UNIVERSIDADE FEDERAL DE ALAGOAS-UFAL
CAMPUS DE ARAPIRACA
CIÊNCIA DA COMPUTAÇÃO

CARLOS HENRIQUE DE MACÊDO

LTP - CONVERSOR DE LADDER PARA REDES DE PETRI COLORIDAS

ARAPIRACA
2017

Carlos Henrique de Macêdo

LtP - Conversor de Ladder para Redes de Petri Coloridas

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Alagoas - UFAL, Campus de Arapiraca.

Orientador: Prof. Dr. Elthon Allex da Silva Oliveira

Arapiraca
2017

Carlos Henrique de Macêdo

LtP - Conversor de Ladder para Redes de Petri Coloridas

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal de Alagoas - UFAL, Campus de Arapiraca.

Data de Aprovação: 08/06/2017

Banca Examinadora

Prof. Dr. Elthon Allex da Silva Oliveira
Universidade Federal de Alagoas
Campus Arapiraca
Orientador

Prof. Dr. Rodolfo Carneiro Cavalcante
Universidade Federal de Alagoas
Campus Arapiraca
Examinador

Prof. Me. André Almeida Silva
Universidade Federal de Alagoas
Campus Arapiraca
Examinador

Dedico este trabalho a Deus e minha família. Principalmente a minha mãe, pelo apoio incondicional.

AGRADECIMENTOS

Agradeço a Deus pela dádiva da vida. Em seguida gostaria de agradecer, principalmente, a minha mãe pelo apoio e motivação desde o início. Agradeço minha família, por entender meus horários de estudo em casa. Gostaria também de agradecer aos professores do curso que me fizeram amadurecer, aprender e desenvolver minhas habilidades durante esses 4 anos, em especial ao Elthon Allex da Silva Oliveira pela orientação e apoio.

Agradeço a todos amigos que fiz durante o curso. Vocês me ajudaram bastante, tanto em debates das disciplinas, quanto em tornar a universidade mais divertida. Vocês se tornaram pessoas especiais para mim – Adriano, Arthur, Pedro, Matheus B., Matheus T., Allan, Wanne, Icaro, Fillipe, Luiz, Luis, Daniel, Irving, Eric, João Paulo, Handerson, Jeillysson, Evertton, Mary, Anderson, Paulo, Juliano, Igor, Monally, Mirelle, Kamila, Welysson, Vanessa, Evellynn, Bartolomeu, Maylson, Carlos.

Por fim, um sentimento especial para o grupo RQL. Vocês, certamente, fizeram esses 4 anos se tornarem mais fáceis.

Aja antes de falar e, portanto, fale de acordo com os seus atos.

Confúcio

RESUMO

Há vários processos críticos na indústria que são automatizados com o objetivo de otimizar os recursos disponíveis e aumento na produção, como processamento de gás e petróleo. Ladder é uma linguagem utilizada no desenvolvimento dos sistemas para indústria que utilizam Controladores Lógico Programáveis. Contudo, os sistemas possuem problemas de garantia de confiabilidade e segurança. Para simular e verificar propriedades dos sistemas, são utilizadas ferramentas matemáticas como as Redes de Petri (PN). Alguns pesquisadores propõem métodos para converter Ladder em alguma classe de PN. Apesar dos vários métodos de conversão existentes, não existem implementações desses métodos que consigam ser utilizadas na indústria sem a implementação de recursos adicionais nas ferramentas existentes. Este trabalho apresenta os métodos de conversão existentes, compara-os brevemente e apresenta uma implementação de um deles. Neste trabalho é apresentado o conversor LtP. O LtP converte Ladder em Rede de Petri Colorida, que é uma classe PN. Além disso, o LtP é comparado com o único conversor encontrado na literatura.

Palavras-chave: Diagramas Ladder. Redes de Petri. Conversão.

ABSTRACT

There are several critical processes in the industry that are automated with the goal of optimizing the available resources and increasing the production such as gas and oil. Ladder is a language used in the development of systems developed to the industry that use Programmable Logic Controllers. However, the systems have reliability and security issues. To simulate and verify the properties of the systems, it is used mathematical tools like Petri Nets (PN). Some researchers propose methods to convert Ladder in some PN class. Despite the large number of existing conversion methods, there are no implementations of these methods that can be used in the industry without the implementation of additional features in the existing tools. This work presents the existing conversion methods, compares them briefly, and presents an implementation of one of them. This work presents the LtP converter. LtP converts Ladder to Colored Petri Nets, which is a PN class. In addition, the LtP is compared with the only converter found in the literature.

Keywords: Ladder Diagrams. Petri Nets. Conversion.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LD	Diagramas Ladder
PN	Redes de Petri
CPN	Redes de Petri Coloridas
CLP	Controladores Lógicos Programáveis

LISTA DE SÍMBOLOS

$\vee$	OU lógico
$\wedge$	E lógico
$\sim$	NÃO lógico

SUMÁRIO

1 INTRODUÇÃO

Há vários processos na indústria que são automatizados com o objetivo de otimizar os recursos disponíveis e de aumentar a produção (OLIVEIRA et al., 2011). Além disso, os processos industriais estão se tornando cada vez mais complexos e mais difíceis de especificar (ASPAR et al., 2015). Alguns exemplos de processos seriam: processamento de gás/petróleo, desenvolvimento automobilístico, entre outros.

Caso algum processo industrial – geralmente crítico – fique instável ou impreciso, pode se tornar uma ameaça à segurança de seres humanos e do meio ambiente. Sendo assim, é importante assegurar que tais sistemas não apresentem falhas e executem corretamente (OLIVEIRA et al., 2011).

Controladores Lógicos Programáveis (CLP) são utilizados para controlar as máquinas industriais. Diagrama Ladder (LD) é a linguagem mais utilizada para programar os CLP (CHEN et al., 2012). Um dos principais problemas do LD é que não dá suporte a nenhuma técnica de verificação, ou seja, não tem como garantir que o sistema desenvolvido não contém falhas. Contudo, existem algumas técnicas de verificação que utilizam linguagens matemáticas na descrição do comportamento do sistema a ser verificado.

Rede de Petri (PN) é uma representação matemática que serve para a modelagem gráfica de sistemas. Através do modelo do sistema descrito em PN, é possível estudar e analisar o sistema a fim de garantir sua confiabilidade.

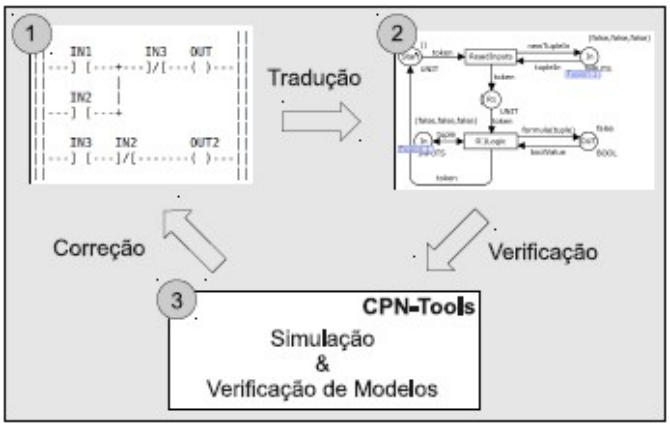
A fim de superar a limitação que o LD possui, alguns pesquisadores[1] propõem a seguinte abordagem – (1) desenvolvimento do sistema em LD, (2) conversão do programa LD criado em uma PN equivalente e (3) análise da PN. Se a PN analisada expor erros do sistema, os erros expostos são corrigidos no programa LD e são realizados os passos (2) e (3) novamente. O processo se repete até não serem encontrados erros. Quando nenhum erro for encontrado, o ciclo termina e o sistema em LD pode ser utilizado. Este processo é ilustrado na Figura 1.

Dois problemas se destacam: a necessidade de tempo extra para modelagem e o treinamento de recursos humanos na linguagem a ser usada[2] (OLIVEIRA et al., 2011). Por conta disso, é natural desejar um conversor automático, a fim de diminuir os custos.

Na literatura, existem várias formas de converter LD em PN. Este trabalho faz uma breve comparação entre os métodos existentes[1] e implementa o mais adequado. O trabalho que foi considerado mais adequado foi o de (OLIVEIRA et al., 2011), cuja justificativa é apresentada na

[1] (OLIVEIRA et al., 2011), (LEE; LEE, 2000), (LATHA; UMAMAHESWARI, 2002) e (CHEN et al., 2012).
[2] Nesse caso PN.

Figura 1 – Visão geral da abordagem seguida por alguns pesquisadores.



Fonte: (OLIVEIRA et al., 2011)

seção 3.3.

Este trabalho implementa o método de conversão criado por (OLIVEIRA et al., 2011). A implementação é denominada de conversor LtP. Como contribuição adicional, criou-se uma biblioteca em JAVA para que seja possível criar uma PN com um maior grau de abstração. Dado que para criar uma PN em tempo de execução era preciso escrever códigos extensos em XML. Com o desenvolvimento da biblioteca, as possíveis implementações dos métodos de conversão propostos são facilitadas (ou qualquer outra aplicação que precise ser modelada em PN).

1.1 OBJETIVOS

1.1.1 Geral

Implementar uma técnica de conversão da linguagem Ladder em Redes de Petri.

1.1.2 Específicos

A fim de atingir o objetivo apresentado, as seguintes tarefas serão desenvolvidas:

- Escolha das ferramentas para a criação/edição de Diagramas Ladder e Redes de Petri Coloridas.
- Escolher a melhor técnica de conversão, entre as existentes na literatura.
- Modelagem UML do sistema.
- Criar uma biblioteca em Java, para que seja possível criar Redes de Petri em tempo de execução, com um maior grau de abstração.

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado da seguinte maneira: no Capítulo 2 é descrita a fundamentação teórica, no Capítulo 3 são descritos trabalhos relacionados, práticos e teóricos. No Capítulo 4 são exibidos detalhes a cerca do desenvolvimento e implementação do conversor LtP. Finalmente, no Capítulo 5, são apresentadas as conclusões e os trabalhos futuros.

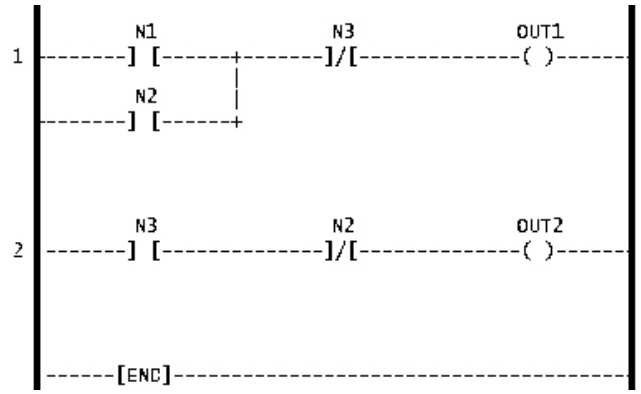
2 Fundamentação Teórica

2.1 LINGUAGEM LADDER

Ladder é uma linguagem de programação gráfica que simula dispositivos de comutação física, baseada nos circuitos de relé (ZHOU; TWISS, 1995). A linguagem é de fácil manipulação, interpretação e representação das conexões físicas entre componentes (sensores e atuadores). Por conta disso, é muito utilizada nos ambientes industriais (OLIVEIRA et al., 2011).

Existem diferentes dialetos de Ladder na indústria, tais como (SIEMENS, 2003) e (RIDLEY, 2004). Os diferentes dialetos servem para CLP diferentes. Tais dialetos possuem suas próprias funções e algumas diferenças visuais. Porém, possuem uma característica em comum, todos seguem o padrão IEC 61131-3[1] (OLIVEIRA et al., 2011).

Figura 2 – Exemplo de diagrama Ladder.



Fonte: Elaborado pelo autor.

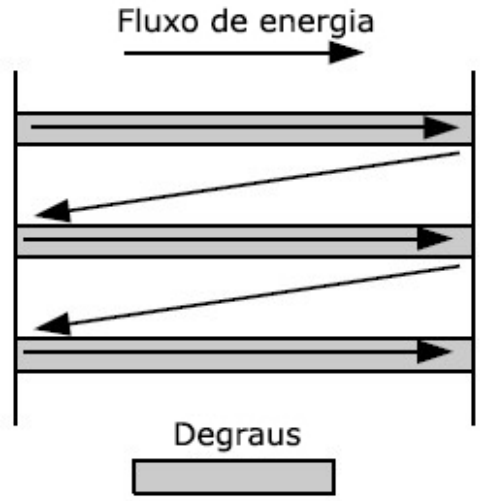
A Figura 2 é um exemplo de um LD. LD representa um circuito elétrico, ou seja, apresenta pelo menos um caminho para a corrente elétrica percorrer. O LD é dividido em degraus[2]. A corrente elétrica percorre da esquerda para direita e de cima para baixo, degrau por degrau, como pode ser visto na Figura 3. A execução do LD – ou melhor, do CLP, afinal, LD é apenas uma abstração dos circuitos de relé – é dividida em ciclos. Chama-se de ciclo quando a corrente elétrica percorre todos os degraus do diagrama. Quando um ciclo finaliza, inicia-se um novo ciclo imediatamente.

Os elementos básicos do LD são os contatos (sensores) e as bobinas (atuadores). Eles são representados, respectivamente, por] [e ( ). Tais elementos podem ser dispostos em séries ou em paralelo. Cada degrau contêm apenas uma bobina, mas pode ter zero ou mais contatos.

[1] Para mais informações sobre IEC 61131-3, consultar (JOHN; TIEGELKAMP, 2010).

[2] O LD da Figura 2 possui dois degraus.

Figura 3 – Fluxo da corrente elétrica.



Fonte: (OLIVEIRA et al., 2011)

Os contatos podem ter dois estados, normalmente abertos ou normalmente fechados, doravante abertos e fechados. Os contatos fechados são representados por] / [. O contato fechado é negação do mesmo contato aberto.

No começo do ciclo, todos sensores (contatos) são lidos e armazenados. Os valores das bobinas são definidos em tempo de execução e dependem dos valores dos contatos. Os valores resultantes das bobinas são liberados apenas no final do ciclo.

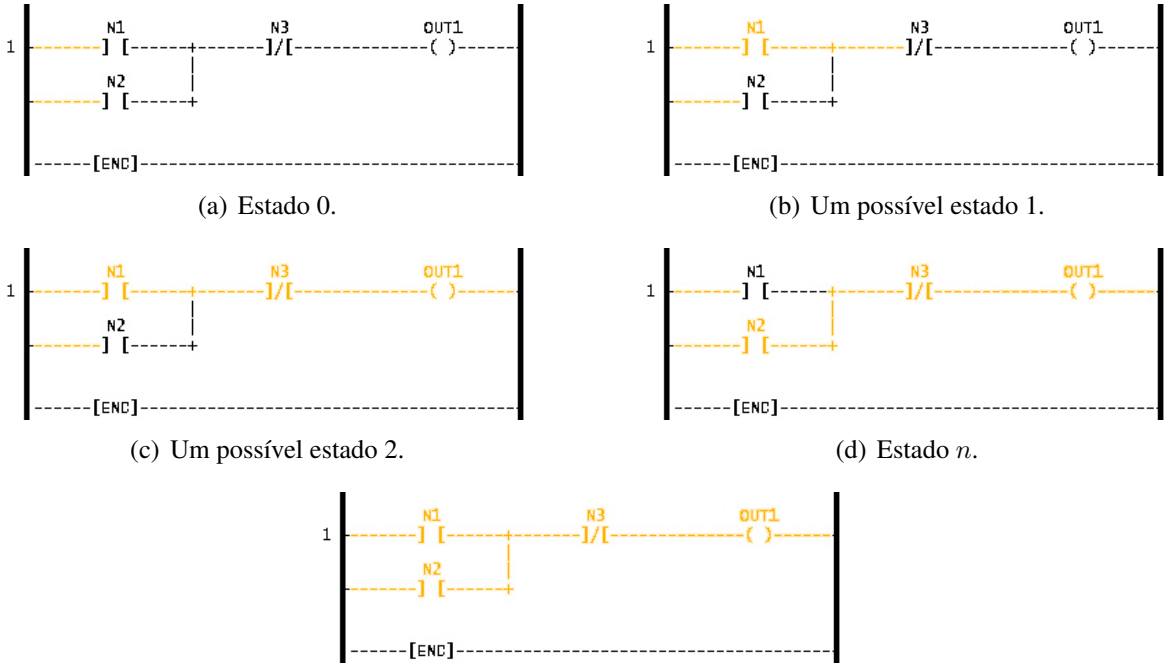
Existem outros elementos – guardas e temporizadores – e operações – atribuição, adição, subtração, divisão e multiplicação – que podem ser utilizados em LD. Contudo, o escopo atual desse trabalho não comporta esses conceitos. Portanto, deixa-se para abordar esses elementos em trabalhos futuros. Caso seja desejado se aprofundar nestes tópicos, procurar em: (OLIVEIRA et al., 2011) e (MADER; WUPPER, 1999).

Degraus que contêm apenas contatos e bobinas são fáceis de expressar os valores resultantes das bobinas, por meio de fórmulas de lógica Booleana. Uma bobina é ativada (tem seu valor 1) quando o fluxo de energia a atinge. Na prática, os contatos funcionam como “barreiras” que impedem (ou não) o fluxo de energia; ou seja, se o contato for aberto e tiver o valor 1 deixa o fluxo passar, se o valor for 0 não deixa. Para o contato fechado o comportamento é exatamente o contrário, valor 1 fluxo bloqueado, valor 0 fluxo livre.

Para entender melhor, o que foi dito no parágrafo anterior, visualize a Figura 4, onde o fluxo de energia é representado pela cor laranja. Quando N1 e N2 valem 0, bloqueiam o fluxo. No momento que N1 muda seu valor para 1, o fluxo segue em diante. Na Figura 4b temos, N1 = 1, N2 = 0, N3 = 1 e OUT1 = 0, ou seja, bobina desativada. Nas Figura 4c, 4d, 4e são ilustrados

alguns exemplos da bobina OUT1 sendo ativada. Com essas imagens é fácil perceber, como dito anteriormente, que a ativação da bobina OUT1 pode ser expressa por uma fórmula Booleana. Nesse caso: (N1 $\vee$ N2) $\wedge$ $\sim$N3.

Figura 4 – Exemplos do fluxo de energia.



(a) Estado 0.

(b) Um possível estado 1.

(c) Um possível estado 2.

(d) Estado n.

(e) Estado z.

Fonte: Elaborado pelo autor.

2.2 REDES DE PETRI COLORIDAS

Redes de Petri (PN) é uma ferramenta matemática bem conhecida. É utilizada para modelagem e análise de sistemas baseados em eventos (LATHA; UMAMAHESWARI, 2002). Sendo adequada, também, para descrever e estudar sistemas concorrentes, assíncronos e distribuídos (MURATA, 1989). Redes de Petri Coloridas (CPN) é uma classe da PN na qual fichas representam tipos de dados complexos, ou seja, conseguem representar vários tipos de dados[3] (OLIVEIRA et al., 2011). Por conta disso, as CPN conseguem ser mais compactas e legíveis, se comparadas às PN. Além disso, existem ferramentas – CPN Tools[4] – que suportam edição, simulação e análise de modelos CPN.

[3] Diferentemente da PN onde as fichas representam um único tipo de dado.

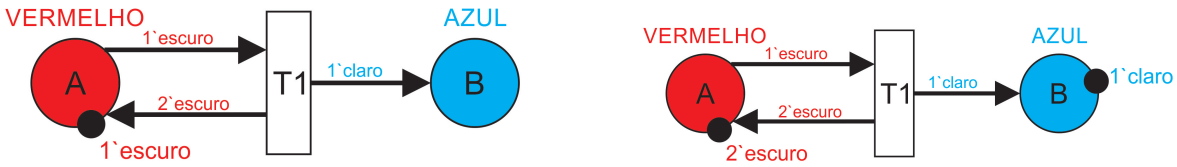
[4] http://cpntools.org/

Nessa seção é feita uma breve introdução a CPN. Para uma abordagem completa consultar os livros (MACIEL et al., 1996) e (JENSEN, 2013).

CPN possui os seguintes elementos básicos:

1. Lugar (círculos ou elipses): representa a disponibilidade de recursos, ou seja, a existência, ou não, de determinadas fichas.
2. Transição (retângulos): evento que deve ocorrer para ela ser disparada.
3. Arcos (setas): são funções de entrada[5] e saída[6].
4. Fichas Coloridas (círculos minúsculos): representam os recursos.

Figura 5 – Exemplo de Rede de Petri Colorida.



(a) Estado Inicial.

(b) Depois de um ciclo.

Fonte: Elaborado pelo autor.

A CPN pode ser entendida, de forma mais simples, como manipulação de fichas, ou seja, várias fichas se movimentando pela rede, a fim de simular um sistema qualquer. Na Figura 5 é mostrado um exemplo de CPN. Será utilizado o exemplo da Figura 5 para explicar o funcionamento da CPN. É importante citar que são ilustrados dois tempos da mesma CPN na Figura 5. Ou seja, (a) é o estado inicial e (b) é o instante seguinte logo após a transição T1 ser disparada.

Alguns conceitos que o leitor deve ter em mente:

- Todo lugar possui um nome e um tipo.
- Toda transição T só é disparada se todos os arcos, que estão ligados a T, do tipo função de entrada, forem ativados.

[5] Função de entrada se o arco tem a direção lugar-transição.
[6] Função de saída se o arco tem a direção transição-lugar.

- Um arco sempre transfere uma certa quantidade de fichas para onde estiver apontando. A quantidade de fichas que o arco transfere é, justamente, a que está sinalizada a cima dele.
- A inscrição do arco é organizada assim: <x>'<nome>. Onde <x> é a quantidade de fichas e <nome> é o nome da ficha.
- Importante: um lugar pode ter qualquer quantidade de fichas. Porém, todas as fichas devem ser do tipo do lugar. Para entender melhor, no exemplo da Figura 5, o lugar é do tipo VERMELHO, então o lugar pode conter fichas do tipo: *vermelho escuro*, *vermelho claro*, etc.

A CPN da Figura 5 possui dois lugares, A e B. Esses lugares são do tipo VERMELHO e AZUL, respectivamente. O lugar A possui uma ficha escuro e o B nenhuma ficha. A CPN possui, também, uma transição T1. Além disso, possui três arcos: (A,T1), (T1,B) e (T1,A). Para a transição T1 ser ativada é necessário que o arco (A,T1) seja ativado. Para o arco (A,T1) ser ativado é necessário que em A tenha pelo menos uma ficha escuro em A, que é o caso[7]. Por conta disso, T1 é ativada. O arco (A,T1) transfere uma ficha de A para T1. T1 ativa todos os arcos do tipo função de saída, que estão conectados a T1 ((T1,A) e (T1,B)). O arco (T1,A) cria duas fichas *escuro* em A e o arco (T1,B) cria uma ficha *claro* em B. O resultado, após isso tudo, é mostrado em 5b. A execução dessa CPN nunca irá parar. Dado que o lugar A sempre terá pelo menos uma ficha para ativar T1 e sempre que T1 é ativada cria novas fichas.

2.3 CONVERSÃO LADDER PARA REDES DE PETRI COLORIDAS

Nessa Seção é apresentado o modo de converter LD em modelos de CPN, desenvolvido em (OLIVEIRA et al., 2011). O método de conversão explicado é o de (OLIVEIRA et al., 2011). Pois, foi o método escolhido para ser implementado. (OLIVEIRA et al., 2011) desenvolve três estágios de conversão: (1) com o LD tendo apenas contatos e bobinas, (2) com contatos, bobinas e variáveis e (3) contatos, bobinas, variáveis e temporizadores. Nesse trabalho não são abordados os estágios (2) e (3), apenas o (1) será apresentado. Pois, na sua atual versão, o conversor LtP suporta apenas contatos e bobinas.

Como dito na seção 2.1, a lógica de controle dos degraus do LD pode ser expressa pela lógica booleana. A fórmula booleana que representa o degrau da bobina OUT1 (Figura 4) é $(N1 \vee N2) \wedge \sim N3$. O modelo CPN equivalente ao LD da Figura 4 é apresentado na Figura 6.

[7] Caso a ficha fosse outra, ou não houvesse fichas, a T1 nunca seria ativada. Por conta disso, a CPN nunca sofreria alterações.

É importante lembrar que o LD tem 5 características importantes, são elas:

(a) Possui um início e um fim do ciclo.

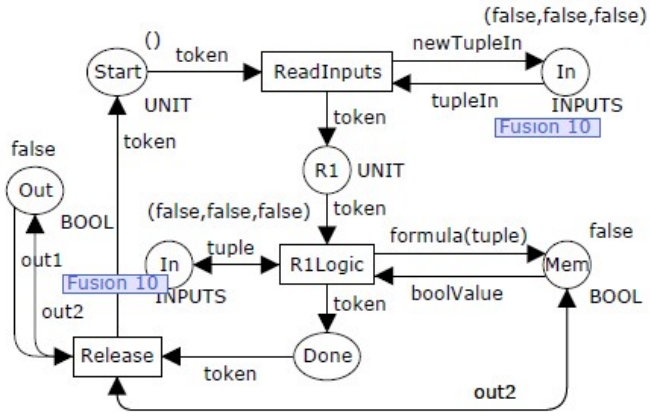
(b) A execução dos ciclos nunca param.

(c) No início do ciclo são lidos os valores de todos sensores.

(d) No fim do ciclo são liberados os valores de todos sensores.

(e) Cada degrau possui uma única bobina. Além disso, é executado um degrau por vez, em ordem.

Figura 6 – Modelo CPN equivalente ao LD da Figura 4



Fonte: (OLIVEIRA et al., 2011)

Os lugares *Start* e *Done* representam a característica (a), ou seja, início e fim do ciclo, respectivamente. O arco (*Release*, *Start*) representa a característica (b). A transição *ReadInputs* faz o papel de (c), enquanto que *Release* exerce (d). Cada degrau (e) é representado pelo lugar *Ri* e pela transição *RiLogic*[8]. Os lugares *In* e *Mem* servem para guardar o valor atual de todos contatos e bobinas, respectivamente. O lugar *Out* é utilizado para mostrar os valores de todas bobinas no final do ciclo. Além disso tudo, todos contatos são representados por uma única tupla – ficha do lugar *In* – onde cada elemento da tupla representa um contato. As bobinas também são representadas por um tupla – ficha do lugar *Mem* – do mesmo modo, cada elemento da tupla representando uma bobina.

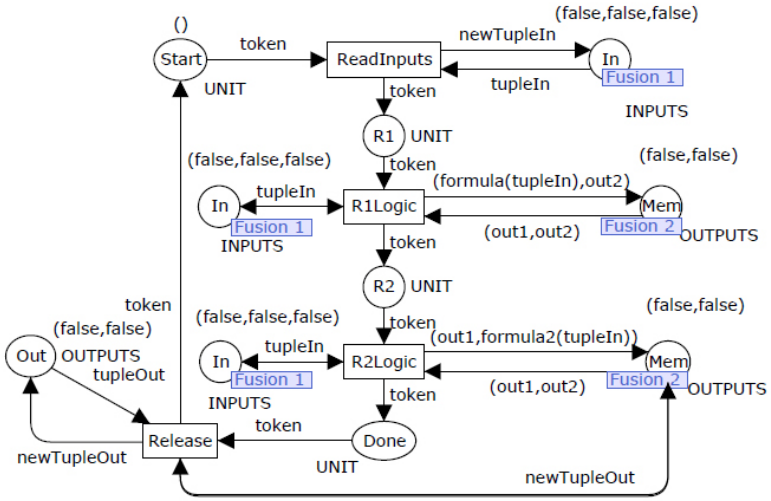
A execução da CPN acontece da seguinte forma – *ReadInputs* é disparada com uma ficha presente em *Start*. Esta transição realiza três ações: retira a ficha de *Start*, atualiza a ficha do lugar *In* e passa uma ficha ao lugar *R1*, que indica que o fluxo de energia está no primeiro degrau.

[8] Nesse caso, R1 e R1Logic, respectivamente.

Dado que a ficha está em *R1*, *R1Logic* dispara e lê todos os contatos (em *In*)[9] e bobinas (em *Mem*). Com base na lógica de controle (*formula(tuple)*), escreve o valor atual da bobina do degrau correspondente[10]. Continuando, *R1Logic* passa uma ficha ao lugar *Done*. Assim, disparando *Release*; que mostra o resultado em *Out* e coloca uma ficha em *Start*, para que comece um novo ciclo.

A função formula representa a lógica de controle. Nesse caso, *(N1 ∨ N2)∧ ∼*N3. Onde recebe os valores dos contatos como parâmetro e retorna o valor da bobina, ou seja, ativada ou não ativada.

Figura 7 – Modelo CPN equivalente ao LD da Figura 2



Fonte: (OLIVEIRA et al., 2011)

A adição de outro degrau ao diagrama não aumenta a complexidade da CPN, apenas demanda a replicação de algumas partes no modelo. Tomando como exemplo o LD da Figura 2, que possui dois degraus. Percebemos (na Figura 7) que as mudanças foram poucas. Apenas (1) adicionou mais um valor na tupla de *Mem*, para representar mais uma bobina. (2) Replicou o lugar *Ri* e *RiLogic*, ou seja, criou *R2* e *R2Logic*.

Nessa rede (Figura 7), *R1Logic* passa o fluxo de energia ao segundo degrau (*R2*). A fórmula tupla agora aparece assim *(formula(tupleIn), out2)* e *(out1, formula2(tupleIn))*; que serve para mostrar que só vai aplicar a fórmula na bobina do degrau correspondente[11].

[9] Esse lugar *In*, é o mesmo que é acessado anteriormente por *ReadInputs*. O lugar apenas está em dois lugares ao mesmo tempo.

[10] Como está no primeiro degrau, escreve o valor no primeiro elemento da tupla.

[11] Vale apena notar que há formula e formula2. Pois, cada degrau possui a sua construção específica implementada por uma função diferente.

3 TRABALHOS RELACIONADOS

Com o objetivo de exibir o estado da arte, nesse capítulo são descritos os principais trabalhos encontrados na literatura que abordam a conversão de Diagramas Ladder (LD) para Rede de Petri (PN). Apesar deste trabalho apresentar um conversor para Rede de Petri Colorida (CPN), não restringiu-se a pesquisa à CPN, ou seja, aborda-se a conversão de LD em qualquer classe PN. Essa escolha foi feita por conta da baixa quantidade de trabalhos em CPN. Por fim, duas tabelas comparativas entre os principais trabalhos encontrados e os motivos da escolha da conversão para a classe CPN são descritos.

Nas últimas décadas, foram desenvolvidos muitos métodos para resolver o problema de confiabilidade dos LD. Os métodos se dividem em:

1) substituir LD por PN, ou seja, baseia-se em comparar LD e PN para decidir qual é mais adequado (normalmente PN) e utilizar apenas um deles;

2) projetar em LD e converter em PN para análise e validação, isto é, mantêm o trabalho com LD, mas utiliza os pontos fortes da PN para análise e validação do sistema;

3) projetar e analisar em PN e converter em LD, ou seja, cria-se todo o sistema em PN analisa-o e, finalmente, converte-o para LD.

Este trabalho enquadra-se no item 2, assim como todos os trabalhos apresentados nesse capítulo. Por conta de espaço, e objetividade, não é abordado nesse trabalho os itens 1 e 3. Não é objetivo deste trabalho identificar qual dos três métodos é o ideal. Cada um dos métodos têm pontos extremamente positivos, são apenas abordagens diferentes. Encontre mais informações dos outros métodos em (PENG; ZHOU, 2004).

Além das diferentes abordagens, citadas no paragrafo anterior, é importante criar mais uma distinção. Os trabalhos relacionados estão divididos em dois grupos: trabalhos teóricos e práticos. Nos trabalhos teóricos são apresentados modos de conversão entre LD e PN propostos por vários autores, entre eles: (OLIVEIRA et al., 2011), (LEE; LEE, 2000), (LATHA; UMAMAHESWARI, 2002) e (CHEN et al., 2012). Enquanto que nos trabalhos práticos são apresentadas propostas de métodos que foram implementados.

O presente trabalho é um trabalho prático. Foi importante criar essa distinção, pois, a partir disso, explica-se o porquê do trabalho (OLIVEIRA et al., 2011) ser o escolhido para ser implementado, entre tantos existentes. Compara-se a implementação deste trabalho com a única encontrada na literatura.

3.1 TRABALHOS TEÓRICOS

3.1.1 Trabalho de Oliveira

Em (OLIVEIRA et al., 2011) é proposta uma técnica de tradução automática da linguagem Ladder para modelos CPN a fim de que a necessidade de tempo para a modelagem da CPN e treinamento de recursos humanos seja minimizada. Além disso, principalmente, para que consiga-se analisar o LD a partir da CPN gerada. Um dos focos desse trabalho foi criar uma CPN bem estruturada e simples de analisar. A CPN, gerada automaticamente, pode ser analisada por ferramentas existentes, dado que, Oliveira não criou nenhum conceito novo, ou seja, utilizou apenas os recursos existentes na CPN.

Aborda a conversão dos seguintes conceitos do LD: contatos, bobinas, temporizadores e operações básicas.

Devido a importância da proposta de Oliveira, para este trabalho, o método de conversão de (OLIVEIRA et al., 2011) foi previamente detalhado na Fundamentação Teórica.

3.1.2 Trabalho de Lee

(LEE; LEE, 2000) propõe um modo de converter LD em PN. Mas, apenas aborda a conversão de contatos isolados, ou seja, converte apenas um contato. O autor denomina os contatos de A e B. Onde o contato A é aberto e o B fechado. O contato A e o B são convertidos para os equivalentes em PN. Para a conversão ser possível, o autor introduz um novo conceito, arcos de preservação. Por conta disso, não é possível analisar a PN resultante com as ferramentas encontradas[1], pois, seria preciso implementar recursos adicionais nas ferramentas de análise. Além disso, compara o LD e a PN resultante, para provar que são equivalentes.

Aborda a conversão dos seguintes conceitos do LD: contatos e bobinas.

3.1.3 Trabalho de Latha

Em (LATHA; UMAMAHESWARI, 2002) mostra como converter LD em PN. Latha cria um cenário industrial fictício e, a partir dele, cria um LD para resolver o problema de automação. Logo em seguida, converte o LD em PN. A conversão é feita de um modo bastante intuitivo, mas difícil de ser automatizado. Basicamente cria a PN de maneira lógica, tomando cuidado para continuar equivalente ao LD, ou seja, analisa-se intuitivamente o LD e converte cada degrau manualmente.

[1] CPN Tools.

Aborda a conversão dos seguintes conceitos do LD: contatos, bobinas e temporizadores.

3.1.4 Trabalho de Chen

(CHEN et al., 2012) propõem um algoritmo para traduzir LD em PN. Primeiramente, cria-se um grafo representando o LD que pretende converter. Logo após isso, converte o grafo resultante em uma PN equivalente. Em trabalhos futuros, o autor pretende diminuir o tamanho da PN gerada.

Aborda a conversão dos seguintes conceitos do LD: contatos e bobinas.

3.2 TRABALHOS PRÁTICOS

3.2.1 Trabalho de Aspar

Em (ASPAR et al., 2015) é proposto e implementado um método automático de converter LD em PN. Diferente dos trabalhos apresentados na seção anterior, este foi o único trabalho encontrado que implementou o método de conversão proposto. O processo de conversão dar-se da seguinte maneira – (1) cria-se uma tabela representando o LD que pretende converter. A tabela é denominada tabela de transição, nela guarda-se todas as possibilidades de ativação das bobinas. (2) A partir da tabela de transição criada, gera-se a PN em forma de grafo (outra tabela).

Aborda a conversão dos seguintes conceitos do LD: contatos e bobinas.

3.3 COMPARAÇÃO E LIMITAÇÕES ENTRE OS TRABALHOS

É possível visualizar na Tabela 1 a comparação entre os trabalhos citados e na Tabela 2 a comparação entre a implementação do método de Oliveira – conversor LtP – e a implementação de Aspar. Nessas tabelas são apresentadas as características positivas e negativas de cada trabalho. Além disso, tem o objetivo de justificar a escolha de implementação do sistema de (OLIVEIRA et al., 2011). Afinal, como já dito, este trabalho é uma implementação do método criado por ele.

Apesar de (LEE; LEE, 2000) ser de grande influência na literatura, a contribuição dele não é suficiente para ser implementado. Pois, trabalha com a conversão de apenas um contato. Não comenta como ficaria a conversão de um LD mais complexo (mais contatos, bobinas e temporizadores). Além disso, o conceito extra que foi criado, arcos de preservação, é um obstáculo. Pois, dificulta a análise da PN criada por softwares existentes (como o CPNTools). Dado que, os softwares não possui tal conceito implementado.

Tabela 1 – Comparação entre os trabalhos teóricos

			Conceitos de LD utilizados				
Trabalho	Classe	Tamanho	Contatos	Bobinas	Temp.	Operações básicas	Sem conceito novo de PN
Oliveira	CPN	3	✓	✓	✓	✓	✓
Lee	PN	2	✓	✓	×	×	×
Latha	PN	5	✓	✓	✓	×	✓
Chen	PN	4	✓	✓	×	×	✓
Aspar	PN	1	✓	✓	×	×	✓

O método (LATHA; UMAMAHESWARI, 2002) não pode ser automatizado facilmente, dado que, é feito de forma manual e intuitiva. Além disso, a PN gerada (principalmente a com temporizadores) é maior e mais confusa que as demais propostas, ou seja, utiliza mais elementos (lugares, transições e arcos) e a disposição desses elementos não é padronizada.

(CHEN et al., 2012) possui problemas parecidos com os de (LATHA; UMAMAHESWARI, 2002), quando refere-se ao tamanho[2] e organização da PN. Isso acontece, também, devido a PN não ser compacta. Dado que, a classe PN não possui muito recursos. Diferentemente da classe CPN – utilizada por (OLIVEIRA et al., 2011) – que possui mais recursos[3] que a PN, fazendo com que a rede criada seja mais concisa.

(ASPAR et al., 2015) foi o único trabalho com implementação encontrado. Isso mostra que apesar de existir métodos consistentes de converter um LD em alguma classe PN, praticamente não existe implementação deles. O método de conversão de Aspar é bastante eficiente, tanto em execução quanto no resultado da PN. A PN resultante é bem menor e fácil de ser lida comparada as de (LATHA; UMAMAHESWARI, 2002) e (CHEN et al., 2012). O trabalho supre praticamente todas necessidades de um projeto real em LD. Contudo, diferente de (OLIVEIRA et al., 2011), o método de Aspar leva em consideração apenas os conceitos de contatos e bobinas; enquanto que, o de (OLIVEIRA et al., 2011) leva em consideração temporizadores e operações básicas, além de, claro, contatos e bobinas. Além disso, o conversor de (ASPAR et al., 2015) não foi encontrado para ser utilizado e testado.

O conversor LtP, que foi implementado nesse trabalho e será apresentado no próximo capítulo, cria uma CPN compatível para ser utilizada e analisada pelo CPNTools; diferente de (ASPAR et al., 2015) que a PN resultante é um grafo (tabela de valores) que não pode ser utilizada diretamente por nenhum programa de análise encontrado.

[2] A coluna tamanho da Tabela 1 é em relação a eles mesmos, ou seja, o número 1 representa o menor das cinco PNs e o número 5 representa a maior.

[3] O principal: cores.

Tabela 2 – Comparação entre a implementação de (ASPAR et al., 2015) e o conversor LtP.

	Conversor	LtP	Aspar
	PN resultante pode ser analisada por alguma ferramenta encontrada	✓	×
	Código-fonte disponível	✓	×
Conceitos de LD utilizados	Contatos	✓	✓
	Bobinas	✓	✓
	Temporizadores	×	×
	Operações básicas	×	×
	Biblioteca (para implementar os conceitos restantes, temporizadores e operações básicas)	✓	×
	Documentação	✓	×

4 Conversor LtP

LtP é um conversor que transforma Diagramas Ladder (LD) em Redes de Petri Coloridas (CPN). Entretanto, a versão atual do LtP, está restrito aos elementos básicos do LD, ou seja, somente aceita arquivos LD que possuam apenas combinações de contatos e bobinas. A implementação dos conceitos de temporizadores e operações básicas será realizada em trabalhos futuros. O código-fonte do conversor está disponível no repositório[1] GitHub.

Há várias ferramentas para edição de LD e CPN, dentre elas: LDmicro e CPN Tools. O conversor foi construído para essas duas ferramentas. O LtP converte o LD criado no LDmicro em uma CPN compatível com o CPN Tools 4.0.1. A Figura 10 é um exemplo de LD feito no LDmicro; esse LD foi convertido pelo LtP, o resultado é mostrado na Figura 12.

O único pré-requisito que o LtP possui para funcionar corretamente é que todos os contatos (sensores) devem começar com a letra "I" e as bobinas (atuadores) devem iniciar com a letra "O". As letras obrigatórias estão destacadas em vermelho na Figura 10.

O processo de conversão foi dividido em quatro partes (Figura 9), que são:

(I) **Extrator**: extrai e lista todos os contatos e bobinas do arquivo LD;

(II) **Gerador**: constrói a expressão booleana em CPN/ML, de cada degrau;

(III) **Configurador CPN**: cria e escreve todas as variáveis e as fórmulas necessárias para a criação da CPN, no CPN Tools;

(IV) **Desenvolvedor Gráfico**: combinada tudo feito até então e, a partir disso, gera a parte gráfica (lugares, transições, arcos, etc.).

4.1 FERRAMENTAS LDMICRO E CPN TOOLS 4.0.1

LDmicro é um editor LD, nele é possível criar um arquivo LD de forma intuitiva. Apenas adiciona-se contatos e bobinas da maneira que preferir, uma bobina por degrau. A Figura 10 é um exemplo de um diagrama gerado pelo LDmicro. Para mais detalhes consulte a documentação do LDmicro[2].

No **CPN Tools** é possível modelar uma CPN, adicionando-se lugares, transições, arcos, cores, etc. O mesmo é dividido em duas partes (Figura 8):

[1] https://github.com/CarlosMacedo/ConversorLadderPetri
[2] http://cq.cx/ladder.pl

(I) **Configuração**, local onde são criadas as fórmulas (fun), as variáveis (var) e as novas cores (colset), destacado na Figura 11 (*Standard declarations*).

(II) **Gráfica**, local no qual é possível adicionar lugares, transições, arcos, cores dos arcos, etc. Isto é, onde é criada a CPN. Mas, só é possível adicionar elementos que foram previamente configurados, ou seja, as cores disponíveis para uso são apenas as declaradas em *Standard declarations* – Figura 11.

Figura 8 – Organização do CPN Tools.

Fonte: Elaborado pelo autor.

Continuando em *Standard declarations*, apresenta exemplos práticos do que foi abordado no parágrafo anterior. O primeiro quadrado em destaque mostra como são as declarações das duas funções necessárias para construir a CPN da Figura 12. No segundo quadrado em destaque, exibe a criação de dois novos tipos de cores, INPUTS e OUTPUTS. Além disso, exibe a criação de 6 variáveis, que são elas: tupleIn, newTupleIn, out1, out2, tupleOut e newTupleOut.

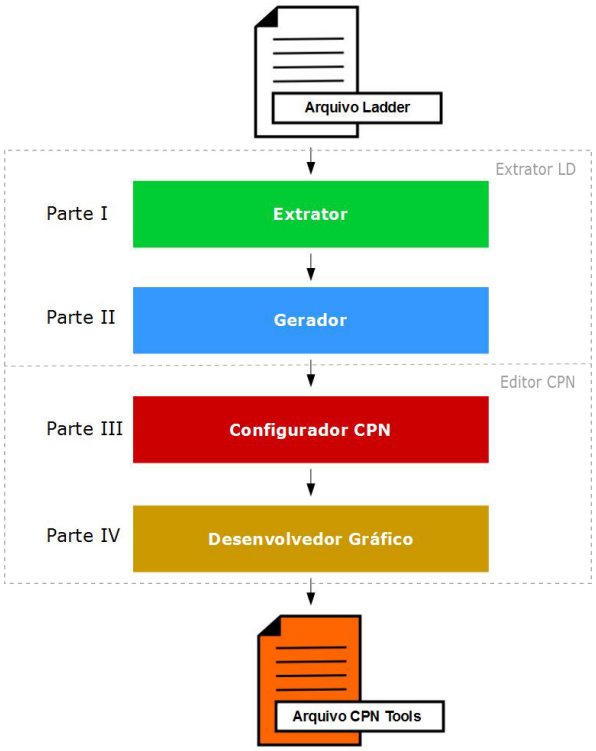
Um detalhe importante a ser mencionado sobre o CPN Tools é que se for preciso que um estado seja clonado, ou seja, esteja em dois lugares ao mesmo tempo – como é o caso do estado In da Figura 12 – é preciso marcar o estado como “Fusion”. Este detalhe é importante para a Parte IV do algoritmo de conversão.

4.2 PROCESSO DE CONVERSÃO ENTRE LD E CPN

Nesta seção explica-se o processo de conversão (Figura 9) de forma geral. Nas seções seguintes, detalha-se como cada parte foi implementada.

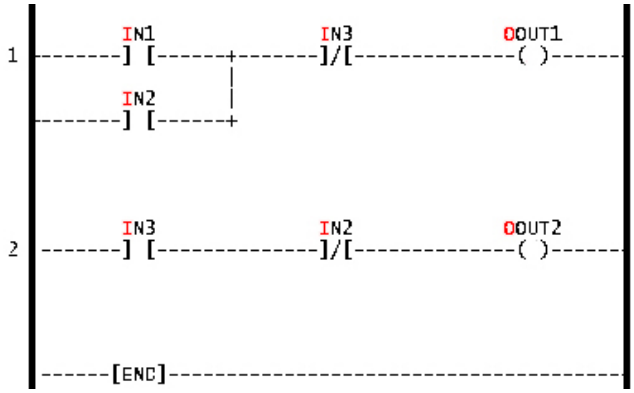
Como exposto anteriormente, o processo de conversão foi dividido em quatro partes. As Partes I e II realizam operações apenas no arquivo de LD. Enquanto que as Partes III e IV constroem a CPN.

Figura 9 – Processo de conversão.



Fonte: Elaborado pelo autor.

Figura 10 – Exemplo de diagrama Ladder, feito no LDmicro.

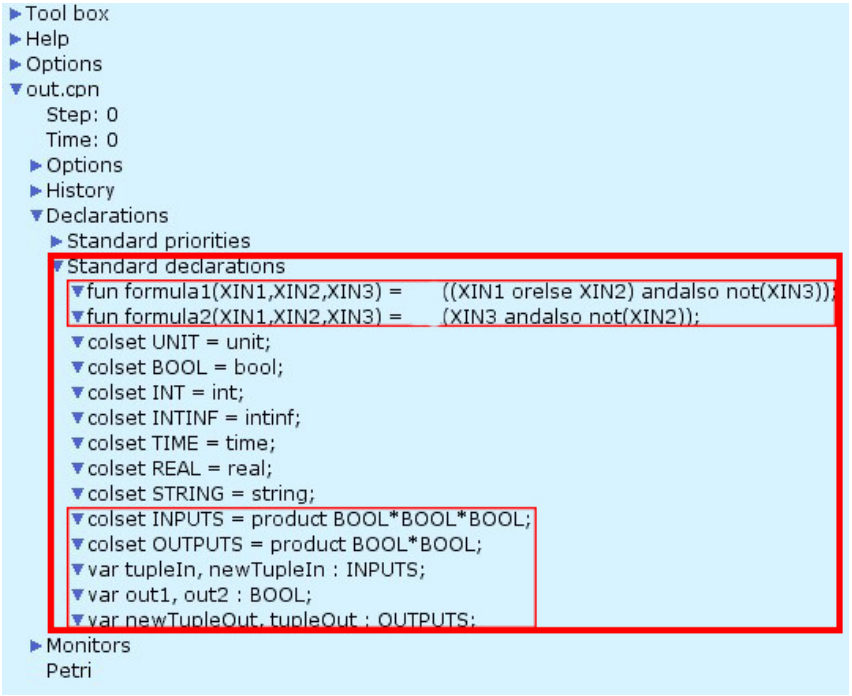


Fonte: Elaborado pelo autor.

A Parte I extrai os nomes e a quantidade de contatos e bobinas; essas informações são enviadas à Parte II. A partir dessas informações, são construídas as expressões booleanas (implementadas em CPN/ML) de cada degrau, que são obtidas de acordo com o método apresentado em (OLIVEIRA et al., 2009).

Para exemplificar as Partes I e II, é utilizado o LD da Figura 10 que possui três contatos

Figura 11 – Configurações da Rede de Petri da Figura 12.



Fonte: Elaborado pelo autor.

(N1, N2 e N3) e duas bobinas (OUT1 e OUT2). Cria-se uma fórmula para cada bobina que pode ser visualizada na Tabela 3. A bobina é ativada sempre que a expressão booleana for verdadeira.

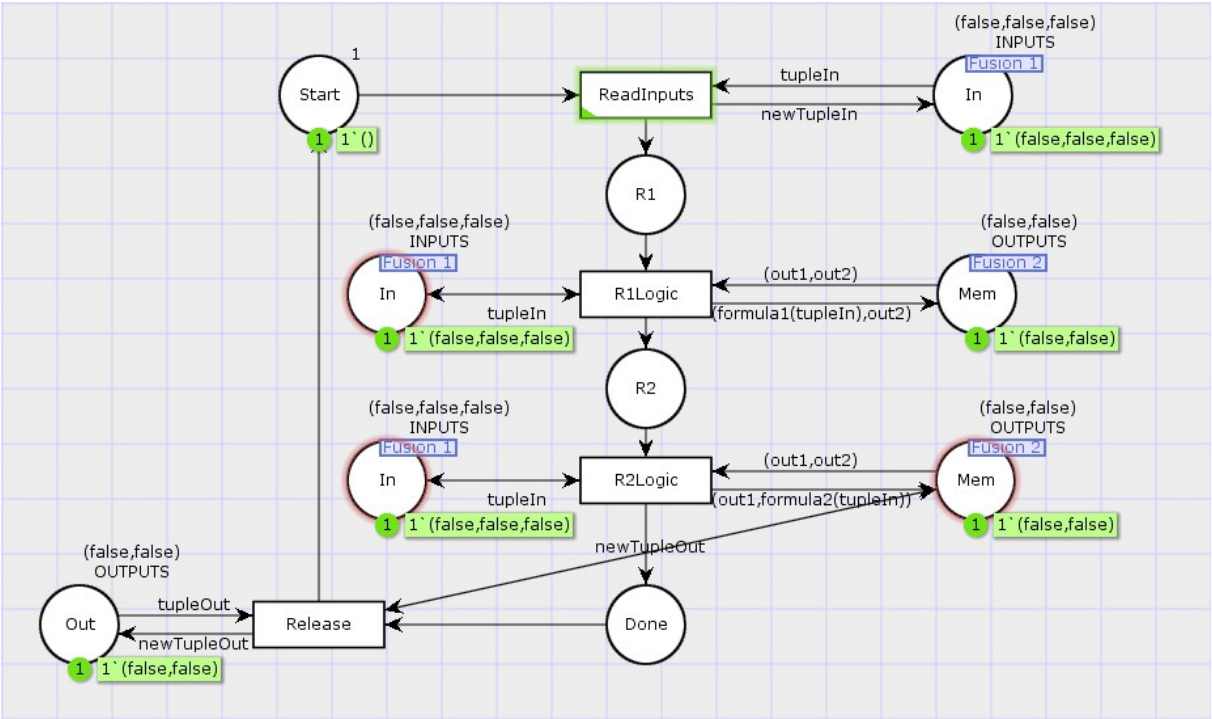
Tabela 3 – Fórmulas do LD da Figura 10, implementadas com CPN/ML.

Bobina	**Fórmula**
OUT1	(N1 orelse N2) andalso ∼N3
OUT2	N3 andalso ∼N2

A Parte III recebe as informações de I e II, que são: quantidade e nome dos contatos e das bobinas, e as expressões booleanas de cada bobina. A quantidade de contatos é utilizada para criação dos tipos (colsets) INPUTS e OUTPUTS, que pode ser visto na Figura 11. Além disso, é utilizado para as declarações das variáveis tupleIn, newTupleIn e tupleOut (ainda na Figura 11), ou seja, por conta do LD (Figura 10) possuir três contatos, tupleIn e newTupleIn possui tamanho três *(“(false,false,false)”)*. Do mesmo modo ocorre para as bobinas, que são duas. Por conta disso, tupleOut possui tamanho dois.

Logo após serem declaradas todas variáveis e funções necessárias na Parte III, na Parte IV é decidido onde, quantos e quais lugares, transições e arcos são necessários. Esses dados são escritos no arquivo CPN, ou seja, a Parte IV é a responsável por desenhar a CPN.

Figura 12 – Rede de Petri gerada para o CPN Tools da Figura 10.



Fonte: Elaborado pelo autor.

4.3 ARQUIVOS FONTES

4.3.1 LDmicro

Nesta seção detalha-se as Partes I e II do processo de conversão.

Em Algoritmo 1 mostra-se o arquivo LD gerado pelo LDmicro, correspondente a Figura 10. O arquivo está organizado em três partes: (A) configuração (linhas 1-4), (B) declaração (linhas 6-12) e (C) definição do circuito (linhas 15-27). O arquivo é lido linha por linha. A partir disso, extrair as informações sobre o LD.

(A) A parte de configuração não é importante para conversão, apenas para o software LDmicro. Por este motivo, é ignorada. (B) Em declaração, é de onde se extrai o nome e a quantidade de contatos e bobinas. Destacadas em vermelho as letras importantes para a diferenciação entre os contatos (N1, N2 e N3) e bobinas (OUT1 e OUT2) no atual algoritmo de conversão. São criadas duas listas, uma de contados e outra com bobinas; em que *XIN1*, *XIN2* e *XIN3* pertencem a lista contatos, enquanto que *OOUT1* e *OOUT2* pertencem a de bobinas.

As letras iniciais (X e Y), que aparecem no início de cada nome, não são importantes para o algoritmo de conversão. No LDmicro as letras X e Y representam se o sinal que o contato (ou bobina) recebe é interno ou externo, respectivamente.

(C) Definição do circuito é o local onde são descritos os degraus do LD. Os degraus estão delimitados por RUNG e END, como nas linhas (16-21). Como esperado, o arquivo tem referência a dois degraus. Dentro desses degraus há N contatos e uma bobina. Os contatos entre PARALLEL e END (linhas 17-18) marcam contatos em paralelo, equivalente a operação OR; ou seja, basta que pelo menos um contato seja ativado. A presença de contatos subjacentes significa que todos eles devem ser ativados para ativar a bobina. A bobina sempre está localizada na última linha do degrau, nesse caso linha 21.

Ainda sobre o degrau das linhas 16-21, ele possui dois sinais que devem ser ativados para que a bobina da linha 21 seja ativada. O primeiro sinal é gerado pelas linhas 16-19. O segundo sinal é gerado pela linha 20. Os números destacados em azul, nos degraus, representam se um contato está negado (marcado com 1) ou não (marcado com 0). Por conta disso tudo, a expressão booleana retirada desse degrau é **(N1 orelse N2) andalso N3**.

Logo após a extração dessas informações (referentes as Partes I e II), é construído o arquivo CPN. Assunto abordado na próxima seção.

É importante salientar que em uma versão futura, não será necessário marcar contatos e bobinas com os I's e O's. É possível retirar essa informação pela quantidade de degraus. Pois, cada degrau só possui uma bobina. Portanto, contar degraus é o mesmo que contar bobinas. No caso do exemplo, os degraus são os que estão entre as linhas 15-22 e 23-27.

Continuando, para se extrair os contatos e bobinas, basta contar todas informações em IO LIST (IO) (linhas 7-11), que são cinco. Logo após, contar a quantidade de degraus (D), que nesse caso são dois. Subtrai a quantidade de degraus da quantidade de itens, no IO List, tendo como resultado a quantidade de contatos (nesse caso, três). A partir dessa informação, pode se afirmar que os n primeiros (nesse caso, $n = 3$) representam os contatos e os restantes representam as bobinas, ou seja: $n = IO - D$.

Algoritmo 1 Arquivo Ladder

```
1: LDmicro0.1
2: CYCLE=10000
3: CRYSTAL=4000000
4: BAUD=2400
5:
6: IO LIST
7:     XIN1 at 0
8:     XIN2 at 0
9:     XIN3 at 0
10:    YOOUT1 at 0
11:    YOOUT2 at 0
12: END
13:
14: PROGRAM
15: RUNG
16:    PARALLEL
17:        CONTACTS XIN1 0
18:        CONTACTS XIN2 0
19:    END
20:    CONTACTS XIN3 1
21:    COIL YOOUT1 1 0 0
22: END
23: RUNG
24:    CONTACTS XIN3 0
25:    CONTACTS XIN2 1
26:    COIL YOOUT2 1 0 0
27: END
```

Fonte: Elaborado pelo autor.

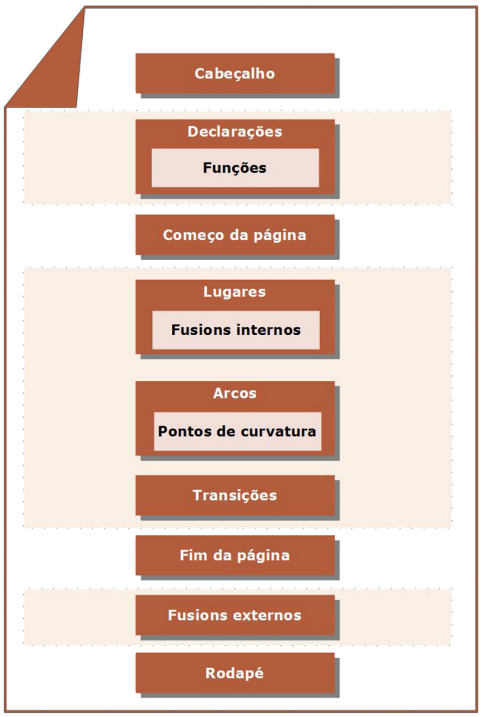
4.3.2 CPN Tools

Nessa seção é detalhado as Partes III e IV, responsável pela escrita do arquivo CPN.

Neste ponto, o processo de conversão possui todas as informações necessárias do LD. Agora o desafio é criar um arquivo CPN compatível com o CPN Tools. Para isso, é preciso entender o como um arquivo CPN é organizado, para que depois consiga-se escrever um arquivo compatível.

Entendendo o arquivo CPN — o arquivo CPN é dividido em 12 partes, do qual, algumas estão contidas em outras. A divisão é ilustrada na Figura 13. As partes que foram destacadas por um retângulo marrom claro são as que sofrem alterações pelo algoritmo; enquanto que na Tabela 4 é representado pela ausência de um asterisco. As outras partes são estáticas e servem apenas para serem escritas na íntegra, para manter o arquivo compatível com o CPN Tools.

Figura 13 – Estrutura do arquivo CPN Tools.



Fonte: Elaborado pelo autor.

É importante perceber que a Figura 13 apresenta a estrutura de um arquivo. Cada uma das 12 partes representa uma quantidade considerável de linhas. Cada bloco é explicado na Tabela 4. O algoritmo de conversão é responsável por escrever cada um destes blocos, alterando-os quando necessário.

Tabela 4 – Estrutura de um arquivo CPN.

Parte	Explicação
Cabeçalho*	Informações importantes para o arquivo funcionar corretamente, mas são constantes. Por conta disso, desprezíveis para o algoritmo de conversão. Sendo assim, são apenas copiadas.
Declarações	Onde são criadas novas funções, cores e variáveis. É o configurador CPN (Parte III).
Funções	Contida em "Declarações". Responsável por criar às funções. O motivo dela ter sido enfatizada (e as cores e variáveis não) foi por conta da maior complexidade em criá-las.
Começo de página*	Marca o início do desenvolvimento gráfico (Parte IV). É irrelevante para o algoritmo de conversão. Por conta disso, é apenas copiado.
Lugares	Onde são escritos os lugares.
Fusions internos	Contido em "Lugares". Se o lugar precisar ter a característica Fusion, é adicionada a ele. Caso contrário, esse bloco não existe.
Arcos	Cria um arco de um lugar/transição A para um transição/lugar B.
Pontos de curvatura*	Contido em "Arcos". É utilizado para arcos mais complexos. Em alguns casos (em versões posteriores) pode ser necessário que um arco realize uma curva, para que a CPN fique mais organizada; aqui é o local responsável por isso.
Transições	Cria uma transição.
Fim da página*	Marca o fim do desenvolvimento gráfico (Parte IV). É irrelevante para o algoritmo de conversão. Por conta disso, é apenas copiado.
Fusions externos	Se tiver adicionado a característica fusion ao lugar (em "Fusions internos") é necessário criar uma referência do fusion logo após "Final da página". Isto é uma restrição do CPN Tools.
Rodapé*	Muitas configurações adicionais, entre elas: tamanho da tela, tokens iniciarem minimizados (ou não), etc. É irrelevante para o algoritmo de conversão. Por conta disso, é apenas copiado.

Uma vez descrito como está organizado um arquivo CPN, será explicado, nos parágrafos seguintes, como cada bloco do arquivo é escrito.

São seguidos dois passos para construir um arquivo CPN. (1) São construídos 12 modelos em XML[3], linguagem em que os arquivos do CPN Tools são escritos. Um modelo para cada bloco possível – blocos da Figura 13. Na mesma figura, percebe-se que os blocos não estão restritos ao que aparece na tela do usuário (como arcos e transições), incluindo também: cabeçalhos, rodapés,

[3] Linguagem de marcação recomendada pela W3C.

funções e características como fusions internos e pontos de curvatura.

(2) É construída instância de um modelo (ou seja, escrita de um lugar, arco, transição, etc) em um arquivo CPN. A construção é feita através de substituições no texto do modelo. Em outras palavras, um modelo é um arquivo de texto que representa a criação de algum elemento na CPN. Em Algoritmo 2 – o equivalente ao 4º bloco (Lugares), da Figura 13 – mostra-se como é um modelo de um lugar. Estão destacados em azul as variáveis do modelo. Essas variáveis são substituídas pelo algoritmo para poder criar um lugar, ou seja, o modelo é lido linha por linha, quando encontra uma variável, substitui pelo valor necessário (uma nova id, posição, etc.). Para cada variável cria-se um identificador. Através do identificador o algoritmo sabe qual é a variável e insere o valor adequado.

Explicando detalhadamente o modelo de um lugar (Algoritmo 2) — todo elemento no CPN Tools precisa ter um identificador (ID), seja lugar, transição, arco, tipo do lugar, etc. O ID do lugar pode ser visto na linha 1. Na linha 2 é apresentada a posição do elemento na tela. Na linha 6 é definido o nome do lugar. Enquanto que na linha 14 é apresentado o ID do tipo do lugar, cujo o nome é INPUTS, linha 19. Na linha 15 é definida a posição do nome. Linha 22 é apresentado o ID de um token que está no lugar. Nesse caso a cor do token é *(false, false, false)* (linha 27). A posição (linha 23) é utilizada para colocar a ficha sobre o lugar. As linhas 30-35 merecem o destaque maior, devido às restrições do CPN Tools. Em Redes de Petri Coloridas é comum utilizar o mesmo lugar em locais diferentes. Para o CPN Tools conseguir simular um lugar com essa característica é preciso marcá-lo como **Fusion**, ou seja, caso esse lugar não fosse Fusion não existiriam as linhas 30-35. Os atributos destacados em azul (linhas 31-32) seguem a mesma lógica dos anteriores, um nome e uma ID única, e uma posição (x,y).

Em suma, as Partes III e IV funcionam dessa forma, substituindo variáveis nos modelos e escrevendo-os em um arquivo principal. A diferença é que a Parte III trata dos blocos Declarações e Funções (Figura 13), enquanto que a Parte IV é responsável pelos blocos restantes. O motivo da divisão ter sido feita dessa forma, foi por conta da Parte de declaração ser mais robusta que a Parte IV.

Realiza-se esse procedimento (instanciação de modelos) para cada um dos blocos da Figura 13, na ordem que é mostrada nesta figura. Depois disso tudo, a CPN está pronta.

Algoritmo 2 Arquivo CPN : Lugar

```
<place id="ID1412347707">
    <posattr x="252.000000" y="137.000000"/>
    <fillattr colour="White" pattern="" filled="false"/>
    <lineattr colour="Black" thick="1" type="Solid"/>
    <textattr colour="Black" bold="false"/>
    <text>In </text>

    <ellipse w="40.000000" h="40.000000"/>
    <token x="-10.000000" y="0.000000"/>
    <marking x="0.000000" y="0.000000" hidden="false">
        <snap snap_id="0" anchor.horizontal="0" anchor.vertical="0"/>
    </marking>

    <type id="ID1412355541">
        <posattr x="291.000000" y="113.000000"/>
        <fillattr colour="White" pattern="Solid" filled="false"/>
        <lineattr colour="Black" thick="0" type="Solid"/>
        <textattr colour="Black" bold="false"/>
        <text tool="CPN Tools" version="4.0.1">INPUTS </text>
    </type>

    <initmark id="ID1412401042">
        <posattr x="326.000000" y="160.000000"/>
        <fillattr colour="White" pattern="Solid" filled="false"/>
        <lineattr colour="Black" thick="0" type="Solid"/>
        <textattr colour="Black" bold="false"/>
        <text tool="CPN Tools" version="4.0.1">(false, false, false)</text>
    </initmark>

    <fusioninfo id="ID1412398726" name="Fusion 1">
        <posattr x="299.000000" y="146.000000"/>
        <fillattr colour="White" pattern="Solid" filled="false"/>
        <lineattr colour="Black" thick="0" type="Solid"/>
        <textattr colour="Black" bold="false"/>
    </fusioninfo>
</place>
```

Fonte: Elaborado pelo autor.

4.4 DESENVOLVIMENTO

Esta seção trata de algumas questões de desenvolvimento, que são elas: diagramas UML, tecnologias e padrões de projetos que foram utilizados.

4.4.1 Tecnologia e padrão de projeto

A aplicação foi desenvolvida utilizando a linguagem de programação Java[4]. A IDE[5] utilizada foi Eclipse Mars.2 (4.5.2). Foi utilizado o padrão de projeto façade[6].

4.4.2 Modelagem UML

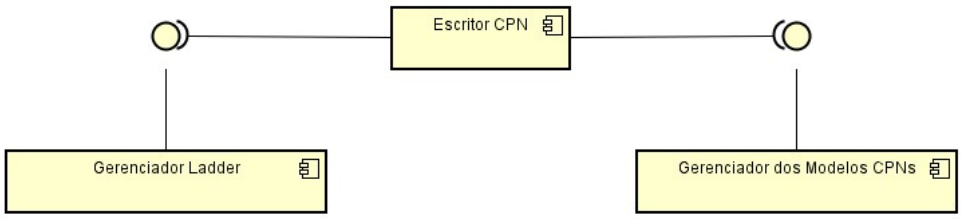
A modelagem de sistema é o processo de desenvolvimento de modelos abstratos do sistema, cada modelo representando uma parte (ou perspectiva) diferente do sistema. Os modelos são utilizados durante o processo de engenharia de requisitos para ajudar na extração dos requisitos e no desenvolvimento do sistema (SOMMERVILLE, 2011). Nesta seção são apresentados alguns modelos que demostram o funcionamento do conversor LtP.

4.4.3 Diagrama de componentes

O diagrama de componentes identifica os componentes que fazem parte de um sistema. Um componente representa tanto um componente de negócio ou processo, arquivos contendo código-fonte, arquivos de ajuda, bibliotecas, etc. (GUEDES, 2009).

Na Figura 14 é ilustrado o diagrama de componentes do conversor LtP. O componente **Gerenciador Ladder** é o responsável por extrair as informações do arquivo LD. Enquanto que o componente **Gerenciador dos Modelos CPNs** é onde encontra-se e organiza-se os modelos CPNs. O **Escritor CPN** pega os dados e os modelos do Gerenciador Ladder e Gerenciador dos Modelos CPNs, respectivamente. A partir disso, o Escritor CPN fica responsável por construir a CPN.

Figura 14 – Diagrama de componentes.



Fonte: Elaborado pelo autor.

[4] É uma linguagem de alto nível orientada a objetos.
[5] Ambiente de desenvolvimento integrado que tem o objetivo de aumentar a produtividade do programador.
[6] É um padrão de projeto que fornece uma interface simplificada, ou seja, define uma interface mais abstrata para tornar o subsistema mais intuitivo de ser utilizado.

4.4.4 Diagrama de atividades

O diagrama de atividade enfatiza a sequência e condições para coordenar comportamentos de baixo nível. É o diagrama com ênfase no algoritmo do problema e apresenta muitas semelhanças com os antigos fluxogramas utilizados para desenvolver a lógica de programação e determinar o fluxo de controle de um algoritmo (GUEDES, 2009).

A Figura 15 mostra o fluxo de controle do conversor LtP. Primeiramente é lido o arquivo LD, onde o usuário seleciona o arquivo através de uma interface de seleção de diretório (igual a abrir um arquivo qualquer em um programa). Logo em seguida, são extraídas todas informações do arquivo LD, que nesse caso são os contatos e bobinas. Verifica-se o arquivo é relevante para ser convertido, ou seja, se possui uma quantidade maior que zero de contatos e bobinas. Caso não tenha, é emitido uma mensagem de erro e é solicitada a entrada de um arquivo válido. Caso contrário, é iniciada a conversão. A conversão nada mais é do que a escrita do arquivo CPN; que é escrito em ordem. Assim, são escritos as declarações, o começo de página e assim sucessivamente como mostra a figura.

4.4.5 Diagrama de classes

Diagrama de classes permite a visualização das classes que comporão o sistema com seus atributos e métodos, bem como em demonstrar como as classes se relacionam (GUEDES, 2009).

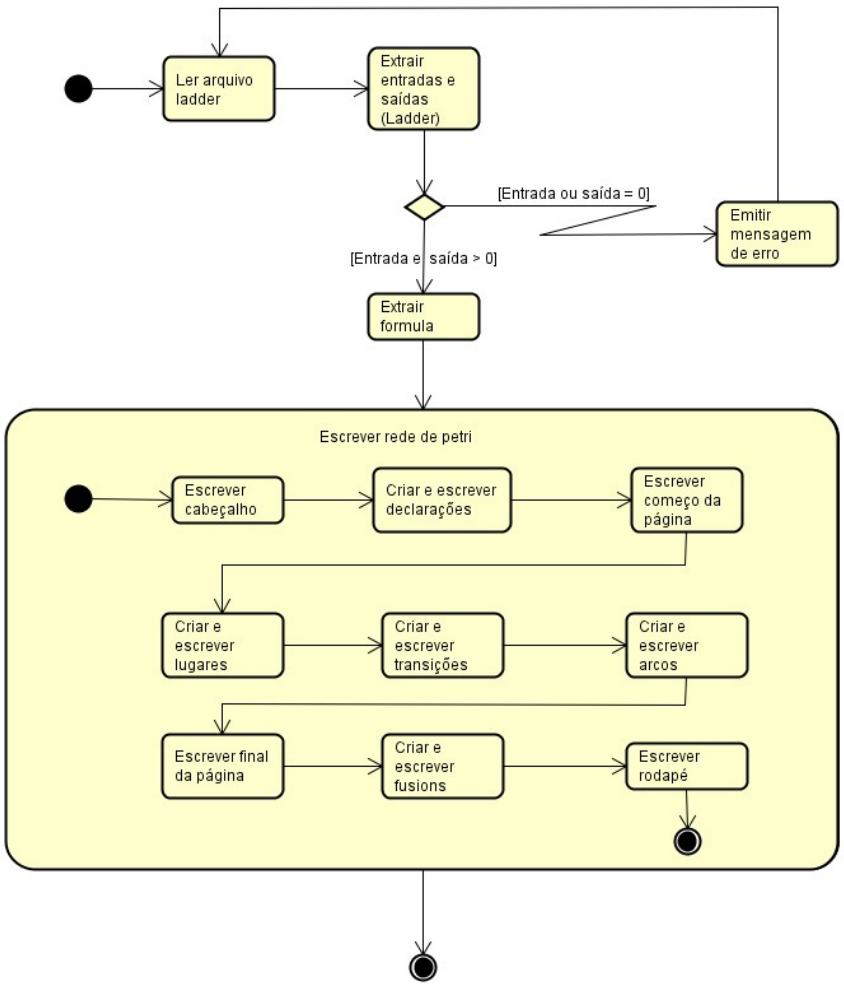
Na Figura 16 mostra-se o diagrama de classe do LtP. Nele é possível ver como as classes estão organizadas. A classe principal é a WriterPetri, é responsável por construir a CPN; é utilizado o padrão de projeto façade, para tornar a escrita da CPN mais intuitiva. Nesse caso, a instancia da classe WritePetri funciona como uma interface mais simples, pois, para escrever uma CPN é preciso chamar funções de criação de lugares, arcos e transições (entre outras) diversas vezes, enquanto que com a instancia de WriterPetri isso tudo é resumido para uma única chamada de função (ladderToPetri). O restante das classes são auto-explicadas pelos seus nomes, como pode ser visto na Tabela 5.

4.4.6 Diagrama de sequência

O diagrama de sequência é um diagrama comportamental que procura determinar a sequência de eventos que ocorrem em um determinado processo (GUEDES, 2009).

É exibido na Figura 17 um exemplo de chamada de função, para a criação de um lugar. As outras funções do sistema tem um comportamento parecido, com exceção dos métodos das classes ManipulatorLadder e WriterPetri. O método addPlace é invocado por uma instância

Figura 15 – Diagrama de atividade.



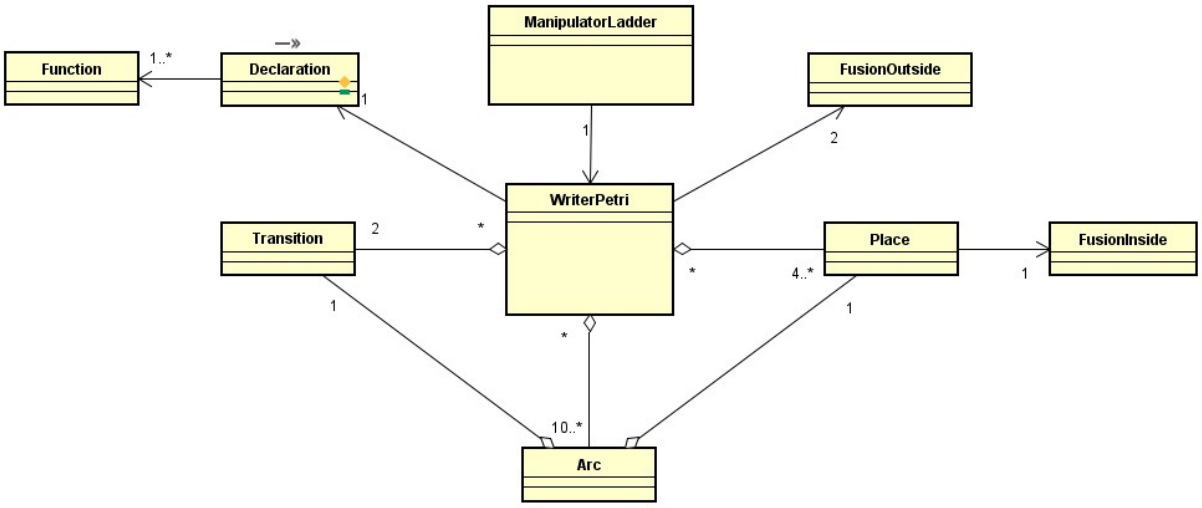
Fonte: Elaborado pelo autor.

Tabela 5 – Responsabilidade de cada classe do sistema.

Classe	**O que faz**
ManipulatorLadder	Extrai as informações do arquivo LD.
Place	Cria os lugares.
FusionInside	Cria os fusions dentro dos lugares.
FusionOutside	Cria as referencias dos fusions, depois do fim de página.
Arc	Cria os arcos.
Transition	Cria as transições.
Function	Cria as funções.
Declaration	Define e cria as declarações.

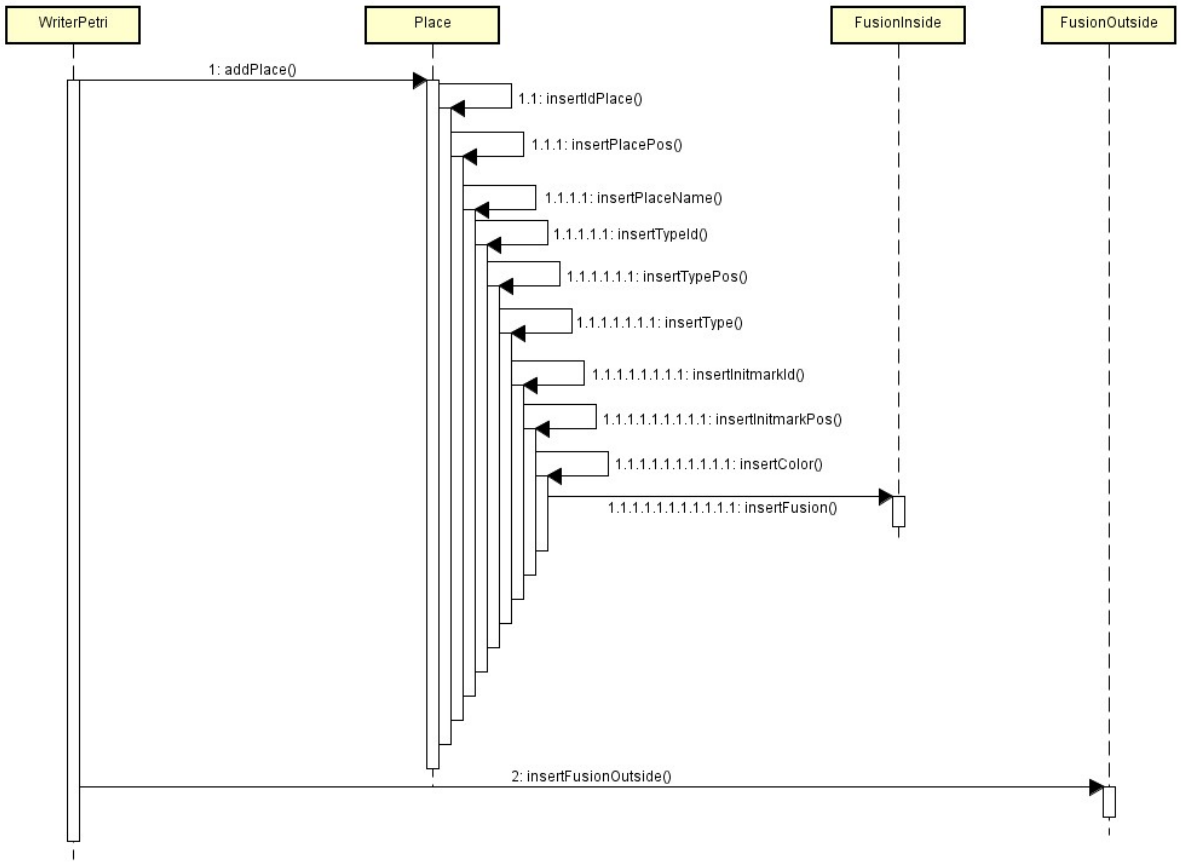
da classe WriterPetri. A partir disso, uma instância da classe Place chama diversos métodos necessários para a criação do lugar; cada uma dessas funções é responsável por substituir uma variável no modelo CPN do lugar. As funções insertFusion e insertFusionOutside, como o nome indica, são para inserir a característica Fusion nos lugares.

Figura 16 – Diagrama de classe.



Fonte: Elaborado pelo autor.

Figura 17 – Diagrama de sequência.



Fonte: Elaborado pelo autor.

5 CONCLUSÕES E TRABALHOS FUTUROS

Nesse trabalho foi apresentado o conversor LtP. A implementação de um conversor LD para CPN é considerada uma boa contribuição. Dado que existem muitas propostas de métodos de conversão, mas nenhuma implementação boa o suficiente[1] para ser utilizada. Com isso, espera-se que a necessidade de tempo extra para modelagem e o custo de treinamento de recursos humanos sejam minimizados.

Como contribuição adicional, este trabalho possibilita a criação de modelos em Rede de Petri Colorida (CPN) em tempo de execução. Afinal, o módulo Gerenciador dos Modelos CPNs (Figura 14) pode ser utilizado como uma biblioteca para escrita de CPN em Java. A documentação do conversor LtP pode ser encontrada no GitHub[2]; nela encontra-se a documentação de todos os módulos da Figura 14.

O conversor LtP ficou restrito a manipulação de contatos e bobinas. Apesar do método de conversão proposto por (OLIVEIRA et al., 2011) também abordar operadores e temporizadores. Em versões futuras, pretende-se implementar os operadores e temporizadores, no LtP. Essa implementação é consideravelmente mais simples se comparado ao desenvolvimento inicial do LtP. Dado que, basta utilizar as funções do módulo Gerenciador dos Modelos CPNs, para desenhar o modelo CPN como proposto em (OLIVEIRA et al., 2011).

[1] Entenda o que foi classificado como “boa o suficiente” no Capítulo 3.

[2] https://github.com/CarlosMacedo/ConversorLadderPetri/tree/master/ConversorLadderPetri/doc

REFERÊNCIAS

ASPAR, Z.; SHAIKH-HUSIN, N.; KHALIL-HANI, M. Algorithm to convert programmable logic controller ladder logic diagram models to petri net models. In: IEEE. **Research and Development (SCOReD), 2015 IEEE Student Conference on**. [S.l.], 2015. p. 156–161.

CHEN, X.; LUO, J.; QI, P. Method for translating ladder diagrams to ordinary petri nets. In: IEEE. **Decision and Control (CDC), 2012 IEEE 51st Annual Conference on**. [S.l.], 2012. p. 6716–6721.

GUEDES, G. T. Uml 2. **Uma Abordagem Prática", São Paulo, Novatec**, 2009.

JENSEN, K. **Coloured Petri nets: basic concepts, analysis methods and practical use**. [S.l.]: Springer Science & Business Media, 2013. v. 1.

JOHN, K.-H.; TIEGELKAMP, M. **IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids**. [S.l.]: Springer Science & Business Media, 2010.

LATHA, K.; UMAMAHESWARI, B. Supervisory control of an automated system with ladder logic programming and analysis using petri nets. In: IEEE. **Systems, Man and Cybernetics, 2002 IEEE International Conference on**. [S.l.], 2002. v. 3, p. 5–pp.

LEE, G. B.; LEE, J. S. The state equation of petri net for the ld program. In: IEEE. **Systems, Man, and Cybernetics, 2000 IEEE International Conference on**. [S.l.], 2000. v. 4, p. 3051–3056.

MACIEL, P. R.; LINS, R. D.; CUNHA, P. R. **Introdução às redes de Petri e aplicações**. [S.l.]: UNICAMP-Instituto de Computacao, 1996.

MADER, A.; WUPPER, H. Timed automaton models for simple programmable logic controllers. In: IEEE. **Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on**. [S.l.], 1999. p. 106–113.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989.

OLIVEIRA, E. A. da S.; SILVA, L. D. da; GORGÔNIO, K.; PERKUSICH, A.; MARTINS, A. F. Obtaining formal models from ladder diagrams. In: IEEE. **Industrial Informatics (INDIN), 2011 9th IEEE International Conference on**. [S.l.], 2011. p. 796–801.

OLIVEIRA, K. V.; GORGôNIO, K.; PERKUSICH, A.; LIMA, A. M. N.; SILVA, L. D. da. Extração Automática de Autômatos Temporizados a Partir de Diagramas Ladder. In: **Anais do IX Simpósio Brasileiro de Automação Inteligente**. Brasília, Brasil: Sociedade Brasileira de Automática, 2009.

PENG, S. S.; ZHOU, M. C. Ladder diagram and petri-net-based discrete-event control design methods. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 34, n. 4, p. 523–531, 2004.

RIDLEY, J. **Mitsubishi FX programmable logic controllers: applications and programming**. [S.l.]: Elsevier, 2004.

SIEMENS, A. Ladder logic (lad) for s7-300 and s7-400 programming reference manual. **SIEMENS AG**, 2003.

SOMMERVILLE, I. **Software Engineering**. Pearson, 2011. (International Computer Science Series). ISBN 9780137053469. Disponível em: <https://books.google.com.br/books?id=l0egcQAACAAJ>.

ZHOU, M.; TWISS, E. A comparison of relay ladder logic programming and petri net approach for sequential industrial control systems. In: IEEE. **Control Applications, 1995., Proceedings of the 4th IEEE Conference on**. [S.l.], 1995. p. 748–753.