UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Engenharia de Computação

FERRAMENTA DE CONVERSÃO DE MODELOS DE SISTEMAS DE AUTOMAÇÃO EM REDE DE PETRI PARA LINGUAGEM LADDER

EDUARDO LOPES FREIRE

RA: 759025

SÃO CARLOS - SP

FEVEREIRO, 2025

CONVERSÃO E VALIDAÇÃO DE MODELOS EM REDE DE PETRI DE SISTEMAS AUTOMATIZADOS PARA LINGUAGEM LADDER

EDUARDO LOPES FREIRE

Monografia apresentada ao Departamento de Computação da Universidade Federal de São Carlos como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação

Orientador: Edilson Reis Rodrigues Kato

SÃO CARLOS - SP

FEVEREIRO, 2025

Dedicado aos meus pais Rogério e Eloisa,

ao meu irmão Rogério Júnior e a todos os

amigos que me ajudaram no caminho.

Resumo

Controladores Lógicos Programáveis (CLPs) são utilizados extensivamente na automatização de processos industriais a fim de melhorar a eficiência, qualidade e segurança de fluxos de trabalho sequenciais. Com o aumento da complexidade dos sistemas a serem automatizados, a programação das CLPs se tornou progressivamente mais difícil de ser aprendida, implementada, mantida e construída sobre. Como resposta a esse aumento da complexidade, foram desenvolvidos métodos para a modelagem desses sistemas, implementando-os em linguagens de programação padrão, podendo ser adaptados posteriormente para qualquer CLP. A linguagem mais popular utilizada para a programação de CLPs é a Linguagem em Diagramas Ladder que, apesar de simplificar o entendimento do sistema, pela conversão na maioria das vezes ser feita de forma intuitiva e manual, ainda pode trazer efeitos inesperados por erros de interpretações humanas. Este trabalho tem como objetivo recuperar, adaptar e complementar uma solução de conversor de modelos de sistemas automatizados para linguagem Ladder (proposto por Rodrigo, 2018), tornando o processo automático. O texto aborda o processo de conversão utilizado (proposto por Kato, 2023), as ferramentas auxiliares necessárias para o fluxo de funcionamento do sistema e as decisões tomadas para recuperar o funcionamento do sistema, expandir suas funcionalidades e executá-lo de forma simples.

Palavras-chave: Redes de Petri, diagrama Ladder, linguagem Ladder, CLP, controladores, automação industrial.

Lista de abreviaturas e siglas

CLP - Controlador Lógico Programável

PNML - *Petri Net Markup Language*

LD - *Ladder Diagram*/Diagrama Ladder

IEC - *International Electrotechnical Commission*

POU - *Program Organization Unit*

XML - *Extensible Markup Language*

RP - Rede de Petri

IDE - *Integrated Development Environment*

PIPE - *Platform Independent Petri Net Editor*

Sumário

1. Introdução

Um Controlador Lógico Programável (CLP) é um computador industrial fundamental para os processos de controle de grandes manufaturas automatizadas. A automação industrial desempenha um papel fundamental na melhoria dos processos produtivos, buscando aumentar a eficiência, qualidade e segurança nas indústrias. Com avanços no uso dos CLPs, vários sistemas microprocessados começaram a ser produzidos por empresas prestadoras de serviços em diversas áreas como de alimentos, química, etc, introduzindo novas funcionalidades ao controlador tanto em elementos de controle como na forma de programação, surgindo a linguagem de programação Ladder, que representa computacionalmente a linguagem de projetos elétricos (KATO, 2023).

A programação do CLP se baseia em lógica digital e em lógica de relés, utilizando também sensores e atuadores tanto digitais como analógicos. Com o aumento da complexidade das necessidades atuais, programas para CLPs se tornaram mais difíceis de se desenvolver, manter, utilizar do código em outros projetos, analisar, encontrar fontes de erros e integrar, diminuindo sua flexibilidade. A fim de implementar programas para CLPs de forma mais fácil e prática se tornaram essenciais ferramentas auxiliares à produção dos mesmos, como linguagens de programação estruturadas, formas de modelagem gráfica, etc.

Uma abordagem promissora nesse campo é o uso da modelagem em Redes de Petri, um modelo gráfico e matemático, junto com técnicas de representação de sistemas sequenciais para o projeto de sistemas automáticos. Essa modelagem visual, apesar de não reconhecida diretamente pelos CLPs, facilita a compreensão e análise dos sistemas automatizados. A partir desses modelos, é possível aplicar algoritmos (KATO, 2023), para transformá-los em diagramas Ladder compatíveis que serão aceitos pelos CLPs.

Dentre as várias possíveis formas de se aplicar esses algoritmos de forma automática estão separar as etapas (gerar Rede de Petri, converter e visualizar o resultado) e utilizar software dedicados como PIPE (Redes de Petri) e Beremiz (Diagrama Ladder), usar uma Engine como auxílio para a aplicação da transformação e utilizar software auxiliares para o processo (RODRIGO, 2018).

1.1 Objetivo

Esse trabalho tem por objetivo implementar uma ferramenta de conversão de modelos em Rede de Petri de sistemas automáticos em linguagem de programação CLP Ladder. A linguagem Ladder gerada como um pseudo-código poderá ser adaptada para CLPs comerciais de forma mais fácil e prática.

1.2 Motivação

A programação de CLPs utilizando linguagem Ladder, embora amplamente difundida no ambiente industrial, apresenta desafios significativos ao se propor soluções para projetos complexos ou que exigem alta flexibilidade. Além disso, a manutenção e a depuração de programas em Ladder podem ser árduas em projetos de grande escala, onde o aumento no número de linhas e de elementos compromete a legibilidade e facilidade de análise e implementação de códigos.

Buscando explorar essas limitações e propor soluções para otimizar o desenvolvimento de programas para CLPs sem reduzir as possibilidades de implementação, foram desenvolvidos modelos de conversão que, com o uso de alguma ferramenta de modelagem que representa de forma gráfica de fácil entendimento o sistema automatizado, transcreve de forma metódica o modelo gerado para uma linguagem de programação de um CLP.

No entanto, tal transcrição realizada de forma manual requer uma grande quantidade de tempo e ainda se dispõe a erros de ações e interpretações humanas. Este trabalho busca, através do fornecimento de uma solução para converter de forma automática modelos de sistemas automatizados em Rede de Petri para linguagem Ladder, diminuir os atritos encontrados no processo de programação de CLPs.

1.3 Organização do Trabalho

Este trabalho foi organizado em cinco partes:

1. Contextualização do problema, incluindo objetivos, motivação e organização do trabalho;
2. Conceitos básicos dos principais elementos necessários para implementar esta proposta;
3. Revisão bibliografica sobre conversores similares;
4. Proposta do sistema, metodologia de implementação, materiais/ferramentas utilizadas, a forma de validação do sistema e o fluxo de execução;
5. Apresenta os resultados da implementação e a validação do sistema;
6. Apresenta as considerações finais e sugestões de trabalhos futuros.

2. Conceitos Básicos

Este capítulo apresenta os conceitos que serão utilizados neste trabalho. A Seção 2.1 descreve as Redes de Petri, seus tipos e um padrão de especificação padrão para manipular as redes computacionalmente. A Seção 2.2 descreve CLPs e o algoritmo que converte o sistema modelado em Redes de Petri em um código em Diagrama Ladder (KATO, 2023). A Seção 2.3 introduz as ferramentas utilizadas para operar os arquivos do trabalho (PIPE, Beremiz…).

2.1 Redes de Petri

Redes de Petri são ferramentas robustas e versáteis para modelagem, análise e simulação de sistemas dinâmicos discretos (MURATA, 1989). Essas redes são compostas por cinco elementos que permitem representar de forma clara e precisa a estrutura e o comportamento de sistemas concorrentes, distribuídos e paralelos. Os componentes são lugares, representados com círculos, transições, com barras, arcos e marcas (conectando dois componentes), com setas, e marcações, com pontos pretos dentro de lugares.

Definição: Uma Rede de Petri é uma tupla $PN = (P, T, F, W, M_0)$ no qual:

$P = \{ p_1, p_2, \ldots, p_n \}$ é o conjunto finito de lugares,

$T = \{ t_1, t_2, \ldots, t_m \}$ é o conjunto finito de transições,

$F = (P \times T) \cup (T \times P)$ é o conjunto de arcos,

$W: F \rightarrow \{ 1, 2, 3, \ldots \}$ é a função-peso,

$M_0: P \rightarrow \{ 0, 1, 2, \ldots \}$ é a marcação inicial,

$P \cap T = 0$ e $P \cup T \neq 0$

Os arcos podem ser separados em dois conjuntos:

• Arcos de entrada: $F_1 \rightarrow (P \times T)$, conjunto de arcos que conectam lugares em transições.

• Arcos de saída: $F_2 \rightarrow (T \times P)$, conjunto de arcos que conectam transições em lugares.

Uma transição t está habilitada quando, dado todos os seus arcos de entrada, cada lugar p definido nesses arcos possui uma quantidade de marcações maior ou igual a função-peso daquele arco. A dinâmica de um sistema segue disparos de transições habilitadas que alteram a marcação ou estado da rede. Uma vez que ocorra um disparo de transição, a quantidade da função-peso dos arcos de entrada serão removidos do local de origem e a quantidade dos arcos de saída serão adicionados no local correspondente. A Figura 1 ilustra uma situação de disparo de transição com funções-peso igual a 1, exemplificando a transmissão do token.

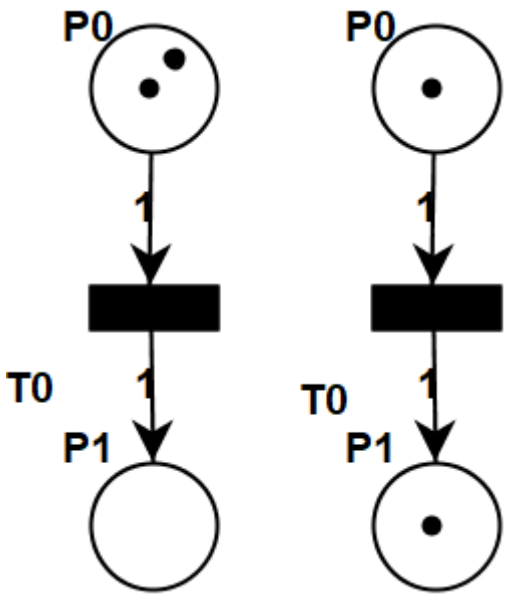


Figura 1 - Disparo de transição em uma Rede de Petri.

2.1.1 Rede binária

Uma rede binária é a representação mais básica de uma Rede de Petri, onde a função-peso é constante 1 e as marcações são binárias. A Figura 2 ilustra uma Rede de Petri Binária.

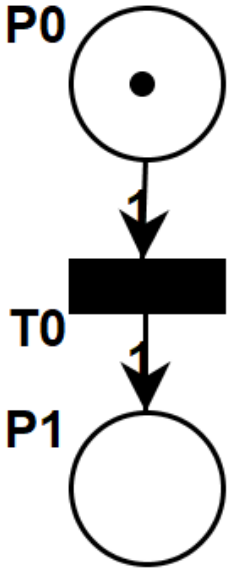


Figura 2 - Rede de Petri binária.

2.1.2 Rede de Petri Lugar-Transição

Outra rede considerada ordinária junto da rede binária, é caracterizada por possuir marcas do tipo inteiro mas, diferentemente das redes binárias, as Redes de Petri Lugar-Transição aceitam números inteiros maiores que 1 em suas marcações e funções-peso. A Figura 3 ilustra uma Rede de Petri Lugar-Transição.

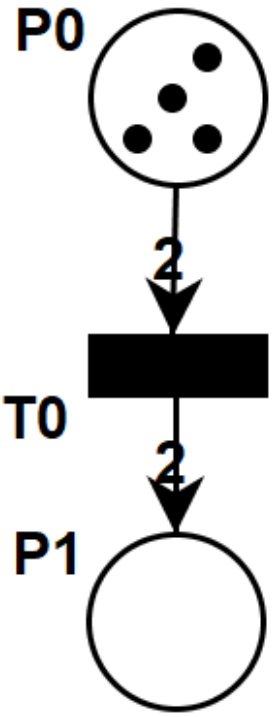


Figura 3 - Rede de Petri Lugar-Transição.

2.1.3 Rede de Petri Sincronizada

A Rede de Petri sincronizada permite descrever entradas e saídas de um sistema de forma bem definida, sendo então aplicada na modelagem de sistemas de controle (FREY, 2000). São redes caracterizadas por uma tupla (P, T, F, W, M_0, E, μ) de forma que a sub tupla (P, T, F, W, M_0) corresponda a uma Rede de Petri conforme descrito em 2.1, E seja um conjunto de eventos externos que influenciam no processo de controle e μ é o mapeamento das transições em T para os eventos E.

2.1.4 Petri Net Markup Language

A *Petri Net Markup Language* (PNML) é uma linguagem baseada em XML desenvolvida para padronizar a representação de modelos em Redes de Petri, facilitando a interoperabilidade entre diferentes ferramentas. O padrão foi definido pela ISO/IEC 15909-2 e foi estruturado de forma a abranger vários tipos de Redes de Petri, sendo possível estender o formato para novos tipos inseridos posteriormente.

Para que um documento PNML seja válido, é preciso que conforme com as especificações da *PNML Core Model*, da sintaxe XML e com as restrições associadas ao próprio tipo da rede; um documento PNML para uma rede Lugar-Transição, por exemplo, necessitaria que a rede tenha um nome, que um lugar na rede tenha um nome e uma marcação inicial e que os arcos sejam de transição para lugar ou de lugar para transição.

2.2 Controladores Lógico Programáveis

Um Controlador Lógico Programável (CLP) é um dispositivo que implementa funções lógicas, aritméticas, de temporização e de contagem para o controle, através de entradas e saídas com comunicação com o ambiente externo, de máquinas e processos. A Figura 4 mostra um exemplo de Controlador Lógico Programável.

Figura 4 - Exemplo de Controlador Lógico Programável.
Fonte: Kato (2023).

Com o surgimento dos CLPs, houve uma evolução significativa na automação industrial. Antes, os sistemas eram projetados com lógica de relés e circuitos elétricos, mas os CLPs revolucionaram o processo ao utilizar linguagens de programação específicas, oferecendo mais flexibilidade e facilidade de implementação.

A modelagem em Redes de Petri passou a ser amplamente usada para o desenvolvimento e análise de sistemas sequenciais, permitindo visualizar e verificar o comportamento do sistema antes de sua implementação, reduzindo riscos e custos associados a falhas na automação ao permitir ao projetista uma visão de alto nível do projeto. No entanto, ainda existem desafios na transcrição dos modelos de Redes de Petri para a linguagem de programação dos CLPs. É necessário estabelecer uma conexão eficiente entre a modelagem em Redes de Petri e a programação do CLP, garantindo a tradução adequada e a implementação correta no controlador. Assim, a evolução dos projetos com CLPs acompanhou o avanço da automação industrial, incorporando linguagens de programação específicas e o uso de modelagem como em Redes de Petri para melhor compreensão e análise dos sistemas automatizados.

As CLPs, de acordo com a norma IEC 61131-3, possuem cinco linguagens padrão: texto estruturado, diagrama de blocos funcionais (FBD), lista de instruções, Grafcet ou carta de função sequencial (SFC) e a mais popular delas devido à sua facilidade de uso para sistemas simples e representação intuitiva para projetistas, diagrama ladder (LD). As linguagens de programação de CLPs são frequentemente representadas utilizando formatos baseados em XML, sendo uma abordagem flexível e padronizada para facilitar a troca de informação entre sistemas e ferramentas de automação. Alguns exemplos de padrões são IEC 61131-10, um *schema* oficial da IEC para a IEC 61131-3, AutomationML e OMAC PackML, não desenvolvidos para CLPs mas que podem ser utilizados para as mesmas, e um mais utilizados, PLCOpen XML.

2.2.1 Diagrama Ladder

Um Diagrama Ladder é uma representação gráfica de variáveis booleanas representadas como se fossem circuitos elétricos. *Program Organization Units* (POUs) em Diagramas Ladder (LD) são divididos em seções conhecidas como redes (IEC 61131-3). O layout gráfico e as conexões de um diagrama Ladder determinam seu processamento, onde conexões indicam se os elementos estão conectados em paralelo ou em série. A Figura 4 ilustra três elementos conectados em paralelo.

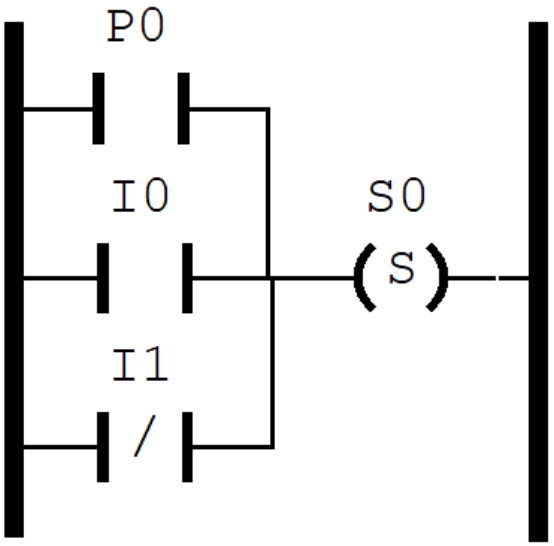


Figura 5 - Conexão em paralelo em Ladder.

Uma rede em Ladder contém quatro elementos: contatos, bobinas, elementos gráficos para sequências de execuções e elementos gráficos para chamadas de funções, os quais serão descritos abaixo.

Contatos são operações lógicas associadas a uma variável. A Tabela 1 descreve os tipos de contatos, com destaque para contatos abertos e fechados.

⊣ ⊢	Contato aberto	Valor igual ao da variável associada
⊣/⊢	Contato fechado	Valor oposto ao da variável associada
⊣P⊢	Contato de transição positiva	TRUE quando a variável mudou de FALSE para TRUE desde o último processamento e FALSE em caso contrário
⊣N⊢	Contato de transição negativa	TRUE quando a variável mudou de TRUE para FALSE desde o último processamento e FALSE em caso contrário

Tabela 1 - Tipos de contatos.

Bobinas (coils) são atuadores que aplicam o resultado calculado na conexão correspondente para uma variável associada. A Tabela 2 descreve os tipos de bobinas, com destaque para *set* e *reset*.

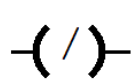
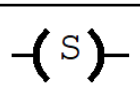
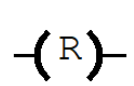
	Coil	Aplica o resultado calculado
	Negated Coil	Aplica o inverso do resultado calculado
	Set Coil	Se o resultado calculado for TRUE, aplica o valor TRUE, caso contrário não muda o valor da variável
	Reset Coil	Se o resultado calculado for TRUE, aplica o valor FALSE, caso contrário não muda o valor da variável

Tabela 2 - Tipos de bobinas.

Elementos gráficos para sequências de execuções e para chamadas de funções ou blocos de funções, que são opcionais subjetivos para a necessidade de cada aplicação, representando um processamento mais complexo que pode ser abstraído ou um componente completo que será adicionado ao projeto.

2.2.2 PLCOpen XML

O PLCOpen XML é um formato baseado em XML desenvolvido pela organização PLCOpen a fim de padronizar a representação e facilitar o intercâmbio de programas de CLPs sem perda de informação. O padrão possui uma estrutura que contém o projeto que, por sua vez, contém um cabeçalho, os tipos (contém o diagrama) e instâncias (contém a configuração do programa da CLP). Dentro da tag *types* são contidos os elementos que compõem o programa, sendo eles *leftPowerRail* e *rightPowerRail* (os trilhos da esquerda e direita), *contacts* (contatos) e *coils* (bobinas). A Figura 6 ilustra a estrutura de um arquivo no formato PLCOpen XML.

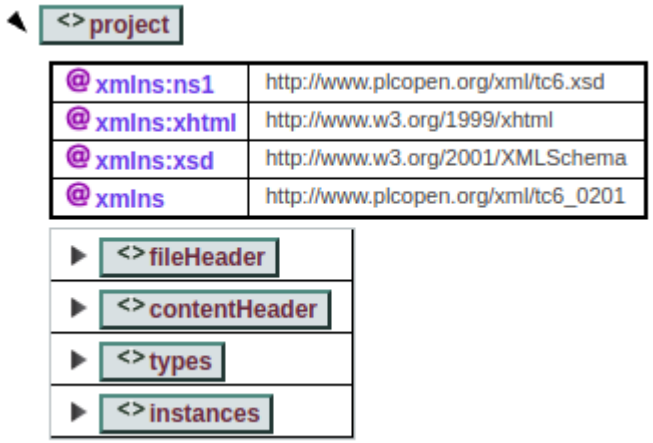


Figura 6 - Estrutura do formato PLCOpen XML.

Fonte: Rodrigo (2018).

2.3 Modelagem de sistemas automáticos com Redes de Petri e conversão para diagramas Ladder

Uma solução para a tradução dos modelos em Redes de Petri para diagramas ladder foi introduzida na forma de um algoritmo (KATO, 2023) que nos permite realizar tal transformação de forma manual ou automática. Para a metodologia proposta, serão utilizadas Redes de Petri Lugar-Transição, aproveitando de sua capacidade de representar todos os elementos básicos de um CLP além de contadores e temporizadores. A Figura 7 ilustra as etapas e suas respectivas descrições do fluxo da metodologia.

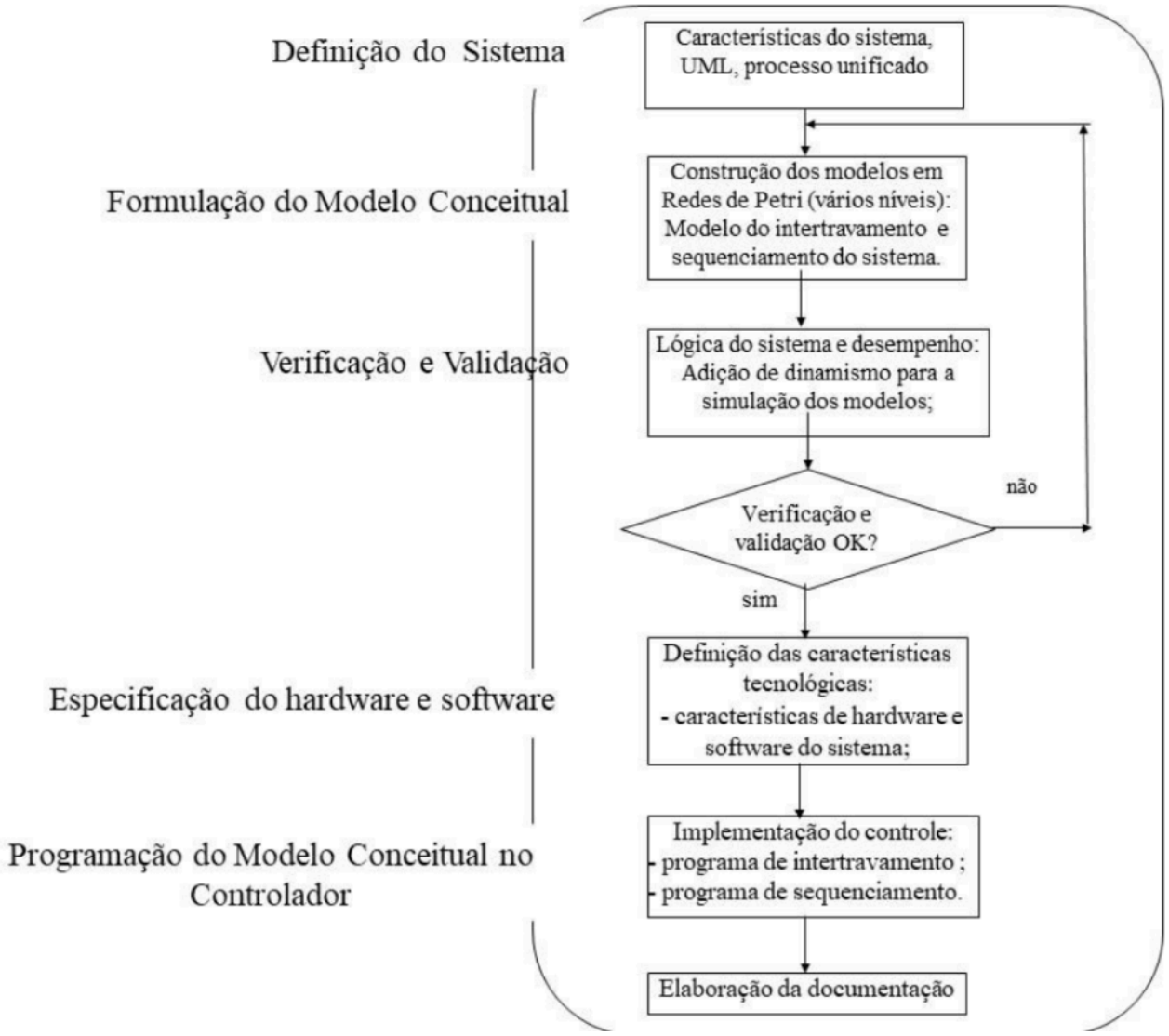


Figura 7 - Ilustração do fluxo da metodologia de projeto de sistemas automatizados utilizando CLP.

Fonte: Kato (2023).

O método contempla etapas de modelagem, implementação, validação e análise de forma iterativa seguindo cinco etapas bem definidas:

2.3.1 Definição do sistema

Define-se de forma clara e criteriosa as informações do projeto de acordo com as necessidades e objetivos do sistema, geralmente com o uso de ferramentas como Diagramas

de Caso de Uso, estabelecendo funções do sistema e conexões externas de acordo com os possíveis fluxos da linha de produção. A Figura 8 ilustra um diagrama de casos de uso contendo todos os possíveis casos de uso para um exemplo de fluxo de produção.

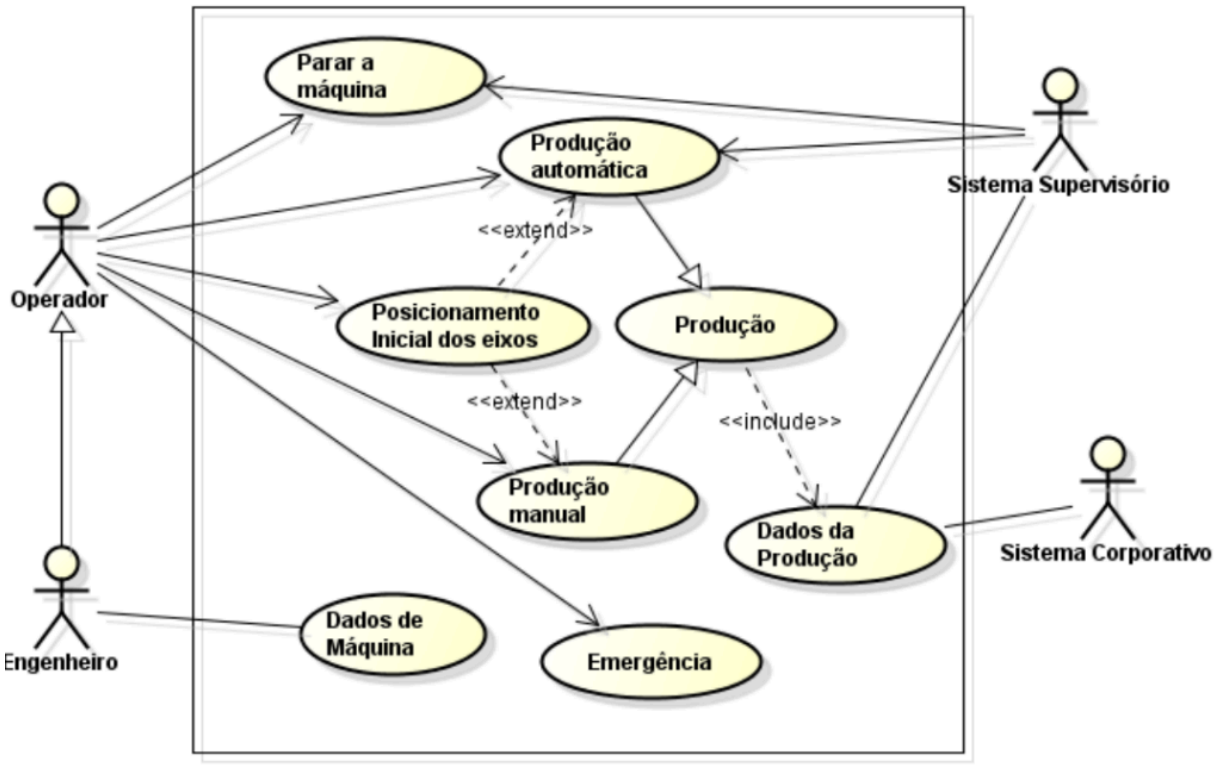


Figura 8 - Exemplo de diagrama de caso de uso.
Fonte: Kato (2023).

Cada elipsóide no diagrama é um caso de uso específico que será detalhado por todas as partes envolvidas no projeto, podendo ser registrados casos de uso alternativos para a mesma ação caso necessário. A Figura 9 ilustra dois dos possíveis casos de uso, o curso normal de produção do exemplo e um curso alternativo onde há falha de carregamento.

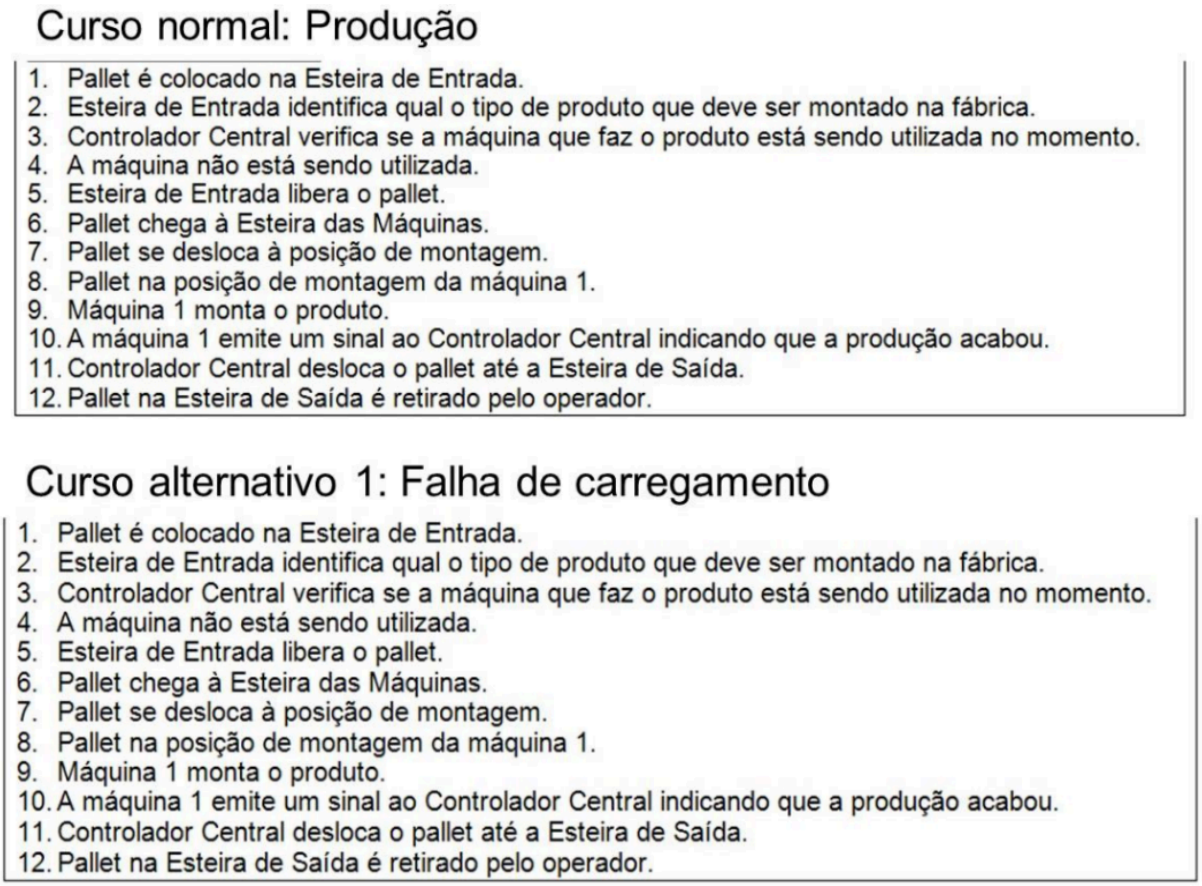


Figura 9 - Exemplo de caso de uso.

Fonte: Kato (2023).

Através dos casos de uso é obtido o diagrama de sequência, permitindo o detalhamento de conexões entre elementos do sistema, trocas de mensagens onde necessário e do fluxo de execução. A Figura 10 ilustra um fluxo de execução de um exemplo de projeto.

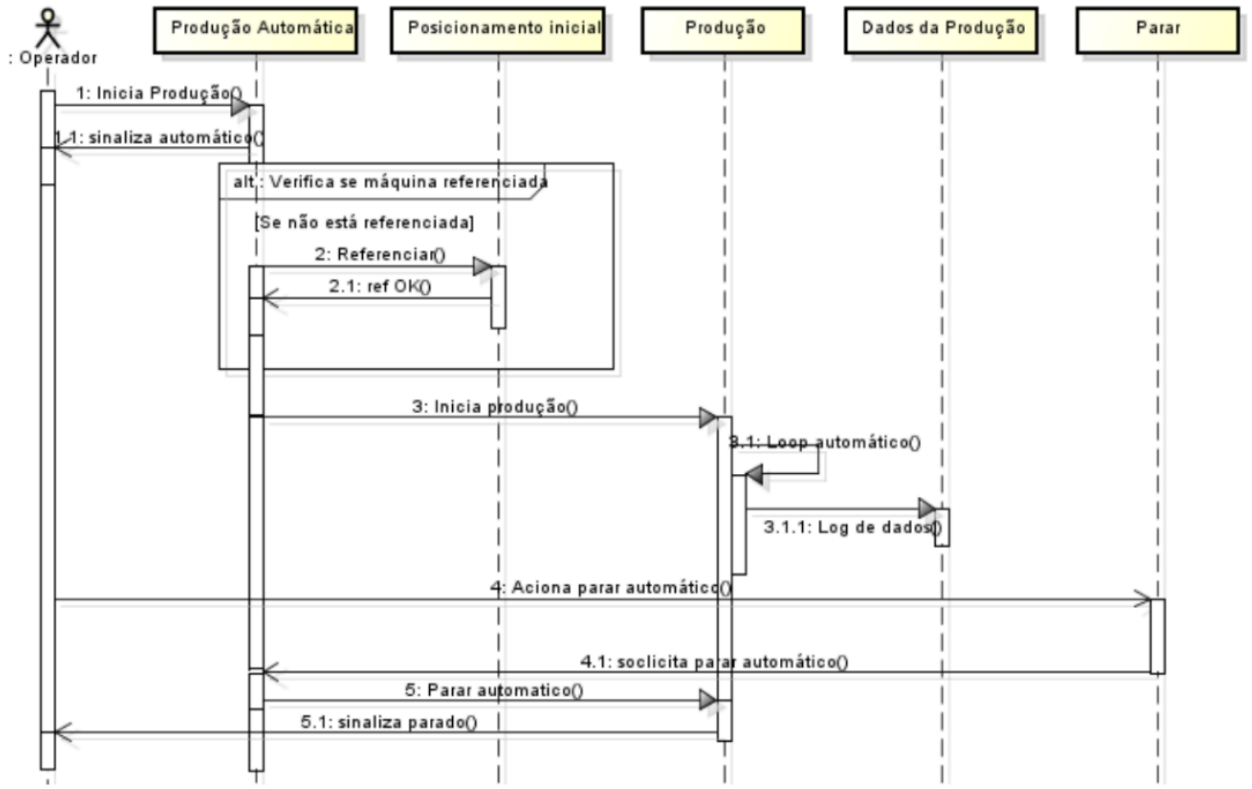


Figura 10 - Exemplo de diagrama de sequência.

Fonte: Kato (2023).

2.3.2 Formulação do modelo conceitual

Modela-se, utilizando Redes de Petri e de forma que os lugares representam estados e as transições, ações de mudança de estado, as funcionalidades e intertravamentos do sistema utilizando como base os diagramas estabelecidos na etapa anterior. Este modelo será iterado sobre nessa etapa, passando por diversos níveis de abstração e sendo refinado até representar uma tradução direta do diagrama possível de ser simulada e implantada em um sistema de automação. A Figura 11 ilustra um modelo conceitual em Rede de Petri em alto nível, na primeira iteração, e a Figura 12 ilustra o mesmo modelo após uma série de iterações, já pronto para conversão direta para diagrama Ladder.

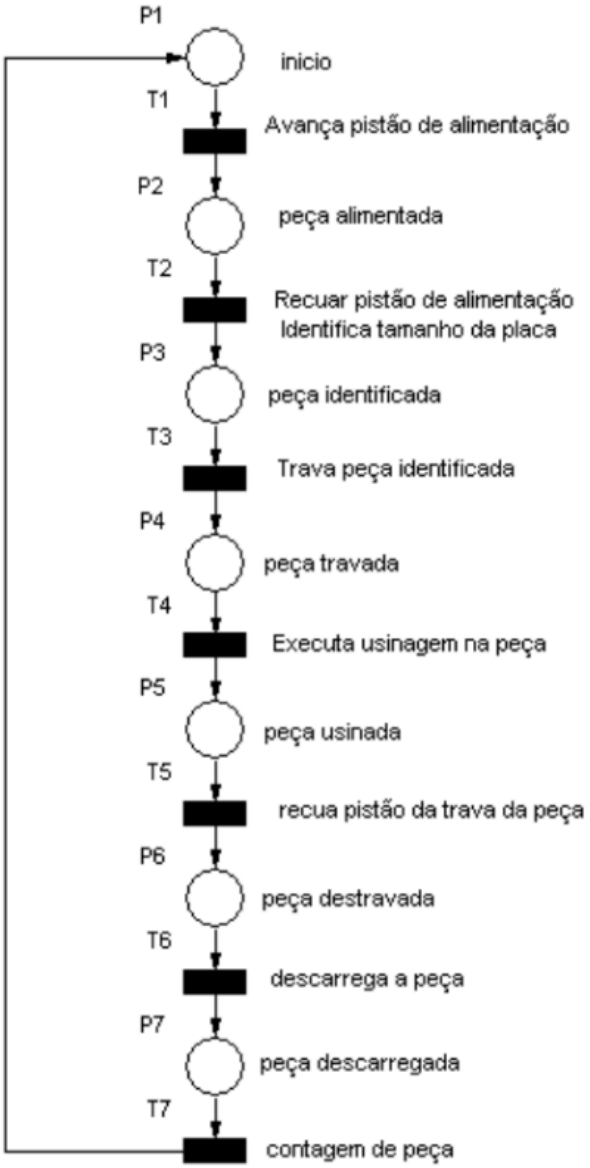


Figura 11 - Exemplo de modelo em alto nível de abstração.

Fonte: Kato (2023).

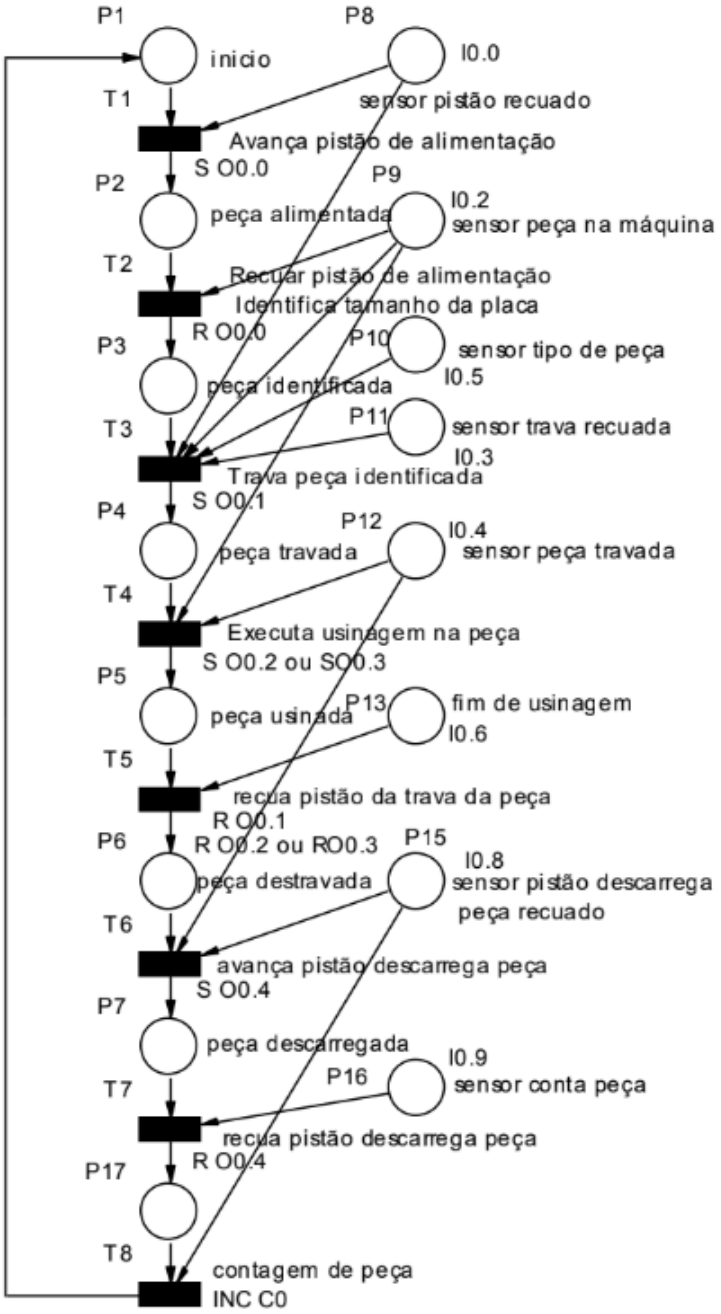


Figura 12 - Exemplo de modelo em baixo nível de abstração.

Fonte: Kato (2023).

2.3.3 Verificação e validação do modelo do sistema de automação

Consiste em analisar o modelo conceitual estabelecido em Redes de Petri na etapa anterior, verificando falhas na modelagem e na dinâmica de funcionamento e analisando o desempenho através da simulação da sequência de funcionamento do sistema.

2.3.4 Especificação do hardware e software

Nesta etapa são estabelecidos os elementos tecnológicos necessários para a execução do projeto, sendo levantadas as entradas e saídas do sistema, os componentes a serem usados nestas entradas e saídas e seus tipos (digital, analógico, flag, mecânico) e o próprio CLP a ser utilizado para implementação. A Figura 13 ilustra um exemplo de lista de entradas e saídas.

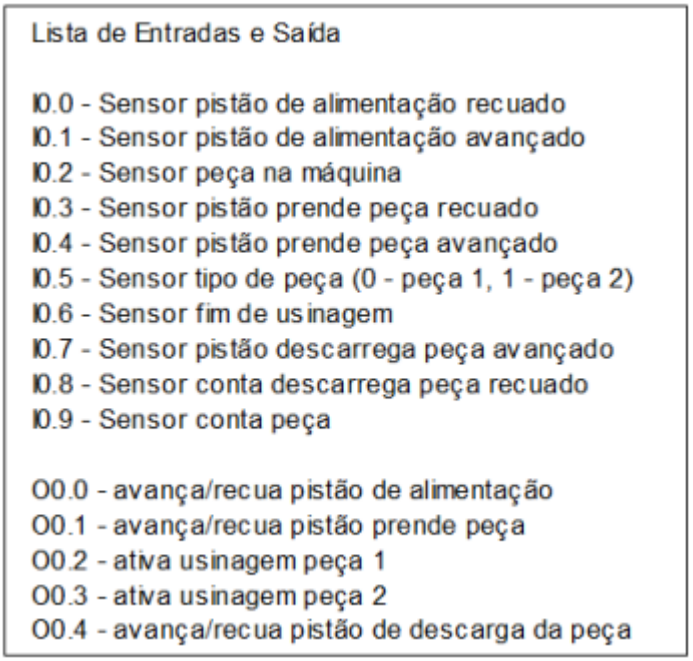


Figura 13 - Exemplo de lista de entradas e saídas de um sistema.
Fonte: Kato (2023).

2.3.5 Programação do modelo conceitual no controlador

Converte-se o modelo em Redes de Petri no menor nível de abstração alcançado para a linguagem de programação do CLP de forma mais direta possível utilizando a relação entre lugares e sinais de entrada ou flags, transições e os acionamentos das saídas e regras de conversão baseadas em lógica E e OU.

As regras de conversão consistem em associar as lógicas E e OU do CLP com sua representação em Redes de Petri e construir um diagrama Ladder correspondente, de forma a

conter uma transição ou ação de mudança de estado por linha. A Figura 14 ilustra as regras de associação utilizadas para associar as Redes de Petri à diagramas Ladder.

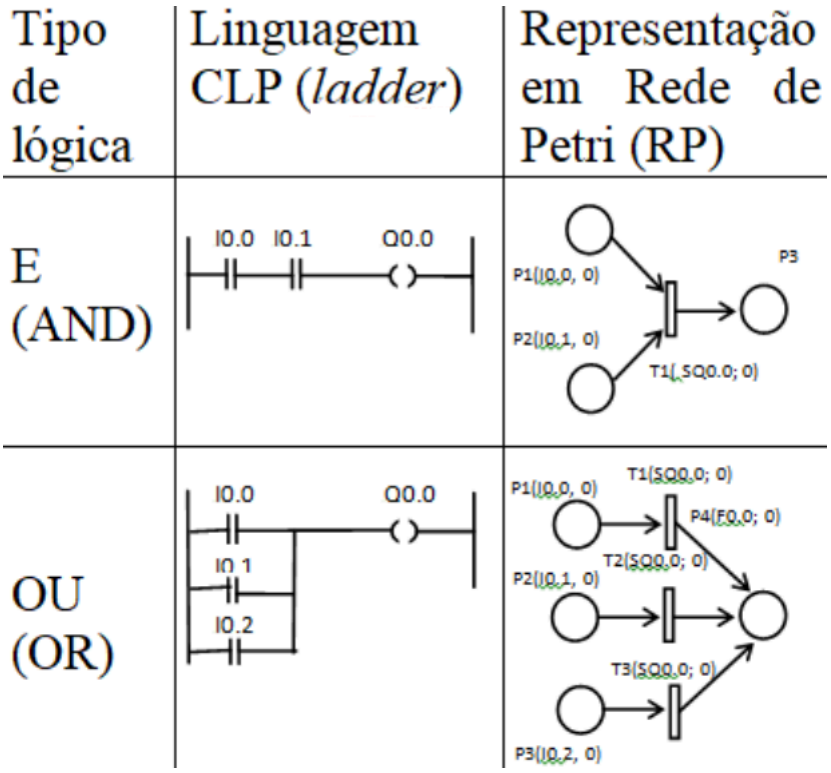


Figura 14 - Lógicas E e OU e suas representações em Ladder e RP.
Fonte: Rodrigo (2018).

Com o modelo conceitual em nível suficientemente baixo, é possível realizar a conversão direta do modelo, onde os lugares representam os contatos e as transições, as bobinas (podendo ser *set* ou *reset*). O modelo a ser convertido deverá conter apenas as lógicas E e/ou OU, com elementos complexos sendo considerados programados à parte. A Figura 15 exemplifica a conversão de um trecho de Rede de Petri para diagrama Ladder e a Figura 16 exemplifica um diagrama Ladder final gerado a partir da metodologia.

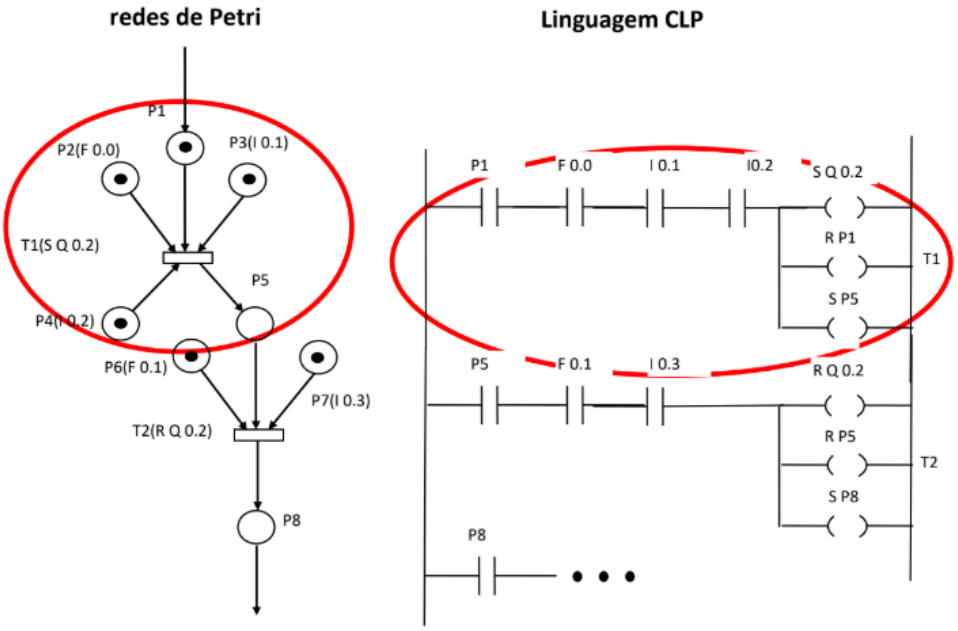


Figura 15 - Exemplo de conversão de um trecho do modelo para linguagem CLP.

Fonte: Kato (2023).

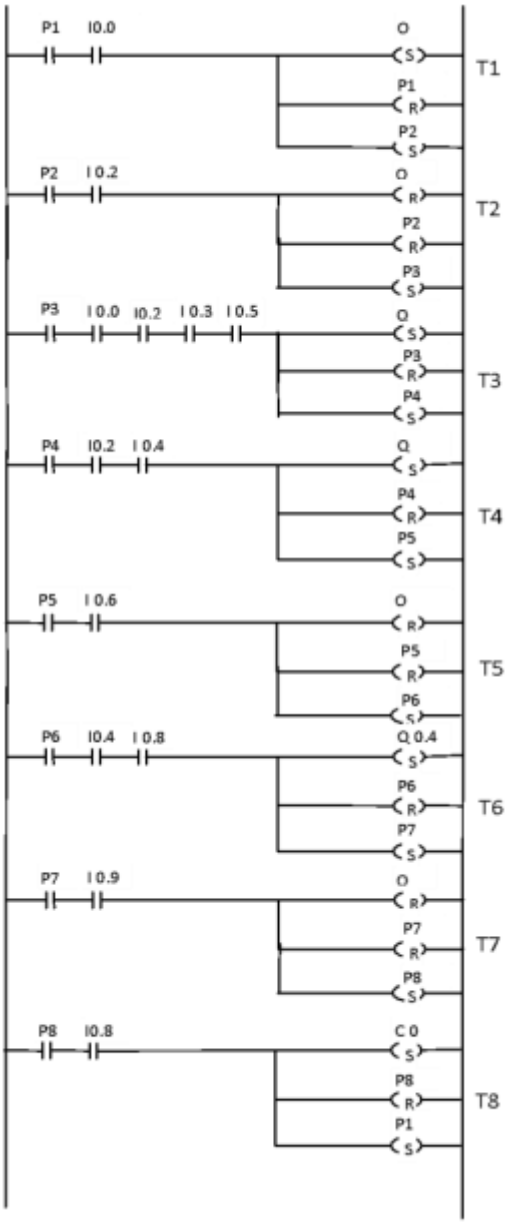


Figura 16 - Exemplo de programação CLP ou diagrama Ladder gerado.

Fonte: Kato (2023).

2.4 Ferramentas de modelagem em Redes de Petri

Esta subseção tem por objetivo apresentar as ferramentas que foram consideradas para este trabalho levando em conta os conceitos e as tecnologias que foram apresentadas até aqui.

PIPE é uma ferramenta de código aberto para criação, simulação e análise de Redes de Petri, permitindo visualizar as redes e importar e exportar arquivos PNML. A Figura 17 mostra um exemplo da tela do software com um projeto aberto.

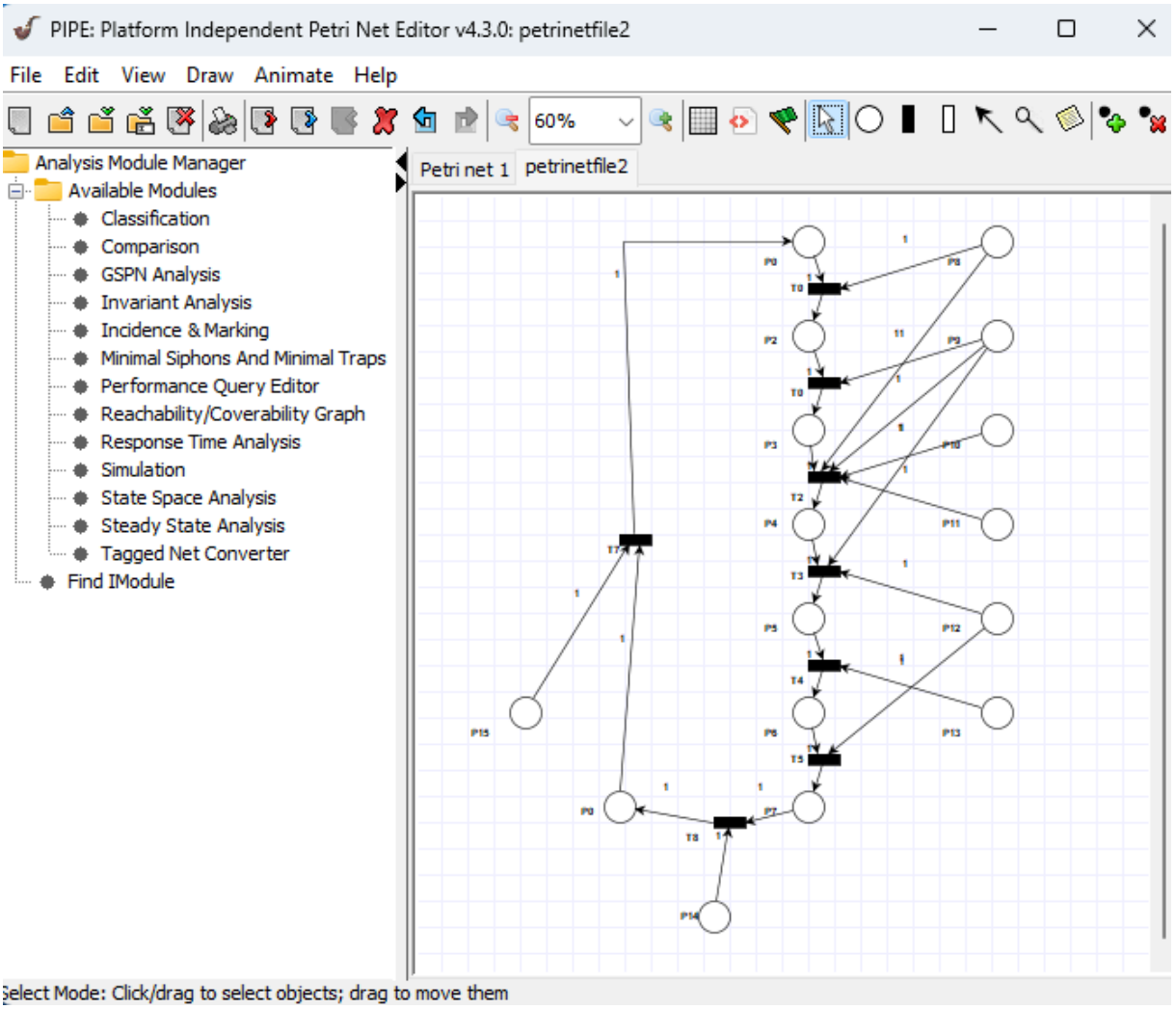


Figura 17 - Exemplo de tela do PIPE

Beremiz é um ambiente de desenvolvimento (IDE) da organização PLCOpen que permite criar e simular projetos de CLPs conforme os padrões da IEC 63113. A Beremiz permite a visualização do programa em Linguagem Ladder. Extraída do site da ferramenta, em tradução livre:

> "Beremiz faz uso de padrões abertos independentes do dispositivo-alvo, e permite que qualquer processador se transforme em

um CLP. Inclui ferramentas para criar interfaces humano-máquina e conectar seus programas em supervisores, bancos de dados ou barramentos.”

A Figura 18 mostra um exemplo da tela do Beremiz com um projeto aberto.

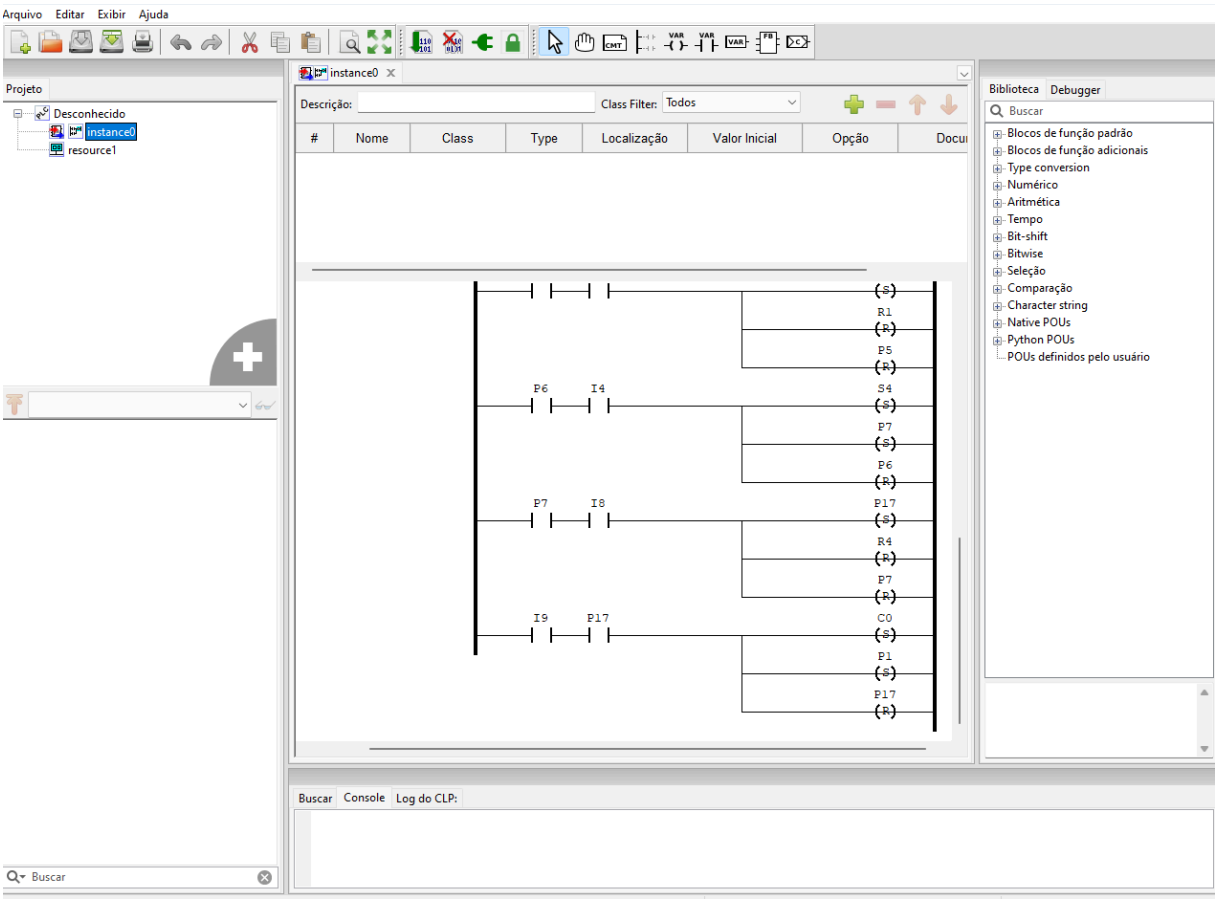


Figura 18 - Exemplo de tela do Beremiz

3. Revisão Bibliográfica

Esta revisão aborda os software de conversão de modelos em Redes de Petri de automação de sistemas em linguagens de programação CLP. A conversão de modelos conceituais para diagramas Ladder utilizando Redes de Petri é a prática mais comum entre software deste nicho, mas estes modelos não se limitam a Redes de Petri Lugar-Transição, apesar de utilizarem a prática padrão da associação das lógicas E e OU e da representação de lugares como contatos e transições como bobinas.

Kato (2023) introduz uma solução para a tradução dos modelos em Redes de Petri para diagramas ladder foi introduzida na forma de um algoritmo que nos permite realizar tal transformação de forma manual ou automática a partir de uma sequência de instruções objetivas utilizando regras de associação entre Redes de Petri Lugar-Transição e a linguagem Ladder, sendo capaz de representar todos os elementos básicos de um CLP além de contadores, temporizadores, comparadores, etc.

Existem atualmente vários software de conversão de modelos conceituais para linguagem de programação de CLPs, a maioria deles utilizando Redes de Petri para obter diagramas Ladder, apesar de poucos estarem em documentos acadêmicos. Uma solução foi desenvolvida aplicando o modelo descrito em 2.3 como software (RODRIGO, 2018) no padrão cliente-servidor em uma única máquina, utilizando a linguagem Python, uma interface com framework GK para interação com o usuário e um software de execução em linha de comando, mas ainda apresentava muitas inconsistências na execução e nos resultados para ser uma ferramenta para uso acadêmico. A ferramenta desenvolvida permite a conversão de forma automática entre Rede de Petri e Diagrama Ladder, realizando o processo apenas para redes válidas, necessitando que o usuário modele a rede através do software PIPE e a exporte em um arquivo em formato PNML, arquivo este que será lido pela ferramenta, gerando outro XML correspondente ao diagrama Ladder, que poderá ser verificado utilizando o software Beremiz. As Figuras 20 e 21 ilustram o fluxo utilizado na aplicação desenvolvida por Rodrigo (2018).

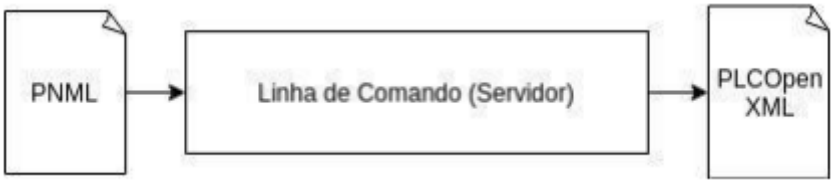


Figura 20 - Fluxo do sistema projetado por Rodrigo (2018).

Fonte: Rodrigo (2018).

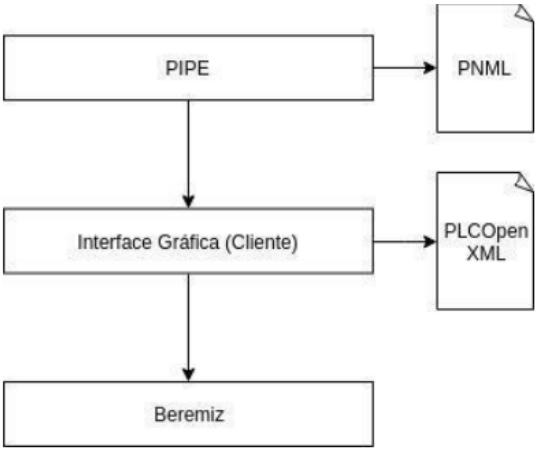


Figura 22 - Fluxo de funcionamento da conversão.

Fonte: Rodrigo (2018).

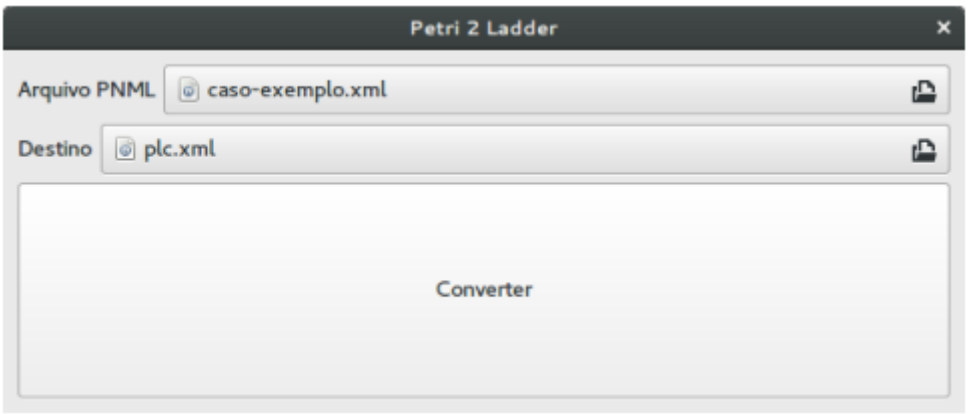


Figura 21 - Interface gráfica do programa de conversão desenvolvido por Rodrigo (2018).

Fonte: Rodrigo (2018).

Macedo e Oliveira (2019) utilizam uma abordagem com Redes de Petri Coloridas, buscando um modelo compacto mas que utiliza tipos de dados complexos, onde os tokens possuem capacidade para representar informações mais estruturadas do que apenas a presença ou não de um recurso. O trabalho buscou na linguagem Java o controle a baixo nível necessário para tal objetivo, utilizando a ferramenta CPN Tools como editor da Rede de Petri, gerando um arquivo XML, e LDmicro para visualizar e validar a conversão para diagrama

Ladder. A Figura 19 ilustra o fluxo utilizado na aplicação desenvolvida por Macedo e Oliveira (2019).

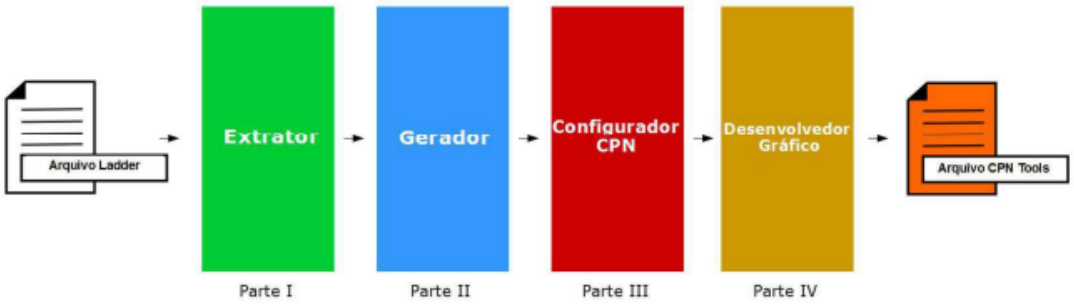


Figura 19 - Fluxo do sistema projetado por Macedo e Oliveira (2019).
Fonte: Macedo e Oliveira (2019).

Kaid, Al-Ahmari e Li (2020) propõem uma abordagem em duas etapas para a implementação de diagramas Ladder em sistemas de manufatura reconfiguráveis (RMS) utilizando Redes de Petri Coloridas Orientadas a Recursos e diagramas Ladder. O algoritmo consiste em utilizar a rede para gerar configurações válidas para o sistema de forma rápida para então traduzir o sistema utilizando iterações aninhadas para percorrer o modelo e regras de associação para adicionar componentes ao diagrama Ladder. A Figura 23 mostra um exemplo de lista de entradas e saídas geradas para traduzir o modelo conceitual para diagrama Ladder

Type	LDCROPN	Address	Description	Type	LDCROPN	Address	Description
Input	I_{S1}	I0.0	Start pushbutton	Output	Q_1	Q0.0	M1 operation
	I_{S2}	I0.1	Stop pushbutton		Q_2	Q0.1	M2 operation
	I_{S3}	I0.2	Part A detection sensor 1		Q_3	Q0.2	M3 operation
	I_{S4}	I0.3	Part B detection sensor 2		Q_4	Q0.3	R1 operation
	I_{S5}	I0.4	Part A detection sensor 3		Q_5	Q0.4	R2 operation
	I_{S6}	I0.5	Part B detection sensor 4		-	-	-
	I_{S7}	I0.6	Part A detection sensor 5		-	-	-
	I_{S8}	I0.7	Part B detection sensor 6		-	-	-

Figura 23 - Exemplo de lista de entradas e saídas utilizada por Kaid, Al-Ahmari e Li (2020).
Fonte: Kaid, Al-Ahmari e Li (2020).

Luo, Zhang, Chen e Zhou (2017) utilizaram Redes de Petri Ordinárias, com marcações simples, na tentativa de criar uma conversão que resolvesse de forma automática condições de corrida, identificando a sub rede que contém tal condição e identificando uma possível correção através da remoção de algum dos elementos de forma a não impactar o projeto. A

Figura 24 exemplifica uma condição de corrida em um programa para CLP e a Figura 25 ilustra um modelo em alto nível utilizado pelo trabalho e seu programa para CLP.

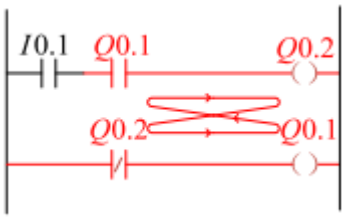


Figura 24 - Exemplo de condição de corrida disponibilizado por Luo, Zhang, Chen e Zhou (2017).

Fonte: Luo, Zhang, Chen e Zhou (2017).

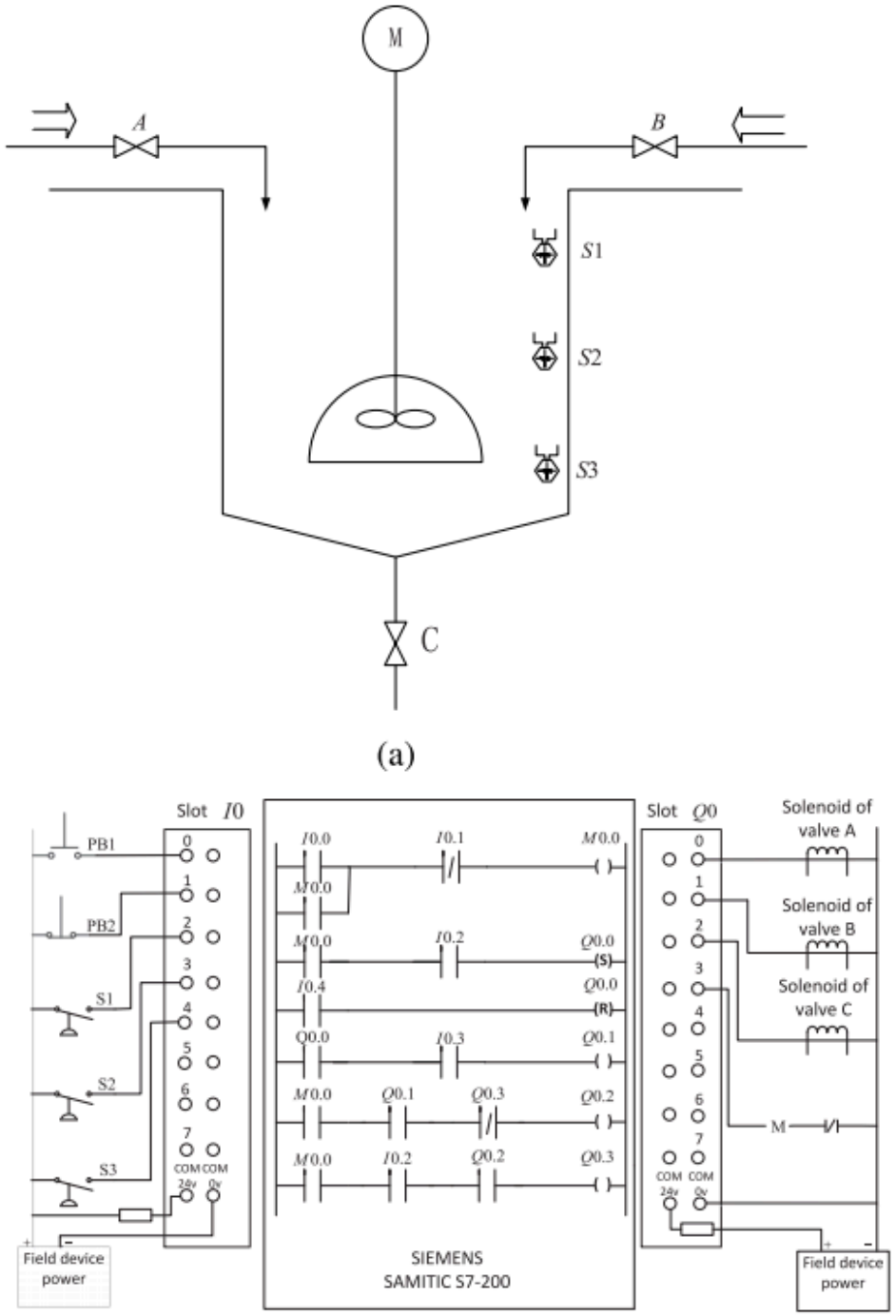


Figura 25 - Modelo em alto nível de um tanque de dosagem (a) e seu programa para CLP (b), conforme disponibilizado por Luo, Zhang, Chen e Zhou (2017).

Fonte: Luo, Zhang, Chen e Zhou (2017).

Feio (2016) propõe um algoritmo que converte Redes de Petri da classe *Input-Output Place-Transition* (IOPT) para diagrama Ladder e listas de instruções utilizando regras de associação entre as entradas e condições e transições e ações/disparos. Foi utilizado no

programa a linguagem PHP e arquivos no formato XML. A Figura 26 ilustra uma tabela de lugares construída pelo algoritmo desenvolvido por Feio (2016) para auxiliar na construção do arquivo XML final.

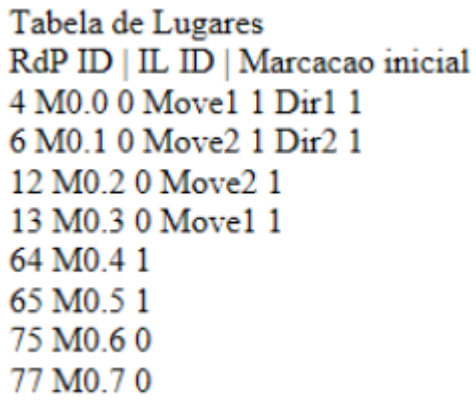


Tabela de Lugares
RdP ID | IL ID | Marcacao inicial
4 M0.0 0 Move1 1 Dir1 1
6 M0.1 0 Move2 1 Dir2 1
12 M0.2 0 Move2 1
13 M0.3 0 Move1 1
64 M0.4 1
65 M0.5 1
75 M0.6 0
77 M0.7 0

Figura 26 - Exemplo de lista de lugares gerada pelo software desenvolvido por Feio (2016).

Fonte: Feio (2016).

Vieira, Santos, Queiroz, Leal, Neto e Cury (2017) propõem um modelo baseado em teoria de controle supervisório (SCT) para criar um modelo utilizando arquitetura de controle dividida em três níveis: supervisores modulares que coordenam a distribuição de tarefas, sistemas de produtos que representam as máquinas e componentes e procedimentos operacionais detalhando as instruções de funcionamento de cada máquina. Deste modelo são obtidos blocos de função e as variáveis necessárias para o funcionamento do sistema e, destes blocos, é possível desenvolver de forma mais fácil programas em diversas linguagens incluídas na IEC 61131-3. A Figura 27 ilustra a estrutura da arquitetura utilizada por Vieira *et al*. e os três níveis utilizados no modelo.

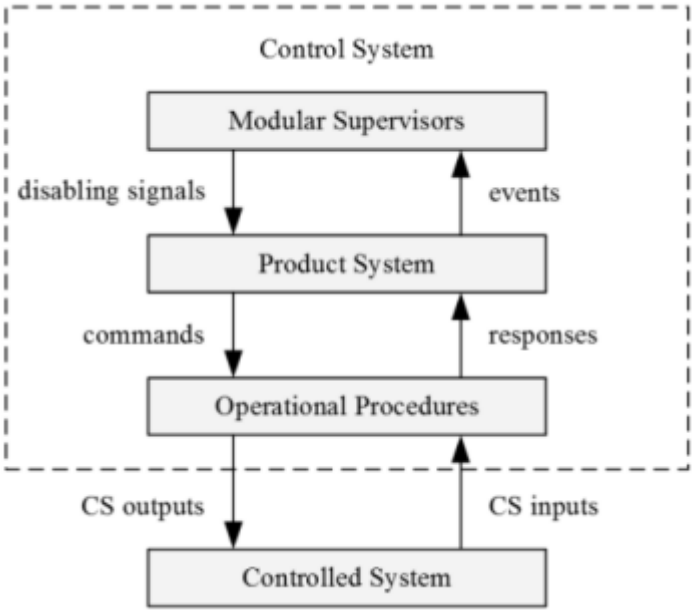


Figura 27 - Estrutura da arquitetura de controle utilizada por Vieira *et al*. (2017).

Fonte: Vieira *et al.* (2017).

4. Proposta

O objetivo deste capítulo é apresentar as decisões e funcionalidades sobre o desenvolvimento do sistema. A solução a ser explorada é a aprimoração de um software que permita implementar, testar e validar o conversor de modelos em Redes de Petri para linguagem Ladder, podendo ser inserida em CLPs para posterior validação, facilitando a detecção de erros e aprimorando a eficiência do processo de transcrição. Por meio do conversor, é possível simular o comportamento do sistema automatizado, verificar seu desempenho e realizar ajustes necessários antes da implementação prática. Dessa forma contribuindo para reduzir riscos e economizar tempo e recursos durante o desenvolvimento de projetos de automação industrial utilizando modelagem em Redes de Petri e CLPs.

O sistema proposto permite converter uma Rede de Petri em um Diagrama Ladder de forma automática, baseando-se em XMLs de Redes de Petri utilizando o formato PLCOpen. O mesmo deve conter contatos, saídas, temporizadores e todas as ferramentas adicionais necessárias para a implementação das aplicações propostas.

O fluxo de execução do sistema consistirá no usuário modelar a Rede de Petri e exportá-la em formato PNML utilizando o software PIPE, abrir o software proposto, especificar o arquivo contendo a rede e iniciar a conversão, onde o programa avisará o usuário tanto caso a conversão seja bem sucedida quanto casa haja algum erro. Ao final da conversão será gerado um outro arquivo XML contendo o diagrama Ladder, que poderá ser importado no Beremiz para fins de validação. Caso algum erro seja identificado após a implementação do programa no CLP, é possível reiterar o desenvolvimento, retornando à fase de modelagem em Rede de Petri em um processo chamado de *as build*. A Figura 28 ilustra o fluxo de execução do sistema descrito.

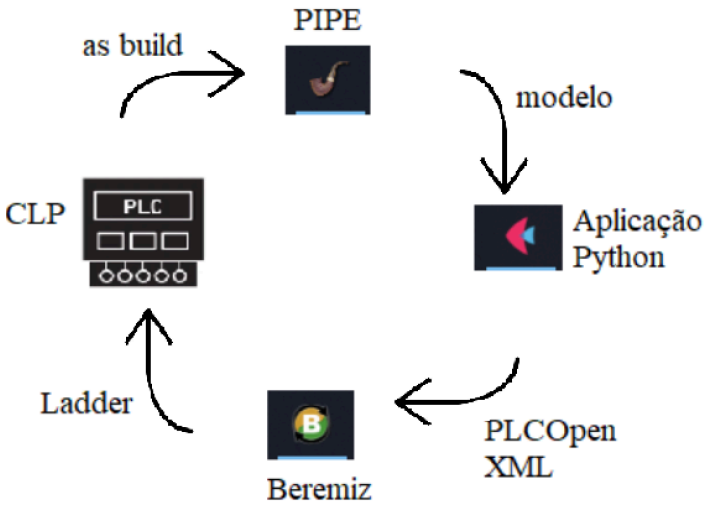


Figura 28 - Ilustração do fluxo proposto para o sistema

O diagrama Ladder irá contar com os elementos *set, reset* e contador (interpretado como uma bobina *set*), onde para que seja possível associar as bobinas com suas respectivas funções, deve-se atribuí-las identificações com início S, R e C, respectivamente, no editor PIPE e em suas respectivas transições, para que o software identifique o tipo do elemento. Para a associação entre bobinas e lugares, são utilizados os termos Px e Ix, onde x é o número identificador do elemento, utilizando para a visualização da rede dentro da aplicação os elementos P0 ou P1 como nó raiz, caso existam.

4.1 Metodologia

A metodologia aplicada para o desenvolvimento deste trabalho segue as seguintes etapas:

1. Estudo do domínio e análise do problema a partir de uma revisão bibliográfica;

2. Avaliação do sistema e planejamento;

3. Recuperação da implementação;

4. Expansão da implementação;

5. Validação do sistema com estudo de casos

As ferramentas escolhidas para auxiliarem no funcionamento do sistema e seu fluxo foram as seguintes:

O PIPE capacita a etapa de criação do modelo conceitual em Redes de Petri necessário para a conversão, fornecendo o modelo em um arquivo PNML, facilitando a leitura do mesmo pelo programa.

O Beremiz permite a importação e a visualização de arquivos PLCOpen XML, possibilitando a validação dos resultados obtidos e a análise dos diagramas Ladder.

A linguagem de programação escolhida para o desenvolvimento foi o Python, facilitando a herança de trabalhos anteriores com linguagem mais alto nível e alta disponibilidade de bibliotecas e pacotes auxiliares.

PyInstaller é um módulo utilizado para ler scripts em Python, analisar os módulos e bibliotecas necessárias para sua execução e organizá-los em um único pacote. O PyInstaller permite a criação de um executável *standalone*, cumprindo a necessidade de uma forma fácil e rápida de utilizar todas as partes fundamentais do programa, sendo necessárias poucas instalações opcionais, sendo a ferramenta escolhida para esta função após ser considerada junto do Docker e *Python Virtual Environment*.

Docker é um conjunto de produtos no formato plataforma como serviço que utilizam virtualização a nível de sistema operacional para entregar software em pacotes chamados de contêineres. Seu uso foi descartado pois não solucionou a necessidade de uma forma fácil e rápida de poder utilizar o programa proposto, havendo dificuldades para o uso de interfaces gráficas.

Python *Virtual Environment* oferece suporte à criação de ambientes virtuais leves, com seus próprios conjuntos independentes de pacotes, bibliotecas e scripts Python instalados a fim de conter uma aplicação em um ambiente isolado. O uso de um Virtual Environment foi descartado pelas dificuldades de utilizar o mesmo para compartilhamento do programa e de suas bibliotecas necessárias entre diferentes usuários, sendo próprio para separar instâncias diferentes destas bibliotecas em um só computador.

O Flet foi o framework escolhido para a interface do programa por criar aplicações desktop multiplataforma de forma intuitiva, rápida e com visual moderno.

O Graphviz é um pacote de ferramentas que permite a criação de imagens para visualização de grafos, sendo de fácil integração com o Flet e permitindo verificar se a Rede de Petri está correta diretamente no software.

4.2 Verificação inicial

A verificação do sistema proposto será realizada de forma não formal por meio da comparação dos resultados obtidos ao final do fluxo de execução, ou seja, do diagrama Ladder gerado pelo conversor, com os diagramas Ladder corretos esperados, sendo utilizados como referência três projetos, sendo eles dois projetos levemente modificados de exemplos disponibilizados por Kato (2023) e Kato (2025), além de um projeto utilizado por Rodrigo (2018). A verificação não garante o funcionamento do sistema para todas as Redes de Petri Lugar-Transição válidas.

4.3 Resultados esperados

Como resultado, espera-se um software executável de conversão de Redes de Petri para diagramas Ladder de forma correta segundo a metodologia utilizada, indicando os mesmos elementos conectados de forma correspondente, representando as mesmas operações lógicas.

5. Implementação

A implementação do código em Python, disponível no apêndice do trabalho, é dividida em cinco partes: o *parser* (parser.py), cuja função é converter o arquivo PNML em um modelo de símbolos contendo os lugares, transições e arcos; o analisador léxico (analyzer.py), que utiliza do modelo gerado pelo *parser* para organizar o XML em uma linguagem intermediária; o gerador de código (ladder_generator.py), responsável por gerar o arquivo PLCOpen XML a partir da linguagem intermediária; o cliente (main.py e petri2ladder_cli.py), que interpreta os comandos do usuário, controla a interface e distribui as chamadas para funções com seus parâmetros corretos; e os modelos de símbolos (Petri.py e AST.py), que definem a organização a baixo nível das informações a serem manipuladas pelo programa.

O fluxo do sistema consiste em criar um projeto no PIPE, utilizando as nomenclaturas Px e Ix para lugares e Sx, Rx e Cx para as transições, onde serão associados a bobinas *set, reset* e contadores, respectivamente, x sendo o número identificador do elemento. É então exportado o arquivo como xml e carregado no sistema iniciado pelo executável. Ao clicar no botão converter, será mostrada uma janela com o *status* da conversão e, caso bem sucedida, gerado um arquivo xml de nome plc, próprio para ser inserido em um projeto do Beremiz. Para visualização e validação do resultado, é então criado um projeto novo no Beremiz e inserido, no lugar de seu código plc.xml padrão, o arquivo obtido do conversor. A Figura 29 ilustra um modelo em Rede de Petri construído no editor PIPE, a Figura 30 mostra a interface da aplicação desenvolvida e a Figura 31 mostra a tela gerada após uma conversão bem sucedida.

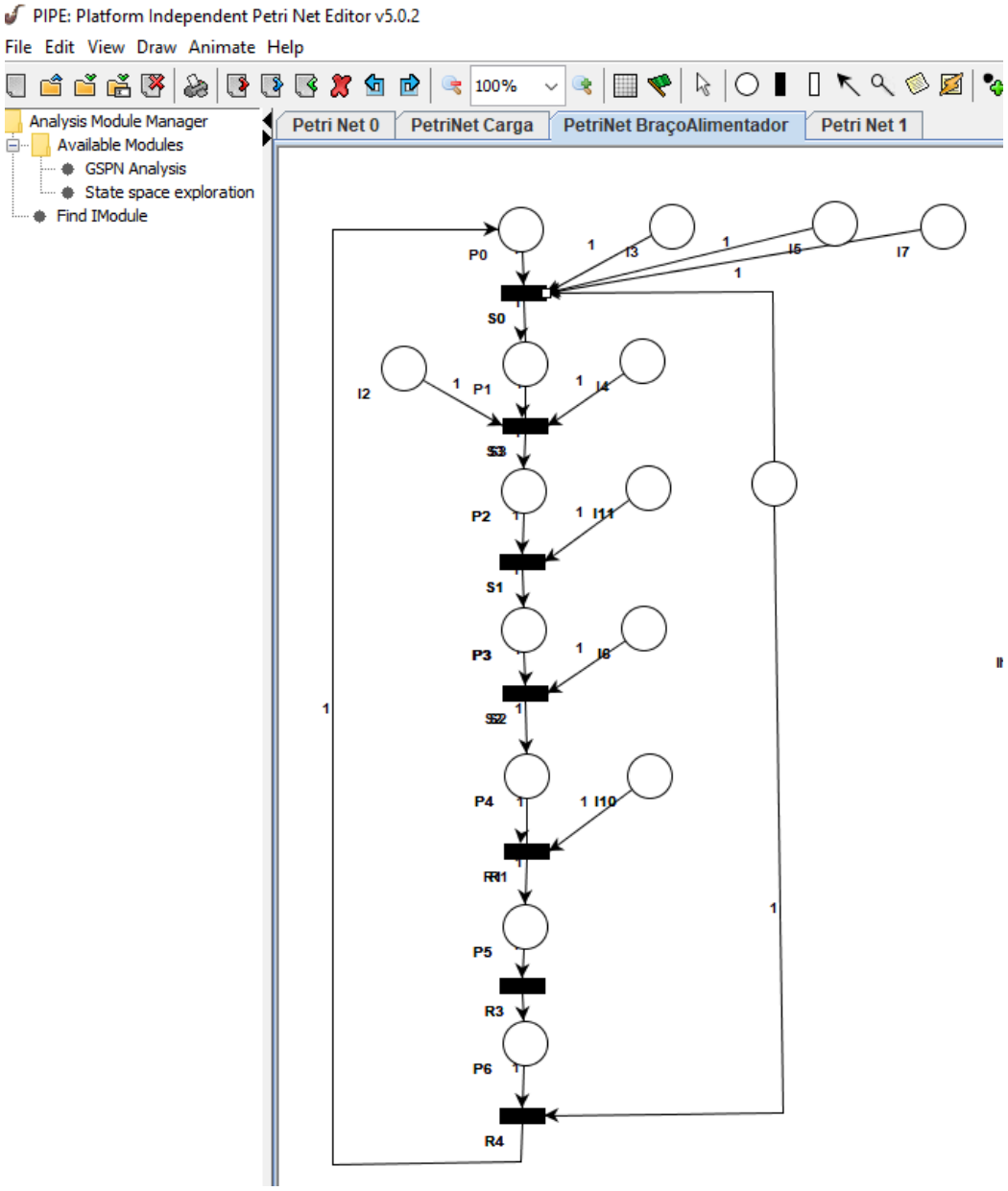


Figura 29 - Interface da ferramenta de edição PIPE.

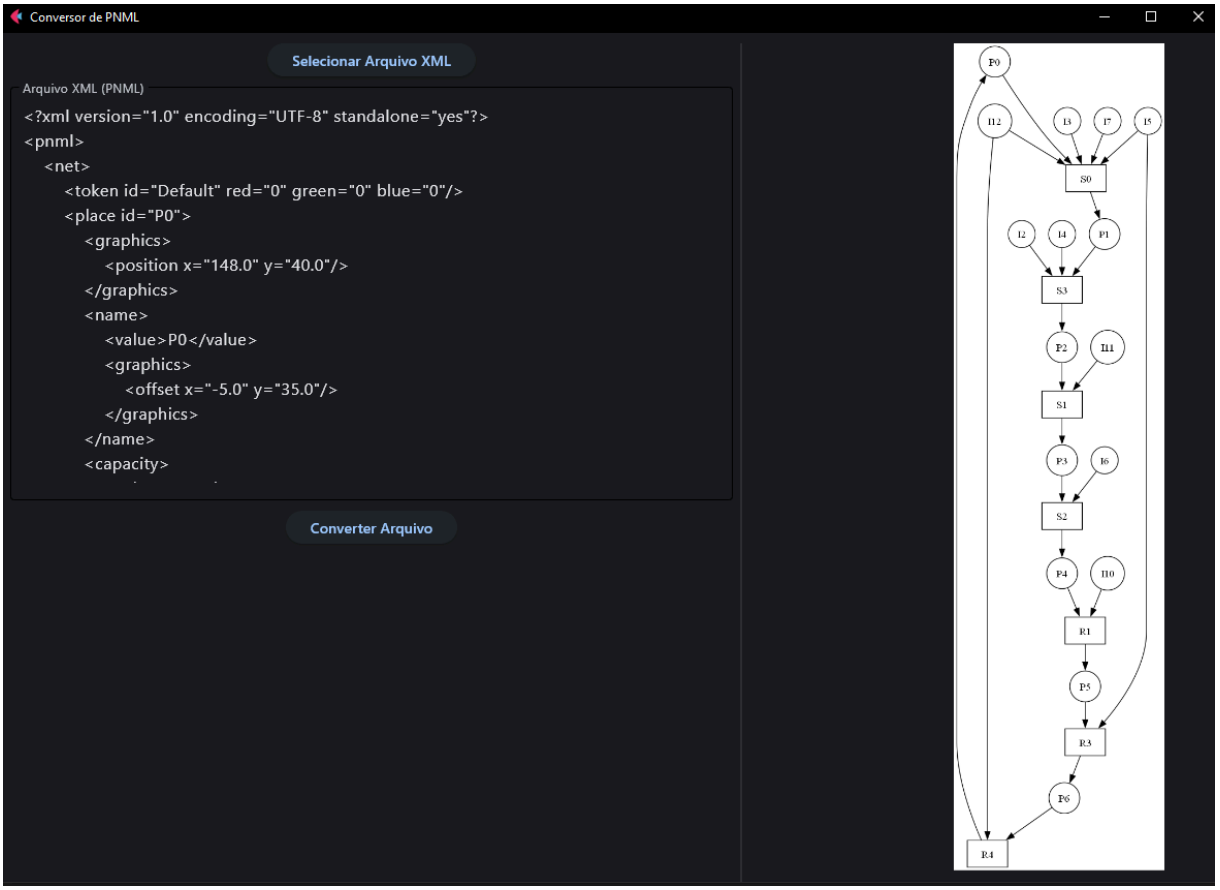


Figura 30 - Interface do conversor de Rede de Petri para linguagem Ladder, com visualização da rede.

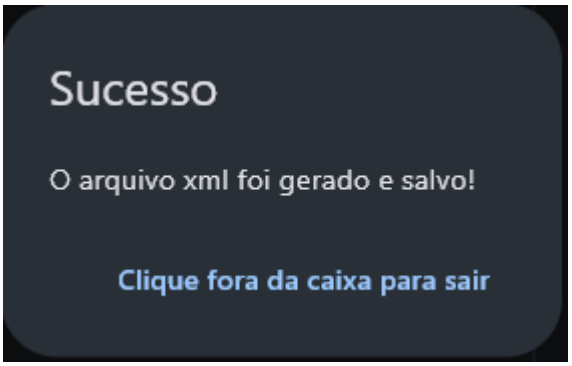


Figura 31 - Mensagem de sucesso obtida após converter o modelo.

A linguagem intermediária utiliza uma transição por linha e é composta de redes/*Nets* com as condições de ativação, uma lista de bobinas *set* e uma lista de bobinas *reset*, sendo então uma lista de redes, sendo muito similar à estrutura da linguagem Ladder. É então gerado

o diagrama Ladder correspondente calculando as posições dos elementos como se estivessem em um grid, sintetizando os elementos da esquerda para a direita, começando por contatos que se conectam primeiramente ao trilho da esquerda seguido de próximos contatos e então associados às bobinas de *set* e *reset,* que por sua vez se conectam no trilho da direita. A Figura 32 ilustra um diagrama Ladder sendo visualizado no editor Beremiz.

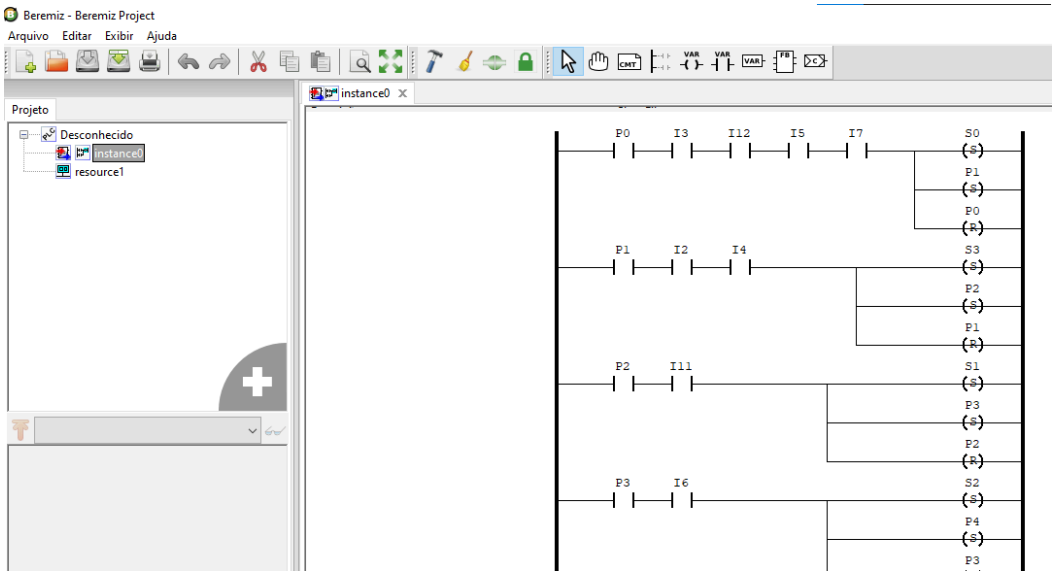


Figura 32 - Interface da ferramenta de edição Beremiz.

5.1 Resultados

Os resultados obtidos aplicando o conversor sobre os modelos em Rede de Petri dos projetos selecionados foram os seguintes:

5.1.1 Braço alimentador

Consiste em um projeto simples de uma sequência única de um alimentador de uma máquina de corte, onde muitos passos se desenvolvem sempre na mesma ordem (Kato, 2023). A Figura 33 ilustra o funcionamento do braço alimentador e a Figura 34 ilustra o modelo final em Redes de Petri para o sistema.

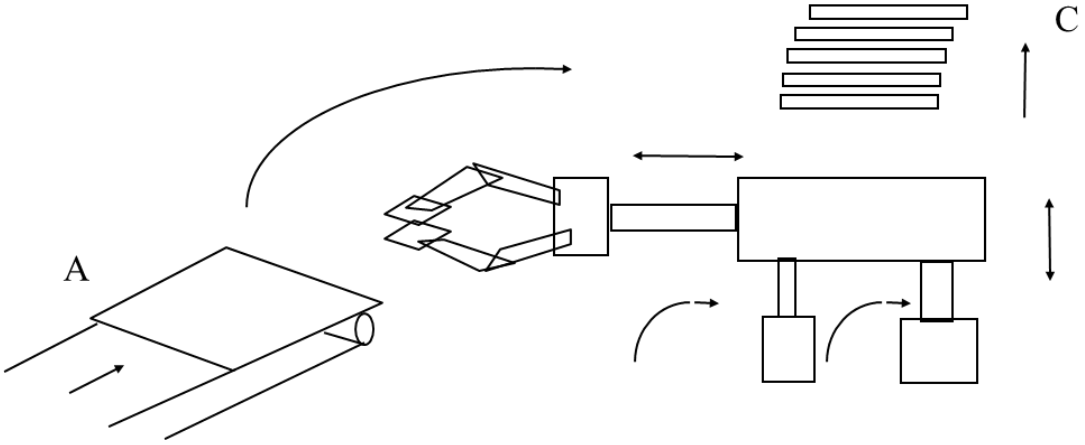


Figura 33 - Ilustração do funcionamento do braço alimentador.

Fonte: Kato (2023).

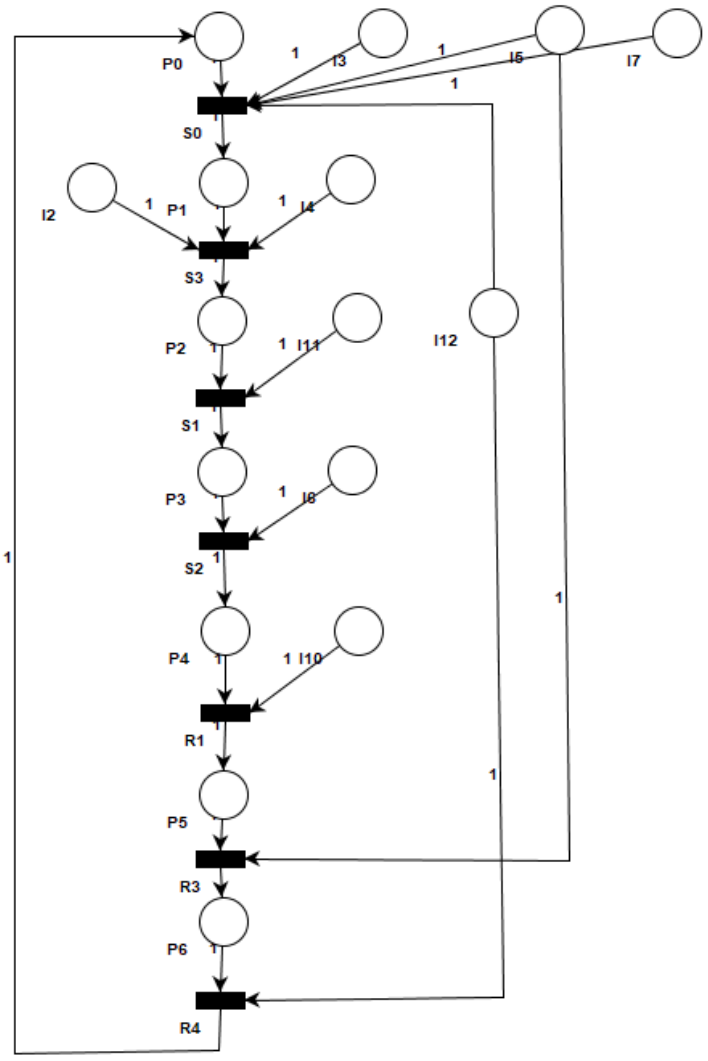


Figura 34 - Modelo da Rede de Petri do projeto de braço alimentador.

A Tabela 3 ilustra a lista de entradas e saídas para o sistema.

I2	Sensor peça presente na esteira
I3	Sensor braço recuado
I4	Sensor braço avançado
I5	Sensor braço embaixo
I6	Sensor braço posição alta
I7	Sensor braço à esquerda
I10	Sensor braço rotacionando
I11	Sensor peça presa
I12	Sensor peça solta*
S0	Avança braço
S/R1	Sobe/Desce braço
S2	Rotaciona braço
S/R3	Fecha/Abre a pinça
R4	Recua e rotaciona braço para posição inicial

Tabela 3 - Lista de entradas e saídas do projeto de braço alimentador.

A Figura 35 mostra o diagrama Ladder convertido manualmente para verificação e a Figura 36 mostra o diagrama Ladder convertido de forma automática pela aplicação.

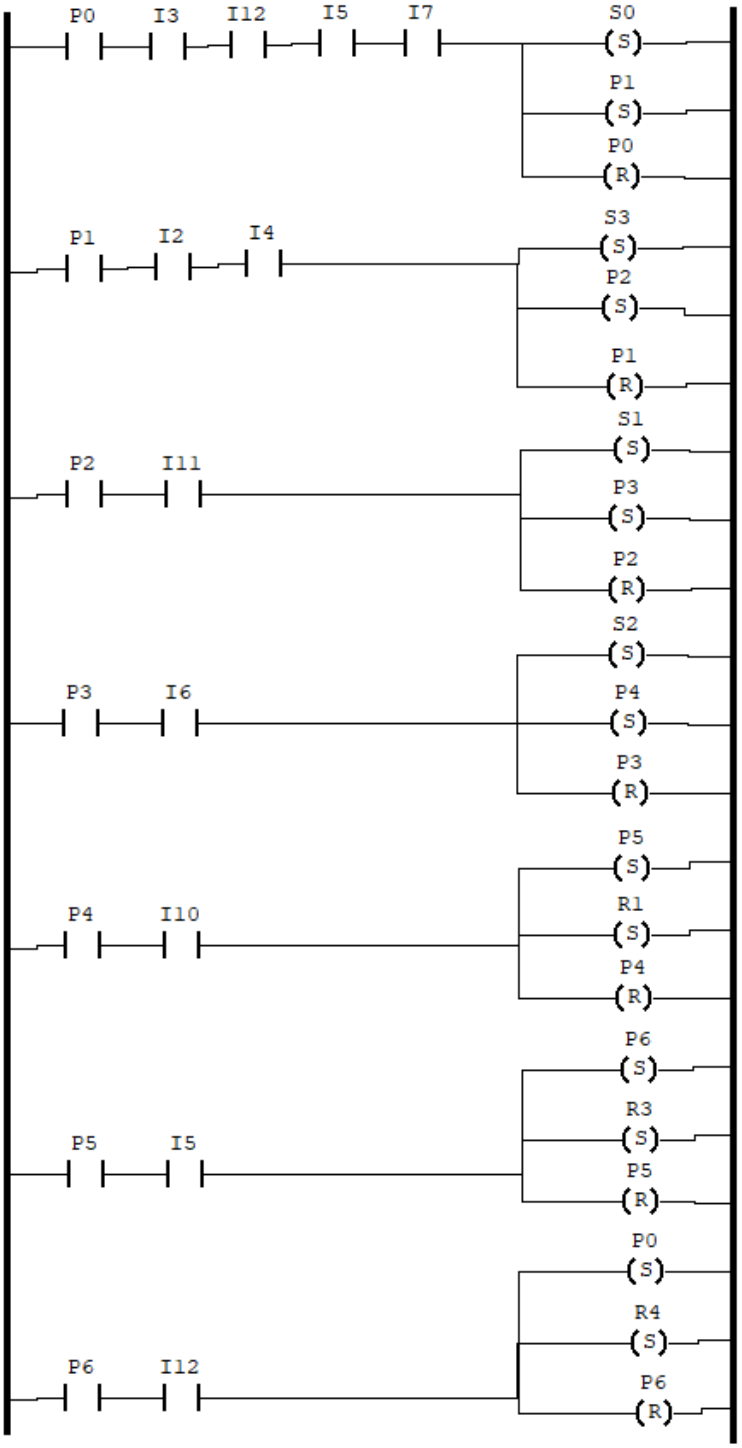


Figura 35 - Diagrama Ladder convertido manualmente para o projeto do braço alimentador.

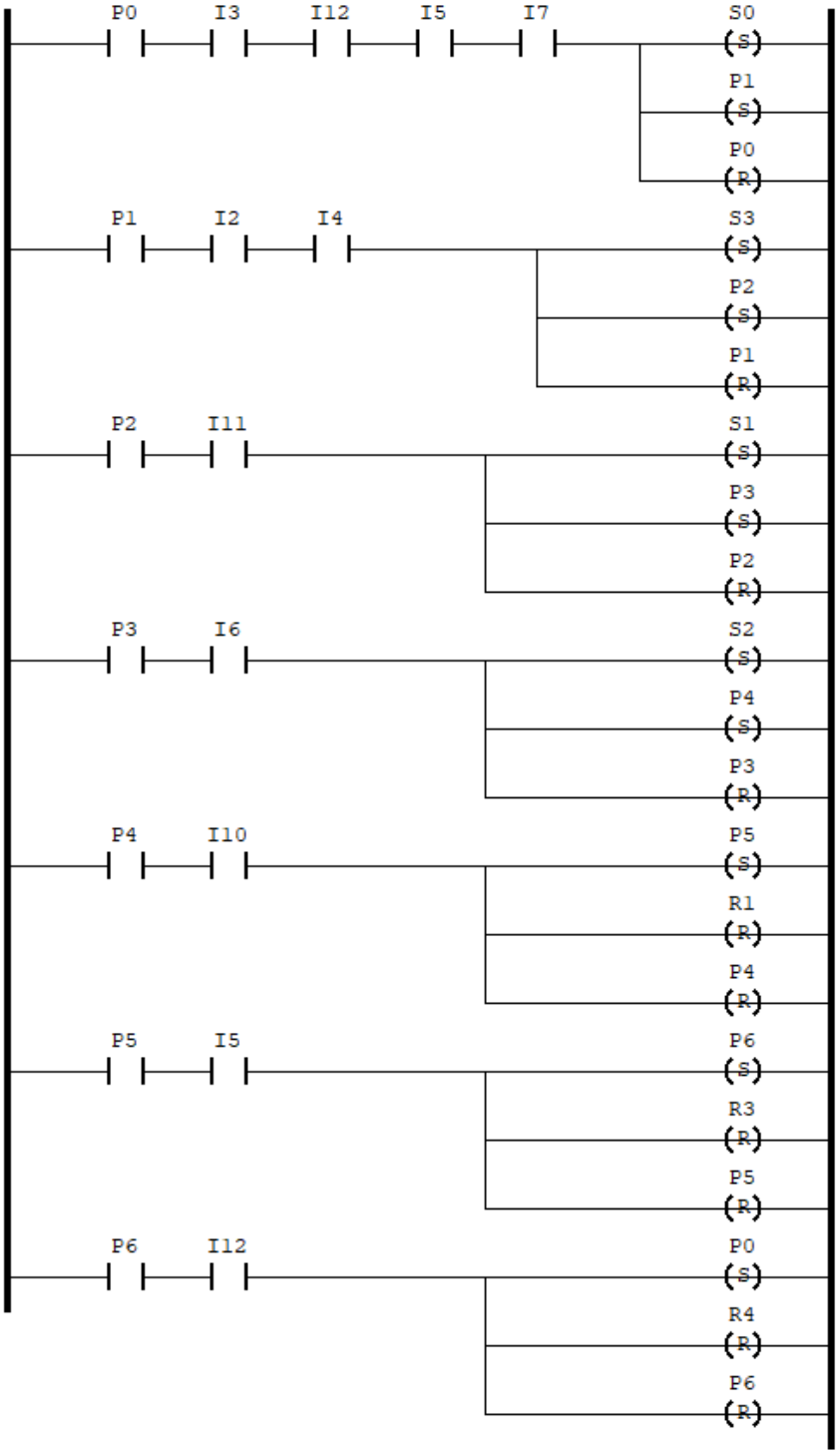


Figura 36 - Diagrama Ladder obtido pelo programa para o projeto do braço alimentador.

5.1.2 Controle de Mistura

Projeto com fluxo sequencial para controlar a mistura correta de quatro substâncias, determinando um produto final. O fluxo do sistema consiste em: a botoeira liga para iniciar o processo e desliga para interromper o processo; caso ligado, a válvula de entrada é aberta até

o nível máximo ser atingido, ligando então o motor do agitador por 10 segundos. Depois dos 10s o motor do agitador é desligado e a válvula de saída é aberta até que o tanque esteja vazio. Ao ser detectado que o tanque está vazio, a válvula de saída é fechada terminando o ciclo (Kato, 2025). A Figura 37 ilustra o funcionamento do controle de mistura e a Figura 38 ilustra o modelo final em Redes de Petri para o sistema.

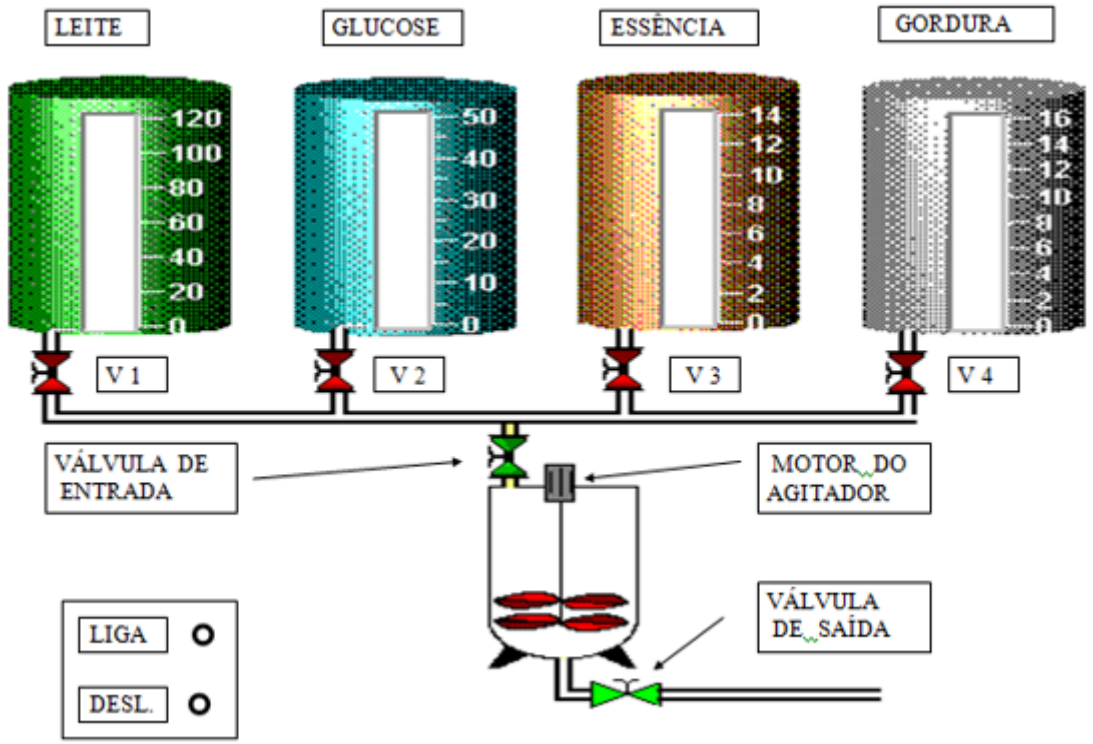


Figura 37 - Ilustração do funcionamento do controle de mistura.

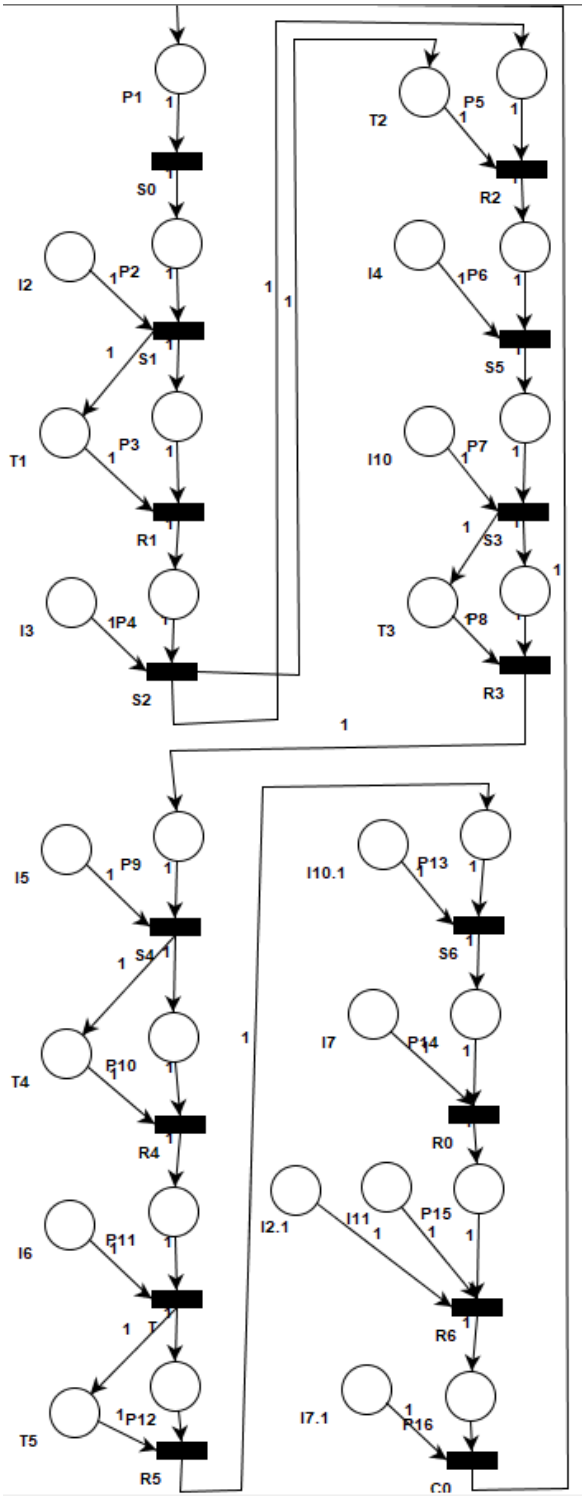


Figura 38 - Modelo da Rede de Petri do projeto de controle de mistura.

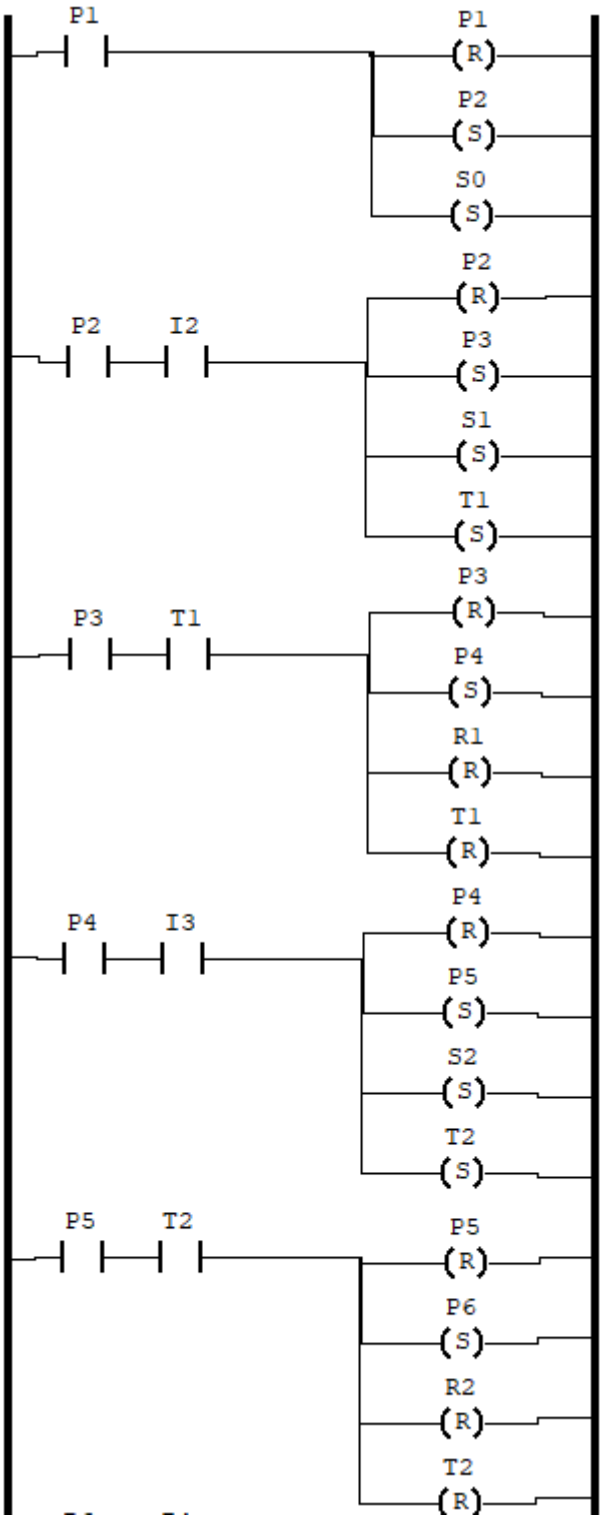
A Tabela 4 ilustra a lista de entradas e saídas para o sistema.

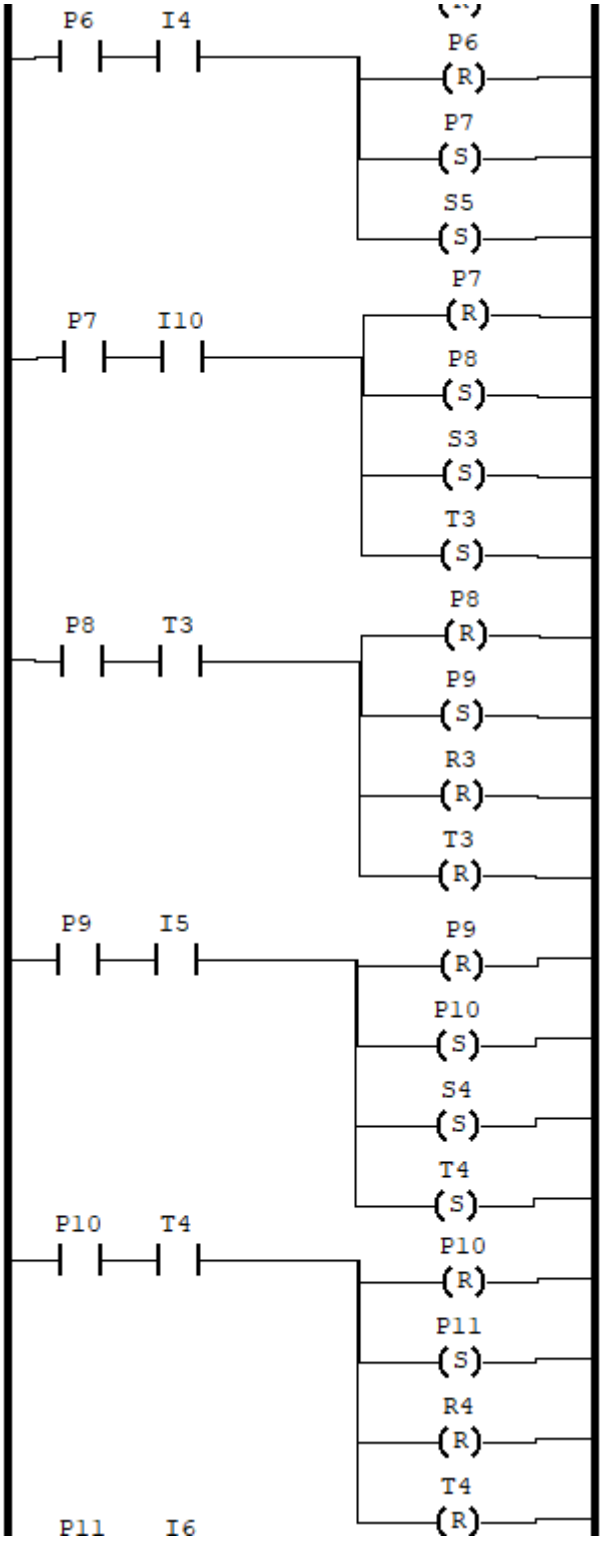
I2/I2.1	Sensor válvula de entrada aberta
I3	Sensor válvula de leite aberta

I4	Sensor válvula de glucose aberta
I5	Sensor válvula de essência aberta
I6	Sensor válvula de gordura aberta
I7/I7.1	Sensor válvula de saída aberta
I10/I10.1	Sensor motor ligado
I11	Sensor tanque vazio
S/R0	Abre/Fecha válvula de entrada
S/R1	Abre/Fecha válvula de leite
S/R2	Abre/Fecha válvula de glucose
S/R3	Abre/Fecha válvula de essência
S/R4	Abre/Fecha válvula de gordura
S/R5	Liga/Desliga motor do agitador
S/R6	Abre/Fecha válvula de saída
T1/T4	Temporizador de 10 segundos
T2/T5	Temporizador de 15 segundos
T3	Temporizador de 5 segundos

Tabela 4 - Lista de entradas e saídas do projeto de controle de mistura.

A Figura 39 mostra o diagrama Ladder convertido manualmente para verificação e a Figura 40 mostra o diagrama Ladder convertido de forma automática pela aplicação.


P1
P1
(R)
P2
(S)
S0
(S)
P2
(R)
P2
I2
P3
(S)
S1
(S)
T1
(S)
P3
(R)
P3
T1
P4
(S)
R1
(R)
T1
(R)
P4
(R)
P4
I3
P5
(S)
S2
(S)
T2
(S)
P5
(R)
P5
T2
P6
(S)
R2
(R)
T2
(R)


P6
I4
P6
(R)
P7
(S)
S5
(S)
P7
I10
P7
(R)
P8
(S)
S3
(S)
T3
(S)
P8
T3
P8
(R)
P9
(S)
R3
(R)
T3
(R)
P9
I5
P9
(R)
P10
(S)
S4
(S)
T4
(S)
P10
T4
P10
(R)
P11
(S)
R4
(R)
T4
(R)
P11
I6

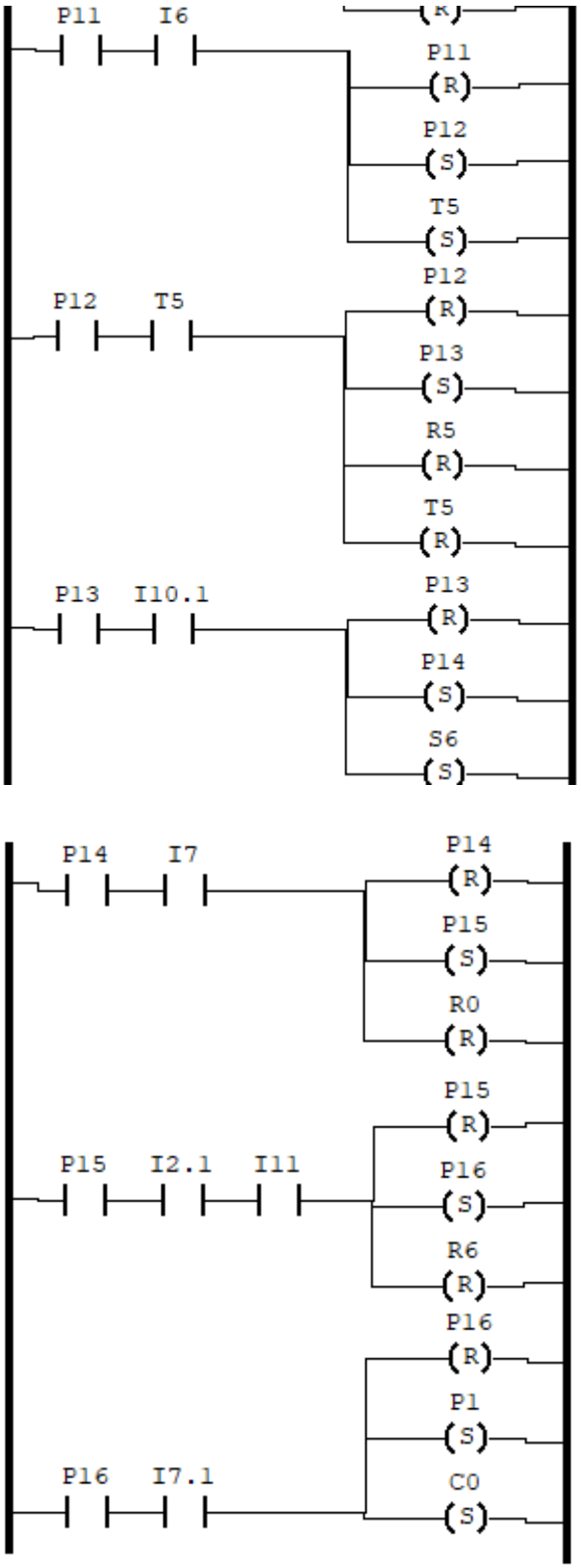
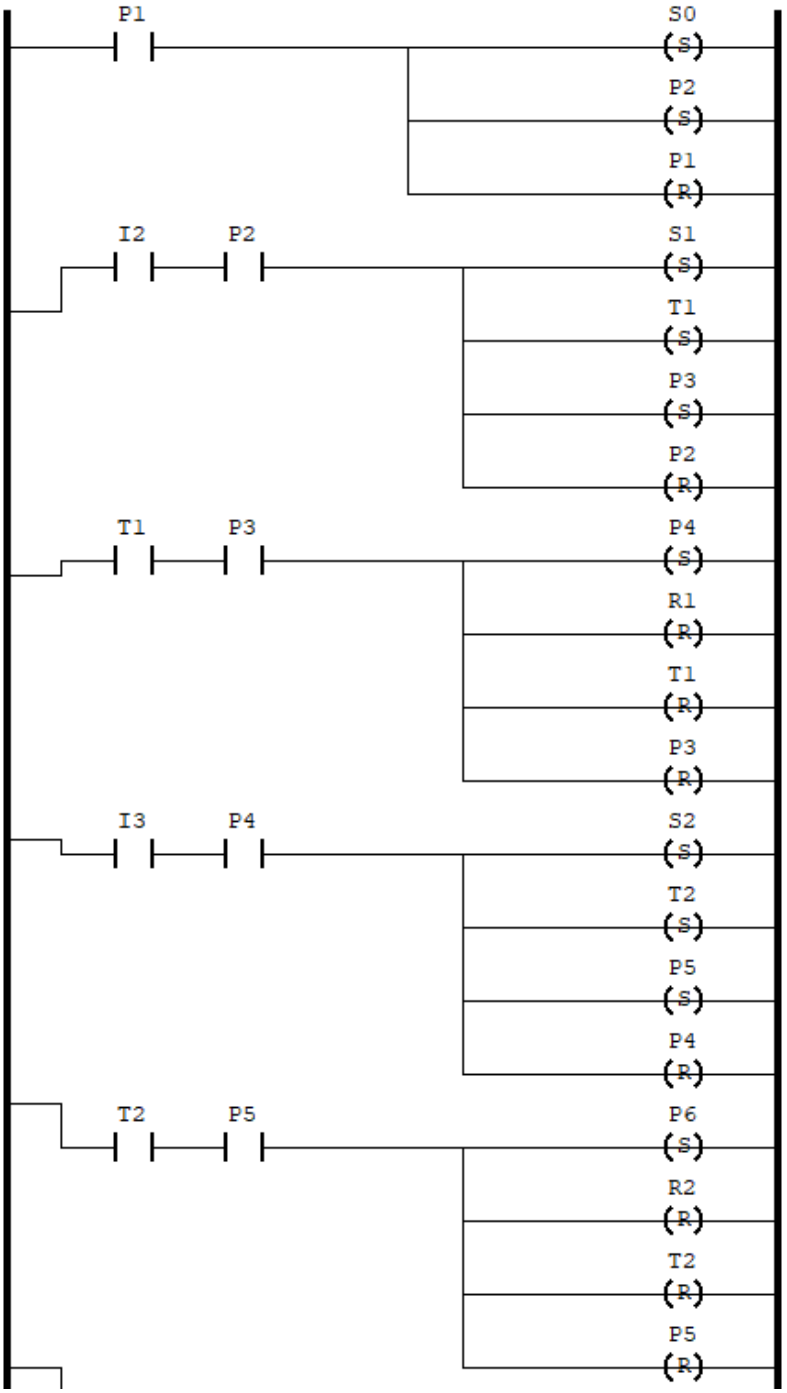
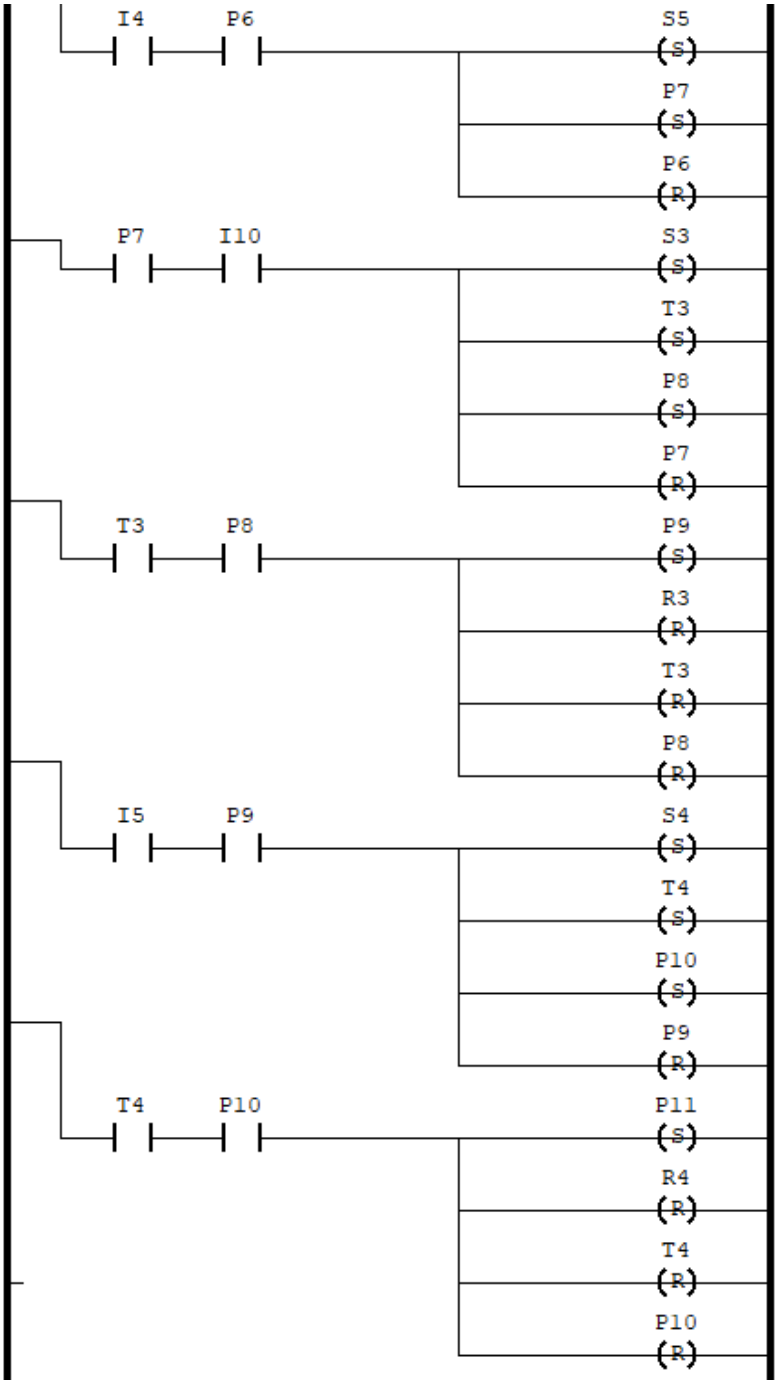


Figura 39 - Diagrama Ladder convertido manualmente para o projeto do controle de mistura.


P1
S0
(S)
P2
(S)
P1
(R)
I2
P2
S1
(S)
T1
(S)
P3
(S)
P2
(R)
T1
P3
P4
(S)
R1
(R)
T1
(R)
P3
(R)
I3
P4
S2
(S)
T2
(S)
P5
(S)
P4
(R)
T2
P5
P6
(S)
R2
(R)
T2
(R)
P5
(R)


I4
P6
S5
(S)
P7
(S)
P6
(R)
P7
I10
S3
(S)
T3
(S)
P8
(S)
P7
(R)
T3
P8
P9
(S)
R3
(R)
T3
(R)
P8
(R)
I5
P9
S4
(S)
T4
(S)
P10
(S)
P9
(R)
T4
P10
P11
(S)
R4
(R)
T4
(R)
P10
(R)

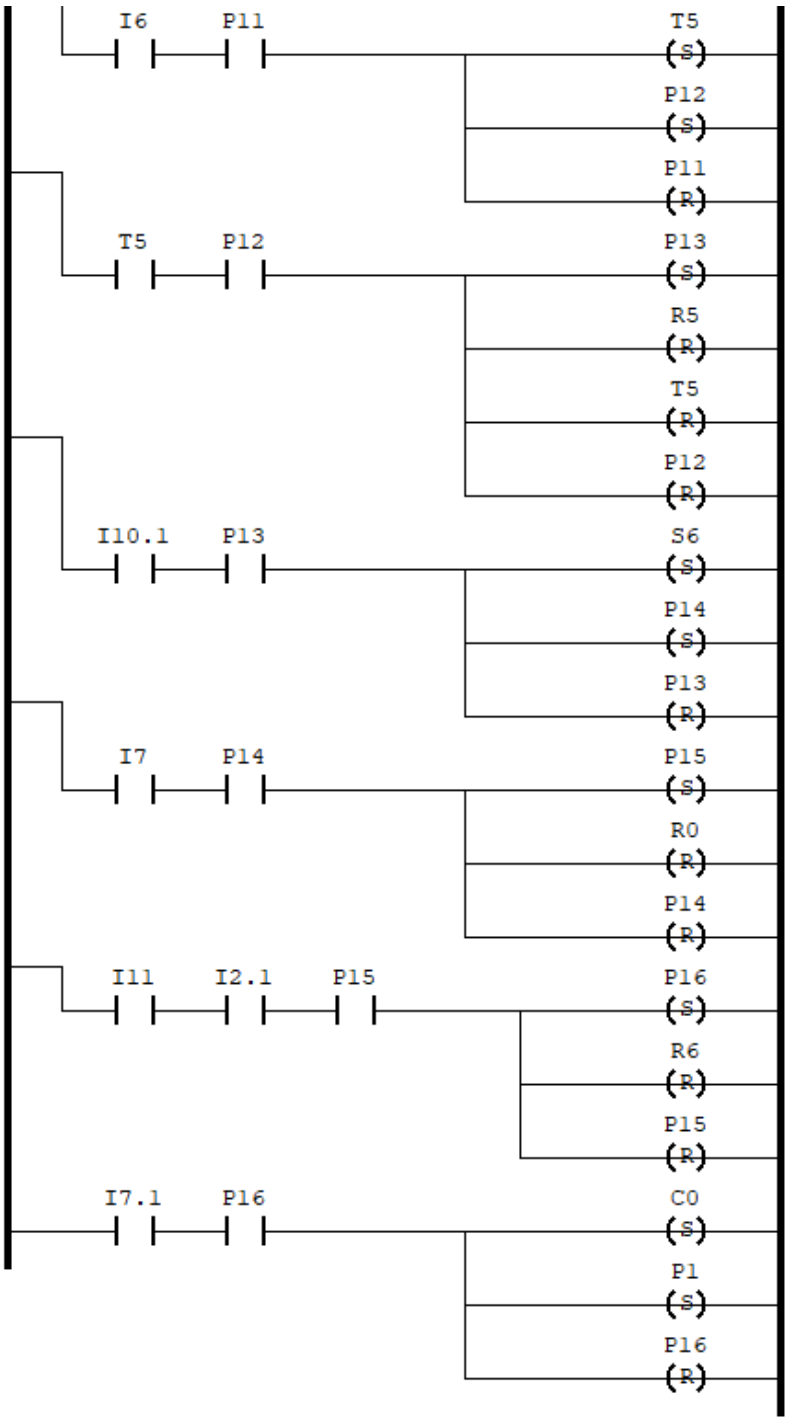


Figura 40 - Diagrama Ladder obtido pelo programa para o projeto do braço alimentador.

5.1.3 Carga e contagem de peças

Consiste em um projeto simples com pistões e contador para carga e descarga de peças. A Figura 41 ilustra o modelo final em Redes de Petri para o sistema (Rodrigo, 2018):

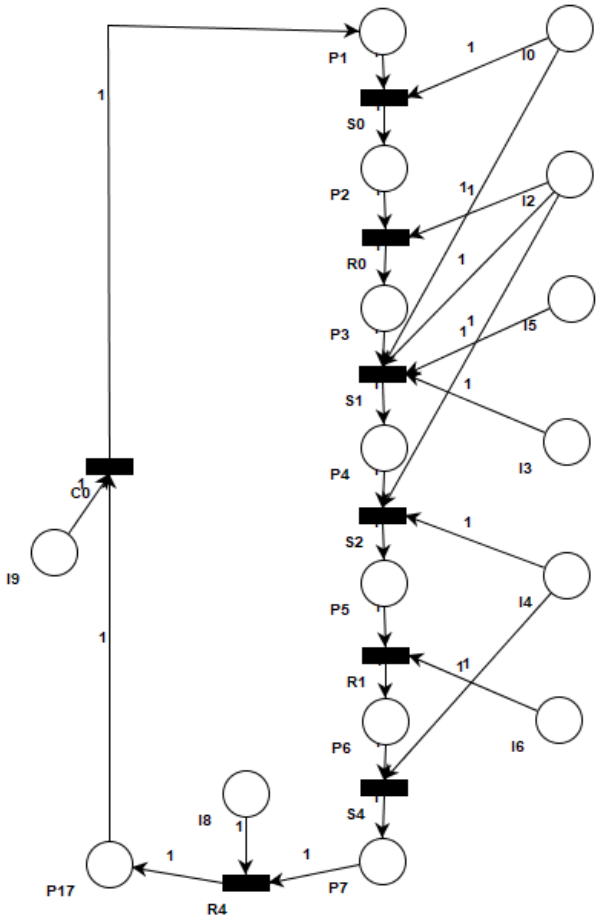


Figura 41 - Modelo da Rede de Petri do projeto de carga de peças.

Fonte: Rodrigo (2018).

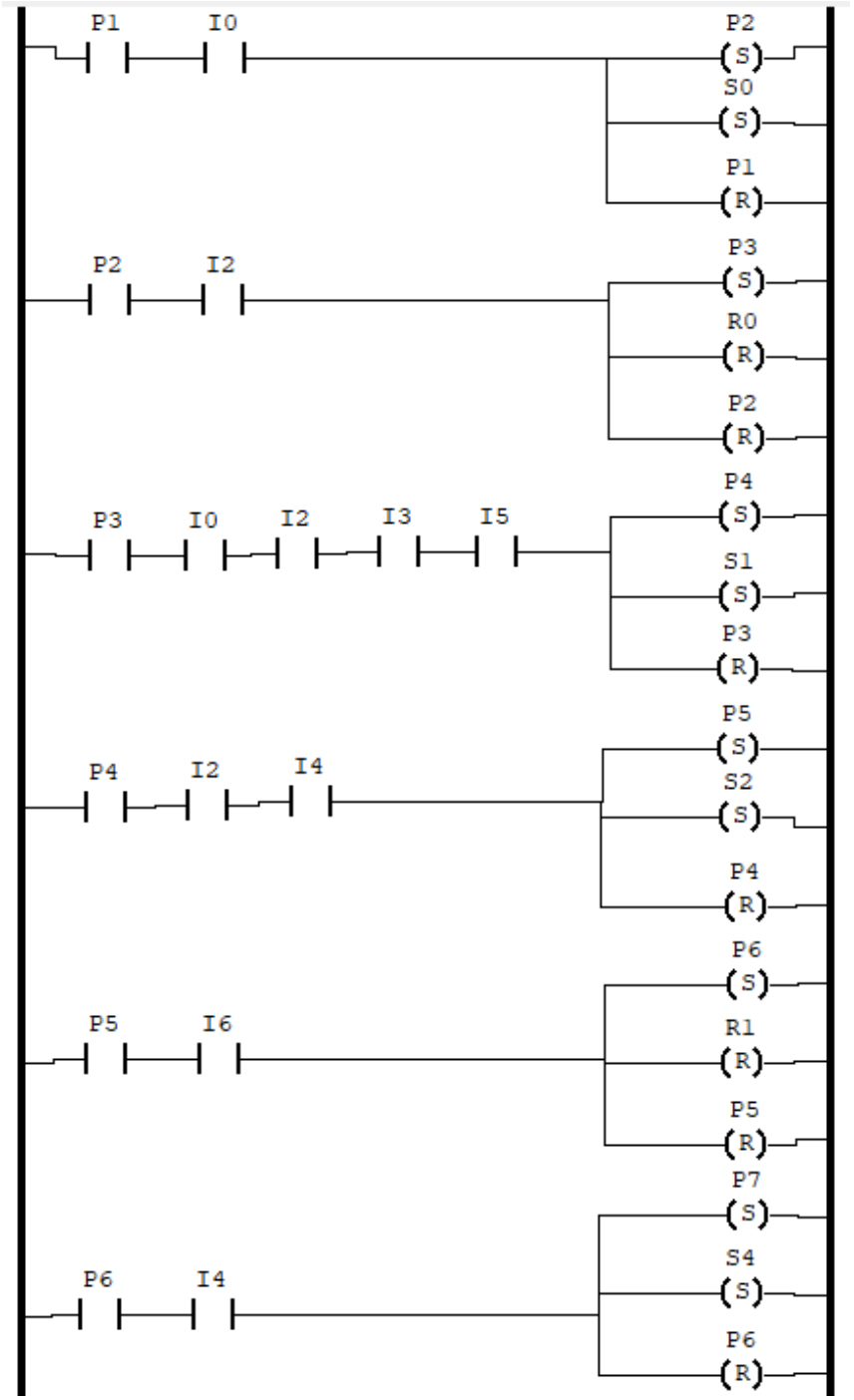
A Tabela 5 ilustra a lista de entradas e saídas para o sistema.

I0	Sensor pistão de alimentação recuado
I2	Sensor peça na máquina
I3	Sensor pistão prende peça recuado
I4	Sensor pistão prende peça avançado
I5	Sensor tipo de peça
I6	Sensor fim de usinagem
I8	Sensor pistão descarrega peça recuado
I9	Sensor conta peça
S/R0	Avança/Recua pistão de alimentação
S/R1	Avança/Recua pistão prende peça

S/R2	Ativa/Desativa usinagem da peça
S/R4	Avança/Recua pistão de descarga da peça

Tabela 5 - Lista de entradas e saídas do projeto de carga de peças.

A Figura 39 mostra o diagrama Ladder convertido manualmente para verificação e a Figura 40 mostra o diagrama Ladder convertido de forma automática pela aplicação.



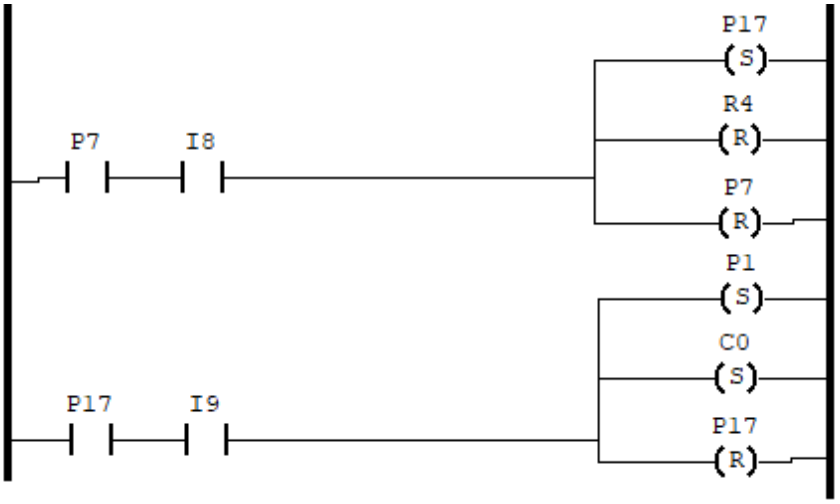
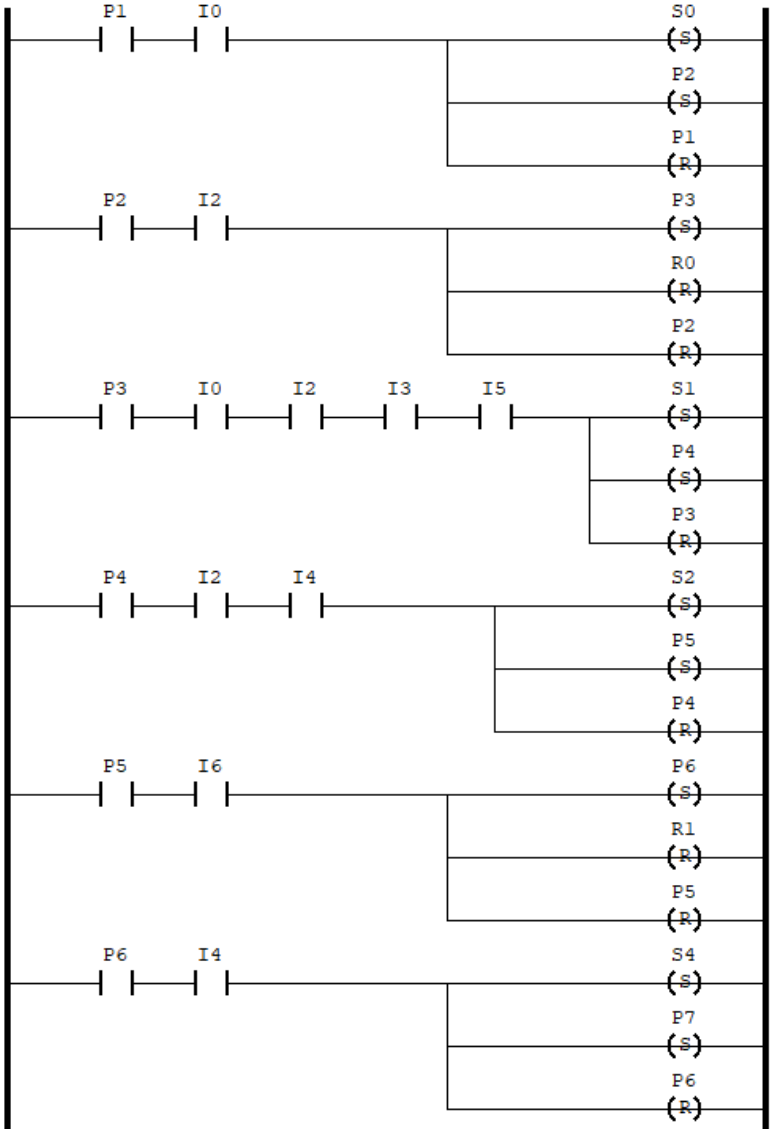


Figura 42 - Diagrama Ladder convertido manualmente para o projeto de carga e descarga de peças.



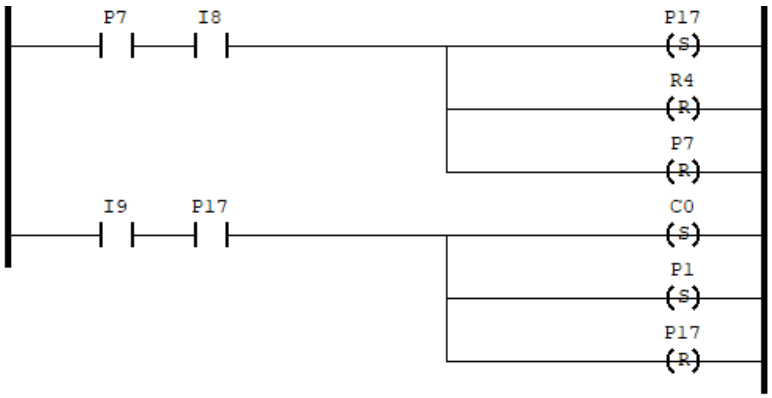


Figura 43 - Diagrama Ladder obtido pelo programa para o projeto de carga e descarga de peças.

6. Conclusão

Pode-se dizer que o trabalho atingiu o objetivo; os programas Ladder obtidos são correspondentes às Redes de Petri apresentadas, traduzindo a rede de forma automática, auxiliando e acelerando a implementação do sistema, avisando o usuário do resultado da conversão e possuindo uma execução prática e rápida, com poucas dificuldades de configuração e preparo, com a aplicação possuindo como limitações não identificar arcos com negação nem múltiplas marcações de forma automática.

As principais dificuldades encontradas no trabalho foram a interpretação e geração do XML, necessitando de lógicas complexas para identificar especificidades de elementos como transições e utilizando esquemas XML estritos com necessidade de cálculo de posições gráficas precisas; e dificuldades com os editores, onde no PIPE não conseguimos representar de forma fácil transições com mais de uma ação, utilizar a mesma identificação para um elemento mais de uma vez ou identificar o tipo da transição de forma nativa.

Sugestões de expansão do sistema para trabalhos futuros incluem:

- Adicionar a capacidade de converter o código para trabalhar com CLPs específicos;
- Adicionar um visualizador do diagrama Ladder para que o usuário possa validar o resultado dentro do próprio software;
- Adicionar mais elementos como PIDs, operadores de byte, operadores binários, etc...

Referências Bibliográficas

KATO, E. R. R.. **Projeto com Controlador Lógico Programável (CLP) - Utilizando Modelagem em Redes de Petri**. [S. l.: s. n.], 2023. Disponível em: https://sites.google.com/ufscar.br/edilsonkato/livros. Acesso em: 8 jul. 2023.

PETRUZELLA, F. D. **Programmable Logic Controllers**. Fifth. ed. [S. l.]: McGraw-Hill Education, 2019. Disponível em: Mc-GrawHill Education. Acesso em: 8 jul. 2023.

BOLTON, W. **Programmable Logic Controllers**. Sixth. ed. [S. l.: s. n.], 2015. Disponível em: Elsevier. Acesso em: 8 jul. 2023.

ALPHONSUS, E. R.; ABDULLAH, M. O. **A review on the applications of programmable logic controllers (PLCs)**. Renewable and Sustainable Energy Reviews, [s. l.], 27 fev. 2016. Disponível em: Science Direct. Acesso em: 8 jul. 2023.

WANG, J. **Petri Nets for Dynamic Event-Driven System Modeling**. Petri Nets, [s. l.], 1 jan. 2007. Disponível em: ResearchGate. Acesso em: 8 jul. 2023.

PARK, S. C.; KO, M.; CHANG, M. **A reverse engineering approach to generate a virtual plant model for PLC si-mulation**. The International Journal of Advanced Manufacturing Technology, [s. l.], 7 ago. 2013. Disponível em: Springer Link. Acesso em: 8 jul. 2023.

SØRENSEN, J. V.; MA, Z.; JØRGENSEN, B. N. **Potentials of game engines for wind power digital twin development: an investigation of the Unreal Engine**. Energy Informatics, [s. l.], 21 dez. 2022. Disponível em: Springer Link. Acesso em: 8 jul. 2023.

RODRIGO, W. A. S. **Interface Rede de Petri para Ladder de Projeto de Sistemas Automatizados**. Universidade Federal de São Carlos, 2018.

MURATA, T. **Petri Net: Properties, analysis and appli-cations**. Proceedings of the IEEE, v.77, n.4, p.541-579, 1989.

KAID, T; AL-AHMARI, A.; LI, Z. **Colored Resource-Oriented Petri Net Based Ladder Diagrams for PLC Implementation in Reconfigurable Manufacturing Systems**. IEEE Access, [s. l.], 1 dez. 2020. Disponível em: IEEEXplore. Acesso em: 8 ago. 2023.

LUO, J.; ZHANG, Q.; CHEN, X.; ZHOU, M. **Modeling and Race Detection of Ladder Diagrams via Ordinary Petri Nets**. IEEE Transactions on Systems, Man and Cybernetics: Sys-tems, [s. l.], 14 mar. 2017. Disponível em: IEEEXplore. Acesso em: 8 ago. 2023.

MACÊDO, C.; OLIVEIRA, E. **Uma API para Conversão de Programas em Ladder em Modelos de Redes de Petri Coloridas**. Universidade Federal de Alagoas, 2019. Acesso em: 10 fev. 2025.

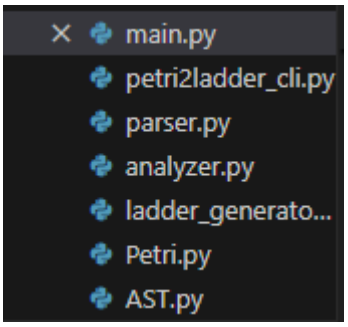
FEIO, R. J. P. M. **Execução de redes de Petri em Autómatos industriais**. Universidade Nova de Lisboa, 2016.

VIEIRA, A. D.; SANTOS, E. A. P.; QUEIROZ, M. H. de; LEAL, A. B.; NETO, A. D. de P.; CURY, J. E. R. **A Method for PLC Implementation of Supervisory Control of Discrete Event Systems**. IEEE Transactions on Control Systems Technology, vol. 25, no. 1, pp. 175-191, Jan. 2017. Disponível em IEEEXplore. Acesso em: 11 fev. 2025.

KATO, E. R. R. **Modelagem da Programação CLP.** Universidade Federal de São Carlos, 2025.

Apêndice

Os códigos são apresentados neste apêndice e, para facilitar futuros usos do sistema, os códigos e as ferramentas utilizadas estarão disponíveis pelo GitHub: https://github.com/EduardoLSS/ConversorPetri2Ladder



main.py

```
import flet as ft
import petri2ladder_cli as p2l
from flet import FilePicker, FilePickerResultEvent, TextField, ElevatedButton, Page, Row, Column, VerticalDivider, \
    MainAxisAlignment, CrossAxisAlignment, Image
import graphviz
import base64
from io import BytesIO
import xml.etree.ElementTree as ET


def main(page: Page):
    page.title = "Conversor de PNML"
    page.window_width = 1600
    page.window_height = 900

    origin_file_path = ""
    destination_file_path = ""

    xml_content = TextField(label="Arquivo XML (PNML)", multiline=True, width=700, height=400)
    graph_image = Image(width=600, height=800)

    def show_dialog(title, text):
        dialog = ft.AlertDialog(
            title=ft.Text(title),
```

```
            content=ft.Text(text),
            actions=[ft.TextButton("Clique fora da caixa para sair", on_click=lambda e:
close_dialog(dialog, page))],
        )
        page.overlay.append(dialog)
        dialog.open = True
        page.update()

    def close_dialog(dialog, page):
        dialog.open = False
        page.overlay.remove(dialog)
        page.update()

    def on_file_picker_result(e: FilePickerResultEvent):
        nonlocal origin_file_path, destination_file_path
        if e.files:
            try:
                with open(e.files[0].path, 'r') as file:
                    origin_file_path = e.files[0].path
                    diretorio, _, _ = origin_file_path.rpartition("\\")
                    destination_file_path = f"{diretorio}\\plc.xml"
                    xml_content.value = file.read()
                page.update()
                display_graph(xml_content.value)
            except Exception as ex:
                print(f"Erro ao ler o arquivo: {ex}")

    def on_convert_click(e):
        if not origin_file_path:
            show_dialog("Erro", "Selecione um arquivo antes de iniciar a conversão")
            return

        if xml_content.value:
            try:
                p2l.petri2ladder(origin_file_path, destination_file_path)
                print("Sucesso na conversão do arquivo")
                show_dialog("Sucesso", "O arquivo xml foi gerado e salvo!")
            except Exception as ex:
                print(f"Erro na conversão do arquivo: {ex.stderr}")

    def display_graph(xml_str):
        try:
```

```
        dot = pnml_to_dot(xml_str)
        print(f"DOT: {dot}")
        graph = graphviz.Source(dot)
        img = BytesIO()
        img.write(graph.pipe(format='png'))
        img.seek(0)
        img_data = base64.b64encode(img.read()).decode('utf-8')
        graph_image.src_base64 = img_data
        page.update()
    except Exception as ex:
        print(f"Erro ao exibir o gráfico: {ex}")

def pnml_to_dot(xml_str):
    try:
        root = ET.fromstring(xml_str)
        dot = ['digraph G {']
        dot.append('    rankdir=TB;')

        # Mapeia os nós
        nodes = {}
        for place in root.findall(".//place"):
            place_id = place.get('id')
            nodes[place_id] = place_id
            dot.append(f'    {place_id} [shape=circle,label="{place_id}"];')

        for transition in root.findall(".//transition"):
            transition_id = transition.get('id')
            nodes[transition_id] = transition_id
            dot.append(f'    {transition_id} [shape=box,label="{transition_id}"];')

        top_node = "P0" if "P0" in nodes else "P1" if "P1" in nodes else None
        if top_node:
            dot.append(f'    {{rank=source; {top_node};}}')

        # Mapeia as arestas
        for arc in root.findall(".//arc"):
            source = arc.get('source')
            target = arc.get('target')
            dot.append(f'    {source} -> {target};')

        dot.append('}')
        return '\n'.join(dot)
```

```
        except Exception as ex:
            print(f"Erro ao converter PNML para DOT: {ex}")
            return ""

    file_picker = FilePicker(on_result=on_file_picker_result)
    page.overlay.append(file_picker)

    btn_select_file = ElevatedButton(text="Selecionar Arquivo XML", on_click=lambda _:
file_picker.pick_files())
    btn_convert = ElevatedButton(text="Converter Arquivo", on_click=on_convert_click)

    page.add(
        Row(
            [
                Column(
                    [
                        btn_select_file,
                        xml_content,
                        btn_convert
                    ],
                    alignment=MainAxisAlignment.START,
                    horizontal_alignment=CrossAxisAlignment.CENTER
                ),
                VerticalDivider(),
                Column(
                    [
                        graph_image
                    ],
                    alignment=MainAxisAlignment.START,
                    horizontal_alignment=CrossAxisAlignment.CENTER
                ),
            ],
            alignment=MainAxisAlignment.SPACE_BETWEEN,
            height=1080
        )
    )


ft.app(target=main)
```

petri2ladder_cli.py

```
import parser
import analyzer as AL
import ladder_generator as LG

def petri2ladder(source, destination):
  filepath = source
  dest = destination
  petrinet = parser.netFromFile(filepath)
  astNet = AL.analyze_net(petrinet)
  LG.generate_xml(astNet,dest)
```

parser.py

```
import xml.etree.ElementTree as ElementTree
import Petri

def xml2Net(xml):
  net = Petri.Net()
  net.places = [placeFromElement(x) for x in xml.findall(".//place")]
  net.arcs = [arcFromElement(x) for x in xml.findall(".//arc")]
  net.transitions = [transitionFromElement(x) for x in xml.findall(".//transition")]
  return net

def arcFromElement(element):
  source = element.get("source")
  destination = element.get("target")
  inhibitor = element.find("type").get("value") == "inhibitor"
  return Petri.Arc(source,destination, inhibitor)

def placeFromElement(element):
  identifier = element.get("id")
  name = element.find("name").find("value").text
  return Petri.Place(identifier, name)

def transitionFromElement(element):
  identifier = element.get("id")
  name = element.find("name").find("value").text
  return Petri.Transition(identifier, name)

def netFromFile(filepath=None):
  with open(filepath) as xmlFile:
```

```
    xml = ElementTree.parse(xmlFile)
    return xml2Net(xml.getroot())
```

analyzer.py

```
import re
import AST
import Petri

def analyze_net(petrinet):
  # Validate Petri Net
  for arc in petrinet.arcs:
    hasPlaceSource = [place for place in petrinet.places if arc.source_id == place.id]
    hasPlaceDestination = [place for place in petrinet.places if arc.destination_id == place.id]
    hasTransitionSource = [transition for transition in petrinet.transitions if arc.source_id ==
transition.id]
    hasTransitionDestination = [transition for transition in petrinet.transitions if
arc.destination_id == transition.id]

  if hasPlaceSource and hasPlaceDestination:
    raise ValueError("Petri Net not suported")
  if hasTransitionSource and hasTransitionDestination:
    raise ValueError("Petri Net not suported")

  # Construct Symbols
  arc_destinations = [arc.destination_id for arc in petrinet.arcs]
  arc_sources = [arc.source_id for arc in petrinet.arcs]
  flags_places = [place for place in petrinet.places if place.id in arc_destinations and
place.id in arc_sources]
  input_places = [place for place in petrinet.places if place.id not in arc_destinations and
place.id in arc_sources]
  laddernetList = []

  for transition in petrinet.transitions:
    arcs_input = [arc.source_id for arc in petrinet.arcs if arc.destination_id == transition.id]
    normal_inputs = [place for place in petrinet.places if place.id in arcs_input and not
arc.negated]
    negated_inputs = [place for place in petrinet.places if place.id in arcs_input and
arc.negated]
    arcs_output = [arc.destination_id for arc in petrinet.arcs if arc.source_id == transition.id]
    outputs = [place for place in petrinet.places if place.id in arcs_output]
```

```
    if not normal_inputs and not negated_inputs:
      raise ValueError("Petri Net invalida: Não contem input")
    if not outputs:
      raise ValueError("Petri Net invalida: não contem output")

    normalConditions = [AST.Input(place.id, False) for place in normal_inputs]
    negatedConditions = [AST.Input(place.id, True) for place in negated_inputs]
    resetOutputList = [AST.ResetOutput(place.id) for place in normal_inputs if place in
flags_places]
    setOutputList = [AST.SetOutput(place.id) for place in outputs]

    outTransition = outputForTransition(transition)
    if isinstance(outTransition,AST.SetOutput):
      setOutputList.insert(0, outTransition)
    if isinstance(outTransition,AST.ResetOutput):
      resetOutputList.insert(0, outTransition)

    conditions = normalConditions
    conditions.extend(negatedConditions)

    laddernet = AST.LadderNet(conditions, setOutputList, resetOutputList)
    laddernetList.append(laddernet)

  return sorted(
    laddernetList,
    key=lambda item: (
      min(
        (int(input.identifier[1:]) for input in item.conditions if input.identifier.startswith('P')),
        default=float('inf')
      )
    )
  )

def outputForTransition(transition):
  if transition.id.startswith('S') or transition.id.startswith('C'):
    return AST.SetOutput(transition.id)
  elif transition.id.startswith('R'):
    return AST.ResetOutput(transition.id)
  else:
    return None
```

ladder_generator.py

```
#!/usr/bin/env python3
import xml.etree.ElementTree as ElementTree
import copy
import AST

localId = 3
rightTrailOffset = 300
positionX = 256
positionY = 150

def generate_xml(ast, filepath):
  calculateRightTrailDistance(ast)
  project = xmlProject(ast)
  xmlString = ElementTree.tostring(project, encoding='utf-8', method='xml').decode()
  with open(filepath, 'w') as xmlFile:
    xmlFile.write(xmlString)

def xmlFileHeader(root):
  header = ElementTree.SubElement(root,'fileHeader')
  header.set('companyName','UFSCar')
  header.set('productName','Conversor PNML2Ladder')
  header.set('productVersion','1')
  header.set('creationDateTime','2025-01-15T12:13:14')

def xmlContentHeader(root):
  header = ElementTree.SubElement(root,'contentHeader')
  header.set('name','Desconhecido')
  header.set('modificationDateTime','2025-01-15T12:13:14')
  info = ElementTree.SubElement(header, 'coordinateInfo')
  pageSize = ElementTree.SubElement(info,'pageSize')
  pageSize.set('x','3000')
  pageSize.set('y','3000')
  fbd = ElementTree.SubElement(info, 'fbd')
  fdb_scaling = ElementTree.SubElement(fbd, 'scaling')
  fdb_scaling.set('x','0')
  fdb_scaling.set('y','0')
  ld = ElementTree.SubElement(info, 'ld')
  ld_scaling = ElementTree.SubElement(ld, 'scaling')
  ld_scaling.set('x','0')
  ld_scaling.set('y','0')
  sfc = ElementTree.SubElement(info, 'sfc')
```

```
  sfc_scaling = ElementTree.SubElement(sfc, 'scaling')
  sfc_scaling.set('x','0')
  sfc_scaling.set('y','0')

def xmlLeftTrail(root, ast):
  leftTrail = ElementTree.SubElement(root, 'leftPowerRail')
  setLocalId(leftTrail, 1)
  xmlPosition(leftTrail, positionX, positionY - 20)
  size_for_each_net = [len(net.setOutputList) + len(net.resetOutputList) for net in ast]
  offset = 0
  current_size = 0
  for i in range(len(ast)):
    if i > 0:
      offset += size_for_each_net[i-1]
    current_size = 20 + 40 * offset
    connectionPointOut = ElementTree.SubElement(leftTrail, 'connectionPointOut')
    connectionPointOut.set('formalParameter','')
    relPosition = ElementTree.SubElement(connectionPointOut, 'relPosition')
    relPosition.set('x','3')
    relPosition.set('y',str(current_size))
  leftTrail.set('width','3')
  leftTrail.set('height',str(current_size+20))

def calculateRightTrailDistance(ast):
  global rightTrailOffset
  maxLength = 0
  for conditionList in ast:
    if len(conditionList.conditions) > maxLength:
      maxLength = len(conditionList.conditions)
  rightTrailOffset = 300 + 30 * (maxLength + 1)

def xmlRightTrail(root, ast):
  rightTrail = ElementTree.SubElement(root, 'rightPowerRail')
  setLocalId(rightTrail, 2)
  xmlPosition(rightTrail, positionX + rightTrailOffset, positionY - 20)
  size_for_each_net = [len(net.setOutputList) + len(net.resetOutputList) for net in ast]
  totalOutputs = sum(size_for_each_net)
  current_size = 0
  for i in range(totalOutputs):
    current_size = 20 + 40 * i
  rightTrail.set('width','3')
  rightTrail.set('height',str(current_size+20))
```

```
  return rightTrail

def xmlLadderNet(root, rightTrail, ast):
  line = 0
  for i in range(len(ast)):
    net = ast[i]
    previousId = '1'
    previousPosX = 0
    previousPosY = 0
    for j in range(len(net.conditions)):
      conn = net.conditions[j]
      referenceId = copy.deepcopy(previousId)
      previousId, previousPosX, previousPosY = xmlContact(root,conn, referenceId, line, j, j
!= 0)
    setOutNumber = len(net.setOutputList)
    for j in range(setOutNumber):
      conn = net.setOutputList[j]
      xmlCoil(root, conn, rightTrail, copy.deepcopy(previousId), (previousPosX,
previousPosY), line + j, len(net.conditions) + 1, True)
    line = line + setOutNumber
    resetOutNumber = len(net.resetOutputList)
    for j in range(resetOutNumber):
      conn = net.resetOutputList[j]
      xmlCoil(root, conn, rightTrail, copy.deepcopy(previousId), (previousPosX,
previousPosY), line + j, len(net.conditions) + 1, False)
    line = line + resetOutNumber

def xmlContact(root, conn, previousId, line, offset, correctId):
  contact = ElementTree.SubElement(root, 'contact')
  updateLocalId(contact)
  if conn.negated:
    contact.set('negated','true')
  else:
    contact.set('negated','false')
  x,y = xmlContactCoilPosition(contact,line,offset)
  previousX, previousY = contactCoilPosition(line,offset-1)
  xmlContactCoilConnectionIn(contact,previousId, (previousX+21,previousY), (x,y),
correctId)
  xmlContactCoilConnectionOut(contact)
  xmlContactCoilName(contact, conn.identifier)
  xmlContactCoilSize(contact)
```

```
  global localId
  return localId, x, y

def xmlCoil(root, conn,rightTrail, previousId, previousPos, line, offset, isSet):
  coil = ElementTree.SubElement(root,'coil')
  updateLocalId(coil)
  coil.set('negated','false')
  if isSet:
    coil.set('storage','set')
  else:
    coil.set('storage','reset')
  x,y = xmlContactCoilPosition(coil,line,offset,1)
  previousX, previousY = previousPos
  xmlContactCoilConnectionIn(coil,previousId,(previousX + 21, previousY),(x,y), True)
  xmlContactCoilConnectionOut(coil)
  xmlContactCoilName(coil, conn.identifier)
  xmlContactCoilSize(coil)
  global localId
  xmlContactCoilConnectionIn(rightTrail,localId, (x,y), (positionX + rightTrailOffset,
positionY + 21), True)

  return x, y

def xmlContactCoilConnectionIn(root,referenceId, previousPosition, currentPosition,
correctId = False):
  connIn = ElementTree.SubElement(root,'connectionPointIn')
  relPosition = ElementTree.SubElement(connIn, 'relPosition')
  relPosition.set('x','0')
  relPosition.set('y','8')
  conn = ElementTree.SubElement(connIn, 'connection')
  if correctId:
    conn.set('refLocalId',str(referenceId-1))
  else:
    conn.set('refLocalId',str(referenceId))

  currentX, currentY = currentPosition
  previousX, previousY = previousPosition

  xmlPosition(conn, currentX, currentY + 8)
  if previousY != currentY:
    xmlPosition(conn, round((previousX + currentX)/2), currentY + 8)
    xmlPosition(conn, round((previousX + currentX)/2), previousY + 8)
```

```
  xmlPosition(conn,previousX, previousY + 8)

def xmlPosition(root, x, y):
  position = ElementTree.SubElement(root, 'position')
  position.set('x',str(x))
  position.set('y',str(y))

def contactCoilPosition(line, offset):
  global positionY
  global positionX
  y = line * 40 + positionY - 8
  x = (offset + 1) * 60 + positionX
  return (x,y)

def xmlContactCoilPosition(root, line, offset, isCoil = 0):
  global positionX, rightTrailOffset
  x,y = contactCoilPosition(line,offset)
  if isCoil == 1:
    x = positionX + rightTrailOffset - 60
  xmlPosition(root, x, y)
  return x,y

def xmlContactCoilConnectionOut(root):
  out = ElementTree.SubElement(root,'connectionPointOut')
  relPosition = ElementTree.SubElement(out, 'relPosition')
  relPosition.set('x','21')
  relPosition.set('y','8')

def xmlContactCoilName(root, name):
  variable = ElementTree.SubElement(root, 'variable')
  variable.text = name

def xmlContactCoilSize(root):
  root.set('height','15')
  root.set('width','21')

def updateLocalId(root):
  global localId
  setLocalId(root, localId)
  localId = localId + 1

def setLocalId(root, newId):
```

```
  root.set('localId',str(newId))

def xmlLadder(root, ast):
  ladder = ElementTree.SubElement(root,'LD')
  xmlLeftTrail(ladder, ast)
  rightTrail = xmlRightTrail(ladder, ast)
  xmlLadderNet(ladder, rightTrail, ast)

def xmlTypes(root, ast):
  types = ElementTree.SubElement(root,'types')
  dataTypes = ElementTree.SubElement(types, 'dataTypes')
  pous = ElementTree.SubElement(types, 'pous')
  pou = ElementTree.SubElement(pous, 'pou')
  pou.set('name','instance0')
  pou.set('pouType','program')
  interface = ElementTree.SubElement(pou,'interface')
  body = ElementTree.SubElement(pou, 'body')
  xmlLadder(body, ast)

def xmlInstanceConfig(root):
  instances = ElementTree.SubElement(root,'instances')
  configurationList = ElementTree.SubElement(instances,'configurations')
  configuration = ElementTree.SubElement(configurationList,'configuration')
  configuration.set('name','config0')
  resource = ElementTree.SubElement(configuration,'resource')
  resource.set('name','resource1')
  task = ElementTree.SubElement(resource,'task')
  task.set('name','task0')
  task.set('priority','0')
  task.set('interval','T#20ms')
  pou = ElementTree.SubElement(task,'pouInstance')
  pou.set('name','instance0')
  pou.set('typeName','program0')

def xmlProject(ast):
  root = ElementTree.Element('project')
  root.set('xmlns','http://www.plcopen.org/xml/tc6_0201')
  root.set('xmlns:ns1','http://www.plcopen.org/xml/tc6.xsd')
  root.set('xmlns:xhtml','http://www.w3.org/1999/xhtml')
  root.set('xmlns:xsd','http://www.w3.org/2001/XMLSchema')

  xmlFileHeader(root)
```

```
  xmlContentHeader(root)
  xmlTypes(root,ast)
  xmlInstanceConfig(root)
  return root
```

Petri.py

```
class Place:
  def __init__(self, place_id, place_name):
    self.id = place_id
    self.name = place_name

  def __repr__(self):
    return "Place(%s, %s)" % (self.id, self.name)

class Transition:
  def __init__(self, transition_id, transition_name):
    self.id = transition_id
    self.name = transition_name

  def __repr__(self):
    return "Transition(%s, %s)" % (self.id, self.name)

class Arc:
  def __init__(self, source_id, destination_id, negated=False):
    self.source_id = source_id
    self.destination_id = destination_id
    self.negated = negated

  def __repr__(self):
    return "Arc(%s, %s, %s)" % (self.source_id, self.destination_id, self.negated)

class Net:
  def __init__(self):
    self.places = []
    self.transitions = []
    self.arcs = []

  def __repr__(self):
    return " places: %s \n transitions: %s \n arcs: %s " % (str(self.places), str(self.transitions),
str(self.arcs))
```

AST.py

```
# --- AST ---
# --- [Condition] [Resets] [Sets]
class LadderNet:
  def __init__(self, conditions, setOutputList, resetOutputList):
    self.conditions = conditions
    self.setOutputList = setOutputList
    self.resetOutputList = resetOutputList

  def __repr__(self):
    return "AST.LadderNet: conditions=%s, sets=%s, resets=%s " % (str(self.conditions),
str(self.setOutputList), str(self.resetOutputList))

class Input:
  def __init__(self, identifier, negated = False, output = None):
    self.identifier = identifier
    self.negated = negated
    self.output = output

  def __repr__(self):
    if self.output is None:
      return "AST.Input(id=%s, negated=%s, output=None)" % (self.identifier, self.negated)
    else:
      return "AST.Input(id=%s, negated=%s, output=%s)" % (self.identifier,
self.negated,self.output)

class ResetOutput:
  def __init__(self, identifier):
    self.identifier = identifier
  def __repr__(self):
    return "AST.ResetOutput(id=%s)" % self.identifier

class SetOutput:
  def __init__(self, identifier):
    self.identifier = identifier
  def __repr__(self):
    return "AST.SetOutput(id: %s)" % self.identifier
```