Uma API para Conversão de Programas em Ladder em Modelos de Redes de Petri Coloridas

An API for Ladder Program Conversion in Colored Petri Nets Models

Carlos Macêdo, Elthon Oliveira

Laboratório Multidisciplinar de Computação
Núcleo de Ciências Exatas – NCEx
Universidade Federal de Alagoas – Campus Arapiraca

carlosmacedo025@gmail.com, elthon@arapiraca.ufal.br

Abstract. *There are several critical processes in the industry that are automated with the goal of optimizing available resources and increasing production of products such as gas and oil. Ladder diagrams is a language used in systems developed to manufacturing industry that uses Programmable Logic Controllers. Such systems have reliability and security issues. To simulate and verify the properties of systems, mathematical tools like Petri Nets (PN) are used. Some researchers propose methods to convert Ladder in some PN class. This work presents some conversion methods, compares them briefly, and presents an implementation of one of them. Such an implementation converts Ladder code to Colored Petri Nets model.*

Resumo. *Há vários processos críticos na indústria que são automatizados com o objetivo de otimizar os recursos disponíveis e aumentar a produção de produtos, tais como gás e petróleo. Diagramas Ladder é uma linguagem utilizada em sistemas desenvolvidos para indústria de manufatura que utiliza Controladores Lógico Programáveis. Tais 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 de converter Ladder em alguma classe de PN. Este trabalho apresenta alguns métodos de conversão existentes, compara-os brevemente e apresenta uma implementação de um deles. Tal implementação converte código Ladder em modelos de Rede de Petri Colorida.*

1. Introdução

Há vários processos na indústria que são automatizados com o objetivo de otimizar os recursos disponíveis e de aumentar a produção [da Silva 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 deste tipo são o processamento de gás/petróleo, o desenvolvimento automobilístico, entre outros.

Para se ter uma ideia sobre a criticidade de sistemas deste tipo, caso algum processo industrial fique instável ou impreciso, este pode se tornar uma ameaça à segurança de seres humanos e do meio ambiente. Sendo assim, antes de por algum sistema de manufatura desta natureza em funcionamento, é importante assegurar que o mesmo não apresente falhas e seja executado corretamente [da Silva Oliveira et al. 2011].

Para controlar sistemas de manufatura, bem como plantas de processamento de gás/petróleo, são utilizados dispositivos chamados de Controladores Lógicos Programáveis (CLPs). CLPs são utilizados para controlar máquinas industriais e podem ser programados. Diagrama Ladder, Ladder (ou simplesmente LD), é a linguagem visual mais utilizada na programação de CLPs [Chen et al. 2012]. Um dos principais problemas de LD é que a linguagem não oferece suporte a técnica de verificação alguma. Ou seja, não há como garantir que o sistema desenvolvido vá se comportar perfeitamente conforme planejado. Contudo, existem algumas técnicas de verificação que utilizam linguagens matemáticas na descrição do comportamento do sistema a ser verificado.

Redes de Petri (em inglês *Petri Nets*, ou apenas PN) são uma representação matemática que servem para a modelagem gráfica de sistemas. Por meio 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 LD possui, [da Silva Oliveira et al. 2011], [Lee and Lee 2002], [Latha and Umamaheswari 2002] e [Chen et al. 2012] propõem o seguinte processo: (i) desenvolvimento do sistema em LD; (ii) conversão do programa LD em uma PN equivalente; e (iii) análise da PN. Se o processo de análise da PN identificar erros do sistema, estes são corrigidos no programa LD e são realizados os passos (ii) e (iii) novamente. O processo se repete até que não sejam identificados erros (comportamentos não desejados). Após este processo, o sistema desenvolvido em LD pode ser utilizado. Este processo é ilustrado na Figura 1.

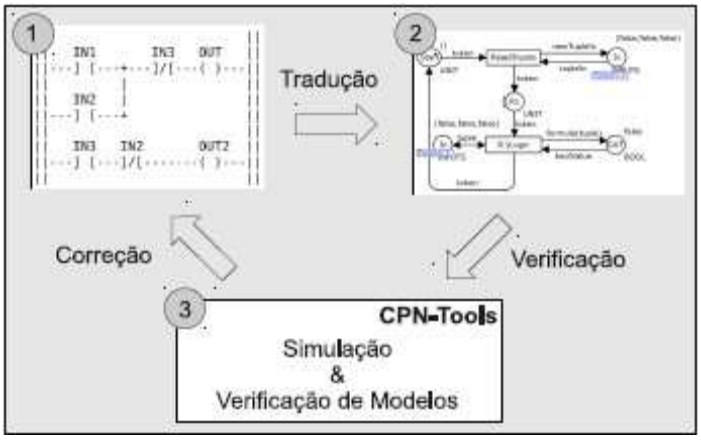


Figura 1. Visão geral do processo.
Fonte: [da Silva Oliveira et al. 2011]

Entretanto, há dois problemas inerentes ao processo descrito: a necessidade de tempo extra para a modelagem do sistema e o treinamento de recursos humanos na linguagem formal a ser usada, neste caso PN [da Silva Oliveira et al. 2011]. Uma forma de eliminar, ou ao menos diminuir, os custos associados aos dois problemas é utilizar um conversor automático para converter programas LD em modelos PN.

Na literatura, existem várias formas de converter LD em PN. Este trabalho faz uma breve comparação entre os métodos existentes: [da Silva Oliveira et al. 2011], [Lee and Lee 2002], [Latha and Umamaheswari 2002] e [Chen et al. 2012]; explicando, brevemente, os algoritmos de conversão que foram propostos nesses trabalhos. Além disso, implementa um deles. O trabalho considerado mais adequado foi o de [da Silva Oliveira et al. 2011], cuja justificativa é apresentada na Seção 3.1.

O restante deste artigo está organizado da seguinte maneira: na Seção 2 é apresentada a fundamentação teórica necessária ao entendimento deste trabalho; na Seção 3 são descritos e comparados os trabalhos correlatos; o conversor de programas Ladder para modelos em Redes de Petri Coloridas é detalhado na Seção 4; na Seção 5 é apresentada a API desenvolvida com auxílio de um pequeno exemplo; e, por fim, são apresentadas as conclusões na Seção 6.

2. Fundamentação Teórica

Nesta seção é apresentado uma breve explicação sobre Ladder e Redes de Petri Coloridas.

2.1 Ladder

Ladder (LD) é uma linguagem visual, baseada nos circuitos de relé, usada na programação de Controladores Lógico Programáveis (CLPs) [Zhou and 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.

Existem diferentes dialetos de LD na indústria [Siemens 2003, Ridley 2004]. Os diferentes dialetos servem para CLPs diferentes. Tais dialetos possuem suas próprias funções e algumas diferenças visuais. Porém, possuem uma característica em comum, todos seguem o padrão IEC 61131-3 – para mais informações sobre IEC 61131-3, consultar [John and Tiegelkamp 2010].

A cada lógica de controle presente num programa em LD, dá-se o nome de degrau. LD é dividido em degraus e seu funcionamento ocorre de acordo com uma passagem de “corrente elétrica”. A “corrente elétrica” percorre da esquerda para direita e de cima para baixo, degrau por degrau (Figura 2), e indica a sequência de leitura e execução dos comandos. O funcionamento dos CLPs é dividido em ciclos. Por se tratar de uma abstração dos circuitos de relé, a execução de LD segue o mesmo funcionamento. Chama-se de ciclo quando a “corrente elétrica” percorre todos os degraus. Quando um ciclo finaliza, outro inicia imediatamente.

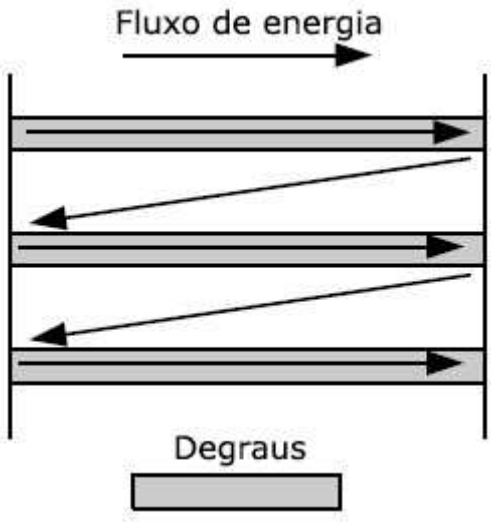


Figura 2. Fluxo da corrente elétrica.
Fonte: [da Silva Oliveira et al. 2011]

Os elementos básicos do LD representam contatos (sensores) e bobinas (atuadores). Eles são representados, respectivamente, por **] [** e **( )**. Tais elementos podem ser dispostos em série, paralelo ou uma combinação de ambos. Cada degrau contém apenas uma bobina, mas pode ter zero ou mais contatos. Os contatos podem ter dois estados, abertos ou fechados. Os contatos fechados são representados por **]/[**. O contato fechado é negação do mesmo contato aberto.

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

Existem outros elementos (e.g., guardas e temporizadores) e operações (e.g., atribuição, adição, subtração, divisão e multiplicação) que podem ser utilizados em LD. Contudo, o escopo atual desse trabalho não contempla estes conceitos. Assim, é pretendido dar suporte a tais elementos futuramente. Mais detalhes podem ser obtidos nos trabalhos de [da Silva Oliveira et al. 2011] e [Mader and Wupper 1999].

2.2. Redes de Petri Coloridas

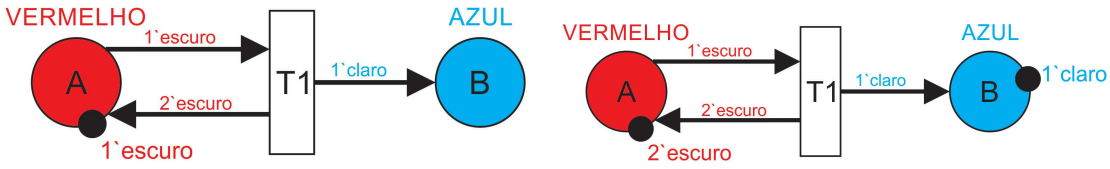
Redes de Petri (PN) é uma ferramenta matemática bem conhecida no contexto de verificação de sistemas. É utilizada para modelagem e análise de sistemas baseados em eventos [Latha and Umamaheswari 2002]. PN é adequada também para descrever e estudar sistemas concorrentes, assíncronos e distribuídos [Murata 1989]. Redes de Petri Coloridas (CPN) é uma classe de PN na qual fichas representam tipos de dados complexos, ou seja, conseguem representar vários tipos de dados[1] [Jensen 2013]. Por isso, modelos CPN conseguem ser mais compactos e legíveis quando comparados aos

[1] Diferentemente de PN ordinária, onde as fichas representam um único tipo de dado.

modelos PN. Há ferramentas (e.g., CPN Tools[2]) que suportam edição, simulação e análise de modelos CPN.

CPN possui os seguintes elementos básicos:

- **lugar** (círculos ou elipses): representa a disponibilidade ou não de recursos;
- **transição** (retângulos): representa um evento ou ação do sistema;
- **arcos** (setas): representam produção[3] e consumo[4] de recursos; e
- **fichas coloridas** (círculos minúsculos): representam os recursos em si.



(a) Estado Inicial. (b) Depois de um ciclo.

Figura 3. Exemplo de Rede de Petri Colorida.

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 3 são ilustrados dois estados distintos de um mesmo modelo CPN. Ou seja, na Figura 3 (a) é ilustrado o modelo antes da execução da transição **T1** e na Figura 3 (b) é ilustrado o modelo no instante seguinte à execução da transição **T1**.

Para o entendimento sobre o funcionamento da CPN, deve-se ter em mente que:

- 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;
- ao ser disparada, a transição remove de seus lugares de entrada as quantidades de fichas indicadas em seus arcos de entrada e adiciona fichas nos lugares de saída de acordo com as quantidades indicadas em seus arcos de saída;
- um arco sempre transfere uma certa quantidade (indicada na inscrição do arco) de fichas para onde estiver apontando;
- caso não haja fichas suficiente em ao menos um lugar de entrada, a transição não poderá ser disparada; e
- a inscrição do arco possui a forma <x>'<nome>, onde <x> é a quantidade de fichas e <nome> é o tipo.

---

[2] http://cpntools.org/

[3] Função de saída: se o arco é do tipo transição-lugar.

[4] Função de entrada: se o arco é do tipo lugar-transição.

É importante ressaltar que um lugar pode ter qualquer quantidade de fichas. Porém, todas as fichas devem ser do tipo daquele lugar. No exemplo da Figura 3, o lugar é do tipo VERMELHO, então o lugar pode conter fichas do tipo: vermelho escuro, vermelho claro, etc.

A CPN da Figura 3 possui dois lugares (**A** e **B**) e uma transição (**T1**). Os lugares são do tipo VERMELHO e AZUL, respectivamente. No primeiro instante (Figura 3 (a)), o lugar **A** possui uma ficha escuro e o **B**, nenhuma ficha. Há três arcos neste modelo: (**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 haja ao menos uma ficha escuro, que é o caso[5]. Assim, **T1** é ativada. O arco (**A**,**T1**) transfere uma ficha de **A** para **T1**, que por sua vez, ativa todos os seus arcos do tipo função de saída: (**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, instante seguinte ao disparo de **T1**, é ilustrado na Figura 3 (b). A execução dessa CPN nunca irá parar, pois, o lugar **A** sempre terá ao menos uma ficha e **T1** sempre criará fichas.

Para mais detalhes sobre PN/CPN, consultar [Maciel et al. 1996] e [Jensen 2013].

3. Trabalhos Relacionados

Existem diversos trabalhos sobre conversão entre LD e PN. Alguns abordam a conversão de LD para PN e outros, o contrário. Neste artigo estão relacionados os trabalhos voltados à conversão de LD para PN. Dentre tais trabalhos, muitos usam as chamadas Redes de Petri Ordinárias (ou lugar/transição). Por isso, as contribuições feitas pelos trabalhos selecionados, e aqui relacionados, sobrepõem as contribuições feitas pelos demais trabalhos existentes na literatura.

Em [Aspar et al. 2015] é proposto e implementado um método automático de converter LD em PN. Diferente dos demais trabalhos apresentados neste artigo, esse foi o único trabalho encontrado que implementou um método de conversão, como proposto neste artigo. O processo de conversão se dá da seguinte maneira. Primeiro é criada uma tabela representando o LD que pretende converter. Esta tabela é denominada tabela de transição e nela são armazenadas todas as possibilidades de ativação das bobinas. Em seguida, a partir da tabela de transição, é gerado o modelo PN. Esse trabalho suporta apenas os conceitos de contatos e bobinas de LD.

No trabalho de [Chen et al. 2012] é proposto um algoritmo para traduzir LD em PN. Não há implementação. Primeiramente, é criado um grafo representando o LD que pretende converter. Logo após a construção do grafo, ele é convertido em um modelo de PN equivalente. A PN resultante é bastante grande. Os próprios autores sugerem diminuir o tamanho da PN gerada como trabalhos futuros. Esse trabalho também suporta apenas os conceitos de contatos e bobinas de LD.

[5] Caso a ficha fosse outra, ou não houvesse fichas, T1 nunca seria ativada. Assim, a CPN nunca sofreria alterações.

Em [Latha and Umamaheswari 2002] é proposto outro algoritmo de conversão de LD para 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 é criada uma PN de maneira lógica, tomando-se cuidado para manter a equivalência ao LD. Ou seja, analisa-se intuitivamente o LD e cada degrau é convertido manualmente, o que é oneroso. Neste trabalho são abordados os conceitos de contatos, bobinas e temporizadores de LD.

No trabalho de [Lee and Lee 2002] é proposta uma conversão de LD para PN. A conversão suporta apenas contatos isolados, isto é, converte apenas um contato. O autor denomina os contatos de A e B, onde o contato A é aberto e o B fechado. Contatos A e 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 disponíveis atualmente (e.g., CPN Tools). Pois, seria preciso implementar recursos adicionais nas ferramentas já existentes/consolidadas. Esse trabalho também suporta apenas os conceitos de contatos e bobinas de LD.

No trabalho de [da Silva Oliveira et al. 2011] é proposta uma técnica de tradução automática da linguagem Ladder para modelos PN coloridas (CPN). 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 os autores não adicionaram novos conceitos na rede. Ou seja, os autores utilizaram apenas os recursos já existentes na formalização de CPN, suportada pela ferramenta CPN Tools. Este trabalho suporta diversos conceitos de LD, tais como: contatos, bobinas, temporizadores e operações básicas.

3.1. Comparação e Limitações entre os Trabalhos

Na Tabela 1 é sumarizada a comparação entre os trabalhos apresentados anteriormente. Nessa tabela são apresentadas as características consideradas positivas e negativas de cada trabalho. Além disso, espera-se que tal sumarização justifique o porquê da escolha em implementar, no trabalho aqui apresentado, o algoritmo proposto no trabalho de [da Silva Oliveira et al. 2011] – o algoritmo é utilizado no processo da Figura 1.

A coluna *Tamanho* da Tabela 1 diz respeito à comparação entre os tamanhos dos modelos gerados pelos trabalhos apresentados. Ou seja, o número 1 indica o menor modelo dentre os trabalhos presentes, enquanto o número 5 indica o maior modelo.

Apesar de [Lee and Lee 2002] ser de grande influência na literatura, o trabalho suporta a conversão de apenas um contato. Além disso, não é comentado no artigo como a conversão seria feita com um LD mais complexo (mais contatos, bobinas e temporizadores). Outro problema é a inclusão do conceito de arcos de preservação, que se torna um obstáculo, pois inviabiliza a análise da PN criada por ferramentas existentes, que não possuem suporte a tal conceito. A inclusão de um novo conceito é considerado uma barreira porque os outros algoritmos propostos resolvem o mesmo problema (e com os mesmos efeitos) sem adicionar novos conceitos. Por conta disso, para esses, não é necessário implementar uma nova ferramenta.

O método de [Latha and Umamaheswari 2002] gera uma PN (principalmente envolvendo temporizadores) maior e mais confusa que as demais propostas apresentadas. Ou seja, o trabalho utiliza mais elementos (lugares, transições e arcos) e a disposição desses elementos não é padronizada/organizada.

O trabalho de [Chen et al. 2012] possui problemas parecidos com os de [Latha and Umamaheswari 2002], quando se refere ao tamanho e à organização da PN. Isso ocorre devido à PN ordinária não ser compacta, visto que, é uma classe de PN que não possui muitos recursos. Já a classe de CPN, utilizada por [da Silva Oliveira et al. 2011], possui mais recursos do que a PN. Assim, podem expressar comportamentos do sistema de forma bem mais concisa do que a classe de PN ordinária.

Como já fora citado, o trabalho de [Aspar et al. 2015] foi o único com implementação. Isso evidenciou que, apesar de existirem métodos de conversão de Ladder para redes de Petri, tais métodos carecem de implementações. O método de conversão de Aspar é eficiente, tanto em execução quanto no resultado da PN. A PN resultante é bem menor e fácil de ser lida quando comparada as de [Latha and Umamaheswari 2002] e [Chen et al. 2012]. Contudo, o método de Aspar suporta apenas os conceitos de contatos e bobinas, enquanto que o de [da Silva Oliveira et al. 2011], suporta temporizadores e operações básicas, além de contatos e bobinas.

A implementação feita neste trabalho será apresentada na próxima seção. É gerada uma CPN compatível para ser utilizada e analisada pela ferramenta CPNTools. Já a PN resultante de [Aspar et al. 2015] é um grafo (tabela de valores) que não pode ser utilizada diretamente por nenhum programa de análise/simulação encontrado.

Tabela 1. Comparação entre os trabalhos teóricos

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

4. Implementação do Conversor Ladder Para CPN

O software que implementa parte do algoritmo apresentado por [da Silva Oliveira et al. 2011] é denominado de LtP. O código-fonte do conversor está disponível no repositório GitHub[6]. Em sua versão atual, o LtP está restrito aos elementos básicos de Ladder. Ou

[6] https://github.com/CarlosMacedo/ConversorLadderPetri

seja, a implementação atual suporta arquivos Ladder que possuam apenas combinações de contatos e bobinas.

Pode-se argumentar que, desta forma, qualquer um dos algoritmos propostos pelos trabalhos apresentados na Seção 3 poderia ter sido implementado. Contudo, a escolha pelo algoritmo adotado facilita, e em alguns pontos possibilita, a futura implementação dos demais conceitos de Ladder. Os demais trabalhos não podem ser estendidos devido a algumas limitações conceituais. Especificamente, os demais trabalhos não possuem suporte a conceitos como temporizadores e operações básicas (justificado na Seção 3.1).

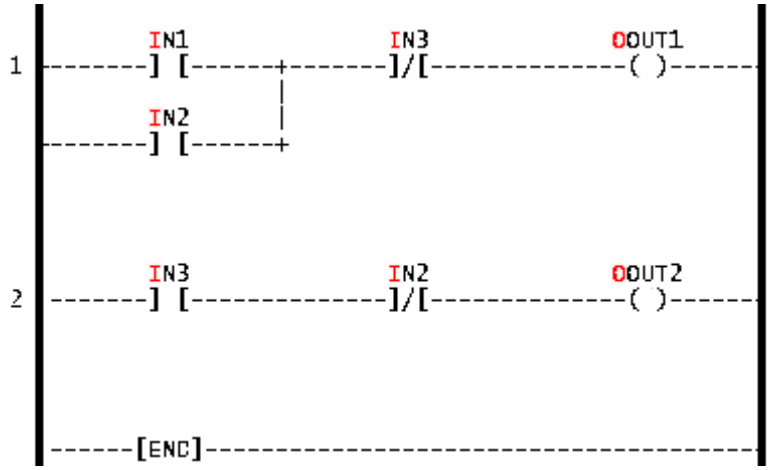


Figura 4. Exemplo de diagrama Ladder feito no LDmicro.

O conversor foi desenvolvido considerando as ferramentas LDmicro[7] e CPN Tools para Ladder e CPN, respectivamente. O LtP converte o LD criado no LDmicro (versão 2.3) em uma CPN compatível com o CPN Tools (versão 4.0.1). Na Figura 4 é ilustrado um exemplo de LD feito no LDmicro.

Esse diagrama Ladder foi convertido pelo LtP. A CPN equivalente gerada pelo conversor é apresentada na Figura 5.

[7] http://cq.cx/ladder.pl

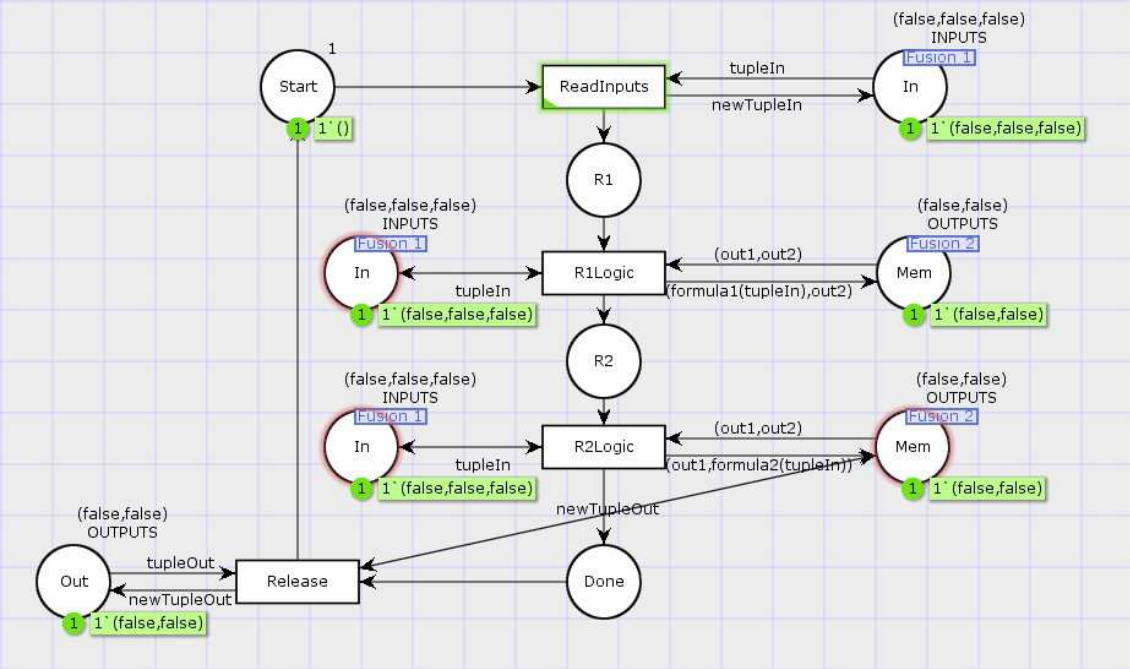


Figura 5. Modelo CPN gerado a partir do LD da Figura 4.

O processo de conversão foi dividido em quatro partes, como ilustrado na Figura 6. Cada uma destas partes é responsabilidade de um componente distinto do software, a saber:

(I) **Extrator:** componente responsável por extrair e listar todos os contatos e bobinas do arquivo LD;

(II) **Gerador:** componente responsável por construir a expressão booleana, no formato específico de CPN/ML, de cada degrau;

(III) **Configurador CPN:** componente responsável por criar e escrever todas as variáveis e as fórmulas necessárias na marcação CPN/ML do CPN Tools; e

(IV) **Desenvolvedor Gráfico:** componente responsável por combinar todos os elementos gerados e construir a parte gráfica (lugares, transições, arcos, etc.) da CPN.

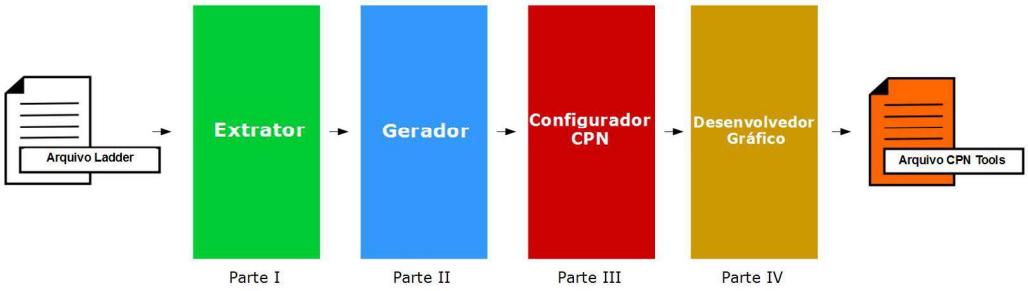
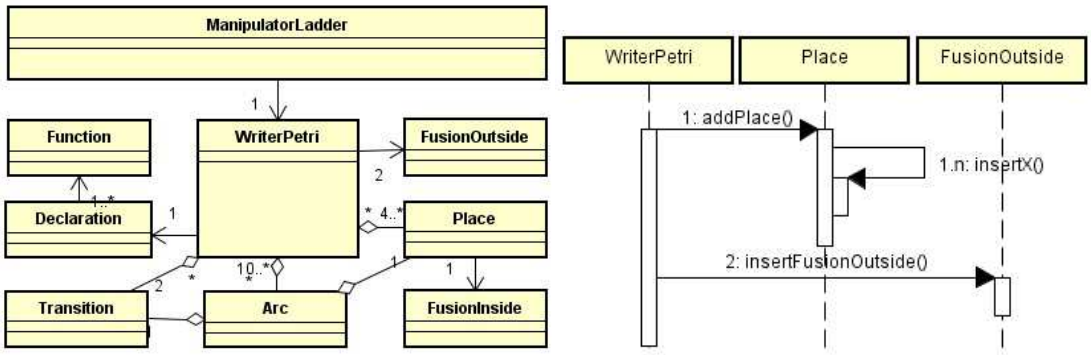


Figura 6. Processo de conversão.

Na Figura 7 são apresentados os diagramas de classe (a) e de sequência (b). Como pode ser visto no diagrama de classe, os elementos da CPN estão representados. O diagrama de sequência ilustra a construção dos lugares na CPN. Dez invocações a métodos foram substituídas por 1.n: *insertX(), a saber: 1.1: insertPlace(), 1.2: insertPlacePos(), 1.3: insertPlaceName(), 1.4: insertTypeId(), 1.5: insertTypePos(), 1.6: insertType(), 1.7: insertInitmarkId(), 1.8: insertInitmarkPos(), 1.9: insertColor() e 1.10: insertFusionInside()*. Tais simplificações nos diagramas se deram devido às restrições de espaço.



(a) Diagrama de classe. (b) Diagrama de sequência.

Figura 7. Versões simplificadas de diagramas UML do conversor.

Para que o modelo CPN seja de fácil leitura, o componente **Desenvolvedor Gráfico** lida com o tamanho da janela da aplicação. Com isso, este componente posiciona cada um dos elementos do modelo de forma a organizar a estrutura do modelo visualmente.

Como supradito, detalhes de implementação podem ser conferidos no código fonte presente no repositório GitHub.

5. API

Como dito anteriormente, a CPN gerada pelo conversor LtP é compatível com o CPN Tools (versão 4.0.1). Para isso ser possível, construiu-se uma API em java para criar uma CPN com maior abstração. Dado que para construir uma CPN em tempo de execução (no CPN Tools) era preciso escrever códigos extensos em XML. Com o desenvolvimento da API, as possíveis implementações dos métodos de conversão propostos são facilitadas – ou qualquer outra aplicação que precise ser modelada em PN. A documentação da API pode ser encontrada no GitHub[8].

5.1 Exemplo

Nesta seção é apresentado o exemplo de um possível primeiro programa feito utilizando a API LtP. Na Figura 8 é apresentado a CPN gerada pelos Algoritmos 1 e 2 (Figura 9 e 10). Esta simples CPN feita em XML possui 793 linhas, enquanto que o equivalente feito em java (utilizando a API) possui 41 linhas.

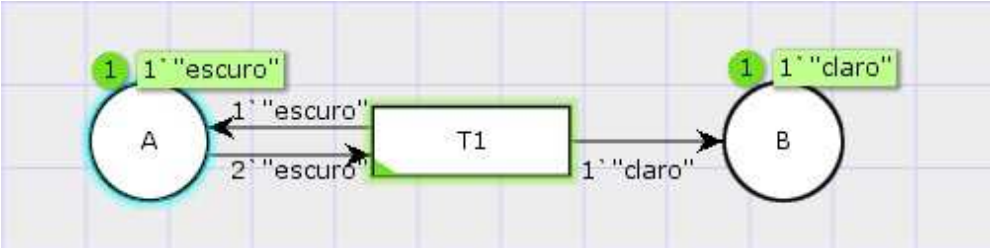


Figura 8. Rede de Petri gerada a partir dos Algoritmos 1 e 2.

Para criar a CPN da Figura 8 basta criar uma classe pública e herdar da API, como ilustrado na Figura 9 (linha 8). A partir da herança, será obrigatório implementar os métodos abstratos da API: *(i) createPlaces, (ii) createTrans, (iii) createArcs*. Tais métodos são responsáveis por criar os lugares, transições e arcos, respectivamente. Ou seja, para criar uma CPN compatível com o CPN Tools em tempo de execução, precisa-se apenas preencher os métodos informando o que deseja inserir.

Na linha 16 do Algoritmo 1 (Figura 9) é criado um lugar A, na posição[9] x0 e y0. O lugar A é o tipo string e possui uma ficha do tipo escuro.

Na linha 17 do Algoritmo 1 (Figura 9) é criado um lugar B, com dois espaçamentos[10] no eixo x. Possui o mesmo tipo do lugar A e uma ficha do tipo claro. Cria-se a transição T1, no ponto médio dos lugares A e B, na linha 22.

Por fim, são criados os arcos (A,T1), (T1,A) e (T1, B). Os arcos[11] possuem uma ficha escuro, duas fichas do tipo escuro e uma ficha do tipo claro, respectivamente.

---

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

[9] (x0,y0) é a origem do plano cartesiano.

[10] *space* é o espaço recomendado entre os elementos da Rede de Petri.

Com o processo realizado acima, a CPN está criada. Para executar o Algoritmo 1 (Figura 9) é necessário chamar o método writerCPN, como pode ser visto no Algoritmo[12] 2 (Figura 10).

```
Algoritmo 1

package MinhaPrimeiraCPN;
import java.io.IOException;
import main.API;

public class MinhaPrimeiraCPN extends API{

    @Override
    public void writerCPN(String path) throws IOException{
        super.writerCPN(path);
    }
    @Override
    protected void createPlaces() {
      createPlace(x0, y0, "A", "STRING", "1`\"escuro\"");
      createPlace(x0+space*2, y0, "B", "STRING", "1`\"claro\"");
    }

    @Override
    protected void createTrans() {
      createTran(x0+space, y0, "T1");
    }

    @Override
    protected void createArcs() {
      createArc("PtoT", "T1", "A", "1`\"escuro\"");
      createArc("TtoP", "T1", "A", "2`\"escuro\"");
      createArc("TtoP", "T1", "B", "1`\"claro\"");
    }

}
```

Figura 9. Algoritmo 1.

[11] PtoT significa arco na direção lugar/transição e TtoP é o contrário.

[12] O arquivo cpn gerado será salvo no diretório “D:/Users/out.cpn".

```
1
2  Algoritmo 2
3
4  package MinhaPrimeiraCPN;
5  import java.io.IOException;
6
7  public class HelloWorld {
8
9      public static void main(String[] args) throws IOException {
10       MinhaPrimeiraCPN cpn = new MinhaPrimeiraCPN();
11       cpn.writerCPN("D:\\Users\\out.cpn");
12     }
13 }
```

Figura 10. Algoritmo 2.

6. Conclusão

Neste trabalho foi apresentada uma implementação de conversão de código Ladder para modelos de Redes de Petri Coloridas: o conversor LtP. A implementação deste conversor é considerada uma boa contribuição, visto que existem muitas propostas de algoritmos de conversão, mas nenhuma implementação que suporte (ou possibilite suporte futuro) aos conceitos presentes na linguagem Ladder[13].

Construiu-se também uma API para criar uma CPN com maior abstração, dado que para construir uma CPN em tempo de execução (no CPN Tools) era preciso escrever códigos extensos em XML. Com a API a construção de uma CPN é facilitada.

A proposta de algoritmos para conversão é muito importante como passo inicial. Contudo, na prática, ainda não resolvem os problemas existentes no dia-a-dia. De posse desta implementação, é esperado que a *necessidade de tempo extra para modelagem* e o *custo de treinamento de recursos humanos* sejam efetivamente minimizados, se não eliminados. Pois, não será necessário que o desenvolvedor do programa em LD tenha conhecimento em PN para gerar os modelos e a conversão será feita automaticamente pelo software.

Considerando as vantagens no uso das linguagens de modelagem formais, tais como precisão matemática e possibilidade de simulação do sistema modelado, espera-se que o conversor auxilie na análise de sistemas industriais. Com isso, assegurando que tais sistemas se comportem conforme esperado.

O conversor LtP ficou restrito à manipulação de contatos e bobinas. Como trabalhos futuros, é pretendido implementar os conceitos de operadores e temporizadores, cujos algoritmos já estão postos em [da Silva Oliveira et al. 2011].

[13] Vide novamente a Seção 3.

Referências

Aspar, Z., Shaikh-Husin, N., and Khalil-Hani, M. (2015). Algorithm to convert programmable logic controller ladder logic diagram models to petri net models. In: *2015 IEEE Student Conference on Research and Development (SCOReD)*, páginas 156–161.

Chen, X., Luo, J., and Qi, P. (2012). Method for translating ladder diagrams to ordinary petri nets. In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, páginas 6716–6721.

da Silva Oliveira, E. A., da Silva, L. D., Gorgônio, K., Perkusich, A., and Martins, A. F. (2011). Obtaining formal models from ladder diagrams. In: *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, páginas 796–801.

Jensen, K. (2013). *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media.

John, K.-H. and Tiegelkamp, M. (2010). *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids. Spinger Science & Business Media.*

Latha, K. and Umamaheswari, B. (2002). Supervisory control of an automated system with ladder logic programming and analysis using petri nets. In: *2002 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, páginas 5–pp.

Lee, G. B. and Lee, J. S. (2002). Conversion of LD program into augmented PN graph. In: *International journal of modelling & simulation*, 22(4), páginas 201–212.

Maciel, P. R., Lins, R. D., and Cunha, P. R. (1996). *Introdução às redes de Petri e aplicações*. UNICAMP-Instituto de Computação.

Mader, A. and Wupper, H. (1999). Timed automaton models for simple programmable logic controllers. In: *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, páginas 106–113.

Murata, T. (1989). Petri nets: Properties, analysis and applications. In: *Proceedings of the IEEE*, 77(4):541–580.

Ridley, J. (2004). *Mitsubishi FX programmable logic controllers: applications and programming*. Elsevier.

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

Zhou, M. and Twiss, E. (1995). A comparison of relay ladder logic programming and petri net approach for sequential industrial control systems. In: *Control Applications, 1995., Proceedings of the 4th IEEE Conference on*, páginas 748–753.