MAN
67	E<> CM.vERIFY_MOTOR_DOWN_MAN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_DOWN_MAN
68	E<> CM.sTOP_DOWN_MAN	Verifies if the state sTOP_DOWN_MAN at CM is reached
69	E<> CM.sTOP_DOWN_MAN	Verifies if the AS_P_X at the state sTOP_DOWN_MAN is always higher than the VB_UPPER_LIMIT
70	A[] CM.sTOP_DOWN_MAN imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state sTOP_DOWN_MAN is always lower than the VB_LOWER_LIMIT
71	E<> CM.sTOP_DOWN_MAN && FE.sTOPPED	Verifies if the motor can be stopped at the state sTOP_DOWN_MAN
72	E<> CM.sTOP_DOWN_MAN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state sTOP_DOWN_MAN
73	E<> CM.sTOP_DOWN_MAN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state sTOP_DOWN_MAN
74	E<> CM.mAN_UP_UPPER_SIGNAL	Verifies if the state mAN_UP_UPPER_SIGNAL at CM is reached
75	A[] CM.mAN_UP_UPPER_SIGNAL imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state mAN_UP_UPPER_SIGNAL is always higher than the VB_UPPER_LIMIT
76	A[] CM.mAN_UP_UPPER_SIGNAL imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state mAN_UP_UPPER_SIGNAL is always lower than the VB_LOWER_LIMIT
77	E<> CM.mAN_UP_UPPER_SIGNAL && FE.sTOPPED	Verifies if the motor can be stopped at the state mAN_UP_UPPER_SIGNAL
78	E<> CM.mAN_UP_UPPER_SIGNAL && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state mAN_UP_UPPER_SIGNAL
79	E<> CM.mAN_UP_UPPER_SIGNAL && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state mAN_UP_UPPER_SIGNAL
80	E<> CM.fAILURE_STATE_GOING_UP	Verifies if the state fAILURE_STATE_GOING_UP at CM is reached
81	A[] CM.fAILURE_STATE_GOING_UP imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_UP is always higher than the VB_UPPER_LIMIT
82	A[] CM.fAILURE_STATE_GOING_UP imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_UP is always lower than the VB_LOWER_LIMIT
83	E<> CM.fAILURE_STATE_GOING_UP && FE.sTOPPED	Verifies if the motor can be stopped at the state fAILURE_STATE_GOING_UP
84	E<> CM.fAILURE_STATE_GOING_UP && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state fAILURE_STATE_GOING_UP
85	E<> CM.fAILURE_STATE_GOING_UP && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state fAILURE_STATE_GOING_UP
86	E<> CM.wAIT_FEEDBACK_2335	Verifies if the state wAIT_FEEDBACK_2335 at CM is reached
87	A[] CM.wAIT_FEEDBACK_2335 imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2335 is always higher than the VB_UPPER_LIMIT
88	A[] CM.wAIT_FEEDBACK_2335 imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2335 is always lower than the VB_LOWER_LIMIT
89	E<> CM.wAIT_FEEDBACK_2335 && FE.sTOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2335
90	E<> CM.wAIT_FEEDBACK_2335 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2335
91	E<> CM.wAIT_FEEDBACK_2335 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2335
92	E<> CM.wAIT_FEEDBACK_1718	Verifies if the state wAIT_FEEDBACK_1718 at CM is reached
93	A[] CM.wAIT_FEEDBACK_1718 imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1718 is always higher than the VB_UPPER_LIMIT
94	A[] CM.wAIT_FEEDBACK_1718 imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1718 is always lower than the VB_LOWER_LIMIT
95	E<> CM.wAIT_FEEDBACK_1718 && FE.sTOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_1718
96	E<> CM.wAIT_FEEDBACK_1718 && FE.sTOPPED	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_1718
97	E<> CM.wAIT_FEEDBACK_1718 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_1718
98	E<> CM.mAN_DOWN_LOWER_SIGNAL	Verifies if the state mAN_DOWN_LOWER_SIGNAL at CM is reached
99	A[] CM.mAN_DOWN_LOWER_SIGNAL imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state mAN_DOWN_LOWER_SIGNAL is always higher than the VB_UPPER_LIMIT
100	A[] CM.mAN_DOWN_LOWER_SIGNAL imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state mAN_DOWN_LOWER_SIGNAL is always lower than the VB_LOWER_LIMIT
101	E<> CM.mAN_DOWN_LOWER_SIGNAL && FE.sTOPPED	Verifies if the motor can be stopped at the state mAN_DOWN_LOWER_SIGNAL
102	E<> CM.mAN_DOWN_LOWER_SIGNAL && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state mAN_DOWN_LOWER_SIGNAL
103	E<> CM.mAN_DOWN_LOWER_SIGNAL && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state mAN_DOWN_LOWER_SIGNAL
104	E<> CM.fAILURE_STATE_GOING_DOWN	Verifies if the state fAILURE_STATE_GOING_DOWN at CM is reached
105	A[] CM.fAILURE_STATE_GOING_DOWN imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_DOWN is always higher than the VB_UPPER_LIMIT
106	A[] CM.fAILURE_STATE_GOING_DOWN imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_DOWN is always lower than the VB_LOWER_LIMIT
107	E<> CM.fAILURE_STATE_GOING_DOWN && FE.sTOPPED	Verifies if the motor can be stopped at the state fAILURE_STATE_GOING_DOWN
108	E<> CM.fAILURE_STATE_GOING_DOWN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state fAILURE_STATE_GOING_DOWN
109	E<> CM.fAILURE_STATE_GOING_DOWN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state fAILURE_STATE_GOING_DOWN
110	E<> CM.wAIT_FEEDBACK_2230	Verifies if the state wAIT_FEEDBACK_2230 at CM is reached
111	A[] CM.wAIT_FEEDBACK_2230 imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2230 is always higher than the VB_UPPER_LIMIT
112	A[] CM.wAIT_FEEDBACK_2230 imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2230 is always lower than the VB_LOWER_LIMIT
113	E<> CM.wAIT_FEEDBACK_2230 && FE.sTOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2230
114	E<> CM.wAIT_FEEDBACK_2230 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2230
115	E<> CM.wAIT_FEEDBACK_2230 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2230
116	E<> CM.wAIT_FEEDBACK_1516	Verifies if the state wAIT_FEEDBACK_1516 at CM is reached
117	A[] CM.wAIT_FEEDBACK_1516 imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1516 is always higher than the VB_UPPER_LIMIT
118	A[] CM.wAIT_FEEDBACK_1516 imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1516 is always lower than the VB_LOWER_LIMIT
119	E<> CM.wAIT_FEEDBACK_1516 && FE.sTOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_1516
120	E<> CM.wAIT_FEEDBACK_1516 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_1516

Tabela 27 – Propriedades da Plataforma - Parte 3 de 3

121	E<> CM.wAIT_FEEDBACK_1516 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_1516
122	E<> CM.gENERAL_FAILURE_STATE	Verifies if the state gENERAL_FAILURE_STATE at CM is reached
123	A[] CM.gENERAL_FAILURE_STATE imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state gENERAL_FAILURE_STATE is always higher than the VB_UPPER_LIMIT
124	A[] CM.gENERAL_FAILURE_STATE imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state gENERAL_FAILURE_STATE is always lower than the VB_LOWER_LIMIT
125	E<> CM.gENERAL_FAILURE_STATE && FE.sTOPPED	Verifies if the motor can be stopped at the state gENERAL_FAILURE_STATE
126	E<> CM.gENERAL_FAILURE_STATE && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state gENERAL_FAILURE_STATE
127	E<> CM.gENERAL_FAILURE_STATE && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state gENERAL_FAILURE_STATE
128	E<> CM.wAIT_FEEDBACK_3136	Verifies if the state wAIT_FEEDBACK_3136 at CM is reached
129	A[] CM.wAIT_FEEDBACK_3136 imply VB_UPPER_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_3136 is always higher than the VB_UPPER_LIMIT
130	A[] CM.wAIT_FEEDBACK_3136 imply AS_P_X<=VB_LOWER_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_3136 is always lower than the VB_LOWER_LIMIT
131	E<> CM.wAIT_FEEDBACK_3136 && FE.sTOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_3136
132	E<> CM.wAIT_FEEDBACK_3136 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_3136
133	E<> CM.wAIT_FEEDBACK_3136 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_3136
134	E<> CM.f1	Verifies if the state f1 at CM is reached
135	E<> CM.f2	Verifies if the state f2 at CM is reached
136	E<> CM.f3	Verifies if the state f3 at CM is reached
137	E<> CM.f4	Verifies if the state f4 at CM is reached
138	E<> CM.f5	Verifies if the state f5 at CM is reached
139	E<> CM.f6	Verifies if the state f6 at CM is reached
140	E<> CM.f7	Verifies if the state f7 at CM is reached
141	E<> CM.f8	Verifies if the state f8 at CM is reached
142	E<> CM.f9	Verifies if the state f9 at CM is reached
143	E<> CM.f10	Verifies if the state f10 at CM is reached
144	E<> CM.f11	Verifies if the state f11 at CM is reached
145	E<> CM.f12	Verifies if the state f12 at CM is reached
146	E<> CM.f13	Verifies if the state f13 at CM is reached
147	E<> CM.f14	Verifies if the state f14 at CM is reached
148	E<> CM.f15	Verifies if the state f15 at CM is reached
149	E<> CM.f16	Verifies if the state f16 at CM is reached
150	E<> CM.f17	Verifies if the state f17 at CM is reached
151	E<> CM.f18	Verifies if the state f18 at CM is reached
152	E<> CM.f19	Verifies if the state f19 at CM is reached
153	E<> CM.f20	Verifies if the state f20 at CM is reached
154	E<> CM.f21	Verifies if the state f21 at CM is reached
155	E<> CM.f22	Verifies if the state f22 at CM is reached
156	E<> CM.f23	Verifies if the state f23 at CM is reached
157	E<> CM.f24	Verifies if the state f24 at CM is reached
158	E<> CM.f25	Verifies if the state f25 at CM is reached
159	E<> CM.f26	Verifies if the state f26 at CM is reached
160	E<> CM.f27	Verifies if the state f27 at CM is reached
161	E<> CM.f28	Verifies if the state f28 at CM is reached
162	E<> CM.f29	Verifies if the state f29 at CM is reached
163	E<> CM.f30	Verifies if the state f30 at CM is reached
164	E<> CM.f31	Verifies if the state f31 at CM is reached
165	E<> CM.f32	Verifies if the state f32 at CM is reached
166	E<> CM.f33	Verifies if the state f33 at CM is reached
167	E<> CM.f34	Verifies if the state f34 at CM is reached
168	E<> CM.f35	Verifies if the state f35 at CM is reached
169	E<> CM.f36	Verifies if the state f36 at CM is reached
170	E<> CM.f37	Verifies if the state f37 at CM is reached
171	E<> CM.f38	Verifies if the state f38 at CM is reached

B.2 Dispositivo de deposição de pó

Tabela 28 – Propriedades do Dispositivo de deposição de pó - Parte 1 de 3

	Recoater	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.STOPPED		Verifies if the state sTOPPED at CM is reached
3	A[] CM.STOPPED imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state sTOPPED is always higher than the VB_BACK_LIMIT
4	A[] CM.STOPPED imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state sTOPPED is always lower than the VB_FRONT_LIMIT
5	E<> CM.STOPPED && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOPPED
6	E<> CM.STOPPED && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOPPED
7	A[] CM.STOPPED imply FE.STOPPED		Verifies if the moto is always stopped at the state sTOPPED
8	E<> CM.vERIFY_HEIGHT		Verifies if the state vERIFY_HEIGHT at CM is reached
9	A[] CM.vERIFY_HEIGHT imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state vERIFY_HEIGHT is always higher than the VB_BACK_LIMIT
10	A[] CM.vERIFY_HEIGHT imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state vERIFY_HEIGHT is always lower than the VB_FRONT_LIMIT
11	E<> CM.vERIFY_HEIGHT && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_HEIGHT
12	E<> CM.vERIFY_HEIGHT && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_HEIGHT
13	A[] CM.vERIFY_HEIGHT imply FE.STOPPED		Verifies if the motor is always stopped at the state vERIFY_HEIGHT
14	E<> CM.STOP_NORMAL		Verifies if the state sTOP_NORMAL at CM is reached
15	A[] CM.STOP_NORMAL imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state sTOP_NORMAL is always higher than the VB_BACK_LIMIT
16	A[] CM.STOP_NORMAL imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state sTOP_NORMAL is always lower than the VB_FRONT_LIMIT
17	E<> CM.STOP_NORMAL && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_NORMAL
18	E<> CM.STOP_NORMAL && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_NORMAL
19	A[] CM.STOP_NORMAL imply FE.STOPPED		Verifies if the motor is always stopped at the state sTOP_NORMAL
20	E<> CM.rECOATER_MIDDLE		Verifies if the state rECOATER_MIDDLE at CM is reached
21	A[] CM.rECOATER_MIDDLE imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state rECOATER_MIDDLE is always higher than the VB_BACK_LIMIT
22	A[] CM.rECOATER_MIDDLE imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state rECOATER_MIDDLE is always lower than the VB_FRONT_LIMIT
23	E<> CM.rECOATER_MIDDLE && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state rECOATER_MIDDLE
24	E<> CM.rECOATER_MIDDLE && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state rECOATER_MIDDLE
25	A[] CM.rECOATER_MIDDLE imply FE.STOPPED		Verifies if the motor is always stopped at the state rECOATER_MIDDLE
26	E<> CM.rECOATER_FRONT		Verifies if the state rECOATER_FRONT at CM is reached
27	A[] CM.rECOATER_FRONT imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state rECOATER_FRONT is always higher than the VB_BACK_LIMIT
28	A[] CM.rECOATER_FRONT imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state rECOATER_FRONT is always lower than the VB_FRONT_LIMIT
29	E<> CM.rECOATER_FRONT && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state rECOATER_FRONT
30	E<> CM.rECOATER_FRONT && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state rECOATER_FRONT
31	A[] CM.rECOATER_FRONT imply FE.STOPPED		Verifies if the motor is always stopped at the state rECOATER_FRONT
32	E<> CM.rECOATER_BACK		Verifies if the state rECOATER_BACK at CM is reached
33	A[] CM.rECOATER_BACK imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state rECOATER_BACK is always higher than the VB_BACK_LIMIT
34	A[] CM.rECOATER_BACK imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state rECOATER_BACK is always lower than the VB_FRONT_LIMIT
35	E<> CM.rECOATER_BACK && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state rECOATER_BACK
36	E<> CM.rECOATER_BACK && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state rECOATER_BACK
37	A[] CM.rECOATER_BACK imply FE.STOPPED		Verifies if the motor is always stopped at the state rECOATER_BACK
38	E<> CM.mAN_BACK		Verifies if the state mAN_BACK at CM is reached
39	A[] CM.mAN_BACK imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state mAN_BACK is always higher than the VB_BACK_LIMIT
40	A[] CM.mAN_BACK imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state mAN_BACK is always lower than the VB_FRONT_LIMIT
41	E<> CM.mAN_BACK && FE.STOPPED		Verifies if the motor can be stopped at the state mAN_BACK
42	E<> CM.mAN_BACK && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_BACK
43	E<> CM.mAN_BACK && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_BACK
44	E<> CM.vERIFY_MOTOR_BACK_MAN		Verifies if the state vERIFY_MOTOR_BACK_MAN at CM is reached
45	A[] CM.vERIFY_MOTOR_BACK_MAN imply VB_BACK_LIMIT<=AS_P_X		Verifies if the AS_P_X at the state vERIFY_MOTOR_BACK_MAN is always higher than the VB_BACK_LIMIT
46	A[] CM.vERIFY_MOTOR_BACK_MAN imply AS_P_X<=VB_FRONT_LIMIT		Verifies if the AS_P_X at the state vERIFY_MOTOR_BACK_MAN is always lower than the VB_FRONT_LIMIT
47	E<> CM.vERIFY_MOTOR_BACK_MAN && FE.STOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_BACK_MAN
48	E<> CM.vERIFY_MOTOR_BACK_MAN && FE.cLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_BACK_MAN
49	E<> CM.vERIFY_MOTOR_BACK_MAN && FE.aNTI_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_BACK_MAN
50	E<> CM.STOP_BACK_MAN		Verifies if the state STOP_BACK_MAN at CM is reached

Tabela 29 – Propriedades do Dispositivo de deposição de pó - Parte 2 de 3

51	A[] CM.STOP_BACK_MAN imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state STOP_BACK_MAN is always higher than the VB_BACK_LIMIT
52	A[] CM.STOP_BACK_MAN imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state STOP_BACK_MAN is always lower than the VB_FRONT_LIMIT
53	E<> CM.STOP_BACK_MAN && FE.STOPPED	Verifies if the motor can be stopped at the state STOP_BACK_MAN
54	E<> CM.STOP_BACK_MAN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state STOP_BACK_MAN
55	E<> CM.STOP_BACK_MAN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state STOP_BACK_MAN
56	E<> CM.mAN_FRONT	Verifies if the state mAN_FRONT at CM is reached
57	A[] CM.mAN_FRONT imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state mAN_FRONT is always higher than the VB_BACK_LIMIT
58	A[] CM.mAN_FRONT imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state mAN_FRONT is always lower than the VB_FRONT_LIMIT
59	E<> CM.mAN_FRONT && FE.STOPPED	Verifies if the motor can be stopped at the state mAN_FRONT
60	E<> CM.mAN_FRONT && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state mAN_FRONT
61	E<> CM.mAN_FRONT && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state mAN_FRONT
62	E<> CM.vERIFY_MOTOR_FRONT_MAN	Verifies if the state vERIFY_MOTOR_FRONT_MAN at CM is reached
63	A[] CM.vERIFY_MOTOR_FRONT_MAN imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state vERIFY_MOTOR_FRONT_MAN is always higher than the VB_BACK_LIMIT
64	A[] CM.vERIFY_MOTOR_FRONT_MAN imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state vERIFY_MOTOR_FRONT_MAN is always lower than the VB_FRONT_LIMIT
65	E<> CM.vERIFY_MOTOR_FRONT_MAN && FE.STOPPED	Verifies if the motor can be stopped at the state vERIFY_MOTOR_FRONT_MAN
66	E<> CM.vERIFY_MOTOR_FRONT_MAN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_FRONT_MAN
67	E<> CM.vERIFY_MOTOR_FRONT_MAN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_FRONT_MAN
68	E<> CM.STOP_FRONT_MAN	Verifies if the state STOP_FRONT_MAN at CM is reached
69	E<> CM.STOP_FRONT_MAN	Verifies if the AS_P_X at the state STOP_FRONT_MAN is always higher than the VB_BACK_LIMIT
70	A[] CM.STOP_FRONT_MAN imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state STOP_FRONT_MAN is always lower than the VB_FRONT_LIMIT
71	E<> CM.STOP_FRONT_MAN && FE.STOPPED	Verifies if the motor can be stopped at the state STOP_FRONT_MAN
72	E<> CM.STOP_FRONT_MAN && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state STOP_FRONT_MAN
73	E<> CM.STOP_FRONT_MAN && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state STOP_FRONT_MAN
74	E<> CM.mAN_BACK_BACK_SIGNAL	Verifies if the state mAN_BACK_BACK_SIGNAL at CM is reached
75	A[] CM.mAN_BACK_BACK_SIGNAL imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state mAN_BACK_BACK_SIGNAL is always higher than the VB_BACK_LIMIT
76	A[] CM.mAN_BACK_BACK_SIGNAL imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state mAN_BACK_BACK_SIGNAL is always lower than the VB_FRONT_LIMIT
77	E<> CM.mAN_BACK_BACK_SIGNAL && FE.STOPPED	Verifies if the motor can be stopped at the state mAN_BACK_BACK_SIGNAL
78	E<> CM.mAN_BACK_BACK_SIGNAL && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state mAN_BACK_BACK_SIGNAL
79	E<> CM.mAN_BACK_BACK_SIGNAL && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state mAN_BACK_BACK_SIGNAL
80	E<> CM.fAILURE_STATE_GOING_BACK	Verifies if the state fAILURE_STATE_GOING_BACK at CM is reached
81	A[] CM.fAILURE_STATE_GOING_BACK imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_BACK is always higher than the VB_BACK_LIMIT
82	A[] CM.fAILURE_STATE_GOING_BACK imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_BACK is always lower than the VB_FRONT_LIMIT
83	E<> CM.fAILURE_STATE_GOING_BACK && FE.STOPPED	Verifies if the motor can be stopped at the state fAILURE_STATE_GOING_BACK
84	E<> CM.fAILURE_STATE_GOING_BACK && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state fAILURE_STATE_GOING_BACK
85	E<> CM.fAILURE_STATE_GOING_BACK && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state fAILURE_STATE_GOING_BACK
86	E<> CM.wAIT_FEEDBACK_2335	Verifies if the state wAIT_FEEDBACK_2335 at CM is reached
87	A[] CM.wAIT_FEEDBACK_2335 imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2335 is always higher than the VB_BACK_LIMIT
88	A[] CM.wAIT_FEEDBACK_2335 imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2335 is always lower than the VB_FRONT_LIMIT
89	E<> CM.wAIT_FEEDBACK_2335 && FE.STOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2335
90	E<> CM.wAIT_FEEDBACK_2335 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2335
91	E<> CM.wAIT_FEEDBACK_2335 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2335
92	E<> CM.wAIT_FEEDBACK_1718	Verifies if the state wAIT_FEEDBACK_1718 at CM is reached
93	A[] CM.wAIT_FEEDBACK_1718 imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1718 is always higher than the VB_BACK_LIMIT
94	A[] CM.wAIT_FEEDBACK_1718 imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1718 is always lower than the VB_FRONT_LIMIT
95	E<> CM.wAIT_FEEDBACK_1718 && FE.STOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_1718
96	E<> CM.wAIT_FEEDBACK_1718 && FE.STOPPED	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_1718
97	E<> CM.wAIT_FEEDBACK_1718 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_1718
98	E<> CM.mAN_FRONT_FRONT_SIGNAL	Verifies if the state mAN_FRONT_FRONT_SIGNAL at CM is reached
99	A[] CM.mAN_FRONT_FRONT_SIGNAL imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state mAN_FRONT_FRONT_SIGNAL is always higher than the VB_BACK_LIMIT
100	A[] CM.mAN_FRONT_FRONT_SIGNAL imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state mAN_FRONT_FRONT_SIGNAL is always lower than the VB_FRONT_LIMIT
101	E<> CM.mAN_FRONT_FRONT_SIGNAL && FE.STOPPED	Verifies if the motor can be stopped at the state mAN_FRONT_FRONT_SIGNAL
102	E<> CM.mAN_FRONT_FRONT_SIGNAL && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state mAN_FRONT_FRONT_SIGNAL
103	E<> CM.mAN_FRONT_FRONT_SIGNAL && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state mAN_FRONT_FRONT_SIGNAL
104	E<> CM.fAILURE_STATE_GOING_FRONT	Verifies if the state fAILURE_STATE_GOING_FRONT at CM is reached
105	A[] CM.fAILURE_STATE_GOING_FRONT imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_FRONT is always higher than the VB_BACK_LIMIT
106	A[] CM.fAILURE_STATE_GOING_FRONT imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state fAILURE_STATE_GOING_FRONT is always lower than the VB_FRONT_LIMIT
107	E<> CM.fAILURE_STATE_GOING_FRONT && FE.STOPPED	Verifies if the motor can be stopped at the state fAILURE_STATE_GOING_FRONT
108	E<> CM.fAILURE_STATE_GOING_FRONT && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state fAILURE_STATE_GOING_FRONT
109	E<> CM.fAILURE_STATE_GOING_FRONT && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state fAILURE_STATE_GOING_FRONT
110	E<> CM.wAIT_FEEDBACK_2230	Verifies if the state wAIT_FEEDBACK_2230 at CM is reached
111	A[] CM.wAIT_FEEDBACK_2230 imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2230 is always higher than the VB_BACK_LIMIT
112	A[] CM.wAIT_FEEDBACK_2230 imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_2230 is always lower than the VB_FRONT_LIMIT
113	E<> CM.wAIT_FEEDBACK_2230 && FE.STOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2230
114	E<> CM.wAIT_FEEDBACK_2230 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2230
115	E<> CM.wAIT_FEEDBACK_2230 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2230
116	E<> CM.wAIT_FEEDBACK_1516	Verifies if the state wAIT_FEEDBACK_1516 at CM is reached
117	A[] CM.wAIT_FEEDBACK_1516 imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1516 is always higher than the VB_BACK_LIMIT
118	A[] CM.wAIT_FEEDBACK_1516 imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_1516 is always lower than the VB_FRONT_LIMIT
119	E<> CM.wAIT_FEEDBACK_1516 && FE.STOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_1516
120	E<> CM.wAIT_FEEDBACK_1516 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_1516

Tabela 30 – Propriedades do Dispositivo de deposição de pó - Parte 3 de 3

121	E<> CM.wAIT_FEEDBACK_1516 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_1516
122	E<> CM.gENERAL_FAILURE_STATE	Verifies if the state gENERAL_FAILURE_STATE at CM is reached
123	A[] CM.gENERAL_FAILURE_STATE imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state gENERAL_FAILURE_STATE is always higher than the VB_BACK_LIMIT
124	A[] CM.gENERAL_FAILURE_STATE imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state gENERAL_FAILURE_STATE is always lower than the VB_FRONT_LIMIT
125	E<> CM.gENERAL_FAILURE_STATE && FE.STOPPED	Verifies if the motor can be stopped at the state gENERAL_FAILURE_STATE
126	E<> CM.gENERAL_FAILURE_STATE && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state gENERAL_FAILURE_STATE
127	E<> CM.gENERAL_FAILURE_STATE && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state gENERAL_FAILURE_STATE
128	E<> CM.wAIT_FEEDBACK_3136	Verifies if the state wAIT_FEEDBACK_3136 at CM is reached
129	A[] CM.wAIT_FEEDBACK_3136 imply VB_BACK_LIMIT<=AS_P_X	Verifies if the AS_P_X at the state wAIT_FEEDBACK_3136 is always higher than the VB_BACK_LIMIT
130	A[] CM.wAIT_FEEDBACK_3136 imply AS_P_X<=VB_FRONT_LIMIT	Verifies if the AS_P_X at the state wAIT_FEEDBACK_3136 is always lower than the VB_FRONT_LIMIT
131	E<> CM.wAIT_FEEDBACK_3136 && FE.STOPPED	Verifies if the motor can be stopped at the state wAIT_FEEDBACK_3136
132	E<> CM.wAIT_FEEDBACK_3136 && FE.cLOCKWISE	Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_3136
133	E<> CM.wAIT_FEEDBACK_3136 && FE.aNTI_CLOCKWISE	Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_3136
134	E<> CM.f1	Verifies if the state f1 at CM is reached
135	E<> CM.f2	Verifies if the state f2 at CM is reached
136	E<> CM.f3	Verifies if the state f3 at CM is reached
137	E<> CM.f4	Verifies if the state f4 at CM is reached
138	E<> CM.f5	Verifies if the state f5 at CM is reached
139	E<> CM.f6	Verifies if the state f6 at CM is reached
140	E<> CM.f7	Verifies if the state f7 at CM is reached
141	E<> CM.f8	Verifies if the state f8 at CM is reached
142	E<> CM.f9	Verifies if the state f9 at CM is reached
143	E<> CM.f10	Verifies if the state f10 at CM is reached
144	E<> CM.f11	Verifies if the state f11 at CM is reached
145	E<> CM.f12	Verifies if the state f12 at CM is reached
146	E<> CM.f13	Verifies if the state f13 at CM is reached
147	E<> CM.f14	Verifies if the state f14 at CM is reached
148	E<> CM.f15	Verifies if the state f15 at CM is reached
149	E<> CM.f16	Verifies if the state f16 at CM is reached
150	E<> CM.f17	Verifies if the state f17 at CM is reached
151	E<> CM.f18	Verifies if the state f18 at CM is reached
152	E<> CM.f19	Verifies if the state f19 at CM is reached
153	E<> CM.f20	Verifies if the state f20 at CM is reached
154	E<> CM.f21	Verifies if the state f21 at CM is reached
155	E<> CM.f22	Verifies if the state f22 at CM is reached
156	E<> CM.f23	Verifies if the state f23 at CM is reached
157	E<> CM.f24	Verifies if the state f24 at CM is reached
158	E<> CM.f25	Verifies if the state f25 at CM is reached
159	E<> CM.f26	Verifies if the state f26 at CM is reached
160	E<> CM.f27	Verifies if the state f27 at CM is reached
161	E<> CM.f28	Verifies if the state f28 at CM is reached
162	E<> CM.f29	Verifies if the state f29 at CM is reached
163	E<> CM.f30	Verifies if the state f30 at CM is reached
164	E<> CM.f31	Verifies if the state f31 at CM is reached
165	E<> CM.f32	Verifies if the state f32 at CM is reached
166	E<> CM.f33	Verifies if the state f33 at CM is reached
167	E<> CM.f34	Verifies if the state f34 at CM is reached
168	E<> CM.f35	Verifies if the state f35 at CM is reached
169	E<> CM.f36	Verifies if the state f36 at CM is reached
170	E<> CM.f37	Verifies if the state f37 at CM is reached
171	E<> CM.f38	Verifies if the state f38 at CM is reached

B.3 Sistema de carregamento 1

Tabela 31 – Propriedades do Sistema de carregamento 1

	Loader 1 Man		
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.sTOPPED		Verifies if the state sTOPPED at CM is reached
3	E<> CM.sTOPPED && FE.vB_ANTICLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOPPED
4	E<> CM.sTOPPED && FE.vB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOPPED
5	A[] CM.sTOPPED imply FE.sTOPPED		Verifies if the moto is always stopped at the state sTOPPED
6	E<> CM.sTOP_NORMAL		Verifies if the state sTOP_NORMAL at CM is reached
7	E<> CM.sTOP_NORMAL && FE.vB_ANTICLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_NORMAL
8	E<> CM.sTOP_NORMAL && FE.vB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_NORMAL
9	A[] CM.sTOP_NORMAL imply FE.sTOPPED		Verifies if the motor is always stopped at the state sTOP_NORMAL
10	E<> CM.mAN_VB_CLOCKWISE		Verifies if the state mAN_VB_CLOCKWISE at CM is reached
11	E<> CM.mAN_VB_CLOCKWISE && FE.sTOPPED		Verifies if the motor can be stopped at the state mAN_VB_CLOCKWISE
12	E<> CM.mAN_VB_CLOCKWISE && FE.vB_ANTICLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_VB_CLOCKWISE
13	E<> CM.mAN_VB_CLOCKWISE && FE.vB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_VB_CLOCKWISE
14	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN		Verifies if the state vERIFY_MOTOR_VB_CLOCKWISE_MAN at CM is reached
15	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
16	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.vB_ANTICLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
17	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.vB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
18	E<> CM.sTOP_VB_CLOCKWISE_MAN		Verifies if the state sTOP_VB_CLOCKWISE_MAN at CM is reached
19	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state sTOP_VB_CLOCKWISE_MAN
20	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.vB_ANTICLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_VB_CLOCKWISE_MAN
21	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.vB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_VB_CLOCKWISE_MAN
22	E<> CM.wAIT_FEEDBACK_2335		Verifies if the state wAIT_FEEDBACK_2335 at CM is reached
23	E<> CM.wAIT_FEEDBACK_2335 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2335
24	E<> CM.wAIT_FEEDBACK_2335 && FE.vB_ANTICLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2335
25	E<> CM.wAIT_FEEDBACK_2335 && FE.vB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2335
26	E<> CM.f23		Verifies if the state f23 at CM is reached
27	E<> CM.f24		Verifies if the state f24 at CM is reached

B.4 Sistema de carregamento 2

Tabela 32 – Propriedades do Sistema de carregamento 2

	Loader 2 Man		
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.sTOPPED		Verifies if the state sTOPPED at CM is reached
3	E<> CM.sTOPPED && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOPPED
4	E<> CM.sTOPPED && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOPPED
5	A[] CM.sTOPPED imply FE.sTOPPED		Verifies if the moto is always stopped at the state sTOPPED
6	E<> CM.sTOP_NORMAL		Verifies if the state sTOP_NORMAL at CM is reached
7	E<> CM.sTOP_NORMAL && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_NORMAL
8	E<> CM.sTOP_NORMAL && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_NORMAL
9	A[] CM.sTOP_NORMAL imply FE.sTOPPED		Verifies if the motor is always stopped at the state sTOP_NORMAL
10	E<> CM.mAN_VB_CLOCKWISE		Verifies if the state mAN_VB_CLOCKWISE at CM is reached
11	E<> CM.mAN_VB_CLOCKWISE && FE.sTOPPED		Verifies if the motor can be stopped at the state mAN_VB_CLOCKWISE
12	E<> CM.mAN_VB_CLOCKWISE && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_VB_CLOCKWISE
13	E<> CM.mAN_VB_CLOCKWISE && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_VB_CLOCKWISE
14	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN		Verifies if the state vERIFY_MOTOR_VB_CLOCKWISE_MAN at CM is reached
15	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
16	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
17	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
18	E<> CM.sTOP_VB_CLOCKWISE_MAN		Verifies if the state sTOP_VB_CLOCKWISE_MAN at CM is reached
19	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state sTOP_VB_CLOCKWISE_MAN
20	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_VB_CLOCKWISE_MAN
21	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_VB_CLOCKWISE_MAN
22	E<> CM.mAN_VB_ANTICLOCKWISE		Verifies if the state mAN_VB_ANTICLOCKWISE at CM is reached
23	E<> CM.mAN_VB_ANTICLOCKWISE && FE.sTOPPED		Verifies if the motor can be stopped at the state mAN_VB_ANTICLOCKWISE
24	E<> CM.mAN_VB_ANTICLOCKWISE && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_VB_ANTICLOCKWISE
25	E<> CM.mAN_VB_ANTICLOCKWISE && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_VB_ANTICLOCKWISE
26	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN		Verifies if the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN at CM is reached
27	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN
28	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN
29	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN
30	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN		Verifies if the state sTOP_VB_ANTICLOCKWISE_MAN at CM is reached
31	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state sTOP_VB_ANTICLOCKWISE_MAN
32	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_VB_ANTICLOCKWISE_MAN
33	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_VB_ANTICLOCKWISE_MAN
34	E<> CM.wAIT_FEEDBACK_2335		Verifies if the state wAIT_FEEDBACK_2335 at CM is reached
35	E<> CM.wAIT_FEEDBACK_2335 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2335
36	E<> CM.wAIT_FEEDBACK_2335 && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2335
37	E<> CM.wAIT_FEEDBACK_2335 && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2335
38	E<> CM.wAIT_FEEDBACK_2230		Verifies if the state wAIT_FEEDBACK_2230 at CM is reached
39	E<> CM.wAIT_FEEDBACK_2230 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2230
40	E<> CM.wAIT_FEEDBACK_2230 && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2230
41	E<> CM.wAIT_FEEDBACK_2230 && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2230
42	E<> CM.wAIT_FEEDBACK_3136		Verifies if the state wAIT_FEEDBACK_3136 at CM is reached
43	E<> CM.wAIT_FEEDBACK_3136 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_3136
44	E<> CM.wAIT_FEEDBACK_3136 && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_3136
45	E<> CM.wAIT_FEEDBACK_3136 && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_3136
46	E<> CM.f19		Verifies if the state f19 at CM is reached
47	E<> CM.f22		Verifies if the state f22 at CM is reached
48	E<> CM.f23		Verifies if the state f23 at CM is reached
49	E<> CM.f24		Verifies if the state f24 at CM is reached
50	E<> CM.f31		Verifies if the state f31 at CM is reached
51	E<> CM.f36		Verifies if the state f36 at CM is reached

B.5 Sistema de carregamento 3

Tabela 33 – Propriedades do Sistema de carregamento 3

	Loader 3 Man		
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.sTOPPED		Verifies if the state sTOPPED at CM is reached
3	E<> CM.sTOPPED && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOPPED
4	E<> CM.sTOPPED && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOPPED
5	A[] CM.sTOPPED imply FE.sTOPPED		Verifies if the moto is always stopped at the state sTOPPED
6	E<> CM.sTOP_NORMAL		Verifies if the state sTOP_NORMAL at CM is reached
7	E<> CM.sTOP_NORMAL && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_NORMAL
8	E<> CM.sTOP_NORMAL && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_NORMAL
9	A[] CM.sTOP_NORMAL imply FE.sTOPPED		Verifies if the motor is always stopped at the state sTOP_NORMAL
10	E<> CM.mAN_VB_CLOCKWISE		Verifies if the state mAN_VB_CLOCKWISE at CM is reached
11	E<> CM.mAN_VB_CLOCKWISE && FE.sTOPPED		Verifies if the motor can be stopped at the state mAN_VB_CLOCKWISE
12	E<> CM.mAN_VB_CLOCKWISE && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_VB_CLOCKWISE
13	E<> CM.mAN_VB_CLOCKWISE && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_VB_CLOCKWISE
14	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN		Verifies if the state vERIFY_MOTOR_VB_CLOCKWISE_MAN at CM is reached
15	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
16	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
17	E<> CM.vERIFY_MOTOR_VB_CLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_VB_CLOCKWISE_MAN
18	E<> CM.sTOP_VB_CLOCKWISE_MAN		Verifies if the state sTOP_VB_CLOCKWISE_MAN at CM is reached
19	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state sTOP_VB_CLOCKWISE_MAN
20	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_VB_CLOCKWISE_MAN
21	E<> CM.sTOP_VB_CLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_VB_CLOCKWISE_MAN
22	E<> CM.mAN_VB_ANTICLOCKWISE		Verifies if the state mAN_VB_ANTICLOCKWISE at CM is reached
23	E<> CM.mAN_VB_ANTICLOCKWISE && FE.sTOPPED		Verifies if the motor can be stopped at the state mAN_VB_ANTICLOCKWISE
24	E<> CM.mAN_VB_ANTICLOCKWISE && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state mAN_VB_ANTICLOCKWISE
25	E<> CM.mAN_VB_ANTICLOCKWISE && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state mAN_VB_ANTICLOCKWISE
26	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN		Verifies if the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN at CM is reached
27	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN
28	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN
29	E<> CM.vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state vERIFY_MOTOR_VB_ANTICLOCKWISE_MAN
30	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN		Verifies if the state sTOP_VB_ANTICLOCKWISE_MAN at CM is reached
31	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN && FE.sTOPPED		Verifies if the motor can be stopped at the state sTOP_VB_ANTICLOCKWISE_MAN
32	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state sTOP_VB_ANTICLOCKWISE_MAN
33	E<> CM.sTOP_VB_ANTICLOCKWISE_MAN && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state sTOP_VB_ANTICLOCKWISE_MAN
34	E<> CM.wAIT_FEEDBACK_2335		Verifies if the state wAIT_FEEDBACK_2335 at CM is reached
35	E<> CM.wAIT_FEEDBACK_2335 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2335
36	E<> CM.wAIT_FEEDBACK_2335 && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2335
37	E<> CM.wAIT_FEEDBACK_2335 && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2335
38	E<> CM.wAIT_FEEDBACK_2230		Verifies if the state wAIT_FEEDBACK_2230 at CM is reached
39	E<> CM.wAIT_FEEDBACK_2230 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_2230
40	E<> CM.wAIT_FEEDBACK_2230 && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_2230
41	E<> CM.wAIT_FEEDBACK_2230 && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_2230
42	E<> CM.wAIT_FEEDBACK_3136		Verifies if the state wAIT_FEEDBACK_3136 at CM is reached
43	E<> CM.wAIT_FEEDBACK_3136 && FE.sTOPPED		Verifies if the motor can be stopped at the state wAIT_FEEDBACK_3136
44	E<> CM.wAIT_FEEDBACK_3136 && FE.VB_CLOCKWISE		Verifies if the motor can rotate clockwise while at the state wAIT_FEEDBACK_3136
45	E<> CM.wAIT_FEEDBACK_3136 && FE.aNTI_VB_CLOCKWISE		Verifies if the motor can rotate anticlockwise while at the state wAIT_FEEDBACK_3136
46	E<> CM.f19		Verifies if the state f19 at CM is reached
47	E<> CM.f22		Verifies if the state f22 at CM is reached
48	E<> CM.f23		Verifies if the state f23 at CM is reached
49	E<> CM.f24		Verifies if the state f24 at CM is reached
50	E<> CM.f31		Verifies if the state f31 at CM is reached
51	E<> CM.f36		Verifies if the state f36 at CM is reached

B.6 Laser

Tabela 34 – Propriedades do Laser

	Laser	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.oFF		Verifies if the state oFF at CM is reached
3	A[] CM.oFF imply VB_LASER_ON == 0		Verifies if at the state oFF the variable VB_LASER_ON is always 0
4	A[] CM.oFF imply LA.oFF		Verifies if at the state oFF the laser is always off
5	E<> CM.mESSAGE_ON_MAN		Verifies if the state mESSAGE_ON_MAN at CM is reached
6	A[] CM.mESSAGE_ON_MAN imply VB_LASER_ON == 0		Verifies if at the state mESSAGE_ON_MAN the variable VB_LASER_ON is always 0
7	A[] CM.mESSAGE_ON_MAN imply LA.oFF		Verifies if at the state mESSAGE_ON_MAN the laser is always off
8	E<> CM.fEED_ON_MAN		Verifies if the state fEED_ON_MAN at CM is reached
9	A[] CM.fEED_ON_MAN imply VB_LASER_ON ==1		Verifies if at the state fEED_ON_MAN the variable VB_LASER_ON is always 1
10	A[] CM.fEED_ON_MAN imply LA.oN		Verifies if at the state fEED_ON_MAN the laser is always on
11	E<> CM.mAN_ON		Verifies if the state mAN_ON at CM is reached
12	A[] CM.mAN_ON imply VB_LASER_ON == 1		Verifies if at the state mAN_ON the variable VB_LASER_ON is always 1
13	A[] CM.mAN_ON imply LA.oN		Verifies if at the state mAN_ON the laser is always on
14	E<> CM.sTOP_MAN		Verifies if the state sTOP_MAN at CM is reached
15	A[] CM.mAN_ON imply VB_LASER_ON == 1		Verifies if at the state sTOP_MAN the variable VB_LASER_ON is always 1
16	A[] CM.mAN_ON imply LA.oN		Verifies if at the state sTOP_MAN the laser is always on
17	E<> CM.fEED_OFF_MAN		Verifies if the state fEED_OFF_MAN at CM is reached
18	A[] CM.fEED_OFF_MAN imply VB_LASER_ON == 0		Verifies if at the state fEED_OFF_MAN the variable VB_LASER_ON is always 0
19	A[] CM.fEED_OFF_MAN imply LA.oFF		Verifies if at the state fEED_OFF_MAN the laser is always off
20	E<> CM.f1		Verifies if the state f1 at CM is reached
21	E<> CM.f2		Verifies if the state f2 at CM is reached

B.7 Sistema de aquecimento

Tabela 35 – Propriedades do Sistema de aquecimento

	Heater	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.oFF		Verifies if the state oFF at CM is reached
3	A[] CM.oFF imply VB_HEATER_ON == 0		Verifies if at the state oFF the variable VB_HEATER_ON is always 0
4	A[] CM.oFF imply HE.oFF		Verifies if at the state oFF the heater is always off
5	A[] CM.oFF imply VB_TEMPERATURE <= 200		Verifies if the temperature is always under 200 degrees
6	E<> CM.mESSAGE_ON_MAN		Verifies if the state mESSAGE_ON_MAN at CM is reached
7	A[] CM.mESSAGE_ON_MAN imply VB_HEATER_ON == 0		Verifies if at the state mESSAGE_ON_MAN the variable VB_HEATER_ON is always 0
8	A[] CM.mESSAGE_ON_MAN imply HE.oFF		Verifies if at the state mESSAGE_ON_MAN the heater is always off
9	A[] CM.mESSAGE_ON_MAN imply VB_TEMPERATURE <= 200		Verifies if the temperature is always under 200 degrees
10	E<> CM.fEED_ON_MAN		Verifies if the state fEED_ON_MAN at CM is reached
11	A[] CM.fEED_ON_MAN imply VB_HEATER_ON ==1		Verifies if at the state fEED_ON_MAN the variable VB_HEATER_ON is always 0
12	A[] CM.fEED_ON_MAN imply HE.oN		Verifies if at the state fEED_ON_MAN the heater is always off
13	A[] CM.fEED_ON_MAN imply VB_TEMPERATURE <= 200		Verifies if the temperature is always under 200 degrees
14	E<> CM.mAN_ON		Verifies if the state mAN_ON at CM is reached
15	A[] CM.mAN_ON imply VB_HEATER_ON == 1		Verifies if at the state mAN_ON the variable VB_HEATER_ON is always 1
16	A[] CM.mAN_ON imply HE.oN		Verifies if at the state mAN_ON the heater is always on
17	A[] CM.mAN_ON imply VB_TEMPERATURE <= 200		Verifies if the temperature is always under 200 degrees
18	E<> CM.sTOP_MAN		Verifies if the state sTOP_MAN at CM is reached
19	A[] CM.mAN_ON imply VB_HEATER_ON == 1		Verifies if at the state sTOP_MAN the variable VB_HEATER_ON is always 0
20	A[] CM.mAN_ON imply HE.oN		Verifies if at the state sTOP_MAN the heater is always off
21	A[] CM.mAN_ON imply VB_TEMPERATURE <= 200		Verifies if the temperature is always under 200 degrees
22	E<> CM.fEED_OFF_MAN		Verifies if the state fEED_OFF_MAN at CM is reached
23	A[] CM.fEED_OFF_MAN imply VB_HEATER_ON == 0		Verifies if at the state fEED_OFF_MAN the variable VB_HEATER_ON is always 0
24	A[] CM.fEED_OFF_MAN imply HE.oFF		Verifies if at the state fEED_OFF_MAN the heater is always off
25	A[] CM.fEED_OFF_MAN imply VB_TEMPERATURE <= 200		Verifies if the temperature is always under 200 degrees
26	E<> CM.f1		Verifies if the state f1 at CM is reached
27	E<> CM.f2		Verifies if the state f2 at CM is reached

B.8 Sistema de refrigeração

Tabela 36 – Propriedades do Sistema de refrigeração

	Refrigeration	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.oFF		Verifies if the state oFF at CM is reached
3	A[] CM.oFF imply VB_REFRIGERATION_ON == 0		Verifies if at the state oFF the variable VB_REFRIGERATION_ON is always 0
4	A[] CM.oFF imply KU.oFF		Verifies if at the state oFF the refrigeration is always off
5	A[] CM.oFF imply VB_TEMPERATURE >= 20		Verifies if the temperature is always over 20 degrees
6	E<> CM.mESSAGE_ON_MAN		Verifies if the state mESSAGE_ON_MAN at CM is reached
7	A[] CM.mESSAGE_ON_MAN imply VB_REFRIGERATION_ON == 0		Verifies if at the state mESSAGE_ON_MAN the variable VB_REFRIGERATION_ON is always 0
8	A[] CM.mESSAGE_ON_MAN imply KU.oFF		Verifies if at the state mESSAGE_ON_MAN the refrigeration is always off
9	A[] CM.mESSAGE_ON_MAN imply VB_TEMPERATURE >= 20		Verifies if the temperature is always over 20 degrees
10	E<> CM.fEED_ON_MAN		Verifies if the state fEED_ON_MAN at CM is reached
11	A[] CM.fEED_ON_MAN imply VB_REFRIGERATION_ON ==1		Verifies if at the state fEED_ON_MAN the variable VB_REFRIGERATION_ON is always 0
12	A[] CM.fEED_ON_MAN imply KU.oN		Verifies if at the state fEED_ON_MAN the refrigeration is always off
13	A[] CM.fEED_ON_MAN imply VB_TEMPERATURE >= 20		Verifies if the temperature is always over 20 degrees
14	E<> CM.mAN_ON		Verifies if the state mAN_ON at CM is reached
15	A[] CM.mAN_ON imply VB_REFRIGERATION_ON == 1		Verifies if at the state mAN_ON the variable VB_REFRIGERATION_ON is always 1
16	A[] CM.mAN_ON imply KU.oN		Verifies if at the state mAN_ON the refrigeration is always on
17	A[] CM.mAN_ON imply VB_TEMPERATURE >= 20		Verifies if the temperature is always over 20degrees
18	E<> CM.sTOP_MAN		Verifies if the state sTOP_MAN at CM is reached
19	A[] CM.mAN_ON imply VB_REFRIGERATION_ON == 1		Verifies if at the state sTOP_MAN the variable VB_REFRIGERATION_ON is always 0
20	A[] CM.mAN_ON imply KU.oN		Verifies if at the state sTOP_MAN the refrigeration is always off
21	A[] CM.mAN_ON imply VB_TEMPERATURE >= 20		Verifies if the temperature is always over 20 degrees
22	E<> CM.fEED_OFF_MAN		Verifies if the state fEED_OFF_MAN at CM is reached
23	A[] CM.fEED_OFF_MAN imply VB_REFRIGERATION_ON == 0		Verifies if at the state fEED_OFF_MAN the variable VB_REFRIGERATION_ON is always 0
24	A[] CM.fEED_OFF_MAN imply KU.oFF		Verifies if at the state fEED_OFF_MAN the refrigeration is always off
25	A[] CM.fEED_OFF_MAN imply VB_TEMPERATURE >= 20		Verifies if the temperature is always over 20 degrees
26	E<> CM.f1		Verifies if the state f1 at CM is reached
27	E<> CM.f2		Verifies if the state f2 at CM is reached

B.9 Sistema de filtragem

Tabela 37 – Propriedades do Sistema de filtragem

	Filter	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.oFF		Verifies if the state oFF at CM is reached
3	A[] CM.oFF imply VB_FILTER_ON == 0		Verifies if at the state oFF the variable VB_FILTER_ON is always 0
4	A[] CM.oFF imply FI.oFF		Verifies if at the state oFF the filter is always off
5	E<> CM.mESSAGE_ON_MAN		Verifies if the state mESSAGE_ON_MAN at CM is reached
6	A[] CM.mESSAGE_ON_MAN imply VB_FILTER_ON == 0		Verifies if at the state mESSAGE_ON_MAN the variable VB_FILTER_ON is always 0
7	A[] CM.mESSAGE_ON_MAN imply FI.oFF		Verifies if at the state mESSAGE_ON_MAN the filter is always off
8	E<> CM.fEED_ON_MAN		Verifies if the state fEED_ON_MAN at CM is reached
9	A[] CM.fEED_ON_MAN imply VB_FILTER_ON ==1		Verifies if at the state fEED_ON_MAN the variable VB_FILTER_ON is always 0
10	A[] CM.fEED_ON_MAN imply FI.oN		Verifies if at the state fEED_ON_MAN the filter is always off
11	E<> CM.mAN_ON		Verifies if the state mAN_ON at CM is reached
12	A[] CM.mAN_ON imply VB_FILTER_ON == 1		Verifies if at the state mAN_ON the variable VB_FILTER_ON is always 1
13	A[] CM.mAN_ON imply FI.oN		Verifies if at the state mAN_ON the filter is always on
14	E<> CM.sTOP_MAN		Verifies if the state sTOP_MAN at CM is reached
15	A[] CM.mAN_ON imply VB_FILTER_ON == 1		Verifies if at the state sTOP_MAN the variable VB_FILTER_ON is always 0
16	A[] CM.mAN_ON imply FI.oN		Verifies if at the state sTOP_MAN the filter is always off
17	E<> CM.fEED_OFF_MAN		Verifies if the state fEED_OFF_MAN at CM is reached
18	A[] CM.fEED_OFF_MAN imply VB_FILTER_ON == 0		Verifies if at the state fEED_OFF_MAN the variable VB_FILTER_ON is always 0
19	A[] CM.fEED_OFF_MAN imply FI.oFF		Verifies if at the state fEED_OFF_MAN the filter is always off
20	E<> CM.f1		Verifies if the state f1 at CM is reached
21	E<> CM.f2		Verifies if the state f2 at CM is reached

B.10 Sistema de vácuo

Tabela 38 – Propriedades do Sistema de vácuo

	Vacuum	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.oFF		Verifies if the state oFF at CM is reached
3	A[] CM.oFF imply VB_VALVE_OPEN == 0		Verifies if at the state oFF the variable VB_VALVE_OPEN is always 0
4	A[] CM.oFF imply VB_MOTOR_ON == 0		Verifies if at the state oFF the variable VB_MOTOR_ON is always 0
5	A[] CM.oFF imply VA.oFF		Verifies if at the state oFF the vacuum is always off
6	A[] CM.oFF imply MO.oFF		Verifies if at the state oFF the motor is always off
7	A[] CM.oFF imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
8	E<> CM.mESSAGE_ON_VALVE_MAN		Verifies if the state mESSAGE_ON_VALVE_MAN at CM is reached
9	A[] CM.mESSAGE_ON_VALVE_MAN imply VB_VALVE_OPEN == 0		Verifies if at the state mESSAGE_ON_VALVE_MAN the variable VB_VALVE_OPEN is always 0
10	A[] CM.mESSAGE_ON_VALVE_MAN imply VB_MOTOR_ON == 0		Verifies if at the state mESSAGE_ON_VALVE_MAN the variable VB_MOTOR_ON is always 0
11	A[] CM.mESSAGE_ON_VALVE_MAN imply VA.oFF		Verifies if at the state mESSAGE_ON_VALVE_MAN the vacuum is always off
12	A[] CM.mESSAGE_ON_VALVE_MAN imply MO.oFF		Verifies if at the state mESSAGE_ON_VALVE_MAN the motor is always off
13	A[] CM.mESSAGE_ON_VALVE_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
14	E<> CM.fEED_ON_VALVE_MAN		Verifies if the state fEED_ON_VALVE_MAN at CM is reached
15	A[] CM.fEED_ON_VALVE_MAN imply VB_VALVE_OPEN == 1		Verifies if at the state fEED_ON_VALVE_MAN the variable VB_VALVE_OPEN is always 1
16	A[] CM.fEED_ON_VALVE_MAN imply VB_MOTOR_ON == 0		Verifies if at the state fEED_ON_VALVE_MAN the variable VB_MOTOR_ON is always 0
17	A[] CM.fEED_ON_VALVE_MAN imply VA.oN		Verifies if at the state fEED_ON_VALVE_MAN the vacuum is always on
18	A[] CM.fEED_ON_VALVE_MAN imply MO.oFF		Verifies if at the state fEED_ON_VALVE_MAN the motor is always off
19	A[] CM.fEED_ON_VALVE_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
20	E<> CM.mESSAGE_ON_MOTOR_MAN		Verifies if the state mESSAGE_ON_MOTOR_MAN at CM is reached
21	A[] CM.mESSAGE_ON_MOTOR_MAN imply VB_VALVE_OPEN == 1		Verifies if at the state mESSAGE_ON_MOTOR_MAN the variable VB_VALVE_OPEN is always 1
22	A[] CM.mESSAGE_ON_MOTOR_MAN imply VB_MOTOR_ON == 0		Verifies if at the state mESSAGE_ON_MOTOR_MAN the variable VB_MOTOR_ON is always 0
23	A[] CM.mESSAGE_ON_MOTOR_MAN imply VA.oN		Verifies if at the state mESSAGE_ON_MOTOR_MAN the vacuum is always on
24	A[] CM.mESSAGE_ON_MOTOR_MAN imply MO.oFF		Verifies if at the state mESSAGE_ON_MOTOR_MAN the motor is always off
25	A[] CM.mESSAGE_ON_MOTOR_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
26	E<> CM.fEED_ON_MOTOR_MAN		Verifies if the state fEED_ON_MOTOR_MAN at CM is reached
27	A[] CM.fEED_ON_MOTOR_MAN imply VB_VALVE_OPEN == 1		Verifies if at the state fEED_ON_MOTOR_MAN the variable VB_VALVE_OPEN is always 1
28	A[] CM.fEED_ON_MOTOR_MAN imply VB_MOTOR_ON == 1		Verifies if at the state fEED_ON_MOTOR_MAN the variable VB_MOTOR_ON is always 1
29	A[] CM.fEED_ON_MOTOR_MAN imply VA.oN		Verifies if at the state fEED_ON_MOTOR_MAN the vacuum is always on
30	A[] CM.fEED_ON_MOTOR_MAN imply MO.oN		Verifies if at the state fEED_ON_MOTOR_MAN the motor is always on
31	A[] CM.fEED_ON_MOTOR_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
32	E<> CM.mAN_ON		Verifies if the state mAN_ON at CM is reached
33	A[] CM.mAN_ON imply VB_VALVE_OPEN == 1		Verifies if at the state mAN_ON the variable VB_VALVE_OPEN is always 1
34	A[] CM.mAN_ON imply VB_MOTOR_ON == 1		Verifies if at the state mAN_ON the variable VB_MOTOR_ON is always 1
35	A[] CM.mAN_ON imply VA.oN		Verifies if at the state mAN_ON the valve is always on
36	A[] CM.mAN_ON imply MO.oN		Verifies if at the state mAN_ON the motor is always on
37	A[] CM.mAN_ON imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
38	E<> CM.sTOP_VALVE_MAN		Verifies if the state sTOP_VALVE_MAN at CM is reached
39	A[] CM.sTOP_VALVE_MAN imply VB_VALVE_OPEN == 1		Verifies if at the state sTOP_VALVE_MAN the variable VB_VALVE_OPEN is always 1
40	A[] CM.sTOP_VALVE_MAN imply VB_MOTOR_ON == 1		Verifies if at the state sTOP_VALVE_MAN the variable VB_MOTOR_ON is always 1
41	A[] CM.sTOP_VALVE_MAN imply VA.oN		Verifies if at the state sTOP_VALVE_MAN the valve is always on
42	A[] CM.sTOP_VALVE_MAN imply MO.oN		Verifies if at the state sTOP_VALVE_MAN the motor is always on
43	A[] CM.sTOP_VALVE_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
44	E<> CM.fEED_OFF_VALVE_MAN		Verifies if the state fEED_OFF_VALVE_MAN at CM is reached
45	A[] CM.fEED_OFF_VALVE_MAN imply VB_VALVE_OPEN == 0		Verifies if at the state fEED_OFF_VALVE_MAN the variable VB_VALVE_OPEN is always 0
46	A[] CM.fEED_OFF_VALVE_MAN imply VB_MOTOR_ON == 1		Verifies if at the state fEED_OFF_VALVE_MAN the variable VB_MOTOR_ON is always 1
47	A[] CM.fEED_OFF_VALVE_MAN imply VA.oFF		Verifies if at the state fEED_OFF_VALVE_MAN the valve is always off
48	A[] CM.fEED_OFF_VALVE_MAN imply MO.oN		Verifies if at the state fEED_OFF_VALVE_MAN the motor is always on
49	A[] CM.fEED_OFF_VALVE_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
50	E<> CM.sTOP_MOTOR_MAN		Verifies if the state sTOP_MOTOR_MAN at CM is reached
51	A[] CM.sTOP_MOTOR_MAN imply VB_VALVE_OPEN == 0		Verifies if at the state sTOP_MOTOR_MAN the variable VB_VALVE_OPEN is always 0
52	A[] CM.sTOP_MOTOR_MAN imply VB_MOTOR_ON == 1		Verifies if at the state sTOP_MOTOR_MAN the variable VB_MOTOR_ON is always 1
53	A[] CM.sTOP_MOTOR_MAN imply VA.oFF		Verifies if at the state sTOP_MOTOR_MAN the valve is always off
54	A[] CM.sTOP_MOTOR_MAN imply MO.oN		Verifies if at the state sTOP_MOTOR_MAN the motor is always on
55	A[] CM.sTOP_MOTOR_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
56	E<> CM.fEED_OFF_MOTOR_MAN		Verifies if the state fEED_OFF_MOTOR_MAN at CM is reached
57	A[] CM.fEED_OFF_MOTOR_MAN imply VB_VALVE_OPEN == 0		Verifies if at the state fEED_OFF_MOTOR_MAN the variable VB_VALVE_OPEN is always 0
58	A[] CM.fEED_OFF_MOTOR_MAN imply VB_MOTOR_ON == 0		Verifies if at the state fEED_OFF_MOTOR_MAN the variable VB_MOTOR_ON is always 0
59	A[] CM.fEED_OFF_MOTOR_MAN imply VA.oFF		Verifies if at the state fEED_OFF_MOTOR_MAN the valve is always off
60	A[] CM.fEED_OFF_MOTOR_MAN imply MO.oFF		Verifies if at the state fEED_OFF_MOTOR_MAN the motor is always off
61	A[] CM.fEED_OFF_MOTOR_MAN imply VB_PRESSURE >= 5		Verifies if the pressure is higher than 5 mbar
62	E<> CM.f1		Verifies if the state f1 at CM is reached
63	E<> CM.f2		Verifies if the state f2 at CM is reached

B.11 Sistema de proteção a gás

Tabela 39 – Propriedades do Sistema de proteção a gás

	Argon	Expected	Description
1	A[] not deadlock		Verifies if there is deadlock in the system
2	E<> CM.oFF		Verifies if the state oFF at CM is reached
3	A[] CM.oFF imply VB_ARGON_ON == 0		Verifies if at the state oFF the variable VB_ARGON_ON is always 0
4	A[] CM.oFF imply AR.oFF		Verifies if at the state oFF the argon is always off
5	A[] CM.oFF imply VB_ARGON <= 95		Verifies if the argon concentration is under 95%
6	E<> CM.mESSAGE_ON_MAN		Verifies if the state mESSAGE_ON_MAN at CM is reached
7	A[] CM.mESSAGE_ON_MAN imply VB_ARGON_ON == 0		Verifies if at the state mESSAGE_ON_MAN the variable VB_ARGON_ON is always 0
8	A[] CM.mESSAGE_ON_MAN imply AR.oFF		Verifies if at the state mESSAGE_ON_MAN the argon is always off
9	A[] CM.mESSAGE_ON_MAN imply VB_ARGON <= 95		Verifies if the argon concentration is under 95%
10	E<> CM.fEED_ON_MAN		Verifies if the state fEED_ON_MAN at CM is reached
11	A[] CM.fEED_ON_MAN imply VB_ARGON_ON ==1		Verifies if at the state fEED_ON_MAN the variable VB_ARGON_ON is always 0
12	A[] CM.fEED_ON_MAN imply AR.oN		Verifies if at the state fEED_ON_MAN the argon is always off
13	A[] CM.fEED_ON_MAN imply VB_ARGON <= 95		Verifies if the argon concentration is under 95%
14	E<> CM.mAN_ON		Verifies if the state mAN_ON at CM is reached
15	A[] CM.mAN_ON imply VB_ARGON_ON == 1		Verifies if at the state mAN_ON the variable VB_ARGON_ON is always 1
16	A[] CM.mAN_ON imply AR.oN		Verifies if at the state mAN_ON the argon is always on
17	A[] CM.mAN_ON imply VB_ARGON <= 95		Verifies if the argon concentration is under 95%
18	E<> CM.sTOP_MAN		Verifies if the state sTOP_MAN at CM is reached
19	A[] CM.mAN_ON imply VB_ARGON_ON == 1		Verifies if at the state sTOP_MAN the variable VB_ARGON_ON is always 0
20	A[] CM.mAN_ON imply AR.oN		Verifies if at the state sTOP_MAN the argon is always off
21	A[] CM.mAN_ON imply VB_ARGON <= 95		Verifies if the argon concentration is under 95%
22	E<> CM.fEED_OFF_MAN		Verifies if the state fEED_OFF_MAN at CM is reached
23	A[] CM.fEED_OFF_MAN imply VB_ARGON_ON == 0		Verifies if at the state fEED_OFF_MAN the variable VB_ARGON_ON is always 0
24	A[] CM.fEED_OFF_MAN imply AR.oFF		Verifies if at the state fEED_OFF_MAN the argon is always off
25	A[] CM.fEED_OFF_MAN imply VB_ARGON <= 95		Verifies if the argon concentration is under 95%
26	E<> CM.f1		Verifies if the state f1 at CM is reached
27	E<> CM.f2		Verifies if the state f2 at CM is reached

Anexos

ANEXO A – Requisitos - Plataforma

O cliente forneceu os requisitos para o funcionamento da plataforma. O documento de requisitos dos outros subsistemas desse trabalho não foram fornecidos pelo cliente.

Documento original fornecido pelo cliente, em inglês.

The necessary requirements to develop the software are described below. The models and the software should meet these requirements, in order to the system be validated.

R1 - The computer must process every command.

R2 - The processing time must be lower than 1 second.

R3 - The computer must send a response for every received command.

R3.1 - The time to be sent must be lower than 5 seconds.

R3.2 - It must be written in a log file in "XML"format.

R3.2.1 - The first field must be the date of the response.

R3.2.2 - The second field must be the hour of the response.

R3.2.3 - The other fields must be in accordance with the protocol described in 1.

R4 - The platform height minimum shall be 400 mm.

R5 - The platform height maximum shall be 0 mm.

R6 - The work distance must be of 300 mm.

R6.1 - The upper limit sensor must be placed at height 50mm.

R6.2 - The lower limit sensor must be placed at height 350mm.

R7 - The command STOP must have the biggest priority

R8 - The platform must stop when it reaches the upper limit sensor.

R9 - The platform must stop when it reaches the lower limit sensor.

R10 - When in manual operation mode, the machine shall only respond to the manual events from GUI.

R10.1 - If the platform is stopped, the controller must verify its position before responding to the commands.

R10.1.1 - If the platform position is lower than the lower limit, the platform is lower than the bottom.

R10.1.1.1 - The controller must inform a failure number 1.

R10.1.1.2 - The severity is considered 5.

R10.1.1.3 - The platform state is at the bottom.

R10.1.2 - If the platform position is higher than the upper limit, the platform is higher than the top.

R10.1.2.1 - The controller must inform a failure number 2.

R10.1.2.2 - The severity is considered 5.

R10.1.2.3 - The platform state is at the top.

R10.1.3 - If it becomes the upper limit signal, the platform is at the top.

R10.1.4 - If it becomes the lower limit signal, the platform is at the bottom.

R10.1.5 - If it does not become any of the signal, the platform is in the middle.

R10.1.6 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to the value "0".

R10.1.7 - If any other command is sent to the controller, it must ignore them.

R10.1.7.1 - The platform must remain at the same place.

R10.1.7.2 - The motor must remain stopped.

R10.2 - If the platform is at the bottom, the controller must wait for the commands from the GUI to begin any motion.

R10.2.1 - In this state, the motor must be stopped.

R10.2.2 - The only allowed motion is in the up direction.

R10.2.3 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.2.3.1 - The controller must inform a failure number 3.

R10.2.3.2 - The severity is considered 5.

R10.2.3.3 - The platform must remain at the same place.

R10.2.3.4 - The motor must remain stopped.

R10.2.4 - If the platform position is higher than the upper limit, it is considered an undesired situation (probable analogic sensor error).

R10.2.4.1 - The controller must inform a failure number 4.

R10.2.4.2 - The severity is considered 7.

R10.2.4.3 - The platform must remain at the same place.

R10.2.4.4 - The motor must remain stopped.

R10.2.4.5 - The system must go to a general failure state.

R10.2.5 - If the controller receives the signal from the up limit sensor, it is considered an undesired situation (probable up limit digital sensor error).

R10.2.5.1 - The controller must inform a failure number 5.

R10.2.5.2 - The severity is considered 7.

R10.2.5.3 - The platform must remain at the same place.

R10.2.5.4 - The motor must remain stopped.

R10.2.5.5 - The system must go to a general failure state.

R10.2.6 - If the controller receives a low level of the signal from the lower limit sensor, it is considered an undesired situation (probable down limit digital sensor error or the platform went down).

R10.2.6.1 - The controller must inform a failure number 6.

R10.2.6.2 - The severity is considered 8.

R10.2.6.3 - The platform must remain at the same place.

R10.2.6.4 - The motor must remain stopped.

R10.2.6.5 - The system must go to a general failure state.

R10.2.7 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to the value "0".

R10.2.8 - If the controller receives the command to go up from the GUI, the platform must move in the up direction.

R10.2.9 - If any other command is sent to the controller, it must ignore them.

R10.2.9.1 - The platform must remain at the same place.

R10.2.9.2 - The motor must remain stopped.

R10.3 - If the platform is at the top, the controller must wait for the commands from the GUI to begin any motion.

R10.3.1 - In this state, the motor must be stopped.

R10.3.2 - The only allowed motion is in the down direction.

R10.3.3 - If the platform position is higher than the up limit, it is considered an undesired situation.

R10.3.3.1 - The controller must inform a failure number 7.

R10.3.3.2 - The severity is considered 5.

R10.3.3.3 - The platform must remain at the same place.

R10.3.3.4 - The motor must remain stopped.

R10.3.4 - If the platform position is lower than the lower limit, it is considered an undesired situation (probable analogic sensor error).

R10.3.4.1 - The controller must inform a failure number 8.

R10.3.4.2 - The severity is considered 7.

R10.3.4.3 - The platform must remain at the same place.

R10.3.4.4 - The motor must remain stopped.

R10.3.4.4 - The system must go to a general failure state.

R10.3.5 - If the controller receives the high signal from the lower limit sensor, it is considered an undesired situation (probable up limit digital sensor error).

R10.3.5.1 - The controller must inform a failure number 9.

R10.3.5.2 - The severity is considered 7.

R10.3.5.3 - The platform must remain at the same place.

R10.3.5.4 - The motor must remain stopped.

R10.3.5.4 - The system must go to a general failure state.

R10.3.6 - If the controller receives a low level signal from the upper limit sensor, it is considered an undesired situation (probable up limit digital sensor error or the platform went up).

R10.3.6.1 - The controller must inform a failure number 10.

R10.3.6.2 - The severity is considered 9.

R10.3.6.3 - The platform must remain at the same place.

R10.3.6.4 - The motor must remain stopped.

R10.3.6.4 - The system must go to a general failure state.

R10.3.7 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.3.8 - If the controller receives the command to go down from the GUI, the platform must move in the down direction.

R10.3.9 - If any other command is sent to the controller, it must ignore them.

R10.3.9.1 - The platform must remain at the same place.

R10.3.9.2 - The motor must remain stopped.

R10.4 - If the platform is in the middle, the controller must wait for the commands from the GUI to begin any motion.

R10.4.1 - In this state, the motor must be stopped.

R10.4.2 - The platform is allowed to go down.

R10.4.3 - The platform is allowed to go up.

R10.4.4 - If the controller receives a high signal from the lower limit sensor, it is considered an undesired situation (probable interference or failure in the digital sensor).

R10.4.4.1 - The controller must inform a failure number 11.

R10.4.4.2 - The severity is considered 8.

R10.4.4.3 - The platform must remain at the same place.

R10.4.4.4 - The motor must remain stopped.

R10.4.4.5 - The system must go to a general failure state.

R10.4.5 - If the controller receives a signal from the up limit sensor, it is considered an undesired situation (probable interference or failure in the digital sensor).

R10.4.5.1 - The controller must inform a failure number 12.

R10.4.5.2 - The severity is considered 8.

R10.4.5.3 - The platform must remain at the same place.

R10.4.5.4 - The motor must remain stopped.

R10.4.5.5 - The system must go to a general failure state.

R10.4.6 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.4.6.1 - The controller must inform a failure number 13.

R10.4.6.2 - The severity is considered 7.

R10.4.6.3 - The platform must remain at the same place.

R10.4.6.4 - The motor must remain stopped.

R10.4.6.5 - The system must go to a general failure state.

R10.4.7 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.4.7.1 - The controller must inform a failure number 14.

R10.4.7.2 - The severity is considered 7.

R10.4.7.3 - The platform must remain at the same place.

R10.4.7.4 - The motor must remain stopped.

R10.4.7.5 - The system must go to a general failure state.

R10.4.8 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.4.9 - If the controller receives the command to go up from the GUI, the platform must move in the up direction.

R10.4.10 - If the controller receives the command to go down from the GUI, the platform must move in the down direction.

R10.4.11 - If any other command is sent to the controller, it must ignore them.

R10.4.11.1 - The platform must remain at the same place.

R10.4.11.2 - The motor must remain stopped.

R10.5 - If the platform is moving down, the controller must wait for the commands from the GUI.

R10.5.1 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.5.1.1 - The controller must inform a failure number 27.

R10.5.1.2 - The severity is considered 9.

R10.5.1.3 - The platform must send a command to stop immediately.

R10.5.1.4 - The system must go to a specific failure state going down, in order to wait the feedback from the motor.

R10.5.2 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.5.2.1 - The controller must inform a failure number 28.

R10.5.2.2 - The severity is considered 5.

R10.5.2.3 - The platform must remain going down.

R10.5.3 - If the controller receives a signal from the up limit sensor, it is considered an undesired situation (probable interference or failure in the digital sensor).

R10.5.3.1 - The controller must inform a failure number 29.

R10.5.3.2 - The severity is considered 5.

R10.5.3.3 - The platform must remain going down.

R10.5.4 - If the controller receives a feedback from the motor that it is already stopped, it is considered an undesired situation.

R10.5.4.1 - The controller must inform a failure number 30.

R10.5.4.2 - The severity is considered 10.

R10.5.4.3 - The system must be in the stop state.

R10.5.5 - If the controller receives the command to go up from the GUI, the platform must stop.

R10.5.5.1 - The controller must inform a failure number 31.

R10.5.5.2 - The severity is considered 9.

R10.5.5.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.5.6 - If the controller receives a high signal from the lower limit sensor, the platform must stop.

R10.5.6.1 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.5.7 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.5.8 - If the controller receives the command to stop from the GUI, the platform must stop.

R10.5.8.1 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.5.9 - If any different commands are sent to the controller, it must ignore them.

R10.5.9.1 - The platform must remain going down.

R10.6 - If the platform is moving up, the controller must wait for the commands from the GUI.

R10.6.1 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.6.1.1 - The controller must inform a failure number 33.

R10.6.1.2 - The severity is considered 9.

R10.6.1.3 - The platform must send a command to stop immediately.

R10.6.1.4 - The system must go to a specific failure state going up, in order to wait the feedback from the motor.

R10.6.2 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.6.2.1 - The controller must inform a failure number 32.

R10.6.2.2 - The severity is considered 5.

R10.6.2.3 - The platform must remain going up.

R10.6.3 - If the controller receives a high signal from the lower limit sensor, it is considered an undesired situation (probable interference or failure in the digital sensor).

R10.6.3.1 - The controller must inform a failure number 34.

R10.6.3.2 - The severity is considered 5.

R10.6.3.3 - The platform must remain going up.

R10.6.4 - If the controller receives a feedback from the motor that it is already stopped, it is considered an undesired situation.

R10.6.4.1 - The controller must inform a failure number 35.

R10.6.4.2 - The severity is considered 10.

R10.6.4.3 - The system must be in the stop state.

R10.6.5 - If the controller receives the command to go down from the GUI, the platform must stop.

R10.6.5.1 - The controller must inform a failure number 36.

R10.6.5.2 - The severity is considered 9.

R10.6.5.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.6.6 - If the controller receives a signal from the up limit sensor, the platform must stop.

R10.6.6.1 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.6.7 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.6.8 - If the controller receives the command to stop from the GUI, the platform must stop.

R10.6.8.1 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.6.9 - If any different commands are sent to the controller, it must ignore them.

R10.6.9.1 - The platform must remain going up.

R10.7 - When in the general failure state, the controller must wait only for the acknowldge command from the GUI.

R10.7.1 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.7.2 - If any different commands or events are sent to the controller, it must ignore them.

R10.8 - When in the failure state going down, the controller must wait only for the feedback from the motor, to confirm that it has been stopped.

R10.8.1 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.8.1.1 - The controller must inform a failure number 15.

R10.8.1.2 - The severity is considered 7.

R10.8.1.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.8.2 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.8.2.1 - The controller must inform a failure number 16.

R10.8.2.2 - The severity is considered 2.

R10.8.2.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.8.3 - If the feedback from the motor is received, the system shall go to the general failure state.

R10.8.4 - If any different commands or events are sent to the controller, it must ignore them.

R10.9 - When in the failure state going up, the controller must wait only for the feedback from the motor, to confirm that it has been stopped.

R10.9.1 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.9.1.1 - The controller must inform a failure number 17.

R10.9.1.2 - The severity is considered 2.

R10.9.1.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.9.2 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.9.2.1 - The controller must inform a failure number 18.

R10.9.2.2 - The severity is considered 7.

R10.9.2.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.9.3 - If the feedback from the motor is received, the system shall go to the general failure state.

R10.9.4 - If any different commands or events are sent to the controller, it must ignore them.

R10.10 - When the platform is waiting for the feedback from motor to confirm that is has stopped in a normal situation, the controller must wait for the feedback from the motor only.

R10.10.1 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.10.1.1 - The controller must inform a failure number 38.

R10.10.1.2 - The severity is considered 5.

R10.10.1.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.10.2 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.10.2.1 - The controller must inform a failure number 37.

R10.10.2.2 - The severity is considered 5.

R10.10.2.3 - The platform must wait the feedback from the motor to confirm that it has been stopped.

R10.10.3 - If the feedback from the motor is received, the system shall go to a stop state

R10.10.4 - If any different commands or events are sent to the controller, it must be ignore them.

R10.10.5 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.11 - When the system is supposed to go up, the controller must wait for the feedback from the motor to confirm that it is going up.

R10.11.1 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.11.1.1 - The controller must inform a failure number 26.

R10.11.1.2 - The severity is considered 8.

R10.11.1.3 - The platform must stop.

R10.11.1.4 - The system must go to a specific failure state going up, in order to wait the feedback from the motor.

R10.11.2 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.11.2.1 - The controller must inform a failure number 25.

R10.11.2.2 - The severity is considered 10.

R10.11.2.3 - The platform must stop.

R10.11.2.4 - The system must go to a specific failure state going down, in order to wait the feedback from the motor.

R10.11.3 - If the controller receives the feedback that the motor is going down, it is considered an undesired situation.

R10.11.3.1 - The controller must inform a failure number 24.

R10.11.3.2 - The severity is considered 10.

R10.11.3.3 - The platform must stop.

R10.11.3.4 - The system must go to a specific failure state going down, in order to wait the feedback from the motor.

R10.11.4 - If the controller receives the feedback that the motor is going up, the system must remain going up.

R10.11.5 - If the controller receives the feedback that the motor is stopped, it is considered an undesired situation.

R10.11.5.1 - The controller must inform a failure number 23.

R10.11.5.2 - The severity is considered 10.

R10.11.5.3 - The platform must go to the the stop state.

R10.11.6 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.11.7 - If any different commands or events are sent to the controller, it must ignore them.

R10.12 - When the system is supposed to go down, the controller must wait for the feedback from the motor to confirm that it is going down.

R10.12.1 - If the platform position is higher than the upper limit, it is considered an undesired situation.

R10.12.1.1 - The controller must inform a failure number 21.

R10.12.1.2 - The severity is considered 10.

R10.12.1.3 - The platform must stop.

R10.12.1.4 - The system must go to a specific failure state going up, in order to wait the feedback from the motor.

R10.12.2 - If the platform position is lower than the lower limit, it is considered an undesired situation.

R10.12.2.1 - The controller must inform a failure number 20.

R10.12.2.2 - The severity is considered 8.

R10.12.2.3 - The platform must stop.

R10.12.2.4 - The system must go to a specific failure state going down, in order to wait the feedback from the motor.

R10.12.3 - If the controller receives the feedback that the motor is going up, it is considered an undesired situation.

R10.12.3.1 - The controller must inform a failure number 19.

R10.12.3.2 - The severity is considered 10.

R10.12.3.3 - The platform must stop.

R10.12.3.4 - The system must go to a specific failure state going down, in order to wait the feedback from the motor.

R10.12.4 - If the controller receives the feedback that the motor is going down, the system must remain going down.

R10.12.5 - If the controller receives the feedback that the motor is stopped, it is considered an undesired situation.

R10.12.5.1 - The controller must inform a failure number 22.

R10.12.5.2 - The severity is considered 10.

R10.12.5.3 - The platform must go to the the stop state.

R10.12.6 - If the controller receives the failure acknowledge from the GUI, the failure number must be set to "0".

R10.12.7 - If any different commands or events are sent to the controller, it must ignore them.