Um método para simular e verificar Redes de Petri Aninhadas

Gustavo Borges Lugoboni

Dissertação apresentada
ao
Centro de Matemática, Computação e Cognição
da
Universidade Federal do ABC
para
obtenção do título
de
Mestre em Ciência da Computação

Programa: Pós-Graduação em Ciência da Computação
Orientador: Profa. Dra. Carla Negri Lintzmayer

Santo André, maio de 2021

Um método para simular e verificar Redes de Petri Aninhadas

Esta versão da dissertação contém as correções e alterações sugeridas
pela Comissão Julgadora durante a defesa da versão original do trabalho,
realizada em 07/05/2021.

Comissão Julgadora:

- Profª. Drª. Carla Negri Lintzmayer (orientadora) - CMCC/UFABC
- Profª. Drª. Kelly Rosa Braghetto - IME/USP
- Prof. Dr. José Reinaldo Silva - Poli/USP

Sistema de Bibliotecas da Universidade Federal do ABC
Elaborada pelo Sistema de Geração de Ficha Catalográfica da UFABC
com os dados fornecidos pelo(a) autor(a).

Borges Lugoboni, Gustavo
Um método para simular e verificar Redes de Petri Aninhadas / Gustavo Borges Lugoboni. — 2021.

115 fls. : il.

Orientadora: Carla Negri Lintzmayer

Dissertação (Mestrado) — Universidade Federal do ABC, Programa de Pós-Graduação em Ciência da Computação, Santo André, 2021.

1. Redes de Petri Aninhadas. 2. Verificação de modelos. 3. Ferramenta de verificação. I. Negri Lintzmayer, Carla. II. Programa de Pós-Graduação em Ciência da Computação, 2021. III. Título.

Este exemplar foi revisado e alterado em relação à versão original, de acordo com as observações levantadas pela banca examinadora no dia da defesa, sob responsabilidade única do(a) autor(a) e com a anuência do(a) (co)orientador(a).

Santo André, 12 de Julho de 2021 .

GUSTAVO Borges Lugoboni

Nome completo e Assinatura do(a) autor(a)

CARLA NEGRI LINTZMAYER

Nome completo e Assinatura do(a) (co)orientador(a)

MINISTÉRIO DA EDUCAÇÃO
Fundação Universidade Federal do ABC
Avenida dos Estados, 5001 – Bairro Santa Terezinha – Santo André – SP
CEP 09210-580 · Fone: (11) 4996-0017

FOLHA DE ASSINATURAS

Assinaturas dos membros da Banca Examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato, GUSTAVO BORGES LUGOBONI realizada em 07 de Maio de 2021:

Prof.(a) JOSÉ REINALDO SILVA
UNIVERSIDADE DE SÃO PAULO

Prof.(a) KELLY ROSA BRAGHETTO
UNIVERSIDADE DE SÃO PAULO

Prof.(a) LUIS ALBERTO MARTINEZ RIASCOS
UNIVERSIDADE FEDERAL DO ABC

Prof.(a) VLADIMIR EMILIANO MOREIRA ROCHA
UNIVERSIDADE FEDERAL DO ABC

Prof.(a) CARLA NEGRI LINTZMAYER
UNIVERSIDADE FEDERAL DO ABC - Presidente

* Por ausência do membro titular, foi substituído pelo membro suplente descrito acima: nome completo, instituição e assinatura

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Agradecimentos

Agradeço à minha esposa Jaqueline pela paciência e apoio, a Deus, à Profa. Dra. Carla Negri Lintzmayer por sua orientação e incentivo nos momentos decisivos e, especialmente à querida Dra. Mirtha Lina Fernández Venero, a quem dedico esta dissertação, pois sem sua criatividade e orientação este trabalho jamais teria existido.

Resumo

O formalismo das Redes de Petri tem sido estendido de várias maneiras para suportar recursos como tipos de dados, hierarquias, tempo, comunicação, prioridades, aninhamento e recursão. Os dois últimos recursos combinados nas Redes de Petri Aninhadas permitem uma modelagem incremental usando múltiplas camadas, que é muito conveniente para lidar com os sistemas cada vez mais complexos de hoje. Apesar disso, atualmente não existe ferramenta para projetar, simular e verificar as propriedades dessas redes. Portanto, na prática, elas devem ser transformadas em redes hierárquicas de uma única camada antes de serem analisadas. Esse processo de achatamento aumenta significativamente o tamanho da rede, dificultando a simulação e a interpretação dos resultados no modelo original após a verificação.

O presente trabalho apresenta um panorama histórico que levou ao desenvolvimento das redes aninhadas e tem como objetivo fornecer um método que permita analisar o comportamento de uma rede de Petri aninhada preservando sua estrutura multicamadas. Para este fim, por meio desta dissertação, foi proposto o uso de uma ferramenta de verificação de modelos usualmente aplicada à verificação de *software multithread*, o SPIN. Em particular, as Redes de Petri Aninhadas foram modeladas a partir de uma proposta de extensão da PNML (*Petri Net Markup Language* - ISO15909-2), e traduzidas de forma automática em modelos de entrada para o verificador SPIN. O resultado da tradução automática foi avaliado a partir da análise de propriedades de redes aninhadas baseadas em modelos que surgem principalmente de dois domínios de aplicação: os sistemas multiagentes e fluxos de trabalho. Isso foi feito utilizando exemplos da literatura para avaliar a eficácia do método e comparar os resultados com os de outras iniciativas.

Palavras-chave: Redes de Petri Aninhadas, Verificação de modelos, Ferramenta de verificação.

Abstract

The Petri nets formalism has been adapted and extended in several ways in order to support features such as data structures, hierarchies, time, communication, priorities, nesting, and recursion. The last two features, combined in the Nested Petri nets, allow incremental modeling by using multiple layers, which is very convenient for dealing with complex systems. In spite of this, currently, there is no tool to simulate and verify the properties of these nets. Therefore, in practice, these multilayer nets are usually turned into flat nets. This flattening process increases significantly the size of the net, making the simulation and the interpretation of the results in the original model after verification difficult.

This work presents a historical overview that led to the development of nested networks and aims to provide a method that allows the analysis of the behavior of a nested Petri network while preserving its multilayer structure. To that end, through this dissertation, the use of a model verification tool usually applied to multithreaded software verification, the SPIN, is proposed. Particularly, the nested Petri nets were modeled in a proposed extension of the PNML (Petri Net Markup Language - ISO15909-2) that is automatically translated into input models for the SPIN verifier. The result of this translation was evaluated by the analysis of some properties of nested networks based on models that arise mainly from two application domains: multiagent systems and workflows. This was done using examples from the literature to assess the effectiveness of the method and to compare the results with other initiatives.

Keywords: Nested Petri Nets, Model Checking, Verification tool.

Sumário

Lista de Figuras

Capítulo 1

Um método para simular e verificar Redes de Petri Aninhadas

Cada dia novos sistemas são construídos ou idealizados de forma a solucionar problemas que surgem com as demandas da modernidade. A natureza destes sistemas é inerentemente complexa e concorrente, pois, além de internamente serem modularizados e compostos de subsistemas, exteriormente competem por recursos e operam em ambientes multiagentes. Isso é latente em tantos ambientes quanto sejam passíveis de análise, desde indústrias de produção como a automobilística até em sistemas de controle aéreo. Agentes autônomos controlados por componentes de *software* operam em consonância entre si em toda parte e cada vez mais.

Métodos formais são cada vez mais aplicados no desenvolvimento de *software* devido a esta complexidade. Os modelos são utilizados nos estágios iniciais do projeto para simular e compreender o comportamento do sistema, ou nos estágios finais para fins de verificação e validação. A construção de um modelo não é uma tarefa trivial e, mesmo quando projetado para fornecer uma visão abstrata e reduzida de um sistema, seu tamanho pode ser grande. Uma abordagem comum para lidar com um modelo complexo é organizá-lo em camadas com diferentes níveis de detalhes. Essa abordagem de vários níveis produz modelos mais fáceis de entender mas também pode esconder falhas sutis que são mais difíceis de detectar e corrigir. No entanto, há poucas pesquisas sobre verificação automática da consistência de um modelo de vários níveis usando métodos incrementais [85].

Redes de Petri são um método formal criado por Carl Adam Petri em sua tese de doutorado no início dos anos 60[1]. Com características gráficas e propriedades matemáticas bem definidas, estas redes são promissoras pois podem descrever e auxiliar o estudo de sistemas de processamento de informações com propriedades não-determinísticas, paralelas, estocásticas, assíncronas e distribuídas [67].

Uma Rede de Petri clássica consiste basicamente em um conjunto de lugares, um conjunto de transições e um conjunto de arcos que conectam os lugares às transições. Transições podem ser interpretadas como eventos que ocorrem no sistema, enquanto lugares são interpretados como condições para que as transições ocorram. Uma forma de indicar se as condições são satisfeitas é atribuindo marcas chamadas *tokens* aos lugares. Conforme os eventos ocorrem, *tokens* são removidos de lugares e acrescentados a outros, de acordo com o que é definido pelos arcos.

[1] Uma grande coleção de informações sobre a história e o desenvolvimento das Redes de Petri pode ser conferida em http://www.informatik.uni-hamburg.de/TGI/PetriNets/index.php.

As Redes de Petri têm sido utilizadas em uma vasta gama de domínios, desde processos de negócios até sistemas biológicos [42, 46, 77, 81]. O formalismo evoluiu rapidamente de um modelo básico de lugares/transições para um modelo de redes de alto nível [44]. Com o tempo, muitas outras derivações foram criadas. Desde 2004, as Redes de Petri são reguladas por uma norma ISO (ISO15909 [41]), contendo suas especificações e abarcando até mesmo possíveis extensões.

Algumas técnicas criadas para facilitar o projeto de sistemas com Redes de Petri foram a introdução de camadas nas redes, de forma que redes inteiras poderiam ser representadas por apenas um lugar em uma outra rede hierarquicamente superior e, posteriormente, os próprios *tokens* passaram a representar outras redes de diversas formas.

As Redes de Petri Aninhadas são um destes tipos de redes capazes de adicionar uma camada de abstração onde os *tokens* da rede podem ser outras redes [59]. Uma Rede de Petri Aninhada define um conjunto de tipos de rede, de forma que os *tokens* de rede devem ser criados, transportados e consumidos da mesma forma que os comuns. Além disso, os *tokens* de rede podem disparar suas próprias transições de maneira independente ou de forma sincronizada junto a outros *tokens* de rede. O comportamento das redes em níveis inferiores pode ser influenciado pelo estado de lugares compartilhados com o nível superior. Uma transição pode ainda construir novos *tokens* usando operações como a composição sequencial, alternativa ou paralela [62].

Dessa forma, o uso das redes aninhadas se torna uma opção para a modelagem de sistemas com subsistemas paralelos e não determinísticos, o que as tornam uma ferramenta interessante para aplicações como sistemas multiagentes e fluxos de trabalho interorganizacionais. Assim, uma ferramenta que auxilie na verificação destes modelos tem o potencial de tornar viável o uso destas redes em detrimento de outras alternativas.

De acordo com Venero e Corrêa da Silva [85], diversas variações de Redes de Petri e formalismos similares, como fluxos de trabalho, processos de negócio e diagramas UML, têm sido analisados através de ferramentas de verificação de modelos, como DVE [53], LTSA [73], NuSMV [22] e SPIN [7, 27, 33, 38, 51, 74, 88], mas apenas algumas destas traduções poderiam ser adaptadas para uma Rede de Petri Aninhada arbitrária. Na maioria dos casos, ou a linguagem da ferramenta não oferece suporte à recursão (DVE, LTSA, NuSMV e STeP), ou a tradução evita múltiplas camadas, ou então restringe a sincronização.

Uma abordagem modular para verificação de Redes de Petri Aninhadas foi descrita por Chang e He [10] para redes de dois níveis. Contudo, nesta tradução alguns importantes recursos de sincronização não são considerados. Dworzánski e Lomazova [20] recentemente utilizaram a conhecida ferramenta CPN Tools para verificação de redes aninhadas de dois níveis com algumas restrições. Algumas propostas de modelagem de Redes de Petri em PROMELA foram feitas por Gannod e Gupta [27] e, também, por Ribeiro e Fernandes [74]. Redes multicamadas e recursivas foram analisadas com SPIN pela primeira vez por Venero e Corrêa da Silva [82, 83], mas a tradução ali proposta deve ser feita manualmente e, além disso, deve ser feita de forma muito específica para cada rede. Esta metodologia de tradução poderia se valer muito de uma ferramenta que automatizasse o processo para uma gama mais generalizada de redes aninhadas pois, conforme evidenciado em uma busca na literatura apresentada no Capítulo 3, uma ferramenta deste tipo para este conjunto de redes não existe.

Esta dissertação pretende contribuir para a construção de um *framework* para analisar de forma automática o comportamento e verificar as propriedades de alguns tipos de redes de Petri Aninhadas.

Nossa proposta inclui uma nova extensão para a parte dois da ISO 15909-2 [41] (que especifica a *Petri Net Markup Leanguage* – PNML), uma nova forma de modelagem das Redes de Petri Aninhadas na linguagem PROMELA e, por fim, a implementação de um tradutor da extensão da PNML para o código PROMELA.

O restante do texto está organizado da seguinte forma. No Capítulo 2 são apresentadas algumas definições que são essenciais para o desenvolvimento da proposta em conjunto com o histórico do paradigma de *tokens* como redes. Seguindo, o Capítulo 3 contém uma revisão sistemática da literatura que evidencia a necessidade de uma ferramenta para verificação de redes de Petri aninhadas. No Capítulo 4 a importância do projeto de pesquisa será apresentada juntamente com a proposta da extensão da PNML seguida, por fim, da modelagem em PROMELA. O Capítulo 5 apresenta uma implementação de um protótipo do tradutor em conjunto com alguns resultados em dois estudos de caso e, por último, algumas considerações finais são apresentadas no Capítulo 6.

Capítulo 2

Redes de Petri e o paradigma de *tokens* como redes

Redes de Petri são um formalismo simples e poderoso para modelar processos complexos com concorrência, recursos limitados ou distribuídos, o que as torna *per se* uma ferramenta de grande utilidade dentro dos contextos modernos de modelagem e simulação de processos nas mais diversas áreas do conhecimento [67]. Através destas redes, pode-se prever o funcionamento de sistemas projetados e garantir sua corretude formal antes da implementação dos mesmos.

Neste capítulo apresentaremos alguns conceitos básicos e definições relacionadas às Redes de Petri. De maneira incremental e progressiva o paradigma de *tokens* como redes será introduzido de forma a conduzir o leitor à definição das Redes de Petri Aninhadas, que servirão de base para a tradução proposta e estudada nos capítulos seguintes, junto a uma discussão sobre os diferentes tipos de Rede de Petri que se adéquam a este paradigma e suas diferenças. Na Seção 2.1 introduziremos formalmente as Redes de Petri em uma de suas formulações mais básicas, seguido de uma breve introdução de suas propriedades na Seção 2.2. Na Seção 2.3 apresentaremos o paradigma de *tokens* como redes junto de uma introdução a multiconjuntos, necessária para as Seções 2.4, 2.5 e 2.6, que apresentam as redes coloridas, por objeto e aninhadas, respectivamente. Por fim, na Seção 2.7 encerraremos o capítulo com uma discussão sobre os diferentes tipos de redes apresentados.

2.1 Redes de Petri

Uma **Rede de Petri** clássica consiste em um conjunto de **lugares**, um conjunto de **transições** e um conjunto de **arcos** que conectam os lugares às transições. Transições podem ser interpretadas como eventos que ocorrem no sistema, enquanto lugares são interpretados como condições para que as transições ocorram. Uma forma de indicar se as condições são satisfeitas é atribuindo marcas chamadas ***tokens*** aos lugares. Assim, as redes clássicas são generalizadas com **pesos** nos arcos, que indicam quantos *tokens* são necessários para que um evento ocorra e quantos *tokens* serão criados quando um evento ocorrer. Esta classe de rede, denominada de **Redes *Place/Transition***, e suas propriedades foram amplamente exploradas [15, 67] e sua estrutura pode ser definida formalmente como a seguir.

Definição 2.1.1. *Uma* **Rede de Petri** *é uma 5-upla,* $PN = (P, T, A, w, I)$*, onde:*

- $P = \{p_1, p_2, \ldots, p_n\}$ *é um conjunto finito de* ***lugares****;*

- $T = \{t_1, t_2, \ldots, t_r\}$ *é um conjunto finito de* ***transições****, onde* $P \cap T = \emptyset$ *e* $P \cup T \neq \emptyset$*;*
- $A \subseteq (P \times T) \cup (T \times P)$ *é um conjunto de* ***arcos****;*
- $w\colon A \to \{1, 2, 3, \ldots\}$ *é uma* ***função de peso*** *nos arcos;*
- $I\colon P \to \{0, 1, 2, 3, \ldots\}$ *é uma* ***função de inicialização****.*

Antes de formalizar o comportamento dessas redes, precisamos de outras definições importantes. O estado ou configuração de uma rede de Petri é chamado de **marcação**, e nada mais é que uma distribuição de *tokens* nos lugares, isto é, uma função que atribui um número a cada lugar, representando a quantidade de *tokens* alocados nele. De maneira formal, podemos definir uma marcação como a seguir.

Definição 2.1.2. *Dada uma rede* $R = (P, T, A, w, I)$*, uma* **marcação** *é uma função* $M : P \to \{0, 1, 2, \ldots\}$*. A marcação* M_0 *tal que* $M_0(p) = I(p)$ *para todo* $p \in P$ *é denominada* **marcação inicial**.

Dada uma marcação M *qualquer de* R*, se para todo* $p_i \in P$ *temos* $M(p_i) = k_i$*, então uma outra forma de denotar* M *é com a notação* $\lessdot p_1 : \{k_1\}, p_2 : \{k_2\}, \ldots, p_n : \{k_n\} \gtrdot$.

Uma representação gráfica das Redes de Petri é possível através de grafos direcionados bipartidos onde lugares e transições são representados por vértices e os arcos entre lugares e transições são representados por arestas direcionadas. Por isso, elas podem ser modeladas a partir de uma simbologia em diagramas, que se mostram bastante úteis para a visualização das mesmas. Tal simbologia tem como base as seguintes convenções:

- lugares são representados por círculos;
- transições são representadas por retângulos verticais;
- arcos são representados por setas indicando o sentido do arco;
- pesos são representados por rótulos nos arcos que, se forem unitários, podem ser omitidos;
- *tokens* podem ser representados por símbolos dentro dos lugares, como por exemplo •. Neste trabalho, um *token* será representado por **[ ]**.

Essa notação gráfica torna intuitiva a seguinte terminologia. Dado um lugar p e uma transição t, dizemos que o arco (p, t) **entra** ou **incide** em t e que o arco (t, p) **sai** de t. Tanto t quanto p são **extremos** desses arcos.

A Figura 2.1 contém um diagrama utilizado para modelar uma Rede de Petri. Nela é possível observar três lugares, $P1$, $P2$ e $P3$, duas transições, $T1$ e $T2$, e seis arcos, $(P1, T1)$, $(T1, P2)$, $(T1, P3)$, $(P2, T2)$, $(P3, T2)$ e $(T2, P1)$, conectando as transições aos lugares. A função w de peso nos arcos, que pode ser observada nos rótulos dos arcos no diagrama, é tal que $w(P1, T1) = 1$, $w(T1, P2) = 2$, $w(T1, P3) = 1$, $w(P2, T2) = 2$, $w(P3, T2) = 1$ e $w(T2, P1) = 2$. A marcação inicial M_0 são os dois *tokens* em $P1$ e pode ser denotada por $\lessdot P1 : \{2\}, P2 : \emptyset, P3 : \emptyset \gtrdot$.

Uma vez que conseguimos representar um estado de uma rede por meio de uma marcação, é possível representar o **comportamento** da rede por meio dos estados que ela pode alcançar. Em outras palavras, o comportamento da rede pode ser visto como uma movimentação de *tokens* pelos

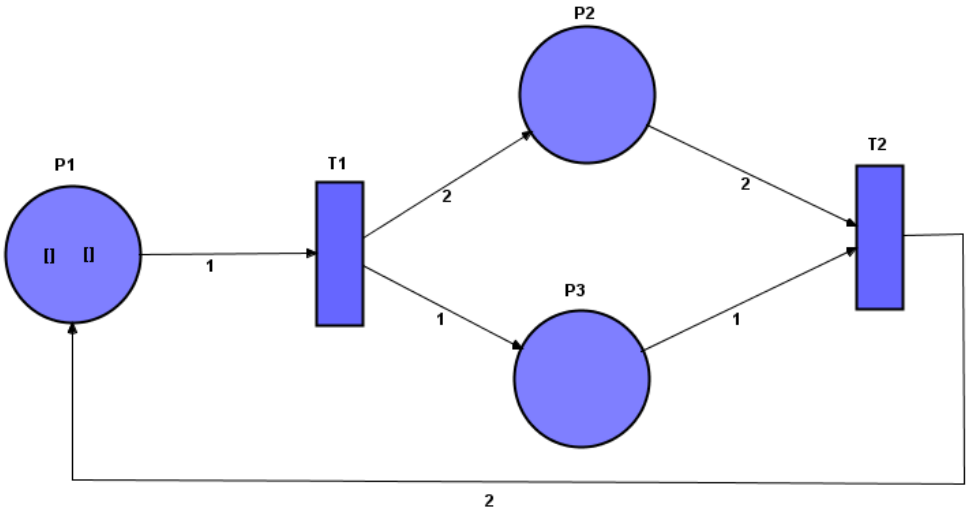

Figura 2.1: *Rede de Petri* $PN_1 = (\{P1, P2, P3\}, \{T1, T2\}, A, w, I)$*, em que o conjunto* A *de arcos e seus pesos podem ser observados no diagrama. A função de inicialização* I*, que dá a marcação inicial, indica que o lugar* $P1$ *possui dois* tokens*.*

lugares. De forma geral, a mudança do estado da rede acontece quando um evento (transição) ocorre, o que chamamos de **disparo** da transição. Para que uma transição possa ser disparada, precisamos observar a marcação atual dos lugares e os pesos dos arcos.

A seguir definimos os conjuntos **pre-set** e **pos-set**, que indicam quais lugares estão diretamente envolvidos com uma determinada transição e, no que tange ao **pre-set**, podem vir a influenciar no seu disparo.

Definição 2.1.3. *Dada uma rede* (P, T, A, w, I)*, o* **pre-set** *de um elemento* $t \in T$*, denotado* $\bullet t$*, é definido como*

$$\bullet t = \{p \in P \colon (p, t) \in A\},$$

isto é, o conjunto de lugares que são extremos de arcos que entram em t*.*

Definição 2.1.4. *Dada uma rede* (P, T, A, w, I)*, o* **pos-set** *de um elemento* $t \in T$*, denotado* $t\bullet$*, é definido como*

$$t\bullet = \{p \in P \colon (t, p) \in A\},$$

isto é, o conjunto de lugares que são extremos de arcos que saem de t*.*

No que segue, considere a rede $PN = (P, T, A, w, I)$. Como dissemos antes, as transições mudam o estado da rede através da marcação dos lugares. O disparo de uma transição pode ocorrer a partir de uma determinada marcação se e somente se a transição estiver **habilitada**, o que significa ter uma condição de disparo satisfeita. Formalmente, uma transição $t \in T$ é considerada habilitada por uma marcação M se para todo $p \in \bullet t$ vale que $M(p) \geq w(p, t)$, isto é, a quantidade de *tokens* no lugar p é pelo menos o peso do arco (p, t). O disparo desta transição t gera uma nova marcação M'

definida a partir de M da seguinte forma. Para cada $p \in P$:

$$M'(p) = \begin{cases} M(p) - w((p,t)) & \text{se } p \in {\bullet t} \setminus t{\bullet} \\ M(p) + w((t,p)) & \text{se } p \in t{\bullet} \setminus {\bullet t} \\ M(p) & \text{caso contrário}\,. \end{cases}$$

Utilizamos a notação $M[t\rangle M'$ para indicar que a marcação M' foi gerada a partir de M através do disparo da transição t. Dizemos ainda que M' é sucessora de M na rede PN.

No exemplo da rede na Figura 2.1, a transição $T1$ está habilitada. Após seu disparo, um *token* é consumido do lugar $P1$, dois novos *tokens* são produzidos em $P2$ e um novo *token* é produzido em $P3$, de acordo com os rótulos dos arcos. A Figura 2.2 mostra a nova marcação. Observe que um novo disparo da transição $T1$ pode ser feito, pois a mesma permanece habilitada devido ao *token* remanescente em $P1$. Além disso, outras condições de disparo foram alcançadas com a produção dos *tokens* em $P2$ e $P3$, impostas pelos arcos $(P2, T2)$ e $(P3, T2)$ e, portanto, a transição $T2$ também encontra-se habilitada para ser disparada.

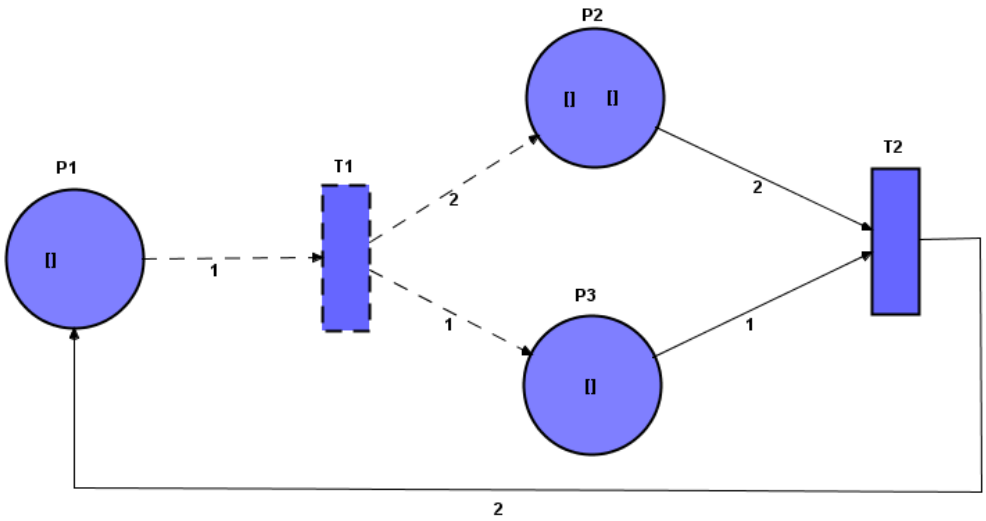

Figura 2.2: *Nova marcação da rede da Figura 2.1 após o disparo da transição* $T1$.

Uma **sequência de disparos** σ pode ser denotada por $\sigma = M_0 t_{a_1} M_1 t_{a_2} M_2 \ldots t_{a_\ell} M_\ell$, ou simplesmente $\sigma = t_{a_1} t_{a_2} \ldots t_{a_\ell}$, em que M_0 é a marcação inicial e cada M_i é a marcação gerada a partir da marcação M_{i-1} pelo disparo da transição $t_{a_i} \in T$, para $i \geq 1$. O **comportamento** de uma rede é definido por todas as sequências de disparos que podem ser executadas a partir da marcação inicial.

2.2 Propriedades, corretude e verificação

Quando um sistema é modelado através de um método formal, o objetivo do projetista é atender alguma especificação ou determinados requisitos com a segurança de que seu projeto de fato satisfaz os requisitos. Segundo Girault e Valk [31], diferentes conceitos de corretude existem e um

sistema pode ser classificado como correto quando seu modelo especificado e sua implementação são equivalentes ou quando este sistema exibe uma série de propriedades desejáveis, seja atender uma série de fórmulas lógicas ou conter um conjunto característico de propriedades do tipo de método formal utilizado.

Segundo Murata [67], duas categorias de propriedades podem ser estudadas em uma Rede de Petri, sendo elas as dependentes da marcação inicial, conhecidas como **propriedades comportamentais**, e as independentes da marcação inicial, chamadas de **propriedades estruturais**. No que segue, vamos nos referir a uma rede $PN = (P, T, A, w, I)$ qualquer, sendo M_0 sua marcação inicial.

As principais propriedades comportamentais são:

- **Alcançabilidade**: Uma marcação M da rede PN é dita **alcançável** se existe uma sequência de disparos σ a partir de M_0 da qual M faz parte. Nesse caso, escrevemos $M_0|\sigma\rangle M$. O conjunto de todas as marcações alcançáveis a partir de M_0 em PN é denotado por $R(PN)$;
- **Limitabilidade**: Uma rede é considerada **k-limitada** se a quantidade de *tokens* em cada lugar não excede o número inteiro finito k para qualquer marcação alcançável a partir de M_0, isto é, se $M(p) \leq k$ para todo $p \in P$ e toda marcação $M \in R(PN)$. Uma rede é dita **segura** se for 1-limitada. Um exemplo de rede 2-limitada pode ser visto na Figura 2.3;
- **Vivacidade**: Uma rede é considerada **viva** se qualquer transição possa ser disparada em alguma sequência de disparos a partir de M_0. Um exemplo de rede viva pode ser observado na Figura 2.4.

Vivacidade é uma propriedade ideal para muitos sistemas, contudo sua verificação de forma exaustiva é impraticável por ser muito custosa. Por isso, algumas definições mais relaxadas de vivacidade foram definidas. Denotamos o conjunto de todas as sequências de disparo possíveis a partir de M_0 em PN por $L(PN)$. Uma transição $t \in T$ é denominada:

1. **L0-viva**, ou **morta**, se t não aparece em nenhuma sequência de disparo em $L(PN)$;
2. **L1-viva**, ou **potencialmente disparável**, se t pode ser disparada pelo menos uma vez em alguma sequência de disparo em $L(PN)$;
3. **L2-viva** se t pode ser disparada pelo menos k vezes em alguma sequência de disparo em $L(PN)$, para algum inteiro k;
4. **L3-viva** se t aparece infinitamente, frequentemente em alguma sequência de disparo em $L(PN)$;
5. **L4-viva**, ou **viva** caso t seja L1-viva para toda marcação em $R(PN)$.

Uma rede é chamada Lk-viva se toda transição da rede for Lk-viva, para $k = 0, 1, 2, 3, 4$. A L4-vivacidade é a definição mais forte e corresponde à definição dada inicialmente. É possível verificar também que L4-vivacidade $\Rightarrow$ L3-vivacidade $\Rightarrow$ L2-vivacidade $\Rightarrow$ L1-vivacidade, onde $\Rightarrow$ significa "implica". Uma transição é chamada **estritamente Lk-viva** caso seja Lk-viva mas não seja L$(k+1)$-viva, para $k = 1, 2, 3$.

O funcionamento de uma Rede de Petri pode ser inicialmente analisado com base na execução da simulação do modelo criando as sequências de disparo na rede de forma não determinada. Contudo,

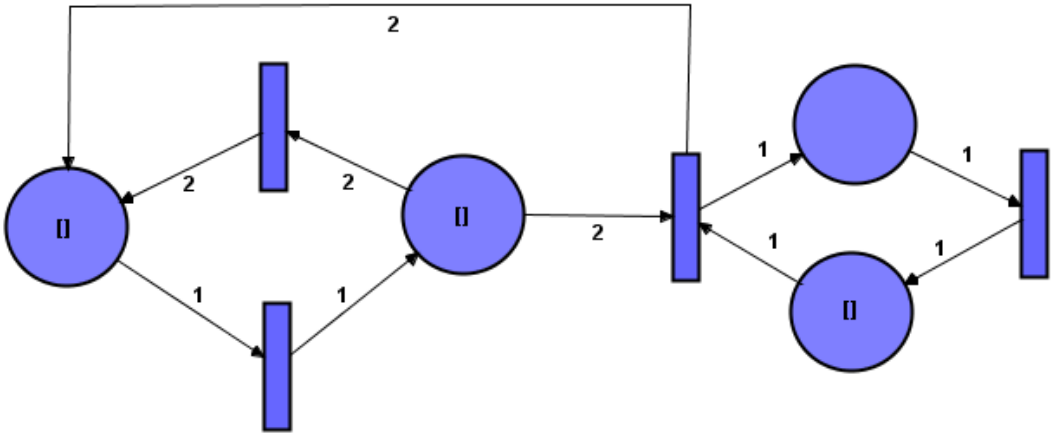

Figura 2.3: *Exemplo de uma rede 2-limitada. Não importa qual seja a sequência de disparos nesta rede, o número de* tokens *em qualquer dos lugares será sempre menor ou igual a dois.*

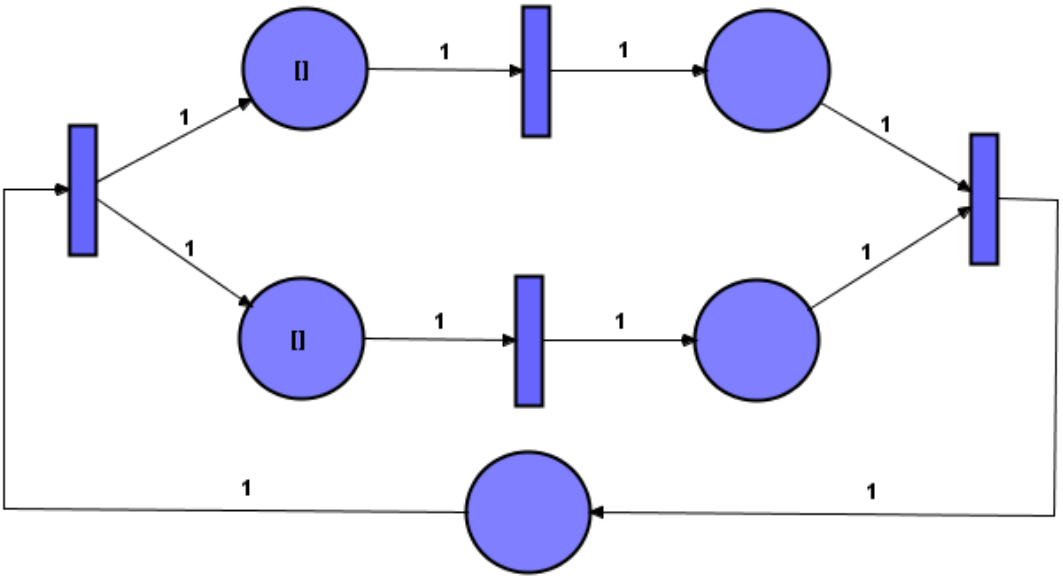

Figura 2.4: *Exemplo de uma rede viva. Nesta rede, todas as transições podem ser disparadas infinitamente em qualquer sequência de disparos partindo da marcação inicial.*

este método permite apenas extrair uma compreensão do funcionamento geral da rede e ajuda a descobrir alguns erros superficiais do modelo mas nada ajuda quanto a checagem da corretude pois, em sistemas complexos, é virtualmente impossível esgotar todas as possibilidades de sequências de disparo por meio de simulações aleatórias. Para assegurar a corretude é necessário, portanto, verificar as propriedades estabelecidas e requeridas no projeto.

Com respeito a análise dessas propriedades nas Redes de Petri, de acordo com Girault e Valk [31], as técnicas podem ser classificadas como técnicas de enumeração ou técnicas dirigidas a rede. As técnicas dirigidas a rede buscam obter informações sobre as redes por meio de seu comportamento a partir da marcação inicial e costumam utilizar-se de transformações, geralmente reduções, nas redes para possibilitar este tipo de análise.

Já as técnicas de enumeração consistem em criar um grafo de alcançabilidade, total ou parcial e, caso a rede seja limitada, utilizar este grafo para provar propriedades através de verificação de fórmulas de lógica temporal ou por meio de outras técnicas que podem ser automatizadas. Neste trabalho iremos focar em uma técnica de enumeração que consistirá em, a parir do código PROMELA de uma rede, utilizar o verificador de modelos SPIN para calcular o grafo de alcançabilidade e buscar verificar algumas propriedades por meio da funcionalidade de verificação de fórmulas de lógica temporal do SPIN[1].

2.3 O paradigma de *tokens* como redes

Através da crescente complexidade dos sistemas a serem modelados, as Redes de Petri de Alto Nível foram criadas, possibilitando que lugares e *tokens* carregassem mais informação e, com isso, pudessem simplificar o projeto de sistemas mais complexos. Einar Smith [77] fornece uma boa introdução a estas redes em seu trabalho. Estas redes, mais tarde, foram especializadas em diversos tipos de técnicas de abstração da representação[2] a fim de obter uma forma mais adequada de modelagem para problemas de naturezas diferentes, desde sistemas dependentes de lógica *fuzzy* [11] até sistemas multivariados [42].

Para nós, contudo, destacam-se para o presente trabalho as Redes de Petri Coloridas e as Redes de Petri por Objetos, devido ao fato de que a abordagem das redes nestes tipos, especialmente nas Redes de Petri por Objetos [79], serviu de base para o paradigma de *tokens* como Redes de Petri. As Redes de Petri Coloridas surgiram a partir da combinação das Redes de Petri com o conceito de linguagem de programação funcional [43]. Mais especificamente, estas redes utilizam os conceitos de tipo, valor, operação, expressão, variável, atribuição e avaliação da mesma forma como estes são utilizados em linguagens de programação funcionais. Por sua vez, as Redes de Petri por Objeto se originam da combinação das Redes de Petri com a tecnologia de modelagem de orientação por objetos da mesma forma como esta é aplicada a linguagens de programação orientadas a objetos [78]. Apesar de facilitarem a modelagem de sistemas complexos, estas diferentes Redes de Petri não possuem um poder de expressividade maior do que as Redes *Place/Transition*, pois as Redes Coloridas são passíveis de serem transformadas em Redes *Place/Transition* e vice-versa [42]. Da mesma forma, as Redes por Objeto podem ser abordadas apenas como um aprimoramento das técnicas de abstração utilizadas nas Redes Coloridas através do paradigma de orientação por

[1] http://spinroot.com/spin/Man/ltl.html

[2] Uma classificação dos tipos diferentes de Redes de Petri pode ser consultada no seguinte endereço mantido pela comunidade: http://www.informatik.uni-hamburg.de/TGI/PetriNets/classification/.

objetos [49].

Nas seções seguintes, precedidas de uma breve introdução a multiconjuntos, estas redes serão apresentadas com maiores detalhes de forma a desnudar o paradigma de *tokens* como Redes de Petri, onde mais adiante tomaremos alguns conceitos destas definições para introduzir as redes que são o objetivo do presente trabalho.

2.3.1 Multiconjuntos

Algumas definições utilizadas neste trabalho são construídas em cima do conceito de multiconjuntos, assim, convém apresentarmos uma breve introdução ao tema através de algumas definições.

De maneira informal, um multiconjunto consiste de um conjunto de elementos que podem se repetir. Formalmente, podemos definir um multiconjunto da seguinte forma.

Definição 2.3.1. *Um* **multiconjunto** M *sobre um conjunto não vazio* A *é uma função* $M\colon A \to \mathbb{N}$ *em que* $M(a)$ *é o número de vezes que o elemento* $a \in A$ *aparece em* M*. Representamos um multiconjunto como a soma simbólica de seus componentes:*

$$\sum_{a \in A} M(a)'a\,,$$

onde a notação $K'e$ *denota um multiconjunto no qual o elemento* e *possui multiplicidade* K*.*

Note que as marcações sobre as redes *place/transition* podem ser vistas como multiconjuntos.

Denotaremos como $Bag(A)$ o conjunto de todos os multiconjuntos sobre A. Em vários momentos escreveremos A^* para denotar um multiconjunto sobre um conjunto A. A Definição 2.3.2 apresenta o funcionamento de algumas operações básicas sobre multiconjuntos.

Definição 2.3.2. *Dados dois multiconjuntos* M_1 *e* M_2 *sobre um mesmo conjunto* A*, as operações de união, diferença, comparação e tamanho são definidas da seguinte forma:*

- União*:* $M_1 + M_2 = \sum_{a \in A}(M_1(a) + M_2(a))'a$*;*
- Diferença*:* $(M_1 - M_2)(a) = \max\{M_1(a) - M_2(a), 0\}$*;*
- Comparação*:* $M_1 \neq M_2$ *se existe* $a \in A$ *tal que* $M_1(a) \neq M_2(a)$*;* $M_1 \leq M_2$ *se para todo* $a \in A$ *vale que* $M_1(a) \leq M_2(a)$*;*
- Tamanho*:* $|M_1| = \sum_{a \in A} M_1(a)$*.*

2.4 Redes de Petri Coloridas

Os *tokens* em uma rede de Petri não possuem estrutura ou informação. Assim, eles são chamados de *tokens* incolores e são usualmente representados como pontos pretos (*black dots*). Devido ao fato de que estes são indiferenciáveis, um conjunto de pontos pretos é geralmente substituído por seu tamanho. Dessa forma, para uma rede (P, T, A, w, I), a função w é definida de A para $\mathbb{N}$ e uma marcação é definida de P para $\mathbb{N}$.

Em uma **Rede de Petri Colorida** cada lugar tem um tipo e pode armazenar *tokens* com valores ou cores diferentes e cada transição possui uma *guarda*. Cada arco é etiquetado por uma

expressão que, quando avaliada, possui o mesmo tipo do lugar do seu arco. De forma simplificada, consideramos uma transição habilitada a disparo se avaliarmos as expressões dos arcos que entram na mesma e tivermos *tokens* suficientes para isso. Para avaliar as expressões dos arcos, estas precisam ter valores atribuídos às variáveis e esses valores são referentes às guardas das transições.

Para definir formalmente uma Rede de Petri Colorida, precisamos dos seguintes conceitos, que são utilizados da mesma forma em linguagens funcionais [42, 49]:

- um tipo T é um conjunto de valores;
- o tipo de uma variável v é denotado por $Type(v)$. Se V é um conjunto de variáveis, a notação $Type(V)$ é definida como $\{Type(v) \colon v \in V\}$;
- o conjunto de expressões é denotado por $Expr$;
- o tipo de uma expressão e é denotado por $Type(e)$;
- o conjunto de variáveis em uma expressão e é denotado por $Var(e)$;
- uma atribuição b (*binding*) a um conjunto V de variáveis é a associação de um valor $b(v) \in Type(v)$ para cada $v \in V$;
- o valor de uma expressão e sob uma atribuição b é denotado por $e\langle b \rangle$ e é obtido pela substituição de cada $v \in Var(e)$ por $b(v)$.

As expressões que usaremos poderão ser de dois tipos quando avaliadas: *bool* (o tipo que contém dois valores, verdadeiro ou falso) ou um multiconjunto sobre um tipo. A definição a seguir formaliza a estrutura das redes coloridas.

Definição 2.4.1. *Uma* **Rede de Petri Colorida** *é uma 9-upla* $CPN = (\Sigma, P, V, T, A, \tau, G, E, I)$ *em que:*

- Σ *é um conjunto finito de tipos não vazios, que são também chamados de* ***cores****;*
- P *é um conjunto finito de* ***lugares****;*
- V *é um conjunto finito de* ***variáveis****;*
- T *é um conjunto finito de* ***transições****, com* $P \cap T = \emptyset$*;*
- $A \subseteq (P \times T) \cup (T \times P)$ *é um conjunto finito de* ***arcos****, com* $A \cap (P \cup T) = \emptyset$*. Para todo* $a \in A$*, dizemos que* $p(a)$ *é o lugar do arco* a*;*
- $\tau \colon P \to \Sigma$ *é uma* ***função de cor****, que mapeia cada lugar* p *a uma cor, ou tipo;*
- $G \colon T \to Expr$ *é uma* ***função de expressões de guarda****, definida de forma que para toda* $t \in T$*, a expressão tenha tipo booleano, isto é,* $Type(G(t)) = bool$*, e cada variável tenha tipo em* Σ*, isto é,* $Type(Var(G(t))) \subseteq \Sigma$*;*
- $E \colon A \to Expr$ *é uma* ***função de expressão de arco****, definida de forma que para todo* $a \in A$*, a expressão tenha tipo multiconjunto sobre a cor/tipo* $\tau(p(a))$*, isto é,* $Type(E(a)) = \tau^*(p(a))$*, e as variáveis das expressões tenham tipo em* Σ*, isto é,* $Type(Var(E(a))) \subseteq \Sigma$*;*

- $I\colon P \to Expr$ *é uma* ***função de expressão de inicialização****, definida de forma que, para todo* $p \in P$*,* $I(p)$ *é uma expressão fechada (independente de variável) e* $Type(I(p)) = \tau^*(p)$ *para todo* $p \in P$*.*

Com relação ao comportamento das redes coloridas, precisamos ainda definir o que são *tokens* e marcações nessas redes. No que segue, considere uma rede colorida $CPN = (\Sigma, P, T, A, \tau, G, E, I)$.

Um ***token*** em CPN é um par (p, c) com $p \in P$ e $c \in \tau(p)$. Qualquer multiconjunto sobre o conjunto de todos os *tokens* é chamado de **marcação**. A marcação inicial M_0 é a marcação obtida através da avaliação da expressão de inicialização, de forma que $M_0(p) = I(p)$ para todo $p \in P$.

Uma atribuição a uma transição t é uma função b que atribui valores às variáveis de $Var(G(t))$ de tal forma que $G(t)\langle b\rangle$ é verdadeiro. Um **elemento de atribuição** em CPN é um par (t, b) com $t \in T$ e b sendo uma atribuição a t. Qualquer multiconjunto sobre o conjunto de todos os elementos de atribuição é chamado de **passo**.

Finalmente, dizemos que um passo Y está **habilitado em uma marcação** M se e somente se

$$\sum_{(t,b)\in Y} E(p,t)\langle b\rangle \leq M(p) \qquad \text{para todo } p \in P\,,$$

isto é, existe uma quantidade suficiente de *tokens* em p quando comparados aos *tokens* exigidos pelas expressões dos arcos que saem de p para transições t que fazem parte dos elementos de atribuição do passo. As transições envolvidas em Y são também ditas habilitadas ou aptas para disparar. Quando estas transições disparam, o passo Y habilitado em M **ocorre** e a marcação M é levada até uma nova marcação M' definida como:

$$M'(p) = M(p) - \sum_{(t,b)\in Y} E(p,t)\langle b\rangle + \sum_{(t,b)\in Y} E(t,p)\langle b\rangle \qquad \text{para todo } p \in P\,,$$

isto é, M' é a marcação obtida através da operação de remoção dos *tokens* resultantes da função de expressão de todos os arcos incidentes nas transições habilitadas, e da adição dos *tokens* resultantes da avaliação da função de expressão de todos os arcos que saem.

O Exemplo 2.4.1, referente à Figura 2.5, exemplifica o funcionamento de uma Rede de Petri Colorida.

Exemplo 2.4.1. *A Figura 2.5 mostra um exemplo de uma rede colorida* $(\Sigma, P, V, T, A, \tau, G, E, I)$ *onde as expressões nos arcos* $(P1, T1)$*, que requer dois* tokens *de tipos diferentes, e* $(P2, T1)$*, que requer dois* tokens *do mesmo tipo, são satisfeitas pelas marcações presentes nos lugares* $P1$ *e* $P2$ *e, com isso, a transição* $T1$ *está habilitada. Ao ser disparada, a transição* $T1$ *leva a marcação inicial* M_0 *para uma nova marcação* M' *de acordo com as expressões nos arcos de saída* $(T1, P3)$ *e* $(T1, P4)$*, alterando o estado da rede.*

Aqui, assumimos que o conjunto de tipos Σ *contém apenas um tipo, a saber, o tipo "Cor", composto pelas cores* ***vermelho*** *(*red*),* ***verde*** *(*green*) e* ***azul*** *(*blue*), denotadas pelas letras* **R**, **G** *e* **B***, respectivamente. Também assumimos que todos os lugares e variáveis desta rede são deste tipo, ou seja* $\forall p \in P,\ Type(p) = Cor$ *e* $\forall V \in \{x, y, z\},\ Type(V) = Cor$*. A transição* $T2$*, por sua vez, possui uma etiqueta com uma função de guarda* $G(T2) = x\ != \mathbf{R}$*, que restringe sua habilitação pela condição de que a variável* x *tenha atribuído a si um valor diferente de um* token *do tipo vermelho, ou* **R**.

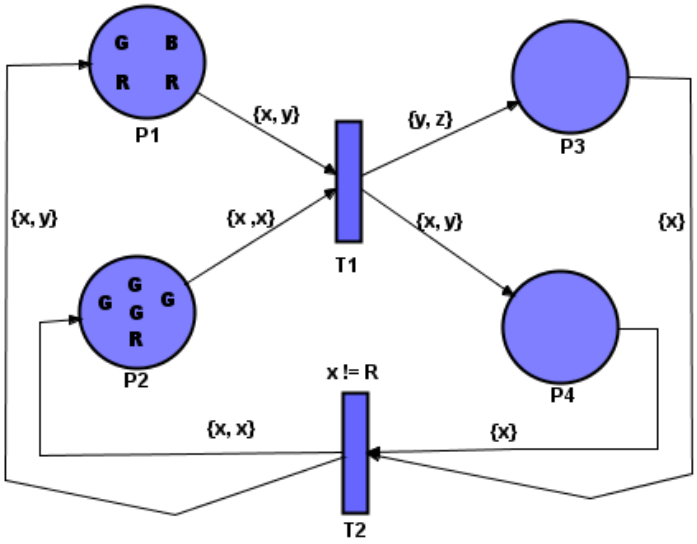

Figura 2.5: *Exemplo de uma rede colorida* $(\Sigma, P, T, A, \tau, G, E, I)$*. Neste exemplo, as expressões nos arcos* $(P1, T1)$ *e* $(P2, T1)$ *satisfazem as condições necessárias e a transição* $T1$ *está habilitada para disparar, levando a marcação inicial* M_0 *para uma nova marcação* M'*, alterando, assim, o estado da rede.*

Note que a marcação inicial desta rede é $\{(P1, R), (P1, R), (P1, G), (P1, B), (P2, R), (P2, G), (P2, G), (P2, G), (P2, G)\}$*. Neste caso, a transição* $T1$ *está habilitada pois existe uma quantidade suficiente de* tokens *nos lugares dos arcos* $(P1, T1)$ *e* $(P2, T1)$ *de forma a satisfazer as expressões destes arcos com base em alguma atribuição para as variáveis destas expressões e, portanto,* $T1$ *pode disparar uma vez. Por causa da marcação de* $P2$ *e da inscrição no arco de entrada* $(P2, T1)$*, a variável* x *deve ter atribuída a si o valor* $\mathbf{G}$*. Contudo, o disparo pode produzir uma dentre seis diferentes marcações, pois* y *pode estar atribuído tanto com* $\mathbf{R}$ *quanto com* $\mathbf{B}$ *e à* z *pode ser atribuída qualquer outra cor. Assim, após disparar* $T1$ *utilizando a atribuição* $b = \{x = \mathbf{G}, y = \mathbf{R}, z = \mathbf{B}\}$*, a nova marcação da rede será* $\{(P1, R), (P1, B), (P2, R), (P2, G), (P2, G), (P3, R), (P3, B), (P4, G), (P4, R)\}$*. Independentemente da atribuição, o* token *verde é removido de* $P1$*. Agora,* $T1$ *não está mais habilitada para disparar, mas* $T2$ *está, pois existe uma quantidade suficiente de* tokens *nos lugares dos arcos* $(P3, T2)$ *e* $(P4, T2)$ *que satisfazem as expressões destes arcos com base em alguma atribuição para as variáveis destas, de forma a obedecer a restrição imposta pela função de guarda de* $T2$*.*

2.5 Redes de Petri por Objeto

Nas Redes de Petri Coloridas, os *tokens* passaram a carregar mais informação, através de sua representação por meio de tipos, do que simplesmente representar estados nos fluxos de controle ou recursos, como os *tokens* indistinguíveis das redes *Place/Transition*. Nas **Redes de Petri por Objeto** os *tokens* passam a carregar informações de seu comportamento, analogamente ao paradigma de linguagens de programação orientadas a objeto. Assim, geralmente os *tokens* nestas redes representam outras Redes de Petri. Muitas características do paradigma de orientação a objetos foram incorporadas às Redes de Petri, sendo inicialmente os conceitos de instanciação de classes

e encapsulamento e, posteriormente, os conceitos de herança e polimorfismo. A evolução deste paradigma a partir das Redes Coloridas Modulares, nas quais Redes de Petri Coloridas poderiam ser organizadas e referenciadas em páginas, e como estas se relacionam pode ser observada nos trabalhos de Lakos [49, 50].

No presente trabalho iremos apresentar a definição de Redes de Petri por Objeto proposta por Valk [79], que tem uma abordagem que será utilizada e referenciada mais adiante na apresentação das Redes Aninhadas utilizadas neste trabalho e das ferramentas utilizadas para simular seu comportamento.

Valk [79] na verdade apresentou um tipo de redes por objeto mais restritas, chamadas *Sistemas Objetos Elementares* (*Elementary Object Systems*). Estes são formados por uma rede sistema (*System Net*) cujos lugares podem ser de dois tipos: os que contêm *tokens* pretos e os que contêm *tokens* redes, ou *redes-objeto* (*Object Nets*). As redes-objeto, nesses sistemas elementares, não podem, por sua vez, conter redes também.

Como qualquer *token* em uma rede, as redes-objeto podem se mover pela rede sistema. Como elas próprias são redes, existem dois tipos de semântica de comportamento: por *referência*, quando a aparição de uma rede-objeto em um lugar é uma referência a uma rede-objeto específica, e por *valor*, quando essas aparições são cópias independentes.

Além de serem movidas pela rede sistema, as redes-objetos também podem se comportar de forma autônoma ou então ter transições sincronizadas com transições das redes sistema. Essa última situação é chamada de **interação**.

A definição a seguir formaliza a estrutura dos Sistemas Objetos Elementares.

Definição 2.5.1. *Um* **Sistema Objeto Elementar** *é uma 4-upla* $OS = (SN, \mathcal{ON}^0_M, \varrho, R_0)$ *onde:*

- $SN = (\hat{P}, \tau, \hat{T}, \hat{W})$ *é uma rede chamada* ***Rede Sistema****, para a qual:*
  - $\hat{P}$ *é um conjunto de* ***lugares****;*
  - $\tau\colon \hat{P} \to \{ob, bt\}$ *é uma* ***função de tipo*** *para os lugares que determina se um lugar comporta* tokens *do tipo objeto (ob) ou* tokens *pretos (bt). Assim, o conjunto de lugares* $\hat{P}$ *pode ser dividido em dois subconjuntos, de acordo com seus tipos, e podemos escrever* $\hat{P} = \hat{P}_{ob} \cup \hat{P}_{bt}$*, onde* $\hat{P}_{ob} = \{p \in P\colon \tau(p) = ob\}$ *e* $\hat{P}_{bt} = \{p \in P\colon \tau(p) = bt\}$*;*
  - $\hat{T}$ *é um conjunto de* ***transições****, com* $\hat{P} \cap \hat{T} = \emptyset$*;*
  - $\hat{W}\colon (\hat{P} \times \hat{T}) \cup (\hat{T} \times \hat{P}) \to \mathbb{N}$ *é uma função que define o* ***conjunto de arcos****. Caso* $\hat{W}(x, y) > 0$*, o arco* (x, y) *é chamado* ***arco objeto*** *se* $\{x, y\} \cap \hat{P}_{ob} \neq \emptyset$ *e é chamado* ***arco black-token*** *se* $\{x, y\} \cap \hat{P}_{bt} \neq \emptyset$*. Caso* $\hat{W}(x, y) = 0$*, o arco* (x, y) *não existe.*
- $\mathcal{ON}^0_M = \{(ON_1, M^0_1), \ldots, (ON_k, M^0_k)\}$ *é um conjunto finito de redes Place/Transition marcadas, chamadas* ***redes-objeto****, ou seja,* $ON_i = (P_i, T_i, A_i, w_i, I_i)$ *e* $M^0_i(p) = I(p)$ *para* $p \in P_i$ *e para* $1 \leq i \leq k$*, como na Definição 2.1.1. Definimos* $\mathcal{ON} = \{ON_1, \ldots, ON_k\}$*, isto é, o conjunto formado pelas redes não marcadas;*
- $\varrho \subseteq \hat{T} \times T$ *é a* ***relação de interação*** *entre as transições, onde* $T = \bigcup_{i=1}^{k} T_i$*, que indica o sincronismo entre transições da rede sistema e transições das redes-objeto;*
- $R_0\colon \hat{P} \to \mathbb{N} \cup Bag(\mathcal{ON})$ *especifica a* ***distribuição inicial de tokens*** *e respeita que* $R_0(\hat{p})$ *só é um número se* $\hat{p}$ *for um lugar do tipo* token *preto, isto é,* $R_0(\hat{p}) \in \mathbb{N} \iff \hat{p} \in \hat{P}_{bt}$*.*

Como já mencionado, existem três tipos de ocorrências em uma rede por objetos: interação, transporte e ação autônoma. A **interação** é quando uma transição da rede sistema está relacionada com uma transição de uma rede-objeto e ambas são ativadas. O **transporte** ocorre quando uma rede-objeto é movida de lugar na rede sistema. Uma **ação autônoma** ocorre quando uma rede-objeto modifica seu próprio estado sem se mover na rede sistema. As definições dessas ações dependem de qual semântica está sendo considerada, por **referência** ou por **valor**.

Para definir as regras de ocorrência sob a semântica por referência, é necessário definir o que é uma marcação em um Sistema Objeto Elementar, a fim de desassociar a marcação de uma rede-objeto do *token* objeto que representa sua instância.

Definição 2.5.2. *Seja* $OS = (SN, \mathcal{ON}^0_M, \varrho, R_0)$ *um sistema objeto elementar. Uma* **marcação de** OS **sob a semântica por referência** *é um par* (R, M) *em que:*

- $R\colon \hat{P} \to \mathbb{N} \cup Bag(\mathcal{ON})$ *é uma distribuição de redes-objeto e* tokens *pretos onde* $R(\hat{p}) \in \mathbb{N} \iff \hat{p} \in \hat{P}_{bt}$*;*
- $M = (M_1, \ldots, M_k)$ *é o vetor onde* M_i *é uma marcação da rede objeto* ON_i*, para cada* $1 \leq i \leq k$*.*

Fazendo $M_0 = (M^0_1, \ldots, M^0_k)$*, definimos* (R_0, M_0) *como sendo a* **marcação inicial** *de* OS*.*

Definição 2.5.3. *Seja* (R, M) *uma marcação de um Sistema Objeto* $OS = (SN, \mathcal{ON}^0_M, \varrho, R_0)$*, com* $M = (M_1, \ldots, M_k)$*. As* **regras de ocorrência da perspectiva da semântica por referência** *são definidas da seguinte forma:*

- **Interação***: Seja* $(\hat{t}, t) \in \varrho$*, com* $t \in T_i$ *para alguma* $ON_i = (P_i, T_i, A_i, w_i, I_i)$*. Dizemos que* $(\hat{t}, t)$ *está* ***habilitada em*** (R, M) *se:*

  1. $R(\hat{p}) \geq \hat{W}(\hat{p}, \hat{t})' ON_i$ *para todo* $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{ob}$*;*
  2. $R(\hat{p}) \geq \hat{W}(\hat{p}, \hat{t})$ *para todo* $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{bt}$*;*
  3. *a transição* t *está habilitada pela marcação* M_i*.*

  A regra de interação habilitada, ao ocorrer, irá gerar uma nova marcação (R', M')*, definida por:*

  1. $R'(\hat{p}) = R(\hat{p}) - \hat{W}(\hat{p}, \hat{t})' ON_i + \hat{W}(\hat{t}, \hat{p})' ON_i$ *para todo* $\hat{p} \in \hat{P}_{ob}$*;*
  2. $R'(\hat{p}) = R(\hat{p}) - \hat{W}(\hat{p}, \hat{t}) + \hat{W}(\hat{t}, \hat{p})$ *para todo* $\hat{p} \in \hat{P}_{bt}$*;*
  3. $M' = (M_1, \ldots, M_{i-1}, M'_i, M_{i+1}, \ldots, M_k)$*, em que* M'_i *é a marcação sucessora de* M_i *em* ON_i*.*

  Isto é, quando duas transições com a mesma etiqueta, uma na rede sistema e outra em alguma rede-objeto, tiverem as condições de seus respectivos arcos incidentes satisfeitas pela quantidade de tokens *nos lugares dos arcos e ao menos um* token *objeto que referencie a rede objeto estiver em um dos lugares do arco incidente da transição da rede sistema, estarão ambas habilitadas e poderão ocorrer simultaneamente e gerar uma nova marcação. A nova marcação será produto da distribuição de* tokens *por meio da função dos arcos de saída das transições.*

- **Transporte**: *Seja $\hat{t} \in \hat{T}$ uma transição de SN tal que $\hat{t} \notin dom(\varrho)$, isto é, ela não está envolvida em uma relação com transições das redes-objeto. Dizemos que $\hat{t}$ está* ***habilitada em*** *(R, M) se existir uma rede-objeto ON_i tal que:*

  1. *$R(\hat{p}) \geq \hat{W}(\hat{p}, \hat{t})'ON_i$ para todo $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{ob}$;*
  2. *$R(\hat{p}) \geq \hat{W}(\hat{p}, \hat{t})$ para todo $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{bt}$.*

  A transição habilitada, ao ocorrer, irá gerar uma nova marcação (R', M') de OS definida por:

  1. *$R'(\hat{p}) = R(\hat{p}) - \hat{W}(\hat{p}, \hat{t})'ON_i + \hat{W}(\hat{t}, \hat{p})'ON_i$ para todo $\hat{p} \in \hat{P}_{ob}$;*
  2. *$R'(\hat{p}) = R(\hat{p}) - \hat{W}(\hat{p}, \hat{t}) + \hat{W}(\hat{t}, \hat{p})$ para todo $\hat{p} \in \hat{P}_{bt}$;*
  3. *$M' = M$.*

  Isto é, quando uma transição da rede sistema tiver as condições de seus arcos incidentes satisfeitas pela quantidade de tokens *nos lugares dos arcos, tal transição estará habilitada e poderá ocorrer, gerando uma nova marcação. Nesta nova marcação, os* tokens *objeto envolvidos no disparo serão transportados para os lugares de saída da transição com seus estados preservados.*

- **Ação Autônoma**: *Seja $t \in T_i$ uma transição de uma rede-objeto $ON_i = (P_i, T_i, A_i, w_i, I_i) \in R(\hat{p})$ para algum $\hat{p} \in \hat{P}_{ob}$ tal que $t \notin img(\varrho)$. Se t está habilitada pela marcação M_i em ON_i, então dizemos que t está* ***habilitada em*** *(R, M). Se essa transição ocorrer em ON_i, irá gerar uma nova marcação (R', M') de OS definida por:*

  1. *$R' = R$;*
  2. *$M' = (M_1, \ldots, M_{i-1}, M_i', M_{i+1}, \ldots, M_k)$, em que M_i' é a marcação sucessora de M_i em ON_i.*

  Isto é, quando uma transição de qualquer rede-objeto tiver as condições de seus arcos incidentes satisfeitas pela quantidade de tokens *nos lugares dos arcos incidentes, tal transição estará habilitada e poderá ocorrer, gerando uma nova marcação daquela rede-objeto. Nesta nova marcação, os* tokens *envolvidos no disparo serão consumidos e redistribuídos nos lugares de saída de acordo com as funções dos arcos de saída desta transição de maneira que o estado da rede sistema permanecerá o mesmo, exceto pela marcação da rede-objeto em questão.*

O Exemplo 2.5.1, referente à Figura 2.6, exemplifica o funcionamento de um Sistema Objeto Elementar com uma semântica por referência.

Exemplo 2.5.1. *A Figura 2.6 mostra um Sistema Objeto Elementar que emula o fluxo de trabalho de um sistema de entregas onde um* drone*, representado pela rede objeto D, funciona como um* token *na rede sistema que orquestra as entregas a partir de um conjunto de pedidos representados por* tokens *pretos genéricos no lugar ORDERS.*

Formalmente, temos o sistema $(SN, \mathcal{ON}_M^0, \varrho, R_0)$, onde a rede sistema SN tem como conjunto de lugares $\{ORDERS, SENT, WAITING, STRESS, MAINTENANCE\}$, com os tipos dados por $\tau(ORDERS) = bt$, $\tau(SENT) = ob$, $\tau(WAITING) = ob$, $\tau(STRESS) = bt$, $\tau(MAINTENANCE) = ob$, tem como conjunto de transições $\{T0, T1, T2, T3, T4\}$, em que $T1$ e $T2$ são etiquetadas por <TAKE ORDER> e <DELIVER>. O conjunto de arcos pode ser observado na própria figura. A

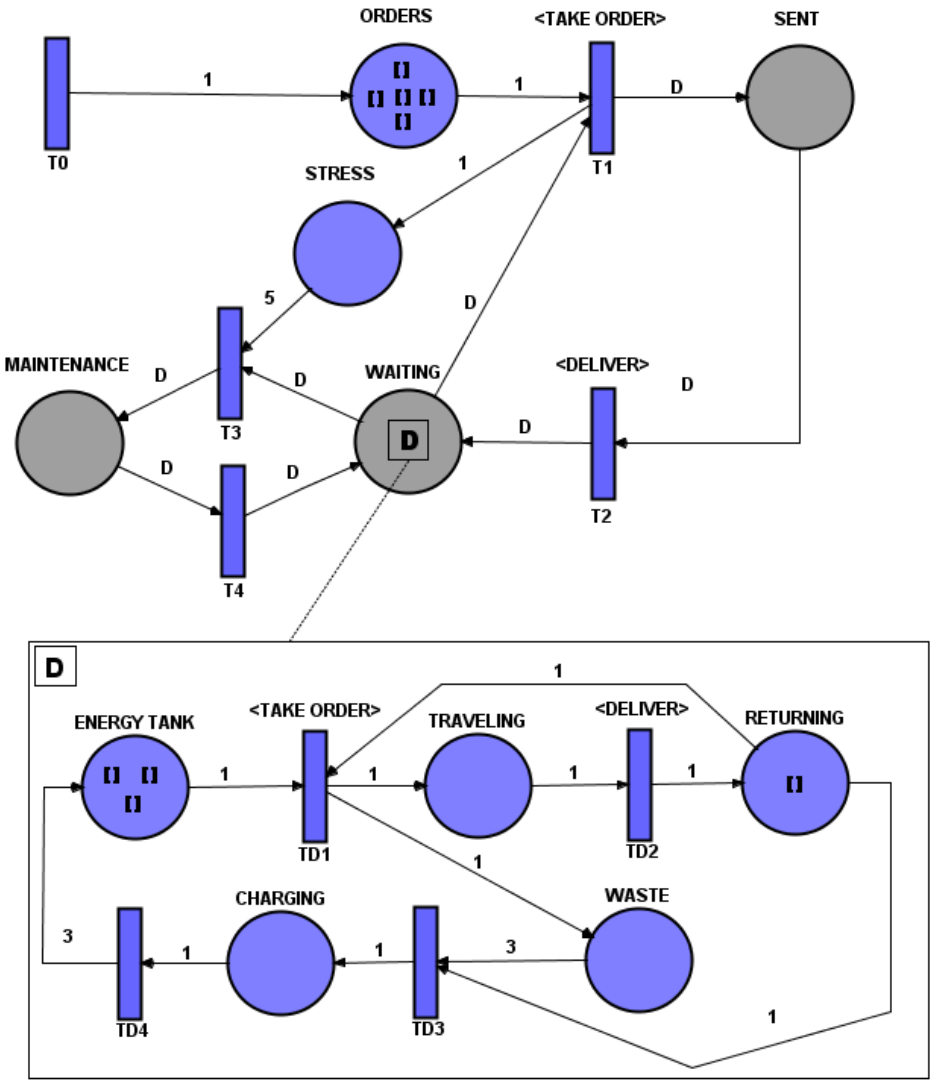

Figura 2.6: *Exemplo de um Sistema Objeto Elementar que emula o fluxo de trabalho de um sistema de entregas onde um* drone*, representado pela rede objeto D, funciona como um* token *na rede sistema que orquestra as entregas a partir de um conjunto de pedidos representados por* tokens *pretos genéricos no lugar ORDERS.*

marcação inicial é dada por $R_0 = \{(ORDERS, 5), (SENT, \emptyset), (STRESS, 0), (MAINTENANCE, \emptyset), (WAITING, \{D\})\}$. *A rede objeto D é uma rede* Place/Transition *com cinco lugares, cujo conjunto é* $\{ENERGY\ TANK, TRAVELING, RETURNING, WASTE, CHARGING\}$, *quatro transições,* $\{TD1, TD2, TD3, TD4\}$, *em que* $TD1$ *e* $TD2$ *são etiquetadas por <TAKE ORDER> e <DELIVER>, e cuja marcação inicial é* $\{(ENERGY\ TANK, 3), (TRAVELING, 0), (WASTE, 0), (RETURNING, 1), (CHARGING, 0)\}$. *A relação* ϱ *entre as transições está indicada pelas etiquetas, de forma que a etiqueta <TAKE ORDER> denota a relação entre as transições* $T1$ *e* $TD1$ *e a etiqueta <DELIVER> denota a relação entre as transições* $T2$ *e* $TD2$.

Inicialmente, o drone *representado pela rede objeto D está aguardando na transição <WAITING> e cinco pedidos estão acumulados no lugar ORDERS, de forma que a transição* $T1$ *e a transição* $TD1$ *estão habilitadas e sincronizadas pela etiqueta <TAKE ORDER>. Ao disparar, a transição* $T1$ transporta *o* token D *para o lugar SENT e, ao mesmo tempo, a transição* $TD1$ *também dispara, criando um* token *preto no lugar TRAVELING e removendo um* token *preto do lugar RETURNING. As novas marcações das redes após o disparo são* $\{(ORDERS, 4), (SENT, D), (STRESS, 1), (MAINTENANCE, \emptyset), (WAITING, \emptyset)\}$ *para* SN *e* $\{(ENERGY\ TANK, 2), (TRAVELING, 1), (WASTE, 1), (RETURNING, 0), (CHARGING, 0)\}$ *para* D. *Nelas, as transições* $T2$ *e* $TD2$ *passam a estar habilitadas e, caso disparem, por fazerem parte da relação <DELIVER>, irão disparar de maneira sincronizada alterando novamente a marcação das redes para* $\{(ORDERS, 4), (SENT, \emptyset), (STRESS, 1), (MAINTENANCE, \emptyset), (WAITING, D)\}$ *em* SN *e* $\{(ENERGY\ TANK, 2), (TRAVELING, 0), (WASTE, 1), (RETURNING, 1), (CHARGING, 0)\}$ *em* D.

As regras definidas sob o ponto de vista da semântica por referência costumam não ser adequadas para expressar mobilidade em sistemas distribuídos. Por isso a semântica por valor é necessária. Antes de definirmos as três regras de ocorrência sob a semântica por valor, é necessário definir o que é uma marcação considerando também esta perspectiva.

Definição 2.5.4. *Seja* $OS = (SN, \mathcal{ON}_M^0, \varrho, R_0)$ *um sistema objeto elementar, e sejam os conjuntos* $\mathcal{M} = \{(M_1, \ldots, M_k) \colon M_i \in Bag(P_i)\}$ *e* $\mathcal{ON}_M = \{(ON_1, M_1), \ldots, (ON_k, M_k) \colon (M_1, \ldots, M_k) \in \mathcal{M}\}$. *Veja que* $\mathcal{M}$ *é o conjunto que contém todas as marcações possíveis das k redes-objeto de OS e* $\mathcal{ON}_M$ *é o conjunto que contém pares de rede-objeto e marcação, para todas as marcações possíveis dessas redes.*

Uma **marcação de** OS **sob a semântica por valor** *é uma função* $V \colon \hat{P} \to \mathbb{N} \cup Bag(\mathcal{ON}_M)$ *que respeita* $V(\hat{p}) \in \mathbb{N} \iff \hat{p} \in \hat{P}_{bt}$. *Ou seja, é uma função que mapeia os lugares da rede sistema para um número natural que representa a quantidade de* tokens *pretos caso seja um lugar deste tipo ou um multiconjunto de* tokens *objeto com suas respectivas marcações caso seja um lugar de objeto.*

A **marcação inicial** *de* OS *é denotada* V_0 *e vale que* $V_0(\hat{p})(ON_i, M_i^0) = R_0(\hat{p})(ON_i)$ *para todo* $\hat{p} \in \hat{P}_{ob}$ *e* $V_0(\hat{p}) = R_0(\hat{p})$ *para todo* $\hat{p} \in \hat{P}_{bt}$.

As definições a seguir usam duas funções, `unify` e `distribute`, cujas descrições dependem de qual refinamento da semântica de valor está sendo considerado: **semântica por distribuição de *tokens*** ou **semântica por histórico de processos**. A ideia geral por trás destas funções é definir a regra pela qual uma sequência de disparos em uma rede com um *token* X, que seja ele mesmo uma instância ou uma referência à uma rede R, deve proceder ao encontrar uma transição onde a partir deste um *token* X dois ou mais devem ser produzidos, cada um como uma instância ou referência à mesma rede R, ou, a partir de dois ou mais destes *tokens* X, que representam esta mesma rede,

apenas um deve ser produzido. O grande desafio nestes cenários é manter a consistência do estado da rede-objeto durante as operações realizadas nos *tokens* que a representam.

As regras de ocorrência para semântica por valor estão na Definição 2.5.5 e utilizaremos as funções `unify` e `distribute` da maneira como são representadas em uma semântica por distribuição de *tokens*. Sejam $M_{i1}, \ldots, M_{is}$ marcações de uma rede ON_i de um Sistema Objeto $OS = (SN, \mathcal{ON}^0_M, \varrho, R_0)$. Definimos `unify` como $\texttt{unify}(\{M_{i1}, \ldots, M_{is}\}) = \sum_{j=1}^{s} M_{ij}$, isto é, a marcação resultante será a somatória das s marcações que foram combinadas. E dada uma única marcação M'_i de uma rede ON_i, definimos então `distribute` como a operação que distribui as marcações em s outros *tokens* de rede de maneira arbitrária, de forma que $\texttt{distribute}(M'_i) = \{(M_{i1}, \ldots, M_{is}) \colon \sum_{j=1}^{s} M_{ij} = M'_i\}$. Resumindo, `distribute` atua de forma a distribuir a marcação nas diferentes instâncias de uma mesma rede objeto, enquanto `unify` atua de forma a juntar a marcação das diferentes instâncias de uma determinada rede-objeto em uma nova e única instância.

Estas regras fundamentam-se na preocupação em manter a consistência do estado das redes-objeto, de forma que as instâncias geradas ao longo do processo da rede sistema não são novas redes *per se*, mas representações de parte do estado de uma única rede-objeto modelo.

Definição 2.5.5. *Seja V uma marcação de um Sistema Objeto $OS = (SN, \mathcal{ON}^0_M, \varrho, R_0)$. As* **regras de ocorrência da perspectiva da semântica por valor** *são definidas da seguinte forma:*

- **Interação***: Seja $(\hat{t}, t) \in \varrho$, com $t \in T_i$ para alguma $ON_i = (P_i, T_i, A_i, w_i, I_i)$. Dizemos que $(\hat{t}, t)$ está* ***habilitada em*** *V se para cada lugar $\hat{p} \in {}^\bullet\hat{t}$:*

  1. *existir um submulticonjunto $V_{\hat{p}} \subseteq V(\hat{p})$ tal que: $V_{\hat{p}} = \hat{W}(\hat{p}, \hat{t})' ON_i$ para todo $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{ob}$;*
  2. *$V(\hat{p}) \geq \hat{W}(\hat{p}, \hat{t})$ para todo $\hat{p} \in {}^\bullet\hat{t} \cup \hat{P}_{bt}$;*
  3. *t está habilitada em $M = \texttt{unify}(\{M'_i \colon (ON_i, M'_i) \in V_{\hat{p}}$ e $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{ob}\})$.*

  A regra de interação habilitada, ao ocorrer, irá gerar uma nova marcação V' definida por:

  1. *$V'(\hat{p}) = V(\hat{p}) - V_{\hat{p}} + \hat{W}(\hat{t}, \hat{p})'(ON_i, M_{\hat{p}})$ para todo $\hat{p} \in \hat{P}_{ob}$, onde $M_{\hat{p}}$ faz parte de $(M_{\hat{p}_1}, \ldots, M_{\hat{p}_q}) \in \texttt{distribute}(M'_i)$, $\{\hat{p}_1, \ldots, \hat{p}_q\} = \hat{t}^\bullet$, e M'_i é sucessora de M_i em ON_i.*
  2. *$V'(\hat{p}) = V(\hat{p}) - \hat{W}(\hat{p}, \hat{t}) + \hat{W}(\hat{t}, \hat{p})$ para todo $\hat{p} \in \hat{P}_{bt}$.*

  Ou seja, quando duas transições com a mesma etiqueta, uma na rede sistema e outra em alguma rede-objeto, tiverem as condições de seus respectivos arcos incidentes satisfeitas pela quantidade de tokens *nos lugares dos arcos (no caso das redes-objeto, será contabilizado o resultado da função* ***`unify`****), e ao menos um* token *objeto que instancie a rede-objeto estiver em um dos lugares do arco incidente da transição da rede sistema, estarão ambas habilitadas e poderão ocorrer simultaneamente e gerar uma nova marcação. A nova marcação será produto da distribuição de* tokens*, conforme a função* ***`distribute`****, por meio da função dos arcos de saída das transições a marcação das redes objeto serão distribuídas.*

- **Transporte***: Seja $\hat{t} \in \hat{T}$ uma transição de SN tal que $\hat{t} \notin dom(\varrho)$, isto é, ela não está envolvida em uma relação com transições das redes-objeto. Dizemos que tal transição está* ***habilitada em*** *V se existir uma rede-objeto ON_i tal que para cada lugar $\hat{p} \in {}^\bullet\hat{t}$:*

  1. *existe um submulticonjunto $V_{\hat{p}} \subseteq V(\hat{p})$ tal que: $V_{\hat{p}} = \hat{W}(\hat{p}, \hat{t})' ON_i$ para todo $\hat{p} \in {}^\bullet\hat{t} \cap \hat{P}_{ob}$;*

2. $V(\hat{p}) \geq \hat{W}(\hat{p}, \hat{t})$ *para todo* $\hat{p} \in {}^\bullet\hat{t} \cup \hat{P}_{bt}$;

A transição habilitada, ao ocorrer, irá gerar uma nova marcação V' *de* OS *definida por:*

1. $V'(\hat{p}) = V(\hat{p}) - V_{\hat{p}} + \hat{W}(\hat{t}, \hat{p})'(ON_i, M_{\hat{p}})$ *para todo* $\hat{p} \in \hat{P}_{ob}$*, onde* $M_{\hat{p}}$ *faz parte de* $(M_{\hat{p}_1}, \ldots, M_{\hat{p}_q}) \in \texttt{distribute}(M_i)$ *e* $\hat{p}_1, \ldots, \hat{p}_q = \hat{t}^\bullet$.
2. $V'(\hat{p}) = V(\hat{p}) - \hat{W}(\hat{p}, \hat{t}) + \hat{W}(\hat{t}, \hat{p})$ *para todo* $\hat{p} \in \hat{P}_{bt}$.

Isto é, quando uma transição da rede sistema tiver as condições de seus arcos incidentes satisfeitas pela quantidade de tokens *nos lugares dos arcos, tal transição estará habilitada e poderá ocorrer, gerando uma nova marcação. Nesta nova marcação, os* tokens *objeto envolvidos no disparo serão transportados para os lugares de saída da transição de forma que o estado das redes-objeto não sofrerá alterações e a marcação será distribuída de acordo com a função* `distribute` *conforme quantas forem as instâncias geradas, mantendo assim o estado consistente.*

- **Ação Autônoma***: Seja* $t \in T_i$ *uma transição de uma rede-objeto* ON_i *tal que* $(ON_i, M_i) \in V(\hat{p})$ *para algum* $\hat{p} \in \hat{P}$ *e* $t \notin img(\varrho)$*. Se* t *está habilitada pela marcação* M_i *em* ON_i*, então dizemos que* t *está* habilitada em V*. Se essa transição ocorrer em* ON_i*, irá gerar uma nova marcação* V' *de* OS *definida por:*

  1. $V'(\hat{p}) = V(\hat{p}) - (ON_i, M_i) + (ON_i, M_i')$ *onde* M_i' *é sucessora de* M_i *em* ON_i.

  Isto é, quando uma transição de qualquer rede-objeto tiver as condições de seus arcos incidentes satisfeitas pela quantidade de tokens *nos lugares dos arcos incidentes, tal transição estará habilitada e poderá ocorrer, gerando uma nova marcação na rede-objeto. Por isso, as redes-objeto envolvidas são apenas atualizadas na nova marcação da rede sistema.*

2.6 Redes de Petri Aninhadas

Uma outra abordagem do paradigma de *tokens* como redes, diferente das redes por objeto, são as **Redes de Petri Aninhadas** propostas por Lomazova [58]. Nesta abordagem, assim como nas redes por objeto de Valk, o sistema é definido através de uma Rede Sistema, um conjunto de redes-objeto que funcionam como os *tokens* da Rede Sistema e uma função que relaciona as transições destes dois tipos de rede. Novamente, as regras de ocorrência se dividem em regras de transporte, interação e ação autônoma, entretanto a semântica destas se difere daquelas apresentadas nas Definições 2.5.3 e 2.5.5 de maneira sutil e o conceito de lugares compartilhados é introduzido na definição. A seguir apresentaremos uma definição das Redes Aninhadas proposta por Venero e Corrêa da Silva [83] que se utiliza de alguns conceitos de Redes Coloridas apresentados anteriormente.

Definição 2.6.1. *Uma* **Rede de Petri Aninhada** *é formalmente definida como uma 4-upla* $N = (\Sigma, P_s, L, (EN_0, EN_1, \ldots, EN_n))$ *onde:*

1. Σ *é um conjunto finito de* ***tipos básicos****;*
2. P_s *é um conjunto finito de* ***lugares compartilhados****;*
3. L *é um* ***conjunto de rótulos****, que é a união dos conjuntos disjuntos de rótulos* L_h*, usados para sincronização horizontal, e* L_v*, usados para sincronização vertical;*

4. $(EN_0, EN_1, \ldots, EN_n)$ *é uma sequência de redes coloridas tais que* EN_0 *é a* ***Rede Sistema*** *e* EN_i*, para* $1 \leq i \leq n$*, são* ***Redes Elementos****. Para* $0 \leq i \leq n$*, portanto, temos que* $EN_i = (\Sigma, P_i, V_i, T_i, A_i, \tau_i, G_i, E_i, I_i, \Lambda_i)$*, onde* P_i*,* V_i*,* T_i*,* A_i*,* G_i*, e* I_i *são como na Definição 2.4.1 e os seguintes termos e restrições são adicionados ou redefinidos:*

   (a) $\tau_i \colon P_i \to \Sigma \cup \mathcal{P}(\{EN_1, \ldots, EN_n\})$ *é uma* ***função de tipo*** *que indica quais tipos de* tokens *cada lugar aceita;*

   (b) $\Lambda_i \colon T_i \to L$ *é uma* ***função parcial de atribuição de rótulos a transições****;*

   (c) $E_i \colon A_i \to Expr(V_i)$ *é uma* ***função de expressão de arco****, definida de forma que* $Type(E(a)) = \tau_i^*(p(a))$ *(a expressão tem tipo multiconjunto sobre a cor/tipo* $\tau_i(p(a))$*) e* $Type(Var(E(a))) \subseteq \Sigma \cup \mathcal{P}(\{EN_1, \ldots, EN_n\})$ *(as variáveis têm tipos em* Σ *ou pertencem a algum tipo de rede elemento) para todo* $a \in A_i$*. Para manter a consistência das redes, seguindo a definição original de Lomazova [60], algumas restrições sobre as funções de expressão dos arcos devem ser adicionadas. São elas:*

      i. *Não podem existir constantes de redes nas expressões de arcos que entram em transições. Ou seja, apenas variáveis podem ser utilizadas nas expressões dos arcos, e nenhum multiconjunto de redes pode ser utilizado nestas expressões. A razão desta restrição será apresentada mais adiante;*

      ii. *Expressões de arcos que entram em transições não contêm mais do que uma ocorrência de cada variável;*

      iii. *Expressões de dois arcos diferentes que entram em uma mesma transição não podem compartilhar variáveis;*

      iv. *Cada variável das expressões dos arcos de saída de uma transição deve ter uma ocorrência em alguma das expressões dos arcos de entrada dessa mesma transição.*

Seguindo a definição apresentada por Venero e Corrêa da Silva [83], podemos definir uma **marcação** nas redes aninhadas da seguinte forma.

Definição 2.6.2. *Dada uma rede aninhada* $N = (\Sigma, P_s, L, (EN_0, EN_1, \ldots, EN_n))$*, seja* $EN_i \in \{EN_1, \ldots, EN_n\}$ *uma rede elemento:*

- *Uma função* M_i *que mapeia cada lugar* $p \in P_i$ *para um multiconjunto sobre* Σ *é uma* **marcação de** EN_i*. A tupla* (EN_i, M_i) *representa um* **token *de rede*** *da rede aninhada;*

- *Seja* $\overline{EN}$ *um conjunto contendo todos os* tokens *de rede* (EN_i, M_i)*. Uma função* M_i *que mapeia cada lugar* $p \in P_i$ *para um multiconjunto sobre* $\Sigma \cup \overline{EN}$ *também é uma* **marcação em** EN_i*;*

- *A* **marcação da rede** N *é a marcação de sua rede sistema* EN_0*, dada por uma função* M_0 *que mapeia cada lugar* $p \in P_0$ *para um multiconjunto sobre* $\Sigma \cup \overline{EN}$*;*

- *A seguinte regra deve ser respeitada: para todo* $p \in P_i$ *com* $0 \leq i \leq n$*,* $M_i(p)$ *será um multiconjunto sobre* $\tau_i(p)$*.*

Observe que as redes elementos são consideradas como sendo tipos e só funcionam nas redes aninhadas como *tokens* de rede pois assim carregam junto de si seu estado através de sua marcação.

É devido a isso que constantes de redes foram restringidas como possíveis dentro das expressões de arco na Definição 2.6.1. Vale ainda notar alguns detalhes importantes que diferem as redes aninhadas originais apresentadas por Lomazova [60] das apresentadas por Venero e Corrêa da Silva [83] e por este trabalho. Diferentemente das redes definidas por Lomazova, as redes na Definição 2.6.1 não permitem que qualquer lugar $p \in P_i$ de uma rede elemento possua uma marcação com *tokens* de tipo distinto do tipo $\tau_i(p)$. Esta restrição foi prevista por Lomazova em seu trabalho, porém não foi definida formalmente. Além disso, um conjunto de lugares compartilhados P_s foi adicionado à definição. Estes lugares compartilhados facilitam um controle de estado da rede e são compartilhados por todos os *tokens* de rede.

Exemplo 2.6.1. *Um exemplo de uma rede de acordo com a Definição 2.6.1 pode ser visto na Figura 2.7. Nela, pode-se observar quatro redes que modelam um sistema de entregas semelhante ao da rede da Figura 2.6, porém, mais complexa e com as características definidas para as redes aninhadas.*

O sistema é composto por quatro redes, SN, R, A e D, onde as transições são etiquetadas de forma a permitir o disparo síncrono entre as redes em passos verticais e horizontais através destas transições. É importante notar que neste sistema Σ contém apenas o tipo `[]` e, por isso, no diagrama as inscrições dos arcos de saída e entrada dos lugares deste tipo são números representando a cardinalidade do multiconjunto. Na prática, este tipo é tratado como um token genérico.

A ideia geral é que a rede sistema coordene o fluxo entre os agentes que são modelados através das outras redes, de forma que a rede A é responsável por criar pedidos (ORDERS), a rede D modela o comportamento dos entregadores responsáveis por retirar o pedido e entregá-lo, e a rede R modela um agente reparador, que realiza a manutenção dos agentes A e D quando estes atingem determinado nível de desgaste.

De maneira mais detalhada, os seguintes elementos de acordo com a definição compõem a rede: $\Sigma = \{[\ ]\}$, $P_s = \{\text{ORDERS}\}$, $L = L_v \cup L_h$, com $L_v = \{$<TAKE ORDER>, <BERECHARGED>, <BACK>, <DELIVER>$\}$ e $L_h = \{$<RECHARGE>$\}$, e (EN_0, EN_1, EN_2, EN_3), *onde:*

- $EN_0(SN) = (\Sigma, P_0, V_0, T_0, A_0, \tau_0, G_0, E_0, I_0, \Lambda_0)$ e
  - $P_0 = \{\text{ORDERS, READY, SENT, ENTRANCE, WAITING, MAINTENANCE}\}$;
  - $V_0 = \{x, y\}$;
  - $T_0 = \{\text{T0, T1, T2, T3, T4, T5}\}$;
  - $A_0 = \{(\text{ORDERS, T0}), (\text{T0, READY}), (\text{READY, T1}), (\text{ENTRANCE, T1}), (\text{T1, SENT}), (\text{SENT, T2}), (\text{T2, ENTRANCE}), (\text{ENTRANCE, T3}), (\text{T3, MAINTENANCE}), (\text{WAITING, T3}), (\text{MAINTENANCE, T4}), (\text{T4, ENTRANCE}), (\text{MAINTENANCE, T5}), (\text{T5, WAITING})\}$;
  - $\tau_0(\text{ORDERS}) = [\ ]$, $\tau_0(\text{READY}) = [\ ]$, $\tau_0(\text{SENT}) = R$, $\tau_0(\text{ENTRANCE}) = R$, $\tau_0(\text{WAITING}) = R$, $\tau_0(\text{MAINTENANCE}) = R$;
  - $E_0(\text{ORDERS, T0}) = 1$, $E_0(\text{T0, READY}) = 1$, $E_0(\text{READY, T1}) = 1$, $E_0(\text{ENTRANCE, T1}) = x$, $E_0(\text{T1, SENT}) = x$, $E_0(\text{SENT, T2}) = x$, $E_0(\text{T2, ENTRANCE}) = x$, $E_0(\text{ENTRANCE, T3}) = x$, $E_0(\text{WAITING, T3}) = y$, $E_0(\text{T3, MAINTENANCE}) = (x, y)$, $E_0(\text{MAINTENANCE, T4}) = x$, $E_0(\text{T4, ENTRANCE}) = x$, $E_0(\text{MAINTENANCE, T5}) = y$, $E_0(\text{T5, WAITING}) = y$;

- I_0(ORDERS) = {[ ], [ ], [ ], [ ], [ ]}, I_0(READY) = ∅, I_0(SENT) = ∅, I_0(ENTRANCE) = {A, D, D}, I_0(WAITING) = {R}, I_0(MAINTENANCE) = ∅;
- Λ_0(T1) = <TAKE ORDER>, Λ_0(T2) = <DELIVER>, Λ_0(T3) = <BERECHARGED>, Λ_0(T4) = <BACK>, Λ_0(T5) = <R_BACK>.

- $EN_1(R) = (\Sigma, P_1, V_1, T_1, A_1, \tau_1, G_1, E_1, I_1, \Lambda_1)$ e
  - P_1 = {R_WAITING, WORKING};
  - $V_1 = \emptyset$;
  - T_1 = {TR0, TR1};
  - A_1 ={(R_WAITING, TR0), (TR0, WORKING), (WORKING, TR1), (TR1, R_WAITING)};
  - τ_1(R_WAITING) = [ ], τ_1(WORKING) = [ ];
  - E_1(R_WAITING, TR0) = 1, E_1(TR0, WORKING) = 1, E_1(WORKING, TR1) = 1, E_1(TR1, R_WAITING) = 1;
  - I_1(R_WAITING) = {[ ]}, I_1(WORKING) = ∅;
  - Λ_1(TR0) = <RECHARGE>, Λ_1(TR1) = <R_BACK>.

- $EN_2(A) = (\Sigma, P_2, V_2, T_2, A_2, \tau_2, G_2, E_2, I_2, \Lambda_2)$ e
  - P_2 = {OPEN, BROKE, WASTE, ORDERS, READY};
  - $V_2 = \emptyset$;
  - T_2 = {TA0, TA1, TA2, TA3};
  - A_2 = {(OPEN, TA0), (TA0, ORDERS), (TA0, WASTE), (WASTE, TA1), (TA1, BROKE), (BROKE, TA2), (TA2, A_READY), (A_READY, TA3), (TA3, OPEN)};
  - τ_2(OPEN) = [ ], τ_2(BROKE) = [ ], τ_2(WASTE) = [ ], τ_2(ORDERS) = [ ];
  - E_2(OPEN, TA0) = 1, E_2(TA0, ORDERS) = 1, E_2(TA0, WASTE) = 1, E_2(WASTE, TA1) = 5, E_2(TA1, BROKE) = 1, E_2(BROKE, TA2) = 1, E_2(TA2, A_READY) = 1, E_2(A_READY, TA3) = 1, E_2(TA3, OPEN) = 5;
  - I_2(OPEN) = [ ], I_2(BROKE) = ∅, I_2(WASTE) = ∅, I_2(ORDERS) = {[ ],[ ],[ ],[ ],[ ]};
  - Λ_2(TA1) = <BERECHARGED>, Λ_2(TA2) = <RECHARGE>, Λ_2(TA3) = <BACK>.

- $EN_3(D) = (\Sigma, P_3, V_3, T_3, A_3, \tau_3, G_3, E_3, I_3, \Lambda_3)$ e
  - P_3 = {RESTING, TRAVELING, RETURNING, WASTE, BROKE, READY};
  - $V_3 = \emptyset$;
  - T_3 = {TD0, TD1, TD2, TD3, TD4, TD5};
  - A_3 = {(RESTING, TD0), (TD0, TRAVELING), (TRAVELING, TD1), (TD1, RETURNING), (RETURNING, TD2), (TD2, WASTE), (WASTE, TD3), (TD3, BROKE), (BROKE, TD4), (TD4, READY), (READY, TD5), (TD5, RESTING)};
  - τ_3(RESTING) = [ ], τ_3(TRAVELING) = [ ], τ_3(RETURNING) = [ ], τ_3(WASTE) = [ ], τ_3(BROKE) = [ ], τ_3(READY) = [ ];

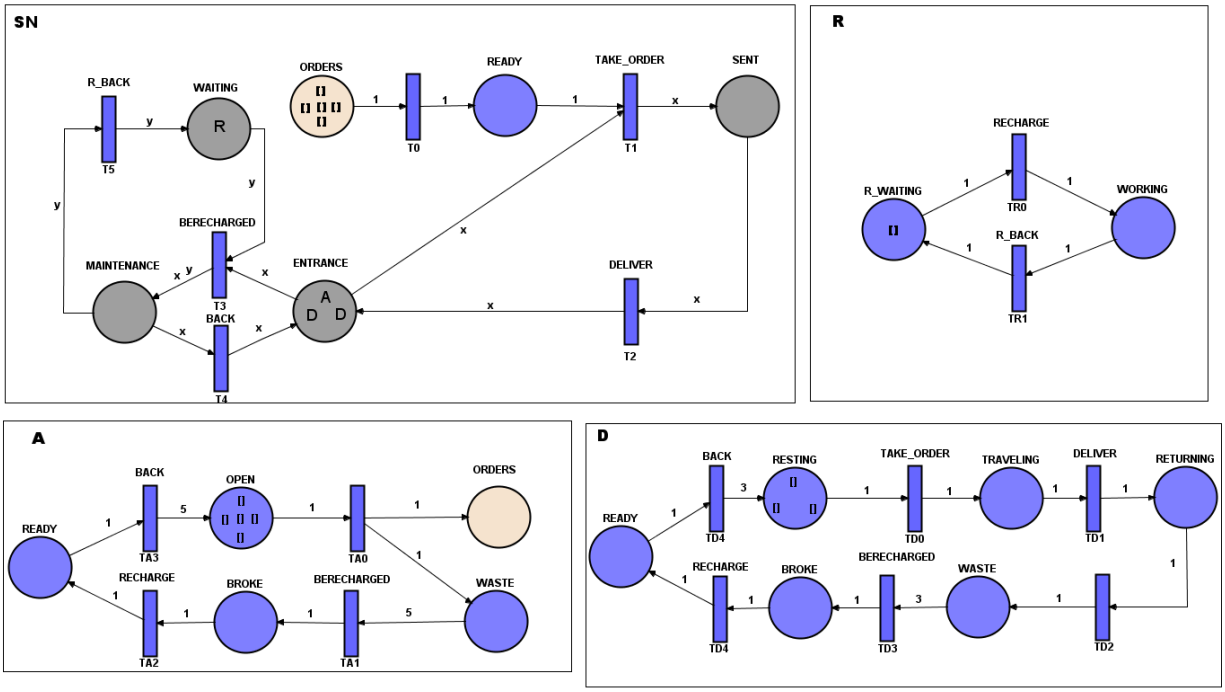

Figura 2.7: *Exemplo de um sistema de entregas modelado por uma Rede de Petri Aninhada conforme a Definição 2.6.1. O sistema é composto por quatro redes (SN, R, A e D), onde as transições são etiquetadas de forma a permitir o disparo síncrono destas entre as redes em passos verticais e horizontais. Além disso, os pedidos são controlados por um lugar compartilhado (ORDERS) que está presente em todas as quatro redes mas foi omitido no diagrama daquelas onde o mesmo não possui relevância para o estado das redes.*

- $E_3(\text{RESTING}, \text{TD0}) = 1$, $E_3(\text{TD0}, \text{TRAVELING}) = 1$, $E_3(\text{TRAVELING}, \text{TD1}) = 1$, $E_3(\text{TD1}, \text{RETURNING}) = 1$, $E_3(\text{RETURNING}, \text{TD2}) = 1$, $E_3(\text{TD2}, \text{WASTE}) = 1$, $E_3(\text{WASTE}, \text{TD3}) = 3$, $E_3(\text{TD3}, \text{BROKE}) = 1$, $E_3(\text{BROKE}, \text{TD4}) = 1$, $E_3(\text{TD4}, \text{READY}) = 1$, $E_3(\text{READY}, \text{TD5}) = 1$, $E_3(\text{TD5}, \text{RESTING}) = 3$;
- $I_3(\text{RESTING}) = \{[\,], [\,], [\,]\}$, $I_3(\text{TRAVELING}) = \emptyset$, $I_3(\text{RETURNING}) = \emptyset$, $I_3(\text{WASTE}) = \emptyset$, $I_3(\text{BROKE}) = \emptyset$;
- $\Lambda_3(\text{TD0}) = <\text{TAKEORDER}>$, $\Lambda_3(\text{TD1}) = <\text{DELIVER}>$, $\Lambda_3(\text{TD3}) = <\text{BERECHARGED}>$, $\Lambda_3(\text{TD4}) = <\text{RECHARGE}>$, $\Lambda_3(\text{TD5}) = <\text{BACK}>$.

A seguir serão definidas as regras de ocorrência para as redes aninhadas. Tais regras são fundamentais para a compreensão das diferentes abordagens do paradigma de *tokens* como redes entre as redes por objeto e as redes aninhadas.

Definição 2.6.3. *Seja $N = (\Sigma, P_s, L, (EN_0, EN_1, \ldots, EN_n))$ uma rede aninhada, seja M uma marcação de N, de acordo com a Definição 2.6.2, e seja $\overline{EN}$ e $\overline{\Sigma}$ os conjuntos de todos os* tokens *de rede e* tokens *atômicos (sem estrutura interna), respectivamente. Seja $W(t)$ o conjunto de todas as expressões dos arcos incidentes em uma transição t e seja b a função que atribui valores em $\overline{\Sigma} \cup \overline{EN}$ às variáveis de $Var(W(t))$. As regras de ocorrência desta rede podem descritas como:*

- **Passo autônomo.** *Seja EN_i uma rede qualquer de N e seja M_i sua marcação. Uma transição $t \in T_i$ com $\Lambda_i(t) = \emptyset$ está habilitada caso exista alguma função de atribuição b tal que*

$$E_i(p,t)(b) \subseteq M_i(p) \qquad \text{para todo } p \in {\bullet t}\,.$$

A transição t poderá ocorrer, gerando uma marcação M_i' para EN_i tal que

$$M_i'(p) = M_i(p) - E_i(p,t)(b) + E_i(t,p)(b) \qquad \textit{para todo } p \in P_i\,.$$

As instâncias de tokens *selecionadas na atribuição b para as variáveis são* **consumidas** *caso as variáveis não estejam presentes nas expressões dos arcos de saída, e são* **transportadas** *caso as variáveis estejam nos arcos de saída. Caso o número de variáveis do mesmo tipo seja maior nas funções dos arcos de saída, as instâncias dos* tokens *atribuídos a estas são copiadas. No passo autônomo, as marcações dos* tokens *de rede presentes na rede EN_i permanecem inalteradas.*

- **Passo vertical.** *Sejam $E_{i_1}, \ldots, E_{i_k}$ um subconjunto das redes $EN_0, \ldots, EN_n$. Sejam $t_{i_1} \in T_{i_1}, \ldots, t_{i_k} \in T_{i_k}$ transições dessas redes, cujas marcações são $M_{i_1}, \ldots, M_{i_k}$, respectivamente, para as quais vale $\Lambda_{i_i}(t_{i_1}) = \cdots = \Lambda_{i_k}(t_{i_k})$, sendo que $\Lambda_{i_1}(t_{i_1}), \ldots, \Lambda_{i_k}(t_{i_k}) \in L_v$. As transições $t_{i_1}, \ldots, t_{i_k}$ estão habilitadas caso exista alguma função de atribuição b tal que, para $i \in \{i_1, \ldots, i_k\}$ vale que*

$$E_i(p_x, t_i)(b) \subseteq M_i(p_i) \qquad \textit{para todo } p_x \in {\bullet t_i}\,.$$

As transições $t_{i_1}, \ldots, t_{i_k}$ poderão ocorrer e, caso ocorram, ocorrerão simultaneamente, gerando marcações M_i', para cada $i \in \{i_1, \ldots, i_k\}$, tais que

$$M_i'(p) = M_i(p) - E_i(p,t_i)(b) + E_i(t_i,p)(b) \qquad \textit{para todo } p \in P_i\,.$$

O mesmo comportamento de **transporte** *dos* tokens *pela atribuição das variáveis descrita no passo autônomo permanece no passo vertical. No passo vertical, apenas uma transição com a etiqueta em L_v por rede pode ser envolvida no disparo, e uma rede só pode ser sincronizada desta maneira com um* token *de rede da sua própria marcação.*

- **Passo horizontal.** *Sejam $t_i \in T_i$ e $t_j \in T_j$ transições de duas redes quaisquer EN_i e EN_j onde $i, j \neq 0$, com marcações M_i e M_j, respectivamente, e sendo $\Lambda(t_i) = \Lambda(t_j)$, onde $\Lambda(t_i), \Lambda(t_j) \in L_h$. As transições t_i e t_j estarão habilitadas caso os* tokens *de rede (EN_i, M_i) e (EN_k, M_j) estejam contidos na marcação $M_k(p_k)$ de um mesmo lugar pertencente a uma rede qualquer EN_k e exista uma função de atribuição b tal que*

$$E_i(p_x, t_i)(b) \subseteq M_i(p_x) \text{ e } E_j(p_y, t_j)(b) \subseteq M_j(p_y) \qquad \textit{para todo } p_x \in {\bullet t_i} \text{ e } p_y \in {\bullet t_j}\,.$$

As transições t_i, t_j poderão ocorrer e, caso ocorram, ocorrerão simultaneamente, gerando as marcações M_i', M_j' e M_k':

$$M_i'(p) = M_i(p) - E_i(p,t_i)(b) + E_i(t_i,p)(b) \qquad \textit{para todo } p \in P_i\,,$$

$$M_j'(p) = M_j(p) - E_j(p,t_j)(b) + E_j(t_j,p)(b) \qquad \textit{para todo } p \in P_j\,,$$

$$M_k'(p_k) = M_k(p_k) - (EN_i, M_i) - (EN_j, M_j) + (EN_i, M_i') + (EN_j, M_i')\,.$$

Novamente, o mesmo comportamento de **transporte** *dos* tokens *pela atribuição das variáveis descrita no passo autônomo permanece no passo horizontal. Como pode ser notado, uma condição para o passo horizontal é que os dois* tokens *de redes necessitam obrigatoriamente estar em um mesmo lugar p_k de alguma outra rede e, ao dispararem, suas marcações serão alteradas mas eles permanecerão no mesmo lugar p_k.*

2.7 Discussão

Ao longo deste capítulo, diversas abordagem ao paradigma de redes como *tokens* em Redes de Petri foram definidas partindo de uma linha de raciocínio inspirada pelo trabalho de Lakos [49], que demonstra que as redes por objeto teriam como referencial teórico inicial as redes coloridas. A diversidade das abordagens deriva da forma como o problema da consistência do estado da rede no sistema modelado é solucionado e, ao final, cada uma destas abordagens parece facilitar o projeto de sistemas com diferentes características.

As redes por objeto com suas duas semânticas preservam o estado do sistema conservando a marcação de todas as redes constituintes deste, de forma que, na semântica por referência cada *token* nada mais é do que uma referência à uma instância de um tipo de rede e instâncias diferentes do mesmo tipo de rede precisam ser referenciados por *tokens* estritamente diferentes. Estas instâncias, após criadas, permanecem mesmo sem nenhuma referência a estas em outras redes do sistema e, assim, o estado de todo o sistema é conservado.

Por outro lado, a semântica por valor mantém a consistência do estado da rede não através das instâncias criadas de um tipo de rede no sistema mas pela conservação da marcação de uma única instância. Assim, todo *token* de um determinado tipo de rede pode carregar parte do estado daquela por meio da participação no total de *tokens* que compõem a marcação, ou por meio da participação na totalidade de processos que compõem a rede. Dessa forma, para se averiguar o estado de uma rede no sistema é necessário a junção das partições espalhadas entre os *tokens* daquela rede.

Estas abordagens semânticas para solucionar o problema da consistência nas redes com *tokens*, segundo Lomazova, são complexas e poderiam ser simplificadas. Desta premissa, as Redes de Petri Aninhadas de Lomazova foram propostas [60]. As Redes Aninhadas semanticamente lidam com os *tokens* de rede como instâncias de um determinado tipo de rede e não como referências, de forma que cada *token* é uma instância *per se* e, diferente da semântica por valor das redes por objeto, esta instância mantém um estado completo e independente. Estes *tokens* podem ser criados e consumidos de forma que instâncias das redes podem ser criadas a partir de sua marcação inicial, como na semântica por referência das redes por objeto, mas também podem ser removidas do sistema.

A grande vantagem do tratamento de *tokens* como instâncias independentes desta forma é que isto possibilita de maneira muito mais simples o projeto de sistemas recursivos [61]. Lomazova, em outros trabalhos, explorou mais as questões da semântica das redes aninhadas e a forma como a consistência do estado nestas é mantida [59, 64].

Mesmo com definições bem estabelecidas e com resultados teóricos contundentes a respeito de sua capacidade de modelar sistemas de forma a garantir formalmente sua consistência, o projeto das redes é realizado por humanos e, portanto, passível de erros. Neste contexto, ferramentas de verificação de modelos criados a partir de uma determinada formulação teórica se fazem indispen-

sáveis. Assim, para tornarem-se uma ferramenta de projeto robusta, as redes aninhadas necessitam de um ecossistema de ferramentas para verificar a aplicação de suas definições e restrições.

A partir desta problemática, Venero e Corrêa da Silva propuseram uma forma de modelagem de sistemas de Redes de Petri Aninhadas por meio da linguagem de verificação PROMELA. Contudo, a definição de redes aninhadas proposta em seu trabalho [83] é um pouco mais restrita do que a proposta inicialmente por Lomazova [60] pois, diferentemente da proposta de Lomazova, na definição dada por Venero e Corrêa da Silva os *tokens* envolvidos em sincronizações verticais são *sempre* consumidos.

Neste trabalho, um complemento a esta contribuição de Venero e Corrêa da Silva é proposto, a saber, uma forma de representação das redes aninhadas na linguagem de marcação PNML e sua tradução automática para PROMELA. Para isso, iremos utilizar ao longo do trabalho a Definição 2.6.1 de Redes de Petri Aninhadas, que consiste da definição apresentada por Venero e Corrêa da Silva [83] sem a restrição do consumo dos *tokens* nos passos de sincronização vertical.

Capítulo 3

Revisão sistemática sobre ferramentas para Redes de Petri Aninhadas

Uma lista bem conhecida de ferramentas de simulação de redes de Petri [1] mantida pela universidade de Hamburgo não apresenta nenhuma ferramenta capaz de realizar a verificação de Redes de Petri Aninhadas. Neste contexto, o presente estudo foi realizado em 2018 com o objetivo de, através de uma revisão sistemática na literatura, identificar as ferramentas não listadas que estão sendo desenvolvidas e utilizadas pela comunidade que sejam capazes de simular e verificar as redes aninhadas e, caso não existam, descobrir quais delas podem ser modificadas para esse fim.

3.1 Método

O presente trabalho de revisão foi conduzido por meio da ferramenta virtual *parsifal*[1]. Através da ferramenta, foi seguido um processo de revisão sistemática nos moldes propostos por Kitchenham [45], que consiste na elaboração de um protocolo, para reduzir o viés da pesquisa, que contém as perguntas de pesquisa, os termos da pesquisa dentro das categorias definidas como *PICOC* (*Population, Intervention, Comparison, Outcome, Context*) e uma série de critérios de qualidade para seleção dos estudos encontrados.

Como parte do processo proposto, o protocolo foi submetido a algumas revisões para o refinamento das questões e, principalmente, das cadeias de busca, a fim de obter uma maior afinidade com o objetivo da revisão. Após a definição do protocolo, foi realizado um processo de pesquisa em uma base de documentos e os critérios de qualidade definidos no protocolo foram aplicados aos resultados. Por fim, houve uma etapa de refinamento até que os principais documentos, conforme o objetivo da revisão, fossem identificados através de uma pontuação e, por fim, os dados de pesquisa foram extraídos destes.

A seguir serão detalhadas cada etapa do processo descrito, e então iremos conduzir uma análise dos resultados.

3.2 Perguntas de pesquisa

No protocolo, as seguintes questões de pesquisa foram levantadas:

[1] https://parsif.al/

- **P1**. Quais são as principais ferramentas de simulação de redes de Petri?
- **P2**. Quais são os verificadores de modelos para redes de Petri?
- **P3**. Quais são as ferramentas para simular redes de Petri coloridas?
- **P4**. Alguma ferramenta consegue simular lugares compartilhados?
- **P5**. Existe alguma ferramenta para simular redes de Petri Aninhadas?

As questões foram elaboradas em uma escala dentro da proposta do objetivo da revisão, partindo do espectro mais global em relação ao assunto e se afunilando quanto a especificidade da tese a ser desenvolvida com base nos resultados.

Primeiro, a questão P1 tem por propósito levantar quais são as ferramentas mais aceitas pela comunidade. Seguindo, P2 direciona a atenção para outro aspecto da revisão que é a investigação a respeito das técnicas de checagem dos modelos e sua integração com as ferramentas de simulação. Já P3 tem por objetivo um estreitamento dos resultados de P1 na direção das redes coloridas, que são um subconjunto das Redes de Petri do qual pode-se derivar as redes aninhadas [49, 79]. O objetivo de identificar ferramentas com tais capacidades a priori seria o de manter um registro das possíveis ferramentas com a possibilidade de serem atualizadas com a funcionalidade de simular as redes aninhadas.

As perguntas P4 e P5 já são mais específicas no tocante às redes de estudo, tendo como propósito a identificação das ferramentas já prontas para simulá-las. Mais especificamente, P4 foca na característica das redes aninhadas não suportada pelas ferramentas mais conhecidas e, por isso, é uma qualidade a ser fortemente considerada durante a revisão.

3.3 Processo de pesquisa

O processo de pesquisa foi realizado manualmente apenas na base *IEEE Digital Library*[2] devido a sua integração com a ferramenta *parsifal* e o período restrito em que a revisão foi conduzida. Para tal, a cadeia de busca utilizada foi

"Petri Net analyzer" OR "Petri Net check" OR "Petri Net tools" OR "Petri Net simulation",

formulada para encontrar qualquer referência a ferramenta ou verificação, contendo basicamente termos sinônimos.

Além da cadeia, outra restrição foi imposta à busca, que foi a data de publicação a partir de 1998 até 2018, ano da execução da revisão, a fim de se obter uma visão ampla, porém atual. Dos resultados obtidos, foram selecionados os 200 primeiros, ordenados por relevância pela ferramenta de busca. Novamente, o ideal seria a inclusão dos mais de 2000 resultados para uma primeira avaliação, contudo, o escasso tempo para a execução do projeto foi um fator impeditivo à amplitude da revisão.

3.4 Critério de inclusão e exclusão

A etapa seguinte do procedimento foi uma triagem dos artigos com base em critérios de inclusão e exclusão. Foram definidos no protocolo os seguintes critérios de inclusão:

[2] http://ieeexplore.ieee.org

1. Contém alguma ferramenta de simulação para algum tipo de rede;
2. Contém alguma referência à verificação.

E os seguintes critérios de exclusão:

1. Não contém nenhuma ferramenta de simulação;
2. É apenas algum caso de uso.

O primeiro critério de inclusão é abrangente e simples, dado o objetivo de levantar quaisquer ferramentas utilizadas pela comunidade. Já o segundo critério foi criado com base na segunda questão de pesquisa, com o objetivo de identificar possíveis métodos de verificação e a possível integração com alguma ferramenta.

Os critérios de exclusão foram igualmente simples. Caso o artigo tratasse apenas de alguma aplicação do formalismo das redes de Petri em algum campo do conhecimento e não citasse alguma ferramenta, ele não deveria ser considerado.

O último critério de exclusão foi levantado para identificar os artigos que continham alguma referência a verificação ou ferramenta de forma ontológica às redes de Petri em algum caso de uso, e não necessariamente sobre algum ferramental de simulação ou verificação das redes *per se*. Alguns exemplos de artigos eliminados por esse critério podem ser vistos nas referências [9, 54, 87].

Além disso, um critério de classificação para "duplicado" foi adotado. Quando os artigos tratavam da mesma ferramenta e haviam sido escritos pelo mesmo autor, apenas o artigo mais antigo foi considerado. Por exemplo, a referência [14] foi considerada duplicada de [13] e [68] foi considerada duplicada de [32].

3.5 Avaliação de qualidade

A avaliação da qualidade para pontuação dos artigos selecionados na triagem anterior foi baseada nos seguintes critérios:

- **C1.** A ferramenta ou método de verificação é original?
  - **RC1.1** Propõe nova técnica ou ferramenta;
  - **RC1.2** Matlab API;
  - **RC1.3** CPN Tools;
  - **RC1.4** Contém uma ferramenta conhecida.
- **C2.** O estudo contém alguma proposta de integração entre ferramenta e verificação?
  - **RC2.1** Contém referência;
  - **RC2.2** Não.
- **C3.** A ferramenta ou método pode lidar com redes aninhadas?
  - **RC3.1** Pode lidar com redes aninhadas;
  - **RC3.2** Não.

Tabela 3.1: *Pontuação dos critérios para avaliação de qualidade.*

Resposta	Pontuação
RC1.1	10.0
RC1.2	3.0
RC1.3	0.0
RC1.4	5.0
RC2.1	7.0
RC2.2	0.0
RC3.1	10.0
RC3.2	0.0

A pontuação de acordo com cada resposta está listada na Tabela 3.1. O nível de corte de qualidade foi uma pontuação maior ou igual a 5.0.

A resposta RC1.1 foi definida com base em uma conhecida lista de ferramentas para simulação de redes de Petri [1]. O primeiro critério de qualidade se destacou dos outros por conter mais de duas possíveis respostas, uma afirmativa e outra negativa, pelo fato de que durante a triagem foi detectado uma grande presença de artigos utilizando a famosa ferramenta CPN Tools e *API's* do Matlab.

Com o propósito de levantar quantos estudos utilizaram CPN Tools, a resposta foi acrescentada com pontuação zerada. Os artigos que continham *API's* de Matlab tiveram uma pontuação prejudicada, contudo não poderiam ser eliminados caso contivessem alguma integração com técnicas de simulação de redes aninhadas ou com alguma técnica de verificação.

3.6 Coleta dos dados

A extração dos seguintes dados foi feita nos estudos que obtiveram a pontuação de corte:

- **D1.** Utiliza alguma ferramenta para modelar as redes ou apenas contém uma proposta de checagem?
- **D2.** Ferramenta utilizada;
- **D3.** Ferramenta é nova?
- **D4.** Qual tipo de rede é suportada?
- **D5.** A ferramenta apenas modela ou simula o funcionamento da rede?
- **D6.** Suporta lugares compartilhados?
- **D7.** Qual técnica ou ferramenta é utilizada para checagem do modelo?

Vale destacar que algumas questões não se aplicavam a alguns artigos e portanto foram respondidas com "NA" (não se aplica).

3.7 Análise dos dados

Os dados extraídos abordam as perguntas de pesquisa, e a relação entre dados e questões pode ser vista na Tabela 3.2.

Tabela 3.2: *Mapeamento entre as questões para extração dos dados e as perguntas de pesquisa abordadas nas questões.*

Dado	Pergunta de Pesquisa
D1	**P1, P2**
D2	**P1**
D3	**P1**
D4	**P3, P5**
D5	**P1**
D6	**P4**
D7	**P2**

3.8 Desvios do protocolo

Como já citado, o fato de muitos estudos utilizarem a CPN Tools gerou a necessidade de um levantamento a respeito da proporção destes para os outros. Dessa forma, a pontuação zerada porém de categoria diferente foi incluída nos critérios de qualidade, gerando assim uma pequena distorção no protocolo. No fim, a abundância de resultados contendo esta ferramenta foi fruto da cadeia de busca, que por sua vez é um reflexo das perguntas de pesquisa, neste caso, da pergunta P3. Além disso, é fato considerável que apenas uma base foi utilizada para a pesquisa e por isso algumas referências podem não ter sido encontradas.

3.9 Resultados

Os resultados da pesquisa podem ser verificados nas seções a seguir.

3.10 Resultados da pesquisa

O gráfico da Figura 3.1 mostra a quantidade de artigos selecionados no processo de triagem antes da aplicação do critério de qualidade e depois do critério aplicado. A Tabela 3.3 mostra os 26 estudos selecionados pelo critério de qualidade identificados por um ID e com sua respectiva referência.

A Figura 3.2 mostra a distribuição dos indicados pelos critérios de inclusão pelo ano de publicação. As Tabelas 3.4 e 3.5 mostram os dados coletados dos estudos.

No gráfico da Figura 3.3 é possível avaliar quais as principais categorias de redes de Petri suportadas pelas ferramentas.

Já no gráfico da Figura 3.4 estão presentes apenas os tipos de rede suportadas pelas ferramentas consideradas novas.

3.11 Avaliação da qualidade

Como discorrido anteriormente, o critério de qualidade foi medido através de uma pontuação distribuída para cada questão. Na Tabela 3.6 é possível visualizar as pontuações de cada estudo.

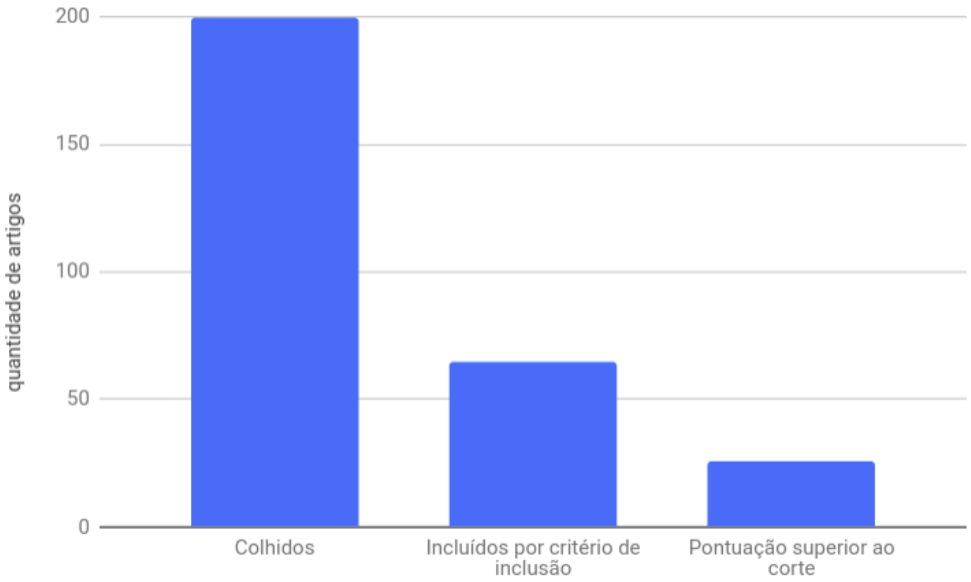

Figura 3.1: *Relação entre os estudos selecionados e aceitos.*

Tabela 3.3: *Descrição e identificação dos artigos selecionados.*

ID	Nome
A1	MathMC: A Mathematica-Based Tool for CSL Model Checking of Deterministic and Stochastic Petri Nets [35]
A2	Evaluation of fuzzy Petri nets with the tool TransPlaceSim [47]
A3	Distributed Simulation of Colored Stochastic Petri Nets With TimeNET 4.0 [90]
A4	Petri net models in computational biology [69]
A5	Software support for teaching Petri nets: P3 [29]
A6	Time Petri Nets Analysis with TINA [86]
A7	Petri net based model checking for the collaborative-ness of multiple processes systems [56]
A8	Applications of the Petri net to simulate, test, and validate the performance and safety of complex, heterogeneous, multi-modality patient monitoring alarm systems [76]
A9	Generalized stochastic Petri Net model based security risk assessment of software defined networks [5]
A10	Developing a New Petri Net Tool for Simulation of Discrete Event Systems [13]
A11	MARCIE - Model Checking and Reachability Analysis Done EffiCIEntly [75]
A12	Software tool for modeling, simulation and real-time implementation of Petri net-based supervisors [30]
A13	The Input-Output Place-Transition Petri Net Class and Associated Tools [32]
A14	Model Checking Control Flow Petri Nets Using PAT [17]
A15	A Term Rewriting Approach to Analyze High-Level Petri Nets [36]
A16	A Petri net software for mission reliability evaluation of PMS [89]
A17	HiPS: Hierarchical Petri Net design, simulation, verification and model checking tool [34]
A18	A new object-oriented Petri net simulation environment based on Modelica [71]
A19	Distributed simulation of timed coloured Petri nets [26]
A20	Petri Net Diagnosability Analyzer [55]
A21	Towards compositional verification of SDL systems [24]
A22	A PIPE Based System for Checking Temporal Constraints in Service Composition [16]
A23	Game theory semantics for PCTL model checking label-extended probabilistic Petri net [57]
A24	Supervisory control of an automated system with ladder logic programming and analysis using Petri nets [52]
A25	Event-driven simulation of timed Petri net models [91]
A26	Model Checking-based Safety Verification of a Petri Net Representation of Train Interlocking Systems [6]

3.12 Fatores de Qualidade

A distribuição da pontuação mostra que as ferramentas que se destacaram o fizeram por serem novas e conterem alguma referência a verificação. Os artigos com pontuação 12.0 assim pontuaram pois continham o uso de uma ferramenta bem conhecida [1], mas também continham alguma

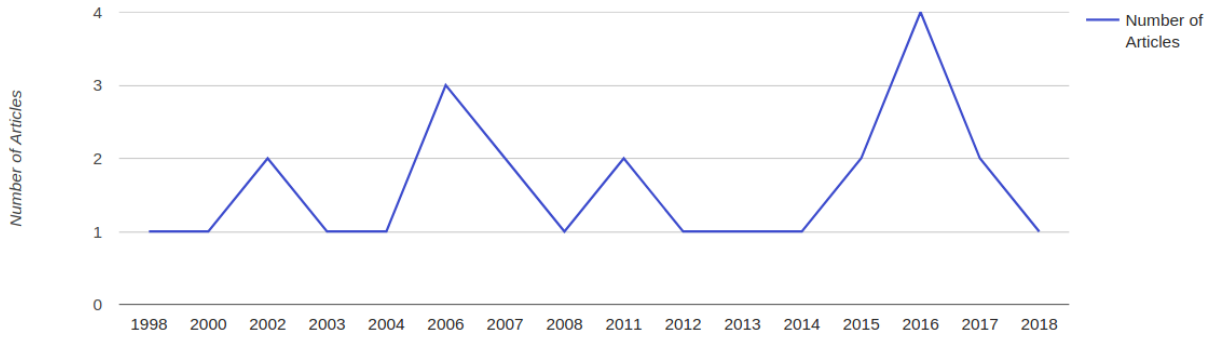

Figura 3.2: *Relação de estudos selecionados por aceitos.*

Tabela 3.4: *Dados coletados dos artigos (D1-D4). A sigla FM significa "Ferramenta para modelar" e a sigla TC significa "Técnica de checagem".*

ID	D1	D2	D3	D4
A1	FM, TC	MathMC	True	Stochastic Petri Net
A2	FM	TransPlaceSim	True	Fuzzy Petri Net
A3	FM	TimeNET	False	High-level Petri Net, Stochastic Petri Net, Timed Petri Net
A4	FM	PIPE,Sirphyco	True	Stochastic Petri Net, High-Level Petri Net
A5	FM	P3	False	Upgraded Petri Net
A6	FM	TINA	False	Timed Petri Net
A7	FM, TC	INA	False	High-level Petri Net, Timed Petri Net
A8	FM	HPSim	False	Stochastic Petri Net, Timed Petri Net
A9	FM	PIPE	False	Stochastic Petri Net, High-Level Petri Net
A10	FM	GPSIM	False	High-level Petri Net, Timed Petri Net
A11	FM, TC	MARCIE	True	Stochastic Petri Net
A12	FM	Petri.NET	True	Petri Net
A13	FM	IOPT	True	Input Output Petri Net
A14	TC	NA	NA	Control Flow Petri Net
A15	FM, TC	PIPE	False	Stochastic Petri Net, High-Level Petri Net
A16	FM, TC	EOOPN Tool	True	Extended Object Oriented Petri Net
A17	FM, TC	HiPS	True	Hierarchical Petri Net
A18	FM, TC	VANESA + PNLib	True	Extended Hybrid Petri Net
A19	FM	RAINBOW	True	Timed Petri Net, Coloured Petri net
A20	TC	PENDA	True	Labeled Petri Net
A21	FM, TC	PEP	False	High-Level Petri Net, Timed Petri Net
A22	FM, TC	PIPE	False	Stochastic Petri Net, High-Level Petri Net
A23	FM, TC	LPPNMV	True	Probrabilistc Petri Net with Label
A24	FM	Visual object net	False	Hybrid Dynamic Net, Hybrid Object Net
A25	FM	TPN-Tools + TPNsim	True	Timed Petri Net
A26	FM	TAPAAL	False	Timed Petri Net

referência à adição de verificação. Outro fator a ser levantado é que são 31 os artigos que faziam referência ao uso de CPN Tools, totalizando 15.5%, e por utilizarem esta ferramenta não atingiram a pontuação necessária pelos critérios de qualidade. Por esta ser uma ferramenta já conhecida e muito utilizada mas que não suporta as Redes de Petri Aninhadas e pelo fato destes artigos não apresentarem novas técnicas, eles acabariam por gerar um viés muito grande nos resultados a favor do CPN Tools. Ainda cabe destacar a descoberta de 13 artigos com ferramentas desconhecidas [1], e que obtiveram uma pontuação suficiente dentro dos critérios de qualidade.

Tabela 3.5: *Dados coletados dos artigos (D5-D7).*

ID	D5	D6	D7
A1	Apenas modela	False	Técnica própria (State spaces)
A2	Apenas modela	False	NA
A3	Modela e Simula	False	NA
A4	Modela e Simula	False	NA
A5	Modela e Simula	False	NA
A6	Modela e Simula	False	NA
A7	Modela e Simula	False	State spaces
A8	Modela e Simula	False	NA
A9	Modela e Simula	False	NA
A10	Modela e Simula	False	NA
A11	Apenas modela	False	Técnica própria (State spaces)
A12	Modela e Simula	False	NA
A13	Modela e Simula	False	NA
A14	NA	False	PAT Process Analysis Toolkit
A15	Modela e Simula	False	Maude
A16	Modela e Simula	NA	Técnica própria
A17	Modela e Simula	False	Técnica própria (State Spaces)
A18	Modela e Simula	False	MODELICA
A19	Modela e Simula	False	NA
A20	NA	False	Técnica própria
A21	Modela e Simula	False	SPIN, State spaces, Partial order based, BDD based checking, Lin. programming
A22	Modela e Simula	False	Checagem Temporal
A23	Apenas modela	False	Técnica própria (Game theory)
A24	Modela e Simula	False	NA
A25	Modela e Simula	False	NA
A26	Modela e Simula	False	NA

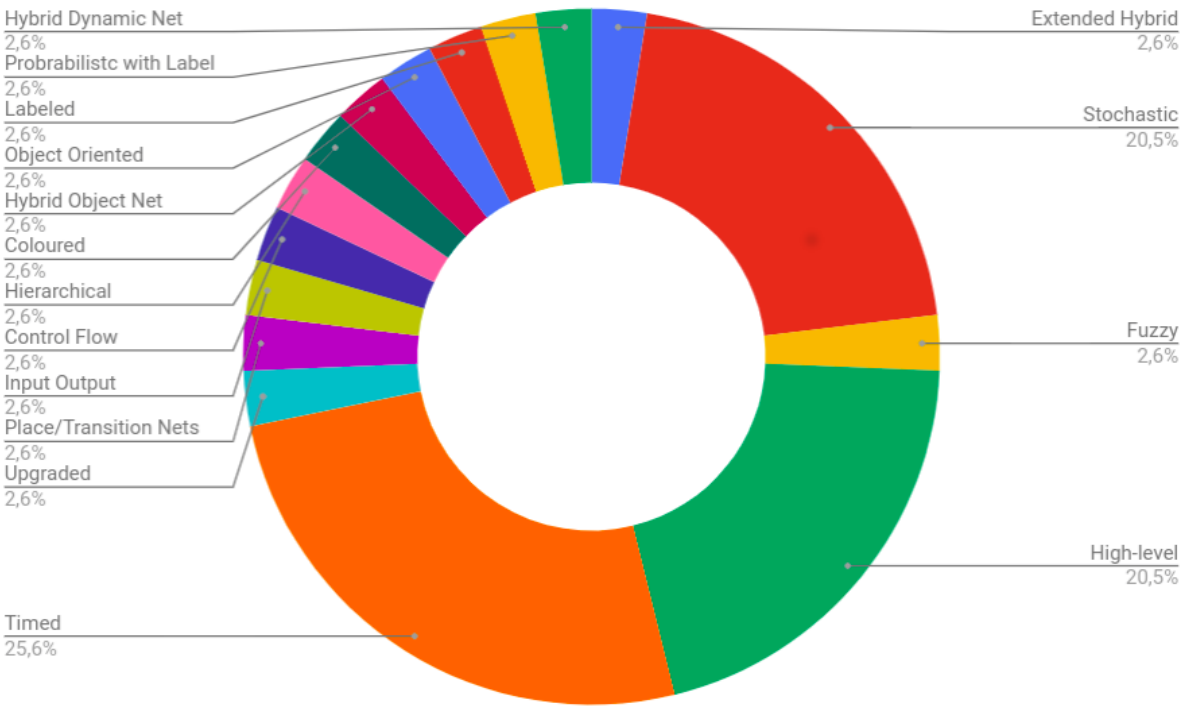

Figura 3.3: *Relação de tipos de rede suportadas por ferramentas ou metodologias automatizadas de simulação e/ou verificação.*

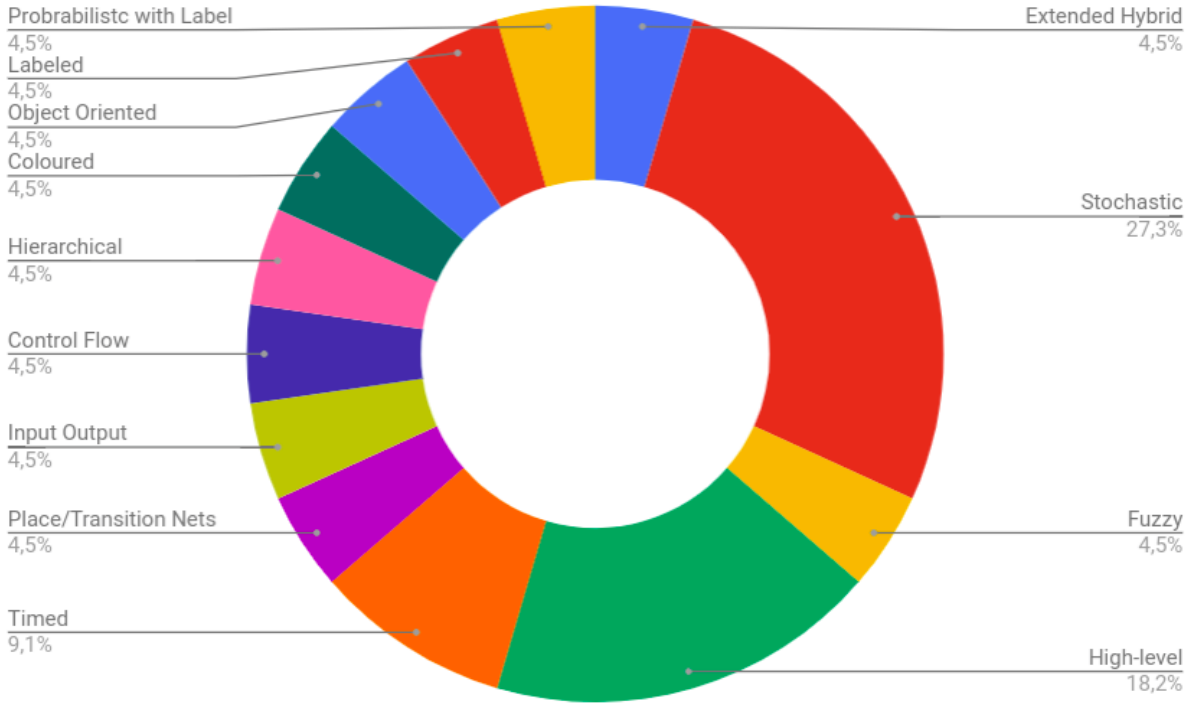

Figura 3.4: *Relação de tipos de rede suportadas pelas ferramentas novas.*

Tabela 3.6: *Pontuação dos artigos selecionados.*

ID	Pontuação	ID	Pontuação	ID	Pontuação
A1	17.0	**A10**	10.0	**A19**	7.0
A2	10.0	**A11**	17.0	**A20**	10.0
A3	5.0	**A12**	10.0	**A21**	12.0
A4	5.0	**A13**	10.0	**A22**	12.0
A5	10.0	**A14**	10.	**A23**	17.0
A6	5.0	**A15**	12.0	**A24**	5.0
A7	12.0	**A16**	17.0	**A25**	7.0
A8	5.0	**A17**	17.0	**A26**	5.0
A9	5.0	**A18**	12.0		

3.13 Discussão

Iremos conduzir agora uma discussão com base nas questões de pesquisa que foram levantadas.

Quais são as principais ferramentas de simulação de redes de Petri? Com base nos resultados apresentados, pode-se afirmar que a ferramenta mais utilizada para simulação das redes de Petri é CPN Tools, contudo, essa era uma ferramenta já conhecida. Destacando os estudos com alto índice de qualidade a partir dos critérios estabelecidos para a revisão, a ferramenta que aparece em mais de um artigo a ferramenta PIPE [5, 16, 36, 69].

Além disso, a revisão também teve sucesso em levantar algumas ferramentas novas [1] que parecem promissoras, como por exemplo EOOPN [89] e RAINBOW [26]. Através do gráfico da Figura 3.3 é possível notar que as chamadas Redes de Petri Temporais são as mais suportadas pelas ferramentas descobertas, seguidas das redes Estocásticas e das redes de Alto Nível.

Contudo, ao se observar o gráfico da Figura 3.4 nota-se que, dentre as ferramentas novas, as que suportam as redes estocásticas são maioria, o que pode indicar uma deficiência nessa área dentre as ferramentas já conhecidas.

Quais verificadores de modelos para redes de Petri? Diversas das publicações da Tabela 3.4 contêm técnicas de verificação de modelos integrados com as ferramentas das redes de Petri. Apesar de boa parte delas serem novas, a estratégia mais comum parece ser a utilização de espaços de estados. Entretanto, um artigo [24] mostra o uso de diversas técnicas e entre elas a utilização de SPIN. Mesmo que o tipo das redes simuladas seja mais simples, isso pode ser um indício positivo da aceitação da utilização do SPIN como verificador pela comunidade.

Quais ferramentas para simular redes de Petri coloridas? Esta questão praticamente retórica, que tinha como objetivo encontrar novas ferramentas para tais redes com capacidade de serem modificadas, também foi sanada de maneira satisfatória. Inicialmente, ficou evidente que CPN Tools é a ferramenta mais utilizada para este tipo de rede e, de certa forma, acabou por enviesar em certo grau a pesquisa. Contudo, uma ferramenta nova para este subconjunto de redes de Petri foi descoberta, a RAINBOW [26].

Alguma ferramenta consegue simular lugares compartilhados? Os resultados da revisão não apontaram nenhuma ferramenta capaz de simular lugares compartilhados. Vale uma nota de que não foi possível inferir se a ferramenta EOOPN [89] contém ou não essa característica. Por se tratar de uma ferramenta com foco em simulação de redes de Petri orientadas a objetos, existe um grande potencial para que ela possa ser modificada para suportar tal funcionalidade, caso já não seja capaz.

Existe alguma ferramenta para simular redes de Petri aninhadas? Nenhuma ferramenta encontrada na revisão mostrou a capacidade imediata de simular as redes aninhadas. Novamente, a ferramenta EOOPN [89] parece ser a mais capaz quanto a isso pelo fato das redes de Petri aninhadas serem um subconjunto das redes orientadas a objeto.

3.14 Outras técnicas para a verificação de Redes de Petri Aninhadas

A verificação das redes garante o correto funcionamento do modelo checando os possíveis comportamentos de acordo com as condições iniciais. Algumas abordagens para tal envolvem por exemplo a verificação dos chamados espaços de estados [28], e outras utilizam de verificadores de *softwares*.

Apesar do poder de modelagem das redes aninhadas, faltam técnicas e ferramentas para analisar suas propriedades. Usando alguns resultados de composicionalidade, pode-se provar a vivacidade, a limitabilidade e a alcançabilidade separadamente para os níveis inferiores e do sistema [19, 66]. Outra abordagem é achatar a estrutura de dois níveis para obter uma rede de Petri ordinária de comportamento equivalente [4, 20, 63]. Então, utilizando a ferramenta CPN Tools, as propriedades são verificadas automaticamente [72]. No entanto, alguns desses métodos são aplicados a subclasses

bastante restritas. Por outro lado, a transformação apresentada por Dworzański e Lomazova [20] produz grandes redes levando a simulações mais difíceis de entender. Além disso, as sequências de disparo correspondentes na rede aninhada devem ser reconstruídas manualmente.

Diversas variações de redes de Petri e formalismos semelhantes (por exemplo, fluxos de trabalho, processos de negócios, diagramas UML) têm sido traduzidos em modelos para verificadores como DVE [53], LTSA [73], NuSMV [22] e SPIN [10, 23, 27, 74, 88]. DVE, LTSA e NuSMV não podem ser usados neste contexto porque não têm suporte para recursão, que é uma propriedade importante quando se trata das aninhadas, onde um *token* de rede pode fazer parte, estar contido, no mesmo tipo de rede que representa. Farwer e Leuschel [23] codificam as redes de objetos de dois níveis no Prolog e as verificam usando o verificador de modelo XTL. Embora o método seja destinado a aninhamento arbitrário, a codificação para a sincronização no caso de vários níveis não é fornecida. A tradução de redes de Petri aninhadas para o PROMELA é mais simples e mais acessível para simulação do que usar regras de reescrita ou programação lógica. Com relação à verificação, o SPIN supera o verificador de modelo do Maude em tempo de execução e requisitos de memória [21]. De acordo com Frappier *et al.* [25], o SPIN é mais rápido que o verificador de modelo XTL e pode manipular um número maior de propriedades e instâncias. Entre as traduções do PROMELA, as apresentadas por Chang e He [10] e por Venero e Corrêa da Silva [83] abordam as redes de Petri aninhadas, mas lidam com subclasses bastante restritas. O primeiro se concentra em redes aninhadas de dois níveis sem sincronização horizontal ou remoção de *tokens* de rede. O último permite analisar redes aninhadas de nível múltiplo e recursivo com uma sincronização bastante restrita e sem passos de transporte.

A revisão realizada tinha como objetivo identificar alguma ferramenta capaz de simular e verificar as chamadas redes de Petri aninhadas mas não encontrou. Uma ferramenta chamada *NPN Tool* [18] promete o feito, porém não está disponível para testes. As redes de Petri aninhadas são propostas como solução para a modelagem de sistemas de vários domínios de aplicação, mas os modelos carecem de uma ferramenta capaz de simular e verificar a corretude destas de uma maneira mais ampla e completa em sua natureza multinível ou recursiva. A capacidade de verificação de redes aninhadas mais gerais através do verificador de software SPIN já foi demonstrada [83]. Apesar destas propostas, atualmente não existe nenhuma ferramenta que permita a tradução automática de uma rede aninhada para a linguagem de um verificador. As traduções precisam ser feitas à mão e isto requer conhecimento da linguagem, funcionamento e recursos do verificador de modelos. Além disso, a tradução reversa também precisa ser feita de forma manual, i.e. interpretar os resultados e os contra exemplos gerados pelo verificador como aspectos do comportamento das redes aninhadas.

De modo indubitável, uma ferramenta gráfica de edição e simulação poderia ajudar muito a modelagem e análise de redes de Petri aninhadas. O desenvolvimento destas ferramentas pode levar um considerável esforço de programação. Por isso, o uso de ferramentas que fornecem funcionalidades de edição e simulação é uma possibilidade que deve ser considerada. Para a verificação uma abordagem que já provou ser bem sucedida é o uso de ferramentas de verificação de modelos. Para a integração de ambas se faz necessária a criação de uma interface entre as ferramentas de simulação e verificação. A ferramenta proposta nesta dissertação tem por objetivo preencher esta lacuna de ferramentas para verificação de redes de Petri aninhadas fornecendo uma interface entre ferramentas de simulação gráficas e o verificador de modelos spin utilizando a *Petri Net Markup Language* como padrão de transferência entre elas.

Capítulo 4

Verificação automática de Redes de Petri Aninhadas

Como o poderoso formalismo para projeto de sistemas que são, as Redes de Petri necessitam de um robusto ecossistema de ferramentas de modelagem, simulação e validação de modelos de forma a simplificar o seu uso em grandes projetos e fornecer uma verificação formal necessária para garantir a integridade, validade e corretude do modelo.

Neste contexto, diversas ferramentas surgiram com foco em diferentes especializações das Redes de Petri. Com isso, surgiu a necessidade da padronização de um formato de representação para transferência, sendo então criada a linguagem de marcação PNML [8] para preencher esta lacuna. O PNML, portanto, pode ser utilizado como formato dos arquivos de modelos de rede a serem verificados e, para realizar esta verificação, a linguagem PROMELA pode ser utilizada como entrada para o verificador de modelos SPIN [39], uma ferramenta eficiente para verificar modelos e *software* distribuído. Porém, para que haja essa integração, precisamos inicialmente propor uma extensão da PNML para Redes de Petri Aninhadas.

Neste Capítulo, introduziremos a PNML na Seção 4.1 em conjunto com nossa proposta de extensão para a especificação. Na Seção 4.2 serão apresentados os conceitos básicos sobre a linguagem PROMELA e o SPIN. A Seção 4.3 introduzirá a metodologia da tradução. E, por fim, a implementação da tradução será apresentada na Seção 4.4.

4.1 A linguagem PNML

O PNML, *Petri Net Markup Language* (ISO/IEC 15909-2) [41], é um padrão baseado em XML proposto inicialmente por Billington *et al.* [8] para suprir uma demanda da comunidade que necessitava de um padrão comum para modelar as Redes de Petri a fim de intercambiar tais modelos entre ferramentas e projetos, facilitando a replicação, validação e verificação. A linguagem foi construída tendo três princípios como ponto de partida [65]:

Legibilidade: O formato deve ser legível e editável com qualquer editor de texto;

Universalidade: O formato não pode excluir nenhum tipo de rede de Petri e deve suportar redes de quaisquer extensões;

Mutualidade: O formato deve permitir a extração do máximo de informações possíveis de

uma rede, mesmo que desconhecida. Dessa forma, o formato deve extrair os princípios comuns e as notações comuns das redes de Petri.

A norma da PNML prevê a proposta de novas extensões para o padrão. A forma corrente de se propô-las é através da definição de um meta-modelo UML contendo os novos pacotes a serem incluídos no padrão. Este meta-modelo não é a definição direta do XML implementado, mas sim um mapeamento das classes da rede com seus atributos [37]. O XML implementado deve ser definido a partir do documento de sintaxe chamado PNTD (*Petri Net Type Definition*).

Nesta seção, iremos propor uma extensão do padrão para as redes aninhadas da Definição 2.6.1. Para isso, será utilizado um meta-modelo UML e posteriormente será apresentada a extensão do padrão XML através do PNTD.

Como uma linguagem de marcação, a sintaxe do PNML é composta de etiquetas (*tags*) que são utilizadas para especificar os elementos das redes e as informações adicionais que podem ser utilizadas pelas aplicações que irão utilizar o modelo. As principais *tags* são:

- **net**. Utilizada para declarar uma rede cujo identificador é `net_id`:

  ```
  <net id=[net_id]> </net>
  ```

- **place**. Utilizada para declarar um lugar cujo identificador é `place_id`:

  ```
  <place id=[place_id]> </place>
  ```

- **transition**. Utilizada para declarar uma transição cujo identificador é `trans_id`:

  ```
  <transition id=[trans_id]> </transition>
  ```

- **arc**. Utilizada para declarar um arco cujo identificador é `arc_id` e que é direcionado de `source` para `target`:

  ```
  <arc id=[arc_id] source=[{trans_id|place_id}] target=[{trans_id|place_id}]> </arc>
  ```

Além das marcações listadas anteriormente, que são referentes a elementos das redes, outras *tags* importantes da linguagem são referentes a características dos elementos. A lista seguinte contém algumas delas:

- **initialMarking**. Utilizada para declarar a marcação inicial de um lugar:

  ```
  <initialMarking> </initialMarking>
  ```

- **inscription**. Utilizada para declarar inscrições dos arcos:

  ```
  <inscription> </inscription>
  ```

- **graphics**. Contém informações gráficas dos elementos a serem interpretados e renderizados de acordo com a implementação de cada ferramenta:

  ```
  <graphics> </graphics>
  ```

A Figura 4.1 apresenta exemplos da representação de dois lugares, uma transição e dois arcos escritos em PNML. As *tags* são compostas como em XML e indicam o estado dos elementos, como a quantidade inicial de *tokens* através da *tag* `<initalMarking>` e também algumas informações para renderização gráfica da rede através da *tag* `<graphics>`. A representação em PNML de algumas redes apresentadas neste trabalho pode ser vista no Apêndice A.

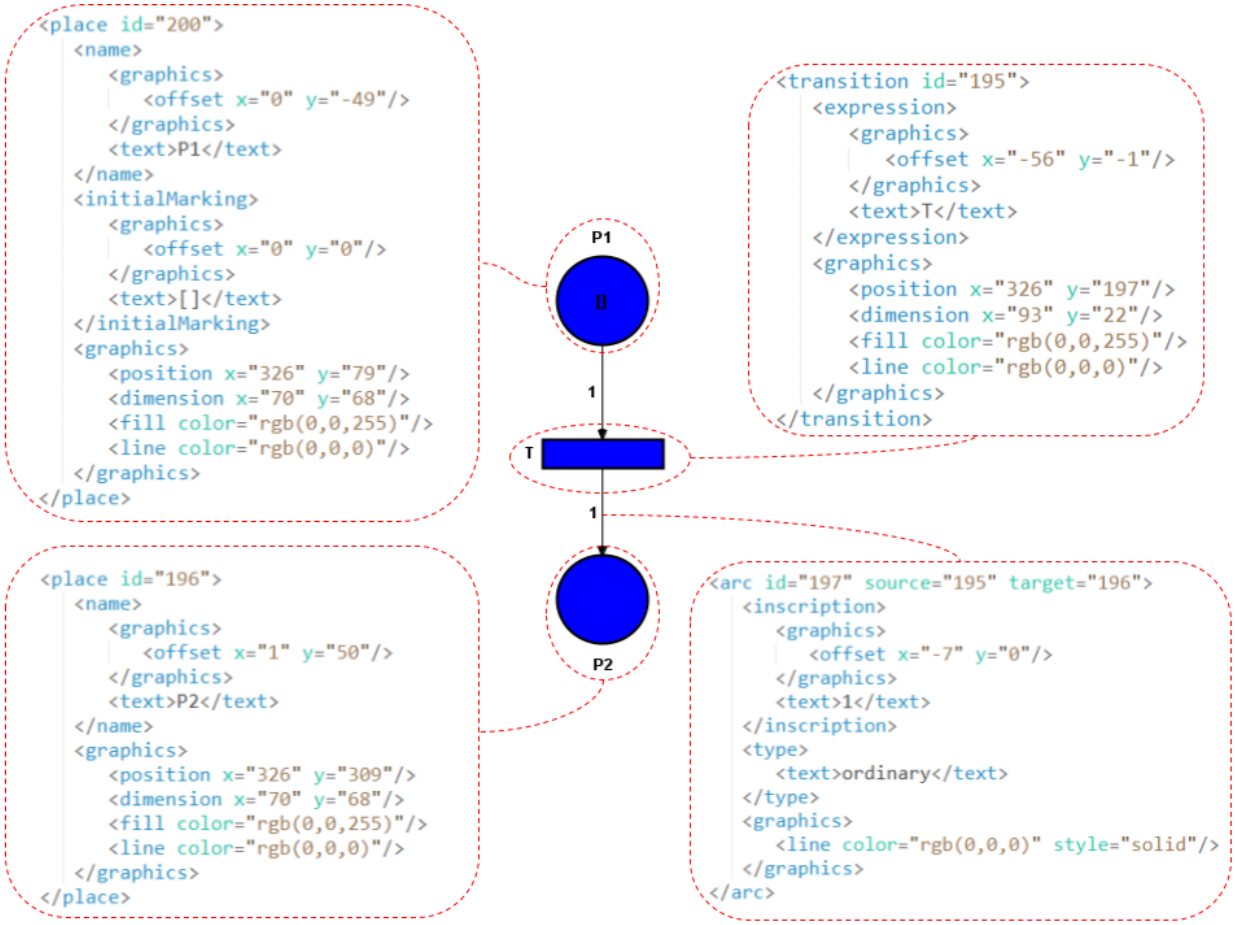

Figura 4.1: *Exemplo de código PNML onde é possível observar o uso das tags nos diferentes elementos que compõem uma Rede de Petri que, neste caso, tem dois lugares e uma transição.*

4.1.1 A extensão NPN (Nested Petri Nets)

A especificação da linguagem PNML prevê a inclusão de extensões capazes de modelar outros tipos de redes de forma a manter a padronização [2]. Qualquer extensão deve ser proposta através de um documento PNTD, escrito na linguagem de esquemas RELAX NG [12], e também através da declaração de um novo um meta-modelo UML integrado com o pacote da especificação padrão [37].

A Figura 4.2 mostra nossa proposta de criação de um novo pacote para a especificação do PNML para Redes Aninhadas onde os pacotes na cor cinza representam inclusões de pacotes terceiros que são importados ou mesclados. Para começar, são amalgamadas as definições do pacote padrão *PT-Net*, que inclui as definições do PNML *Core Model* [37]. As inclusões feitas do tipo *Multisets* e *SyncLabels* são utilizadas para sobrescrever os atributos originais das *Annotations*, *PTMarking* e *PTArcAnnotation*, permitindo que as marcações sejam multiconjuntos ou referências a outras redes (EN-Net). O pacote *SyncLabel*, definido de acordo com a Figura 4.3, é adicionado à *TransitionAnnotation*, permitindo a sincronização entre transições (*Transition*). Além disso, o atributo *type* é redefinido nos lugares (*Place*), adicionando o tipo *shared* para identificar lugares compartilhados.

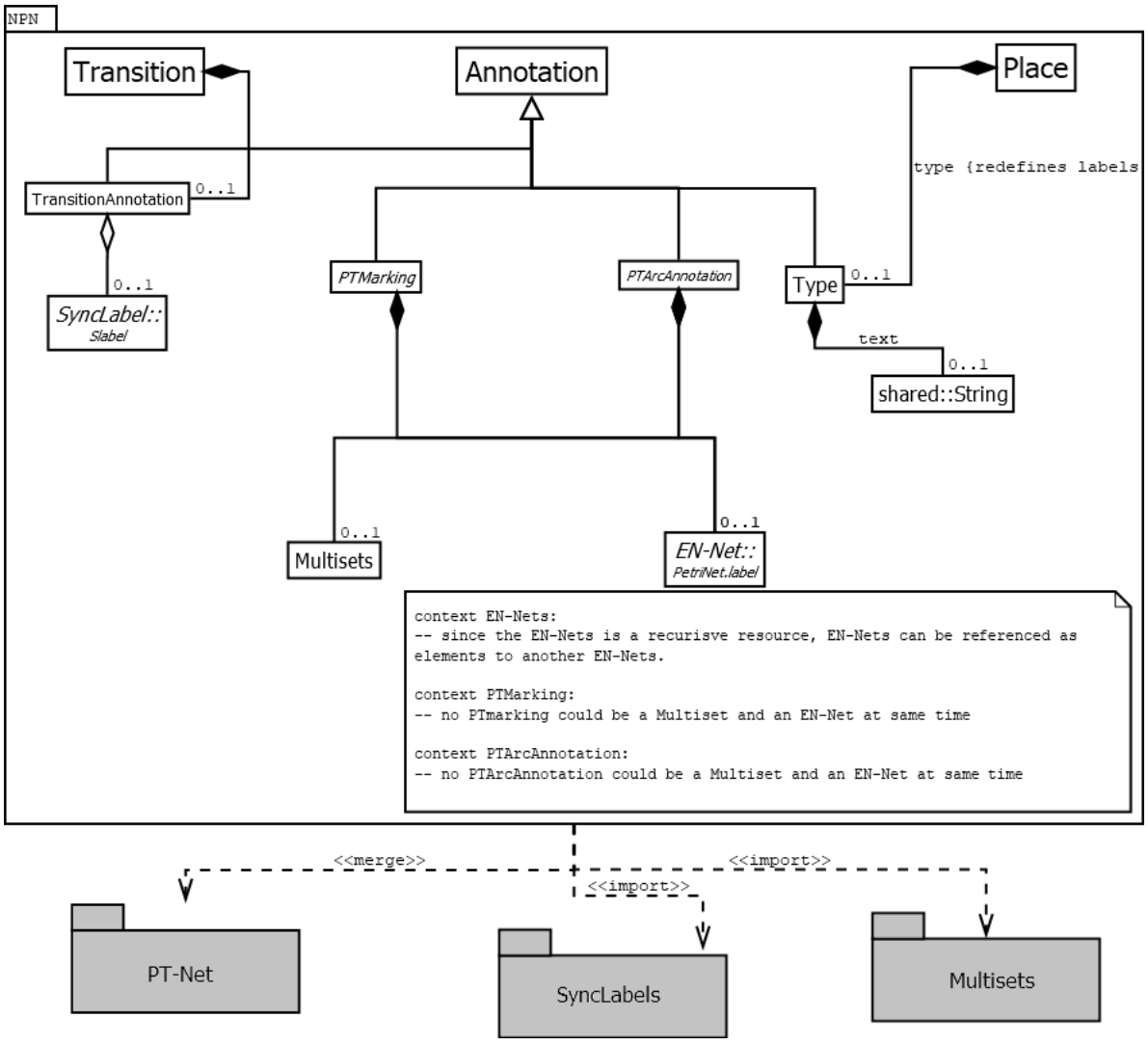

Figura 4.2: *Meta-modelo UML para a extensão NPN da PNML. Neste meta-modelo o pacote base PT-Net é complementado com as etiquetas do pacote* SyncLabels *e com a possibilidade de marcação em multiconjuntos com a inclusão do pacote* Multiset.

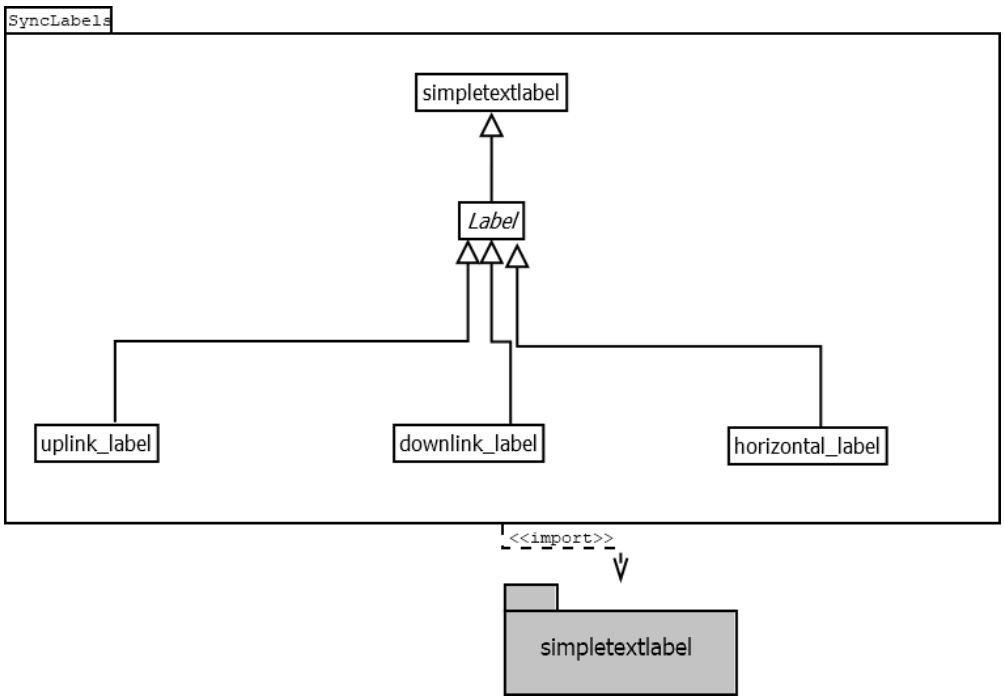

Figura 4.3: *Meta-modelo UML das etiquetas de sincronização para as transições das redes aninhadas.*

O código apresentando no Algoritmo 4.1 a seguir contém a proposta em PNTD para as redes aninhadas. Neste PNTD, assim como no meta-modelo UML, definimos as novas etiquetas para as transições e o novo tipo (*shared*) para os lugares.

```
1  <grammar datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
2    <a:documentation>
3      RELAX NG implementation of Nested Petri nets defined as a expanded form of P/T Nets with some
       ↪ aditional features allowing it to contain multisets and synchronization labels. File name:
       ↪ npn.pntd Version: 2020 (c) Gustavo Borges Lugoboni (UFABC).
4    </a:documentation>
5    <include href="http://www.pnml.org/version-2009/grammar/multisets.rng"/>
6    <include href="http://www.pnml.org/version-2009/grammar/ptnet.pntd"/>
7      <define name="ENnets">
8        <a:documentation> defining simple text reference to child nets. </a:documentation>
9        <element name="text"> <ref name="simpletextlabel.content"/> </element>
10     </define>
11     <define name="PTArcAnnotation">
12       <a:documentation> re-defining arc inscriptions including multisets. </a:documentation>
13       <element name="inscription">
14         <choice>
15           <ref name="positiveintegerlabel.content"/>
16           <ref name="multisets.content"/>
17           <ref name="ENnets"/>
18         </choice>
19       </element>
20     </define>
21     <define name="PTMarking">
22       <a:documentation> re-defining initial marking in nets including multisets. </a:documentation>
23       <element name="initialMarking">
24         <choice>
```

```
25            <ref name="nonnegativeintegerlabel.content"/>
26            <ref name="multisets.content"/>
27            <ref name="ENnets"/>
28          </choice>
29        </element>
30      </define>
31    </include>
32    <define name="nettype.uri" combine="choice">
33      <a:documentation>
34        The URI value for the net type attribute, declaring the type of P/T nets.
35      </a:documentation>
36      <attribute name="type"> <value>NPN</value> </attribute>
37    </define>
38    <define name="SyncLabels">
39      <a:documentation> Defining the synchronization labels. </a:documentation>
40      <choice>
41        <element name="downlink_label"> <ref name="simpletextlabel.content"/> </element>
42        <element name="uplink_label"> <ref name="simpletextlabel.content"/> </element>
43        <element name="horizontal_label"> <ref name="simpletextlabel.content"/> </element>
44      </choice>
45    </define>
46    <define name="Type">
47      <a:documentation>
48        Defining the 'Type' label for a place with the "shared" type option.
49      </a:documentation>
50      <element name="type"> <value>shared</value> </element>
51    </define>
52    <define name="TransitionAnnotation">
53      <a:documentation> assigning the synchonization labels for transitions. </a:documentation>
54      <ref name="SyncLabels">
55    </define>
56    <define name="transition.labels" combine="interleave">
57      <a:documentation>
58        A transition can be labeled with some synchronization labels.
59      </a:documentation>
60      <ref name="TransitionAnnotation"/>
61    </define>
62    <define name="place.labels" combine="interleave">
63      <a:documentation> A place of NPN can be typed as a shared place. </a:documentation>
64      <optional> <ref name="Type"/> </optional>
65    </define>
66  </grammar>
```

Algoritmo 4.1: *Proposta de esquema em PNTD para as redes aninhadas.*

De forma pragmática, a expansão proposta consiste da inclusão das seguintes *tags*:

- **shared**. Utilizada dentro das *tags* `place`, indica que o tipo daquele lugar é compartilhado, de acordo com a Definição 2.6.1:

```
<type> <text> shared </text> </type>
```

- **downlink_label**. Utilizada dentro das *tags* `transition`, indica que a transição contém uma

etiqueta de sincronização vertical e a rede que contém esta transição é a rede que contém os *tokens* de rede que serão sincronizados:

```
<downlink_label> <text> LABEL </text> </downlink_label>
```

- **uplink_label**. Utilizada dentro das *tags* `transition`, indica que a transição contém uma etiqueta de sincronização vertical e a rede que contém esta transição é a rede que será o *token* em uma outra rede:

```
<uplink_label> <text> LABEL </text> </uplink_label>
```

- **horizontal_label**. Utilizada dentro das *tags* `transition`, indica que a transição contém uma etiqueta de sincronização horizontal:

```
<horizontal_label> <text> LABEL </text> </horizontal_label>
```

O uso desta extensão proposta pode ser verificado no Apêndice A, através do código PNML da rede da Figura 2.7.

4.2 A linguagem PROMELA e o verificador de modelos SPIN

O verificador de modelos SPIN[1] é um eficiente sistema de verificação de modelos e *software* distribuído. É um sistema genérico de verificação que suporta o desenho e a verificação de sistemas com processos assíncronos [39]. O SPIN funciona através de uma linguagem utilizada para modelar os processos a serem verificados chamada PROMELA. Essa linguagem tem sintaxe inspirada em C e suprime detalhes que não estão relacionados à interação dos processos. Dessa forma, o comportamento que se pretende analisar é modelado em PROMELA e o SPIN é então utilizado para verificar as propriedades pretendidas. As declarações, expressões e atribuições seguem a sintaxe padrão de C e outras linguagens de programação. No entanto, as instruções `do` e `if` consistem em várias ramificações (chamadas de opções) que podem ser selecionadas não deterministicamente.

Um programa PROMELA tem três principais componentes: processos, canais de mensagem e variáveis. Os processos são definidos como objetos globais, já os canais e as variáveis podem ser globais ou locais dentro do escopo de um processo. Cada processo pode ser instanciado de forma individual dentro de um processo global chamado `init` e cada programa PROMELA deve conter um processo do tipo `init`. Devido às características decorrentes da natureza concorrente de processos paralelos, o PROMELA permite ainda a declaração de sequências atômicas dentro dos processos através da palavra chave `atomic`.

Um processo pode criar outras instâncias de processos que são executadas simultaneamente. Para este fim, o operador `run` é usado, o qual devolve o número da instanciação do processo. Este número é armazenado na variável predefinida, local e de leitura `_pid`. Os processos podem se comunicar enviando e recebendo mensagens através de canais, que podem ser assíncronos (*buffered*) ou síncronos (*rendezvous*). As instruções padrão de envio e recebimento (`!` e `?`, resp.) assumem os canais em *buffer* como listas *fifo*. O PROMELA oferece várias funções predefinidas para canais em *buffer*, como por exemplo `len` para o número de mensagens e `empty`. Ele também inclui outras variantes de envio e recebimento, como o envio classificado (`!!`), para enfileiramento ordenado de mensagens, e o recebimento aleatório (`??`), para remoção aleatória de mensagens. Há ainda

[1] http://spinroot.com/

operadores livres de efeitos colaterais para testar a presença de uma mensagem em um canal em *buffer*. Para um canal síncrono, o tamanho do *buffer* é zero. Como nenhuma mensagem pode ser armazenada, as operações de envio e recebimento por meio de tal canal devem ser executadas simultaneamente por duas instâncias de processo.

Os canais de mensagem têm por objetivo a troca de dados entre os processos e podem carregar um número declarado de dados de um determinado tipo. Por exemplo, a declaração `chan qname = [16] of {short}` corresponde a um canal chamado `qname` com a capacidade de 16 mensagens do tipo `short`. Uma instrução em um processo pode ser executada se o mesmo estiver habilitado; caso contrário, ela é bloqueada. Declarações, atribuições e saltos estão sempre habilitados. Para as demais declarações, existem condições para sua executabilidade. Por exemplo, uma expressão é habilitada se for avaliada como um valor diferente de zero. Uma sequência de instruções é habilitada se a primeira instrução for habilitada. Uma instrução condicional ou de repetição é habilitada se pelo menos uma de suas opções estiver habilitada. Nesse caso, uma opção habilitada é escolhida não deterministicamente. A opção especial prefixada por `else` é habilitada se e somente se nenhuma outra opção for habilitada. Se a instrução atual estiver bloqueada, a execução do processo é suspensa até que a instrução seja habilitada.

O paralelismo no PROMELA possui uma semântica de intercalação assíncrona. Portanto, declarações de diferentes processos não são executadas simultaneamente (exceto a comunicação síncrona). Além disso, a execução de um processo pode ser interrompida por qualquer outro processo habilitado. No entanto, as declarações em uma sequência `atomic` não são intercaladas, a menos que uma delas seja bloqueada. A primeira declaração de uma sequência `atomic`, ou uma opção, é chamada de *guard* e normalmente é separada das instruções restantes usando o símbolo `->`. A ordem de execução dos processos habilitados é não determinística, no entanto, os processos podem ter prioridades. Nesse caso, uma instrução ou processo habilitado é selecionado se não houver outra instrução ou processo habilitado com prioridade mais alta.

O código PROMELA apresentado no Algoritmo 4.2 foi gerado a partir da rede modelada em PNML da Figura 2.1, e o código PNML pode ser consultado no Apêndice A. Nele é possível identificar a execução de duas sequências `atomic` equivalentes ao disparo das transições que modificam o valor das variáveis do tipo `byte pP1`, `pP2` e `pP3`, que correspondem aos lugares e são declaradas de forma a refletir sua marcação inicial. Neste caso, devido ao fato de que os *tokens* da rede são apenas fichas pretas, pode-se representá-los como números naturais, conforme a Definição 2.1.2. Ao final da verificação de um programa PROMELA, o SPIN devolve um relatório com o estado final dos processos, canais e variáveis, assim como outras asserções testadas.

No que tange as propriedades das Redes de Petri apresentadas na Seção 2.2, através das duas propriedades verificadas pelo SPIN (propriedades de lógica temporal e busca de ciclos aceitáveis, ou, ciclos de aceitação), é possível construir testes para verificar as das Redes de Petri. A alcançabilidade e, por consequência, a terminabilidade, ou seja, se a rede alcançou um estado final de aceitação, podem ser verificadas por meio de testes de lógica temporal linear [3], assim como a limitabilidade, onde uma condição de lógica temporal linear pode ser colocada para cada lugar garantindo que a marcação destes seja limitada a uma constante k. Para isso, é necessário utilizar os modificadores `ltl` para declaração de testes por meio de lógica temporal linear e `accept` para marcação de trechos de código que devem ser finalizados em estado de aceitação. Além disso, o modificador `end` deve ser utilizado para marcar um estado final de aceitação para um determinado processo, declarado como

```
active proctype P() {
byte pP1 = 2;
byte pP2 = 0;
byte pP3 = 0;
do
  :: atomic {
       pP3 >= 1 && pP2 >= 2 ->
         pP3 = pP3 - 1;
         pP2 = pP2 - 2;
         pP1 = pP1 + 2;
         printf("\nFiring T2 -> Places status: pP1 = %d pP2 = %d pP3 = %d", pP1, pP2, pP3);
  }
  :: atomic {
       pP1 >= 1 ->
         pP1 = pP1 - 1;
         pP2 = pP2 + 2;
         pP3 = pP3 + 1;
         printf("\nFiring T1 -> Places status: pP1 = %d pP2 = %d pP3 = %d", pP1, pP2, pP3);
  }
  od
}
```

Algoritmo 4.2: *Código PROMELA da rede da Figura 2.1.*

o tipo nativo do PROMELA, `proctype`, para designar processos. A vivacidade, por outro lado, como antes dito, não pode ser verificada facilmente, mas é possível, por meio da testes de lógica temporal linear, verificar se alguma transição é morta, além de se utilizar ciclos de aceitação para os laços nas redes que não terminam.

4.3 Fundamentos da modelagem de Redes de Petri Aninhadas em PROMELA

A modelagem das Redes de Petri Aninhadas em código PROMELA foi inicialmente proposta por Venero e Corrêa da Silva em 2013 [83] e contou com uma revisão e ampliação posterior pelos autores em 2014 [84] e 2016 [85]. Estas propostas servirão como base para a nossa versão, contudo, as redes modeladas pela metodologia por eles na primeira publicação [83] diferem das redes aninhadas da Definição 2.6.1, conforme já discutido em detalhes na Sessão 2.7. A principal diferença reside no fato de que as redes aninhadas definidas por Venero e Corrêa da Silva consomem os *tokens* de rede enquanto as redes da Definição 2.6.1 permitem o transporte destes entre os lugares.

As formulações posteriores são muito mais complexas pois abordam problemas de desempenho na execução do SPIN para verificação das redes modeladas e, especialmente no trabalho mais recente [85], a definição das redes foi ampliada para atender redes com transporte de *tokens* e sincronização horizontal entre mais de duas redes. Como resultado, cada instância de rede aninhada nos exemplos dos trabalhos citados foi convertida para o PROMELA de forma distinta, dificultando o trabalho de se criar um tradutor automático sem alterar alguns pontos de cada iteração apresentada.

Por meio da metodologia de modelagem aqui proposta, todos os elementos e o comportamento das redes aninhadas terão elementos equivalentes em PROMELA com base em sua representação PNML. Com o objetivo de tornar a tradução automática através de uma aplicação tendo como

objeto de entrada um modelo em PNML, faz-se necessário uma padronização na metodologia de modelagem. Esta padronização será proposta a seguir por meio de uma fusão entre os métodos citados com uma simplificação.

Para realizar esta síntese, o trabalho mais recente de Venero e Corrêa da Silva foi tomado como base para a proposta apresentada a seguir. Contudo, fizemos uma simplificação fundamental, que é a utilização de apenas um canal global para sincronização entre os *tokens*. Isso tráz como consequência imediata a imposição de sincronização entre apenas dois *tokens* por vez. Além disso, a estrutura dos laços utilizada será baseada naquela do primeiro trabalho [83]. Esta simplificação pode acarretar em um código PROMELA menos otimizado, sem os ajustes finos específicos de cada modelo, contudo possibilitará a automação desta tradução.

A modelagem de uma Rede de Petri Aninhada em PROMELA pode ser compreendida de maneira sintética através da tradução dos principais componentes das redes:

- **Redes**: Seguindo a Definição 2.6.1, cada rede elemento $EN_1, \ldots, EN_n$ é declarada como um processo `proctype`. Por sua vez, a rede sistema EN_0 é traduzida como o processo principal `init`. Cada processo das redes elemento tem como estrutura dois laços, sendo um interno contendo sequências atômicas de código para cada transição, e um externo contendo as condições de sincronização entre as transições de diferentes redes. O processo principal (`init`), que representa a rede sistema, contém apenas um laço com sequências atômicas para as transições, pois a sincronização entre as transições da rede sistema e as transições das redes elemento acontece de forma *top-down*, não necessitando assim de indicadores complementares de resposta (mensagens de *acknowledge*) para a sincronização.

- ***Tokens***: Os *tokens* de rede são modelados como mensagens que contêm quatro campos, com o primeiro sendo o identificador do processo da rede (`_pid`), o segundo sendo o identificador da etiqueta da transição, o terceiro sendo o identificador da transição, e o último sendo uma *flag* de sincronização. Já os *tokens* pretos são representados por números inteiros que indicam a quantidade destes.

- **Transições e Etiquetas**: Cada transição do sistema é modelada como uma declaração condicional `if` onde a condição para execução é a condição para habilitação do disparo da transição. O disparo da transição resulta em ações de consumo e produção, como exemplificado no Algoritmo 4.3.

  A declaração das etiquetas é feita de forma que cada etiqueta, vertical ou horizontal, é declarada como um número entre 1 e 254. As etiquetas 0 e 255 são reservadas para representar as instâncias dos *tokens*.

- **Lugares**: Os lugares de tipos básicos, um tipo definido como *token* preto sem informação, são traduzidos como variáveis numéricas do tipo `byte`. Lugares de redes ou de *tokens* com tipos distintos do básico, que carregam informação (*tokens* coloridos), são traduzidos como um novo tipo por meio do `typedef`. Esse tipo novo é composto de um canal de comunicação do tipo `chan`, que se comunica através de mensagens no formato `{byte, byte, byte, bit}`, onde os campos representam, respectivamente, o `_pid` do *token* de rede alocado neste lugar, o identificador da etiqueta da transição, o identificador da transição, e a *flag* de sincronização (0 para *request* e 1 para *response*). Os lugares compartilhados são declarados como variáveis globais e podem ser modificados por qualquer processo.

```
:: d_step {
   #### TRANSITION_EN ENABLE CONDITIONS ####
   P1 > 1 ->

   #### TRANSITION CONSUME ACTIONS ####
      P1 = P1 - 1;

   #### TRANSITION PRODUCE ACTIONS ####
      P2 = P2 + 2;
      P3 = P3 + 1;
}
```

Algoritmo 4.3: *Estrutura de modelagem de uma transição de uma Rede de Petri em PROMELA. Neste exemplo, a transição T1 da rede da Figura 2.1 é traduzida e é possível observar a estrutura básica de expressão condicional, seguido das ações de consumo e produção. O comando* `d_step` *em PROMELA designa uma sequência determinística indivisível de código.*

O Algoritmo 4.4 contém a estrutura básica da modelagem de uma Rede de Petri Aninhada em PROMELA de acordo com Venero e Corrêa da Silva [83], com as modificações necessárias para uma automatização do processo de tradução. A estrutura é composta, inicialmente, pelas declarações dos parâmetros globais ao projeto, como o tipo `NetPlace`, que será utilizado para declarar variáveis que representam os lugares que aceitam *tokens* de rede, seguido de funções pré-definidas para facilitar a tradução como, por exemplo, uma função para transportar *tokens* de rede entre lugares. Em seguida as declarações das etiquetas e variáveis globais são feitas e, após estas, são declarados os processos das redes compostas pelos laços que testam as condições de habilitação de cada transição.

O identificador de sequência atômica da PROMELA, `d_step`, foi utilizado nas redes elemento em detrimento do identificador `atomic` por ser mais eficiente durante a verificação do modelo pelo SPIN[2]. Na rede sistema, `atomic` foi utilizado pois não é possível executar outro processo dentro de uma sequência de identificador `d_step` e, portanto, não seria possível executar os processos redes elementos. Esta condição foi contornada nas redes elemento através da execução de novos processos dentro das sequências do laço externo. Na tradução proposta por Venero e Corrêa da Silva [83], os identificadores eram intercambiáveis, de forma que `d_step` poderia ser utilizado em qualquer sequência de qualquer rede com a condição de que nenhum processo fosse executado nesta sequência. Em nossa proposta, padronizamos o uso do `d_step` nas redes elementos para possibilitar a automatização do processo de tradução.

Para a comunicação entre os processos, um canal global (`gbchan`) é utilizado e, através deste, todas as mensagens de sincronização são trocadas entre as redes. Cada processo de rede (`proctype`) também recebe como parâmetro de entrada um canal para comunicação. Contudo, em nossa proposta, apenas as mensagens que representam *tokens* são transmitidas nestes canais para verificação do estado da rede. Em casos de conflito entre duas transições habilitadas simultaneamente com o mesmo lugar de entrada, a transição que dispara deve desabilitar a outra transição através da remoção das mensagens enviadas por aquela no canal global.

As ações de produção e consumo são dependentes do tipo de lugar em que terão efeito, de acordo com os arcos de entrada e saída. As ações de produção em lugares modelados como canais são traduzidas como envio de mensagens que representam os *tokens* que irão ser alocados nestes. Caso seja um *token* de rede, a produção deverá instanciar o processo que representará o *token* e a

[2] http://spinroot.com/spin/Man/d_step.html

```
################## GLOBAL PARAMETERS ##################
################ PREDEFINED FUNCTIONS #################
############## TRANSITION VERTICAL LABELS ##############
############# TRANSITION HORIZONTAL LABELS #############
#################### SHARED PLACES ####################
################ SYSTEM NET VARIABLES #################
################ SYSTEM NET INIT PROC #################
init {

   atomic {
      ########### INITIAL MARKING #############
   }

   do ########### MAIN LOOP #############
      :: atomic {
            #### TRANSITION_A ENABLE CONDITIONS #### ->
               sP(_pid, 3);
               #### TRANSITION CONSUME ACTIONS ####
               #### TRANSITION PRODUCE ACTIONS ####
               sP(_pid, 1);
      }
   od
}

################ ELEMENT NETS PROC ##################
proctype Element_net_name (chan pc) {

   ############# LOCAL VARIABLES ###############

   atomic {
      ########### INITIAL MARKING #############
   }
   do
      :: {
         do ########### LOCAL INNER LOOP #############
            :: d_step {
                  #### TRANSITION_EN ENABLE CONDITIONS ####  ->
                     #### TRANSITION CONSUME ACTIONS ####
                     #### TRANSITION PRODUCE ACTIONS ####
            }
         od
      }

      unless atomic {
         ### SYNCHRONIZATION CONDITIONS ###
         ### LOCAL OUTER LOOP ###
         if
            :: ### SYNCHRONIZATION CONDITION A ###  ->
                  #### TRANSITION CONSUME ACTIONS ####
                  #### TRANSITION PRODUCE ACTIONS ####
            :: ### SYNCHRONIZATION CONDITION B ###  ->
                  #### TRANSITION CONSUME ACTIONS ####
                  #### TRANSITION PRODUCE ACTIONS ####
         fi;
         sP(_pid, 1);
      }
   od;
   sP(_pid, 1);
}
```

Algoritmo 4.4: *Estrutura básica da modelagem em PROMELA de uma Rede de Retri Aninhada (Definição 2.6.1), onde sP representa a função* `set_priority` *do PROMELA.*

```
### PRODUCTIONS ###
:: atomic {
      PRE_CONDITIONS  ->
         sP(_pid, 3);
         BLACK_TOKEN_PLACE = BLACK_TOKEN_PLACE + 1;
         nt = run NETWORK(CHANNEL_PLACE.channel);
         CHANNEL_PLACE.channel ! nt,255,0,0;
         sP(_pid, 1);
}

### CONSUPTIONS ###
:: atomic {
      PRE_CONDITIONS  ->
         sP(_pid, 3);
         BLACK_TOKEN_PLACE = BLACK_TOKEN_PLACE - 1;
         CHANNEL_PLACE.channel ?? NET_TOKEN_PID,255,_,0;
         sP(_pid, 1);
}
```

Algoritmo 4.5: *Estrutura de modelagem de ações de produção e consumo para lugares de tipos diferentes. A primeira sequência atomic contém produções e a segunda sequência contém ações de consumo.*

mensagem enviada para o canal que representa o lugar deverá conter o *pid* do novo processo.

O consumo em lugares modelados como canais deve ser traduzido como uma operação de remoção da mensagem representativa do *token* destes e, caso o *token* represente uma rede, uma mensagem pode ser enviada ao canal global com um sinal para que o processo termine. Porém, encerrar o processo pode ser arriscado pois este pode terminar antes de executar alguma ação necessária, caso a prioridade entre os passos da rede não seja bem balanceada.

Já em lugares de *tokens* sem informação, que são traduzidos como variáveis de tipos numéricos, a produção e o consumo consistem simplesmente da soma e subtração, respectivamente, na quantidade de *tokens* daquele lugar. O Algoritmo 4.5 contém o formato das operações de consumo e produção de forma a ilustrar a explicação.

A sincronização entre as transições, que é realizada por meio do canal global, ocorre nas direções vertical e horizontal, seguindo a Definição 2.6.3. A sincronização vertical é modelada em dois sentidos, *downlink* e *uplink*, e a sincronização horizontal é modelada a partir de um par de testes, da mesma forma que foi feito por Venero e Corrêa da Silva [84].

Cada transição com etiqueta de sincronização deve ter, em suas condições de disparo, um teste de existência de mensagem de sincronização no canal global, de acordo com a estrutura dos Algoritmos 4.6, 4.7 e 4.8, e detalhado a seguir:

- **Sincronização vertical *downlink*.** Em uma sincronização vertical *downlink*, a sequência de código que descreve os testes e ações da transição pode ser do tipo `atomic`, caso seja uma transição de uma rede sistema, ou `d_step`, caso em que as ações devem ser declaradas no laço externo. O teste consiste em verificar se os lugares de entrada contêm *tokens* e se existe mensagem com o identificador da etiqueta no canal de sincronização com o bit de sincronização com o valor 0. Caso a condição seja satisfeita, estas mensagens serão retiradas do canal global e mensagens com o bit final de sincronização com o valor 1 serão enviadas com o identificador da etiqueta e das transições a fim de que as ações declaradas no laço externo, tanto da rede com o *downlink* quanto da rede que enviou a mensagem de sincronização, sejam executadas.

```
:: atomic (or d_step) {
      PRE_CONDITIONS && nempty(INPUT_PLACE.d) && gbchan ?? [_,LABEL_ID,_,0]  ->
         sP(_pid, 3);
         gbchan ?? NT_PID,LABEL_ID,it,0;
         gbchan ! NT_PID, LABEL_ID, it, 1;
         sP(NT_PID, 3)

         #### IF ATOMIC ####
         #### TRANSITION CONSUME ACTIONS ####
         #### TRANSITION PRODUCE ACTIONS ####

         ### IF DSTEP ###
         gbchan ! _PID, LABEL_ID, TRANSITION_ID, 1;
         sP(_pid, 1);
}
### IF DSTEP ###
    .
    .
    .
unless atomic {
   gbchan ?? eval(_pid),_,it,1
   if :: it == TRANSITION_ID ->
            #### TRANSITION CONSUME ACTIONS ####
            #### TRANSITION PRODUCE ACTIONS ####
}
```

Algoritmo 4.6: *Estrutura de modelagem da sincronização via* downlink *em uma Rede de Petri Aninhada em PROMELA.*

Para isso, a prioridade da rede que irá executar a ação complementar é elevada para 3.

- **Sincronização vertical *uplink*.** Em uma sincronização vertical *uplink*, o teste da transição consiste em verificar se não existe nenhuma mensagem de solicitação de sincronização no canal de comunicação e, caso não exista, criar uma mensagem e a enviar. As ações executadas pelo disparo da transição são declaradas no laço externo e são executadas somente quando uma resposta à mensagem de sincronização é dada por alguma transição *downlink*.

- **Sincronização horizontal.** Na sincronização horizontal, todas transições contendo uma etiqueta de sincronização devem ser modeladas em PROMELA da mesma forma. Duas sequên-

```
:: d_step {
      PRE_CONDITIONS && nempty(INPUT_PLACE.d) && ! gbchan ?? [_,LABEL_ID,_,0]  ->
         gbchan ! _PID, LABEL_ID, TRANSITION_ID, 0;
}
.
.
.
unless atomic {
   gbchan ?? eval(_pid),_,it,1
   if :: it == TRANSITION_ID ->
            #### TRANSITION CONSUME ACTIONS ####
            #### TRANSITION PRODUCE ACTIONS ####
}
```

Algoritmo 4.7: *Estrutura de modelagem da sincronização via* uplink *em uma Rede de Petri Aninhada em PROMELA.*

```
:: d_step{
      PRE_CONDITIONS && ! gbchan??[eval(_PID),LABEL,_,0] && gbchan??[_,LABEL,_,0]  ->
         gbchan ?? NT_PID,LABEL,_,0;
         gbchan ! _PID,LABEL,TRANSITION_ID,0;
         gbchan ! _PID,LABEL,TRANSITION_ID,1;
}
:: d_step {
      PRE_CONDITIONS && ! gbchan??[_,LABEL,_,0]  ->
         gbchan!_PID,LABEL,TRANSITION_ID,0;
}
.
.
.
unless atomic {
   gbchan ?? eval(_pid),_,it,1
   if :: it == TRANSITION_ID ->
            #### TRANSITION CONSUME ACTIONS ####
            #### TRANSITION PRODUCE ACTIONS ####
            gbchan ?? eval(_pid),_,it,0;
}
```

Algoritmo 4.8: *Estrutura de modelagem da sincronização horizontal em uma Rede de Petri Aninhada em PROMELA.*

cias devem ser incluídas no laço interno para a transição. Uma destas sequências consiste na verificação de que nenhuma mensagem de sincronização para a transição foi enviada pelo processo de rede e seguida da criação e envio da mensagem. A outra sequência verifica se outro processo de rede enviou uma mensagem e, caso sim, a mensagem deve ser retirada do canal. Em seguida, novas mensagens com o identificador do processo da rede que retirou as mensagens anteriores devem ser enviadas para o canal. Estas mensagens deverão acionar as ações da transição declaradas no laço externo desta rede e habilitar a execução na rede que enviou a primeira mensagem.

Com isso, concluímos a estrutura geral da modelagem de uma rede aninhada em PROMELA. Através desta, é possível verificar que cada elemento das Redes de Petri Aninhadas (arcos, lugares, transições, *tokens*, regras de disparo e mecanismos de sincronização) teve seu comportamento mapeado para o PROMELA. A questão que se coloca então é se o modelo de construção aqui proposto é capaz de, de fato, traduzir o comportamento dinâmico esperado das redes, levando-se em conta o grau de indeterminismo em que a ordem de uma sequência de disparos pode ocorrer. No Seção seguinte apresentaremos uma implementação desta abordagem em uma proposta de tradutor automático, assim como uma implementação básica em um pequeno exemplo para verificar o comportamento destas redes na prática.

4.4 Implementação da tradução

A tradução foi implementada em linguagem Python, versão 3.8[3], e o programa foi dividido em três partes. A primeira parte consiste em processar os arquivos de entrada em PNML e extrair as informações importantes, organizando-as em dicionários intermediários contendo as lógicas imediatas de relacionamento entre os principais elementos da rede (lugares, arcos e transições). A segunda

[3] https://www.python.org/

parte processa os dicionários intermediários de forma a realizar a análise das marcações iniciais, verificar as condições de disparo das transições, e realizar as sequências de consumo e produção. Por fim, a última parte do programa organiza as informações produzidas nas duas outras partes na estrutura do código PROMELA, conforme apresentado no Algoritmo 4.4, e, ao final, escreve o resultado da execução do tradutor em um novo arquivo.

Apesar da especificação PNML NPN apresentada na Seção 4.1.1 permitir o uso de multiconjuntos para tipagem dos *tokens*, o tradutor foi construído e testado seguindo a abordagem de Venero e Corrêa da Silva [83]. Isso significa que o uso de apenas um tipo genérico além dos tipos de redes foi considerado já que, segundo os autores, lugares com tipos em multiconjuntos podem ser desdobrados em múltiplos lugares do mesmo tipo básico. Esta simplificação da tradução, portanto, não traz um prejuízo inicial quanto ao conjunto de redes possíveis de serem modeladas, mas oferece um ganho de redução de complexidade para a construção inicial desta ferramenta.

O Algoritmo 4.9 apresenta a função principal da primeira parte da implementação da tradução. Inicialmente, os arquivos PNML que devem ser passados como argumentos para o programa têm suas informações básicas extraídas validadas, como o nome e a extensão. Os dicionários intermediários são então instanciados. Cada dicionário contém informações a respeito da estrutura da rede organizadas de maneiras diferentes, por questões de desempenho, e é preenchido durante a execução do laço que percorre as árvores geradas pela execução da função `initial_trees = [(ElementTree.parse(INPUT_FILE), INPUT_FILE.split('.')[0]) for INPUT_FILE in INPUT_FILES]`, que utiliza o método `ElementTree` do pacote xml do Python para organizar documentos em árvores a partir de suas *tags*.

As funções `parse_transitions_and_labels`, `parse_places` e `parse_arcs` são responsáveis por realizar a análise dos documentos PNML, buscando as partes relacionadas aos principais elementos (arcos, lugares e transições) e preenchendo os dicionários passados como parâmetro de entrada para cada uma delas. O Algoritmo 4.10 contém a função `parse_places`, onde é possível observar a estrutura básica da extração dos trechos para o preenchimento dos dicionários intermediários. Inicialmente é criada a lista `parsed_place_elements`, com cada lugar extraído da rede através da *tag* `place`, representada pela constante `PLACE_TAG_MODEL_STRING` no código pois o formato do PNML pode influenciar na extração desta *tag* pelo método `ElementTree`. Esta lista é criada a partir da extração dos elementos contidos nas *tags* `id`, `name`, `initialMarking` e `type` dos documentos PNML. Ela é posteriormente percorrida de forma a ter seus dados estruturados nos dicionários de entrada de acordo com as informações importantes como marcação e tipo do lugar.

O Algoritmo 4.11 contém a função da segunda parte do tradutor, que gera a marcação inicial em PROMELA a partir dos dicionários intermediários. A função, a partir do dicionário com os lugares passados como parâmetro de entrada, verifica o tipo de cada lugar no contexto da rede. Caso seja um `channel_place`, um lugar com *tokens* representados por canais do PROMELA, uma declaração do tipo `NetPlace` é gerada e, para cada *token* de rede em sua marcação inicial, uma declaração de execução do processo deste é gerada na forma da uma string formatada. A seguir, os lugares não compartilhados, que são do tipo que não recebem *tokens* de rede, são declarados como `byte` por meio da sentença em forma de string formatada.

Em seguida, para cada transição do sistema o tradutor verifica as condições de disparo e as sequências de consumo e de produção. O processamento no código é realizado através de um laço declarado na função `translate_by_inter_dictionaries` que percorre cada transição e, para cada uma,

```
def criar_nets_info(redes):
   nets_info = novo_dicionario()
   net_label = 1
   para cada rede em redes:
      se rede != rede_sistema:
         nets_info[rede] = novo_dicionario()
         nets_info[rede]['label'] = net_label
         net_label += 1
         nets_info[rede]['tokens'] = novo_conjunto()
         nets_info[rede]['transitions_labels'] = novo_conjunto()
      senão:
         nets_info[rede] = novo_dicionario() caso rede não esteja em nets_info.chaves()
         nets_info[rede]['tokens'] = novo_conjunto()
         nets_info[rede]['transitions_labels'] = novo_conjunto()
         nets_info[rede]['id'] = 0
   devolva nets_info

def init():
   args = pegar_argumentos_de_entrada()
   INPUT_FILES = parsear_nomes_e_criar_lista_com_nomes_dos_arquivos_de_entrada(args)
   splited_filenames = criar_lista_com_arquivos_e_extensao_separadas(INPUT_FILES)

   para cada filename em splited_filenames:
      se tamanho(filenames) < 2:
         devolva Erro

   BASE_FILENAMES = criar_lista_com_nomes_dos_arquivos()
   EXTENSIONS_FILENAMES = criar_conjunto_com_extensoes_dos_arquivos()

   verificar_extensoes(EXTENSIONS_FILENAMES)

   transition_dicts = novo_dicionario_a_partir_das_chaves(BASE_FILENAMES)
   place_dicts = novo_dicionario_a_partir_das_chaves(BASE_FILENAMES)
   only_places_dict = novo_dicionario()
   arc_dicts = novo_dicionario_a_partir_das_chaves(BASE_FILENAMES)
   only_arcs_dict = novo_dicionario()
   only_transitions_dict = novo_dicionario()
   nets_info = criar_nets_info(BASE_FILENAMES)
   uplink = novo_dicionario()
   downlink = novo_dicionario()
   horizontal = novo_dicionario()
   shared_places = novo_conjunto()
   arc_variables = novo_conjunto()

   initial_trees = abrir_arquivos_pnml(INPUT_FILES)

   para cada arquivo em initial_trees:
      parse_transitions_and_labels(arquivo, transition_dicts, nets_info, uplink, downlink,
      ↪ horizontal, only_transitions_dict)
      parse_places(arquivo, place_dicts, nets_info, shared_places, only_places_dict)
      parse_arcs(arquivo, arc_dicts, nets_info, arc_variables, only_arcs_dict)

   vlabels, hlabels = criar_dicionarios_para_sincronizacao(uplink, horizontal)

   devolva BASE_FILENAMES[0], place_dicts, transition_dicts, arc_dicts, nets_info, uplink,
   ↪ downlink, horizontal, vlabels, hlabels, shared_places, arc_variables, only_places_dict,
   ↪ only_arcs_dict, only_transitions_dict
```

Algoritmo 4.9: *Função de extração das informações dos arquivos PNML, do início da tradução.*

```
def parse_places(arquivo, place_dicts, nets_info, shared_places, only_places_dict):
   place_dict = novo_dicionario()
   raw_place_elements = extrair_elementos_place_do_arquivo(arquivo)
   id_rede = extrair_id_rede(arquivo)
   parsed_place_elements = nova_lista()

   para cada item em raw_place_elements:
      lista_temp = nova_lista()
      lista_temp.adicionar(nets_info[id_rede]['id'] + raw_place_elements[item].attrib['id'])
      lista_temp.adicionar(extrair_elemento_com_o_nome(raw_place_elements[item], 'name'))
      lista_temp.adicionar(extrair_elemento_com_o_nome(raw_place_elements[item], 'initialMarking'))
      lista_temp.adicionar(extrair_elemento_com_o_nome(raw_place_elements[item], 'type'))
      parsed_place_elements.adicionar(lista_temp)

   para cada place em parsed_place_elements:
      marking = novo_dicionario()
      se place[2] != Vazio:
         para cada mark em place[2]:
            se mark já está em marking:
               marking[mark] = marking[mark] + 1
            senão:
               marking[mark] = 1
            nets_info[id_rede]['tokens'].adicionar(mark)
         place[2] = marking

      se place[3] != Vazio e place[3][0] == "shared":
         se BLACK_TOKEN está em place[2]:
            shared_places.adicionar((place[1][0], place[2][BLACK_TOKEN]))
         senão:
            shared_places.adicionar((place[1][0], 0))
      place_dict[place[0]] = novo.dicionario([('name', place[1]), ('marking', place[2]), ('type',
      ↪  place[3])])

   only_places_dict.atualizar_com(place_dict)
   place_dicts[id_rede] = place_dict
```

Algoritmo 4.10: *Função que analisa e extrai os trechos relacionados aos lugares dos arquivos PNML.*

```
def create_initial_marking(place_dict, channel_places, net_tokens_list, shared_places):
  initial_marking = nova_lista()
  para cada chave em place_dict:
    se chave está em channel_places:
      create_net_place = string_formatada("NetPlace %s;\n", place_dict[key]['name'][0])
      initial_marking.adicionar(create_net_place)
      para cada marca em place_dict[chave]['marking'].chaves():
        se marca não está em net_tokens_list:
          devolva ERRO
        senão:
          para cada item em place_dict[chave]['marking'][marca]:
            initial_marking.adicionar(string_formatada("nt = run %s(%s.d);\n", marca,
            ↪  place_dict[chave]['name'][0]))
            initial_marking.adicionar(string_formatada("%s.d ! nt,255,0,0;\n",
            ↪  place_dict[key]['name'][0]))
    senão se place_dict[chave]['name'][0] não está em shared_places:
      se place_dict[chave]['marking'].chaves() != Vazio:
        para cada marca em place_dict[chave]['marking'].chaves():
          se marca não está em net_tokens_list:
            marking = string_formatada("byte %s = %s;\n", place_dict[key]['name'][0],
            ↪  place_dict[key]['marking'][mark])
            initial_marking.adicionar(marking)
      senão:
        marking = string_formatada("byte %s = 0;\n", place_dict[key]['name'][0])
        initial_marking.adicionar(marking)
  initial_marking.adicionar("byte it;\n")
  devolva initial_marking
```

Algoritmo 4.11: *Função que processa e gera a marcação inicial de cada rede no formato PROMELA.*

percorre seus arcos de entrada e saída. As condições de disparo, consumo e produção dependem do tipo de transição, de forma que transições do tipo *uplink*, *downlink* e *horizontal* devem ser traduzidas para o PROMELA de formas diferentes, conforme explicado na Seção 4.3. O Algoritmo 4.12 contém o trecho da função `translate_by_inter_dictionaries` responsável pelo processamento de cada tipo de transição.

Após a identificação do tipo de transição e produção das sentenças em PROMELA condizentes com o tipo destas, estas sentenças são armazenadas em listas que serão utilizadas na terceira parte do tradutor para gerar o código de saída. Outra parte importante desta etapa de tradução é a declaração das sequências de transporte dos *tokens* de rede, que pode ser conferida no Algoritmo 4.13. Esse algoritmo consiste de um laço sobre os arcos de cada transição que verifica as variáveis de cada arco de entrada e saída a fim de encontrar correlações entre estas e, caso existam, criar a declaração de transporte em PROMELA e salvar em uma lista de produções para ser utilizada na terceira parte do tradutor. Caso a variável utilizada nos arcos de entrada não seja encontrada em um arco de saída, uma declaração de consumo será incluída na lista de consumo para ser utilizada na terceira parte do tradutor.

A terceira parte do tradutor é composta da função `generate_promela_code(system_net_name, places, transitions, nets, vlabels, hlabels, shared_places, channel_places, enable_tests, consume, produce)`, que recebe como argumento as metainformações da rede fornecidas pela primeira parte do tradutor, como nome das redes e identificação da rede sistema, e recebe as listas de testes, consumo e produção geradas na segunda parte do tradutor. A função tem por objetivo escrever um arquivo de saída na extensão `pml` (PROMELA) com a estrutura básica apresentada anteriormente na Seção 4.3. Sua

```
se dowlink_label em only_transitions[transition]:
   downlink_label = only_transitions[transition]['downlink_label'][0]
   transition_label = downlink_label
   origins = nova_lista()
   para cada arc em arcs:
      se arc[0] está em channel_places:
         origin = only_places[arc[0]]['name'][0]
         origins.adicionar(origin)
   para cada origin em origins:
      downlink_condition = string_formatada("gbchan ?? [_,%s,_,0]", downlink_label)
      sentences['downlink_specific_conditions'].adicionar(downlink_condition)

senão se uplink_label em only_transitions[transition]:
   uplink_label = only_transitions[transition]['uplink_label'][0]
   transition_label = uplink_label
   fire_uplink_condition = string_formatada("it == %s", only_transitions[transition]['promela_id'])
   sentences['outer_loop_conditions'].adicionar(fire_uplink_condition)

   condition = string_formatada("! gbchan ?? [eval(_pid),%s,_,0]", uplink_label)
   sentences['uplink_specific_conditions'].adicionar(condition)

senão se horizontal_label em only_transitions[transition]:
   horizontal_label = only_transitions[transition]['horizontal_label'][0]

   sentences['horizontal_specific_conditions_a'].adicionar(string_formatada("! gbchan??[_,%s,_,0]",
   ↪ horizontal_label))
   sentences['horizontal_specific_conditions_b'].adicionar(string_formatada("!
   ↪ gbchan??[eval(_pid),%s,_,0] && gbchan??[_,%s,_,0]", horizontal_label, horizontal_label))
   sentences['outer_loop_conditions'].adicionar(string_formatada("it == %s",
   ↪ only_transitions[transition]['promela_id']))

   consume_actions['consume'].adicionar(string_formatada("invertMsg(nt, %s, gbchan);\n",
   ↪ transition_label))
   consume_actions['consume'].adicionar("sP(nt, 3);\n")
   produce_actions['general'].adicionar(string_formatada("gbchan ! _pid,%s,%s,0;\n",
   ↪ transition_label, only_transitions[transition]['promela_id']))
   produce_actions['general'].adicionar(string_formatada("gbchan ! _pid,%s,%s,1;\n",
   ↪ transition_label, only_transitions[transition]['promela_id']))
```

Algoritmo 4.12: *Trecho da função* `translate_by_inter_dictionaries` *que processa as condições para cada tipo de transição.*

```
para cada arc em arcs:
   se arc[0] está em channel_places e existem arc[1]['marking'].chaves():
      marks_count = arc[1]['marking']
      para cada mark em marks_count.chaves():
         para cada out_arc em arcs_out:
            out_marks_count = out_arc['marking']
            enquanto marks_count[mark] > 0 e mark está em out_marks_count.chaves() e
            ↪  out_marks_count[mark] > 0:
               se transition_label != Vazio:
                  sentence = string_formatada("invertMsg(nt, %s, gbchan);\nsp(nt, 3);\n",
                  ↪  transition_label)
               senão:
                  sentence = "gbchan ?? nt,_,_,0;\n"
                  se sentence não está em produce_actions['transport']:
                     produce_actions['transport'].adicionar(sentence)

                  sentence = string_formatada("transpNetTok(%s.d, %s.d, nt);\n",
                  ↪  only_places[arc[0]]['name'][0], only_places[out_arc['target']]['name'][0])
                  produce_actions['transport'].adicionar(sentence)
                                 marks_count[mark] = marks_count[mark] - 1
                  out_marks_count[mark] = out_marks_count[mark] - 1

            se marks_count[mark] > 0:
               para cada item em marks_count[mark]:
                  se transition_label != Vazio:
                     sentence = string_formatada("invertMsg(nt, %s, gbchan);\n", transition_label)
                     consume_actions['consume'].adicionar(sentence)
                     sentence = "sP(nt, 3);\n"
                     consume_actions['consume'].adicionar(sentence)
                  sentence = string_formatada("%s.d ?? nt,255,_,0;\n",
                  ↪  only_places[arc[0]]['name'][0])
                  consume_actions['consume'].adicionar(sentence)
                  sentence = "sP(nt, 3);\n"
                  consume_actions['consume'].adicionar(sentence)
                  sentence = string_formatada("consNetTok(%s.d,nt);\n",
                  ↪  only_places[arc[0]]['name'][0])
                  consume_actions['consume'].adicionar(sentence)
```

Algoritmo 4.13: *Trecho da função* `translate_by_inter_dictionaries` *que processa o transporte e consumo dos* tokens.

composição consiste do processamento e saída de três partes principais. A primeira é a impressão das funções fixas auxiliares em PROMELA, a segunda consiste das declarações de variáveis globais e, por fim, a terceira é um laço sobre as transições, utilizando-as como chave para posicionar os dados de saída das duas primeiras partes do tradutor no documento final. O código completo do tradutor encontra-se em `https://github.com/lugoboni/pnmltopml`.

4.4.1 Um exemplo simples da tradução

A Figura 4.4 contém o diagrama de uma rede aninhada utilizada como teste inicial da metodologia e implementação da tradução. Essa rede é composta de uma rede sistema *SN* com três lugares (*PSN1*, *PSN2* e *PSN3*) e duas transições (*TSN1* e *TSN2*), uma rede objeto *A* composta de três lugares (*PA1*, *PA2* e *PA3*) e duas transições (*TA1* e *TA2*) e uma rede objeto *B* composta de dois lugares (*PB1* e *PB2*) e uma transição (*TB1*). Nesta rede, as transições *TSN2* e *TA2* contêm uma etiqueta de sincronização vertical, *Lambda*, e as transições *TA1* e *TB1* contêm uma etiqueta de sincronização horizontal, *Theta*.

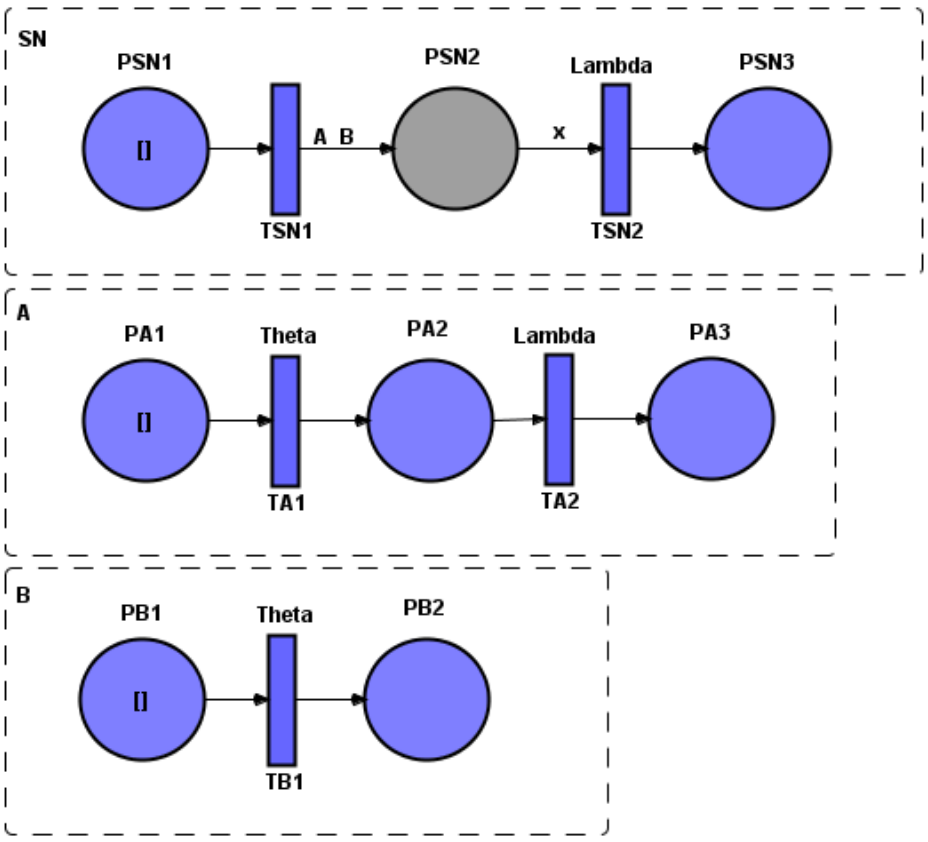

Figura 4.4: *Rede aninhada simples, composta de uma rede sistema* SN*, composta de três lugares e duas transições, e das redes elementos* A*, composta de três lugares e duas transições, e* B*, composta de dois lugares e uma transição.*

Com a marcação inicial de um *token* preto no lugar *PSN1*, a transição *TSN1* já está inicialmente habilitada. Ao disparar, ela cria as instâncias das redes *A* e *B* com marcação inicial de um *token* preto em cada um dos lugares *PA1* e *PB1*. Assim, as transições *TA1* e *TB1* são instanciadas já habilitadas de forma a poderem executar a sincronização horizontal por *Theta* e, ao dispararem,

produzirão um *token* preto em cada um dos lugares *PA2* e *PB2*. Com o *token* preto em *PA2* e os dois *tokens* de rede em *PSN2*, as transições *TSN2* e *TA2* com sincronização vertical pela etiqueta *Lambda* tornam-se habilitadas e, ao dispararem, produzem um *token* preto em cada um dos lugares *PSN3* e *PA3*. Porém, a variável x do arco (*PSN2, TSN2*) neste processo é atrelada a alguma das instâncias das redes A ou B, que deve ser consumida e terminada na ocorrência desta transição. Com isso, o estado final da rede deve com certeza possuir um *token* preto no lugar *PSN3* e um *token* de rede no lugar *PSN2*.

A rede contida na Figura 4.4 foi modelada em PNML de acordo com a definição apresentada na Seção 4.1 e foi traduzida por meio da implementação apresentada na Seção 4.4 de acordo com a metodologia definida na Seção 4.3.

O Algoritmo 4.14 apresenta o resultado da tradução para a rede *SN* contida na Figura 4.4 cada uma de suas partes é comentada a seguir. Nesse algoritmo é possível verificar a definição das duas etiquetas de sincronização *Lambda* e *Theta*, cada uma com um valor inteiro entre 1 e 254, a declaração dos lugares *PSN1* e *PSN3* como variáveis do tipo `byte` e do lugar *PSN2* como uma variável do tipo `NetPlace`, que por sua vez é declarado como um canal de mensagens de quatro posições. Após o comando `init`, que declara o processo principal do programa, a primeira declaração `atomic` contém a declaração da marcação inicial mas, como nenhum *token* de rede é instanciado antes do disparo da primeira transição, esta declaração foi gerada apenas com o *byte it*, uma variável temporária que pode ser utilizada para armazenar estados de outras variáveis em situações de necessidade de troca entre estas.

Seguindo, o laço principal, com rótulo `end:`, contém duas declarações `atomic`, uma para cada transição da rede. A primeira modela a transição *TSN1*: a condição de disparo é $PSN1 > 0$ e, quando ocorre, a prioridade da sequência é elevada para 3, a ação de consumo $PSN1 - 1$ é executada, removendo o *token* do lugar *PSN1*, as duas redes A e B são criadas no lugar *PSN2* e, por fim, a prioridade é reduzida para 1.

A segunda declaração *atomic* modela a transição *TSN2*. A condição de ocorrência é modelada de forma a indicar que a transição irá disparar somente se o lugar *PSN2* não estiver vazio e uma mensagem de sincronização com etiqueta *Lambda* existir no canal global. Com isso, o comportamento da sincronização vertical é modelado nesta rede e, ao ocorrer a transição, uma mensagem de *acknowledge* é enviada para o canal, de forma que a rede que enviou a mensagem inicial possa ter sua sequência pendente da sincronização vertical executada. A ação de consumo desta transição é a remoção do *token* de rede do lugar *PSN2* por meio da remoção da mensagem de seu canal e, como a mensagem não é transportada para outro canal, a mensagem que modela o *token* é consumida pela função `consNetTok`. Por fim, a produção da transição consiste da soma $PSN3 = PSN3 + 1$, que representa criação de um novo *token* preto no lugar *PSN3*.

O código correspondente à rede A pode ser conferido no Algoritmo 4.15. Inicialmente, a rede é declarada como um *proctype* para simular o comportamento de um processo independente do processo principal *init* declarado no código referente à rede *SN*. Os lugares *PA1*, *PA2* e *PA3* foram traduzidos como variáveis do tipo `byte` com sua marcação inicial. Novamente, como nenhum *token* de rede faz parte da marcação inicial, a declaração *atomic* é gerada sem nenhuma instanciação de outros processos de rede. Seguindo, os dois laços principais foram gerados de forma que o laço interno é composto das sequências `d_step` e modela a parte de sincronização e passos autônomos da rede, e o laço externo contém as ações decorrentes das ocorrências das sincronizações.

```
typedef NetPlace {chan d = [255] of {byte, byte, byte, bit}}

#define sP(a,b) set_priority(a,b)

chan gbchan = [255] of {byte, byte, byte, bit};

/*######TRANSITION VERTICAL LABELS########*/
#define Lambda 1

/*######TRANSITION HORIZONTAL LABELS########*/
#define Theta 254

NetPlace PSN2;
byte PSN1 = 1;
byte PSN3 = 0;
init {

    byte nt;
    atomic {
        printf("SN setting initial marking\n\n");
        byte it;
    }

    end: do
        :: atomic {
            PSN1 > 0  ->
                sP(_pid, 3);
                PSN1 = PSN1 - 1;
                nt = run B(PSN2.d);
                PSN2.d ! nt,255,0,0;
                nt = run A(PSN2.d);
                PSN2.d ! nt,255,0,0;
                printf("Firing transition TSN1");
                sP(_pid, 1);
        }
        :: atomic {
            nempty(PSN2.d) && gbchan ?? [_,Lambda,_,0]  ->
                sP(_pid, 3);
                invertMsg(nt, Lambda, gbchan);
                sP(nt, 3);
                PSN2.d ?? nt,255,_,0;
                sP(nt, 3);
                consNetTok(PSN2.d,nt);
                PSN3 = PSN3 + 1;
                printf("Firing transition TSN2");
                sP(_pid, 1);
        }
    od
}
```

Algoritmo 4.14: *Código PROMELA que modela a rede* SN *contida na Figura 4.4. O código foi resultado do tradutor automático e nele é possível observar as declarações das estruturas de dados utilizadas na modelagem, a definição das etiquetas de sincronização, assim como a marcação inicial e o comportamento das duas transições da rede no laço principal.*

```
proctype A (chan pc) {
    byte nt;
    byte v0;
    byte PA2 = 0;
    byte PA1 = 1;
    byte PA3 = 0;
    byte it;
    atomic {
        printf("A setting initial marking\n\n");
    }
    end: do :: {
        end1: do
            :: d_step {
                PA1 > 0 && ! gbchan??[eval(_pid),Theta,_,0] && gbchan??[_,Theta,_,0]  ->
                    sP(_pid, 3);
                    invertMsg(nt, Theta, gbchan);
                    sP(nt, 3);
                    gbchan ! _pid,Theta,1,0;
                    gbchan ! _pid,Theta,1,1;
                    printf("Firing transition TA1");
            }
            :: d_step {
                PA1 > 0 && ! gbchan??[_,Theta,_,0]  ->
                    sP(_pid, 3);
                    gbchan!_pid,Theta,1,0;
                    printf("Firing transition TA1");
            }
            :: d_step {
                PA2 > 0 && ! gbchan ?? [eval(_pid),Lambda,_,0]  ->
                    gbchan ! _pid,Lambda,2,0
                    printf("Firing transition TA2");
            }
        od
    }
    unless atomic {
        gbchan ?? eval(_pid),v0,it,1 ->
            if
                :: it == 0 && v0 == 255 -> break;
                :: it == 1 ->
                    PA1 = PA1 - 1;
                    PA2 = PA2 + 1;
                    dT(it);
                    printf("Firing outer loop transition TA1");
                :: it == 2 ->
                    PA2 = PA2 - 1;
                    PA3 = PA3 + 1;
                    printf("Firing outer loop transition TA2");
            fi;
            sP(_pid, 1);
    }
    od;
    sP(_pid, 1);
}
```

Algoritmo 4.15: *Código PROMELA que modela a rede elemento* A *contida na Figura 4.4. O código foi resultado do tradutor automático e nele é possível observar a declaração dos lugares da rede, e de suas transições através dos dois laços da estrutura utilizados para modelar as ações complementares das sincronizações horizontal e vertical da rede.*

Como descrito no Algoritmo 4.8, a sincronização horizontal em uma transição é decomposta em duas sequências `d_step` e, assim, a transição *TA1* foi traduzida de acordo com a etiqueta de sincronização horizontal *Theta*. A sequência com a condição `PA1 > 0 && ! gbchan??[_,Theta,_,0]` verifica se o lugar *PA1* possui um *token* no lugar *PA1* e se não há mensagens de sincronização no canal global `gbchan` com a etiqueta *Theta*. Caso a condição se confirme, a ação produzida é o envio da mensagem de solicitação sincronização para o canal global, pois a condição `PA1 > 0` do arco *(PA1, TA1)* foi satisfeita e a transição *TA1* está apta para a ocorrer em conjunto com o par da sincronização.

A sequência com a condição `PA1 > 0 && ! gbchan??[eval(_pid),Theta,_,0] && gbchan??[_,Theta,_,0]` verifica se o lugar *PA1* possui um *token* no lugar *PA1*, se não há mensagem de sincronização no canal global `gbchan` com a etiqueta *Theta* enviada pelo próprio processo da rede *A*, e se existe solicitação de sincronização com etiqueta *Theta* de outra rede. Caso as condições sejam satisfeitas, a sequência inverte o bit de sincronização da mensagem da outra rede e envia mensagens para o canal com bit 0, para inibir uma nova requisição, e com bit 1, para iniciar a condição do laço externo que executa as ações de consumo e produção da rede.

A condição no laço externo para executar as produções e consumo da ocorrência da transição *TA1* é a composição das condições `gbchan ?? eval(_pid),v0,it,1` e `it == 1`. Ao ocorrer, o *token* é removido de *PA1* e colocado em *PA2* e a mensagem de sincronização com a condição `it == 1` é removida pela função `dT(1)`. A sequência `d_step` restante modela a condição de sincronização vertical com a rede *SN*. Quando a condição `PA2 > 0 && ! gbchan ?? [eval(_pid),Lambda,_,0]` é satisfeita, ou seja, quando o lugar *PA2* contiver pelo menos um *token* e nenhuma mensagem de sincronização para a etiqueta *Lambda* existir no canal global, a sequência é executada. Sua execução consiste no envio da mensagem de solicitação de sincronização para o canal global. Quando a mensagem é enviada no canal global, a rede *SN* envia a mensagem de *acknowledge* que ativa a condição do laço externo composta das condições `gbchan ?? eval(_pid),v0,it,1` e `it == 2`. Ao ocorrer, esta sequência executa as ações de produção e consumo da transição *TA2*.

O código correspondente à rede *B* pode ser conferido no Algoritmo 4.16. A estrutura básica e comportamento desta rede são similares ao da rede *A* do Algoritmo 4.15. A transição *TB1* é modelada por meio das sequências `d_step` do laço interno e da condição composta das condições `gbchan ?? eval(_pid),v0,it,1` e `it == 1` no laço externo, da mesma forma que a transição *TA1* da rede *A*, pois ambas sincronizam horizontalmente com a etiqueta *Theta*.

Os códigos completos resultantes da tradução podem ser conferidos no repositório do projeto[4] e a saída da execução feita com a versão 6.4.9 do SPIN em um computador com processador Intel Core i5-5200U CPU @ 2.20GHz, 2201 Mhz, 2 Núcleo(s), 4 processadores lógicos e 8 GB de memória RAM pode ser verificada na Figura 4.5, onde é possível observar que três processos foram criados, um para cada instância de cada rede. A ordem da execução de disparo das transições e suas sincronizações ocorreu conforme o comportamento esperado e descrito anteriormente e, da mesma forma, o estado final da rede mostra que o lugar *PSN3* contém um *token*, assim como o lugar *PSN2*, representado pelo canal *d*, contém um *token* de rede.

Através deste exemplo, é possível observar que de fato a implementação de acordo com a metodologia apresentada neste capítulo podem simular o comportamento das redes de Petri aninhadas. No próximo capítulo, alguns estudos de caso mais complexos serão apresentados com o fim de averiguar se verificações podem ser realizadas no código PROMELA de saída do tradutor e comparar

[4] https://github.com/lugoboni/pnmltopml/tree/master/src/Examples/Basic

os resultados com as abordagens de Venero e Corrêa da Silva mencionadas no início deste capítulo.

```
proctype B (chan pc) {
    byte nt;
    byte v0;
    byte PB2 = 0;
    byte PB1 = 1;
    byte it;
    atomic {
       printf("B setting initial marking\n\n");
    }
    end: do :: {
        end1: do
            :: d_step {
                PB1 > 0 && ! gbchan??[eval(_pid),Theta,_,0] && gbchan??[_,Theta,_,0]  ->
                    sP(_pid, 3);
                    invertMsg(nt, Theta, gbchan);
                    sP(nt, 3);
                    gbchan ! _pid,Theta,1,0;
                    gbchan ! _pid,Theta,1,1;
                    printf("Firing transition TB1");
                }
            :: d_step{
                PB1 > 0 && ! gbchan??[_,Theta,_,0]  ->
                    sP(_pid, 3);
                    gbchan!_pid,Theta,1,0;
                    printf("Firing transition TB1");
            }
        od
    }
    unless atomic {
        gbchan ?? eval(_pid),v0,it,1 ->
            if
                :: it == 0 && v0 == 255 -> break;
                :: it == 1 ->
                    PB1 = PB1 - 1;
                    PB2 = PB2 + 1;
                    dT(it);
                    printf("Firing outer loop transition TB1");
            fi;
            sP(_pid, 1);
    }
}
```

Algoritmo 4.16: *Código PROMELA que modela a rede elemento* B *contida na Figura 4.4. O código foi resultado do tradutor automático e nele é possível observar a declaração dos lugares da rede, e de suas transições através dos dois laços da estrutura utilizados para modelar as ações complementares para a sincronização horizontal da rede.*

```
     SN setting initial marking

  5: setting priority of proc 0 (:init:) to 3
     Firing transition TSN1 12: setting priority of proc 0 (:init:) to 1
         B setting initial marking

             A setting initial marking

 20: setting priority of proc 2 (A) to 3
             Firing transition TA1 25: setting priority of proc 1 (B) to 3
 30: setting priority of proc 2 (A) to 3
          Firing transition TB1          Firing outer loop transition TB1 41: setting priority of proc 1 (B) to 1
             Firing outer loop transition TA1 48: setting priority of proc 2 (A) to 1
             Firing transition TA2 58: setting priority of proc 0 (:init:) to 3
 63: setting priority of proc 2 (A) to 3
 65: setting priority of proc 1 (B) to 3
     Firing transition TSN2 71: setting priority of proc 0 (:init:) to 1
             Firing outer loop transition TA2 77: setting priority of proc 2 (A) to 1
     timeout
#processes: 3
               queue 2 (gbchan):
               queue 1 (d): [2,255,0,0]
               PSN1 = 0
               PSN3 = 1
 82:    proc  2 (A:1) .\\net_translated.pml:167 (state 27) <valid end state>
 82:    proc  1 (B:3) .\\net_translated.pml:215 (state 23) <valid end state>
 82:    proc  0 (:init::1) .\\net_translated.pml:131 (state 40) <valid end state>
3 processes created
```

Figura 4.5: *Resultado da execução do programa PROMELA resultante da tradução do código PNML da rede da Figura 4.4.*

Capítulo 5

Estudos de caso da tradução

Neste capítulo mostramos alguns resultados da tradução de modelos em PNML para PROMELA em alguns estudos de caso. O tradutor automático foi testado em dois modelos de fluxo de trabalho para análise da tradução.

A análise foi realiza em duas etapas, sendo a primeira através de uma execução aleatória do modelo via SPIN e a segunda por meio de testes de verificação em busca de checar propriedades das redes. O primeiro caso, detalhado na Seção 5.1, foi retirado do artigo de Venero e Corrêa da Silva [83] e servirá de *benchmark* para a tradução, pois já possui uma versão modelada em PROMELA. O exemplo consiste de um *workflow* interorganizacional [80] modelado como uma rede aninhada. O segundo caso, detalhado na Seção 5.2, será o rede da Figura 2.7, que modela um sistema de entrega por *drones* que, diferente do primeiro caso, não possui um estado final da rede. Além disso, ele contém todas as principais características das redes aninhadas. Por conveniência, a Figura 2.7 encontra-se repetida na Figura 5.4.

5.1 *Workflow* interorganizacional

Um *workflow* interorganizacional é um conjunto de fluxos de trabalho acoplados de forma a serem executados de maneira independente, com pontos de contato entre os processos. Cada processo de fluxo de trabalho representa uma organização e pode ser modelado como uma rede de Petri chamada rede *workflow* [80], sendo responsável por seus recursos e pela execução de sua tarefa. Contudo, algumas etapas do processo podem depender de que a execução de outro processo de outro fluxo de trabalho alcance algum estado específico, de forma que um *workflow* para gerência dos outros pode ser necessário. A comunicação entre estes processos pode acontecer de forma assíncrona, através da troca de mensagens, ou de forma síncrona através da sincronização entre tarefas de dois processos de fluxo de trabalho diferentes [80]. A Rede de Petri que contém todas as redes *workflow* em um sistema coordenado como descrito é chamada de rede *workflow* interorganizacional.

A Figura 5.1 contém um *workflow* interorganizacional modelado como uma Rede de Petri aninhada. Esta rede é composta por uma rede sistema, duas redes que representam os fluxos de trabalho locais (LWF, de *local workflow*) e uma rede que modela um canal de comunicação assíncrona entre os fluxos de trabalho. Inicialmente, a rede sistema cria três *tokens* de rede no lugar `p2`, um para cada outra rede do sistema. A transição `T2` recebe uma etiqueta de sincronização vertical `Lambda` com as transições `F1T7` e `F2T6` das redes `LWF1` e `LWF2`. As transições `F1T3` e `F2T3` são etiquetadas com `L3`, para haver uma sincronização horizontal entre estas, de forma a modelar uma comunicação síncrona

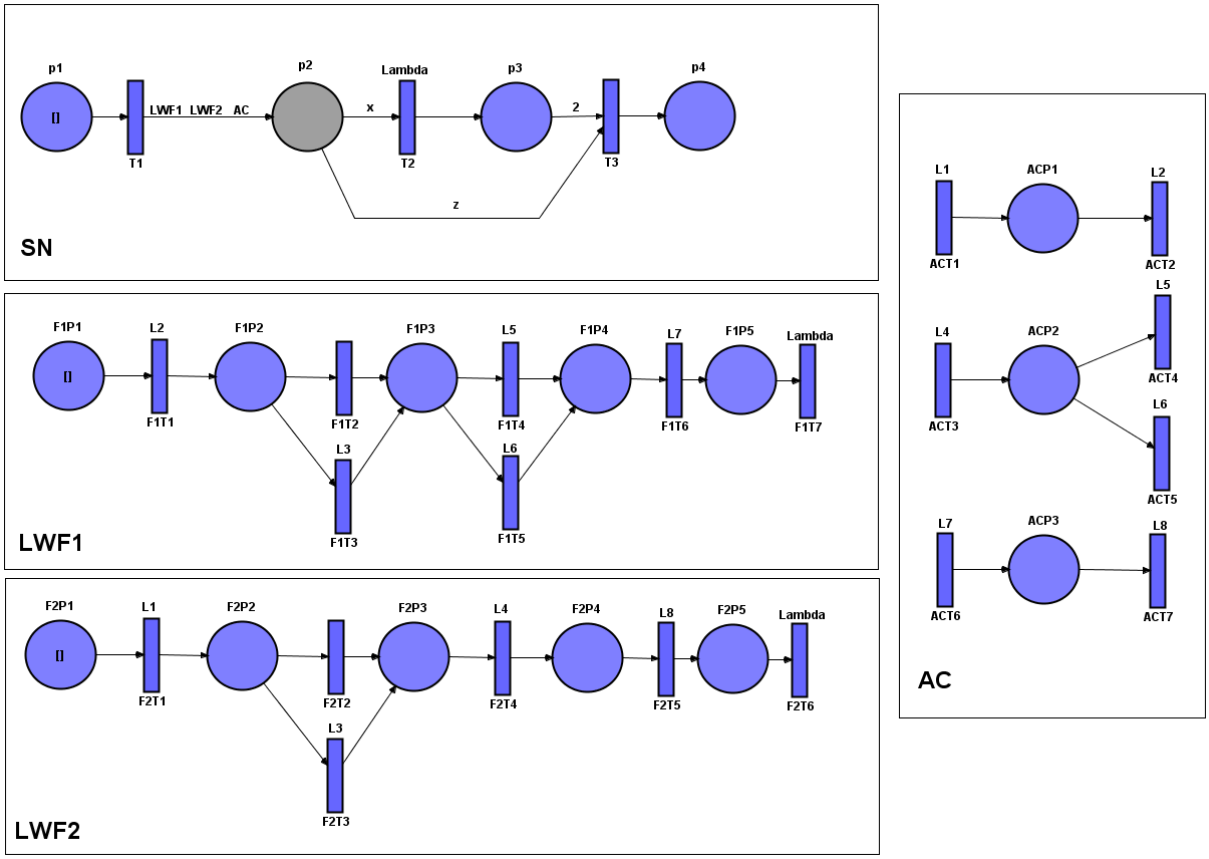

Figura 5.1: *Rede* workflow *interorganizacional modelada através de uma Rede de Petri aninhada. A rede contém uma rede sistema, duas redes* workflow *locais e uma rede para comunicação assíncrona.*

entre os dois fluxos de trabalho. Todas as outras transições com etiquetas nas redes `LWF1` e `LWF2` assim o são a fim de sincronizá-las com as transições da rede `AC` para modelar uma comunicação assíncrona entre as redes `LWF1` e `LWF2`.

Como dito anteriormente, esta rede foi escolhida para testes pois também foi utilizada antes por Venero e Corrêa da Silva [83] como um teste de modelagem de uma rede aninhada em PROMELA e, como consequência, podemos utilizar a modelagem já realizada como parâmetro para a tradução automática deste trabalho. A comparação deve ser feita através do resultado da execução do SPIN sobre o código PROMELA, de forma a assegurar a verificação realizada por este processo. A propriedade *soundness* foi escolhida para ser verificada no código PROMELA gerado, pois Yamaguchi [88] já demonstrou a possibilidade desta verificação através de uma proposta de modelagem de uma rede de *worflow* em PROMELA. De acordo com Venero e Corrêa da Silva [83], uma rede de *workflow* pode ser considerada *sound* se, e somente se, toda marcação inicial é levada até uma marcação final da rede, a marcação final deve ser a única marcação alcançável com um *token* em um lugar do qual não saem transições, e toda tarefa deve ser executada ao menos uma vez durante o fluxo.

No que tange as redes *workflow* interorganizacionais, de acordo com Prisecaru [70], estas satisfazem a propriedade de *soundness* caso (i) suas redes *workflow* locais forem *sound* e (ii) toda marcação inicial é levada até uma marcação final da rede. Ainda segundo Venero e Corrêa da Silva [83], as redes `LWF1` e `LWF2` da rede interorganizacional da Figura 5.1 são, *sound* e, portanto, a verificação da propriedade pode ser realizada através da condição (ii). Para isso, é possível utilizar a

função de verificação por fórmula de lógica temporal `ltl<>` do PROMELA para checar o estado final desejado da rede. A fórmula `ltl s {<> (p4>0)}`, onde`<>` significa *eventualmente*, testa se em algum ponto dentro da árvore de estados do modelo a variável `p4` terá valor maior do que zero.

A primeira etapa do processo foi a modelagem da rede no formato PNML conforme especificado na Seção 4.1. O código resultante dessa modelagem pode ser verificado no repositório público do projeto[1]. O processo de modelagem adotado foi a criação do diagrama através da ferramenta de modelagem e simulação de redes por objeto com semântica por referência *Renew* [48]. O diagrama criado foi exportado para PNML através da ferramenta, que exporta no formato PNML definido pelos mantenedores do *Renew*. Este PNML foi modificado para atender a especificação conforme a extensão proposta neste trabalho, na Seção 4.1, de forma que cada subrede foi modelada em um arquivo PNML homônimo diferente.

Estes arquivos foram utilizados como entrada para o programa tradutor e é importante ressaltar que devido às particularidades de cada rede não é possível gerar as restrições para verificação automaticamente, ficando a cargo do projetista do sistema adicionar as condições de verificação no código PROMELA. O código PML resultante também pode ser conferido no repositório do projeto[2].

Uma verificação foi realizada com a versão 6.4.9 do SPIN em um computador com processador Intel Core i5-5200U CPU @ 2.20GHz, 2201 Mhz, 2 Núcleo(s), 4 processadores lógicos e 8 GB de memória RAM, utilizando os parâmetros `-DVECTORSZ=16384 -DBITSTATE -O2` e seu resultado se encontra na Figura 5.2. Como comparação, uma verificação com os mesmos parâmetros foi realizada na tradução publicada por Venero e Corrêa da Silva [83] e seu resultado pode ser conferido na Figura 5.3.

A verificação mostra que a propriedade testada foi satisfeita pois nenhum erro foi reportado pelo SPIN, e destaca-se que algumas linhas do código da rede foram apresentados como nunca alcançadas. Estas linhas são aquelas que modelam a execução das transições `F1T3` e `F2T3` e não são executadas com a mesma prioridade de execução das transições `F1T2` e `F2T2` pois, por se tratarem de transições sem sincronização, `F1T2` e `F2T2` sempre serão executadas primeiro e desabilitarão `F1T3` e `F2T3`.

Em comparação com a verificação executada na tradução feita manualmente por Venero e Corrêa da Silva [83], a tradução automática apresentou um incremento considerável de 191240 estados a serem verificados em uma profundidade que cresceu de 163 para 189.

5.2 Logística de entregas multiagente

A Figura 5.4 contém o modelo de Rede de Petri aninhada de um sistema multiagente que executa um sistema de logística de entregas de pedidos. O funcionamento do sistema consiste de três agentes autônomos e uma rede coordenadora. Cada agente é modelado como uma rede com comportamento independente das outras redes e a rede coordenadora é modelada como a rede sistema. Nesta rede sistema os agentes são coordenados quanto a suas ações conjuntas e aos recursos compartilhados entre as redes.

Especificamente, o principal recurso compartilhado são os pedidos, que são modelados na rede pelo lugar compartilhado `ORDERS`. O agente `A` representa um atendente do sistema e tem como

[1] https://github.com/lugoboni/pnmltopml/tree/master/src/Examples/Workflow
[2] https://github.com/lugoboni/pnmltopml/blob/master/src/Examples/Workflow/wkfw_net_translated.pml

```
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> .\run.exe -a -m10000000
Depth=     189 States=   1e+006 Transitions= 2.01e+006 Memory=   476.889        t=     27.9 R=  4e+004

(Spin Version 6.4.9 -- 17 December 2018)

Bit statespace search for:
        never claim             + (s)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 4176 byte, depth reached 189, errors: 0
   616987 states, stored (1.23397e+006 visited)
  1240679 states, matched
  2474653 transitions (= visited+matched)
  4551109 atomic steps

hash factor: 108.769 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
 2464.239       equivalent memory usage for states (stored*(State-vector + overhead))
   16.000       memory used for hash array (-w27)
   38.147       memory used for bit stack
  419.617       memory used for DFS stack (-m10000000)
    1.615       other (proc and chan stacks)
    1.510       memory lost to fragmentation
  476.889       total actual memory usage


unreached in init
        .\wkflw_saida.pml:8, state 30, "p2.d??eval(nt),_,_,0"
        .\wkflw_saida.pml:8, state 48, "p2.d??eval(nt),_,_,0"
        .\wkflw_saida.pml:14, state 57, "gbchan??eval(_pid),1,_,_"
        .\wkflw_saida.pml:178, state 69, "-end-"
        (4 of 69 states)
unreached in proctype AC
        .\wkflw_saida.pml:332, state 233, "set_priority(_pid, 1)"
        .\wkflw_saida.pml:332, state 234, "-end-"
        (2 of 234 states)
unreached in proctype LWF1
        .\wkflw_saida.pml:460, state 169, "F1P3 = (F1P3+1)"
        .\wkflw_saida.pml:89, state 171, "gbchan??eval(_pid),_,it,0"
        .\wkflw_saida.pml:472, state 199, "set_priority(_pid, 1)"
        .\wkflw_saida.pml:472, state 200, "-end-"
        (4 of 200 states)
unreached in proctype LWF2
        .\wkflw_saida.pml:578, state 127, "F2P3 = (F2P3+1)"
        .\wkflw_saida.pml:89, state 129, "gbchan??eval(_pid),_,it,0"
        .\wkflw_saida.pml:590, state 157, "set_priority(_pid, 1)"
        .\wkflw_saida.pml:590, state 158, "-end-"
        (4 of 158 states)
unreached in claim s
        _spin_nvr.tmp:6, state 6, "-end-"
        (1 of 6 states)

pan: elapsed time 34.7 seconds
pan: rate 35527.423 states/second
```

Figura 5.2: *Resultado da verificação da propriedade* `ltl s {<> (p4>0)}` *no código PROMELA resultante da tradução do PNML modelado a partir da rede da Figura 5.1. No resultado é possível notar que a condição foi satisfeita pois nenhum erro foi encontrado na busca da propriedade.*

```
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> .\run_benchmark_wkfl.exe -a -m1000000

(Spin Version 6.4.9 -- 17 December 2018)

Bit statespace search for:
        never claim             + (s)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 188 byte, depth reached 163, errors: 0
   425747 states, stored (851493 visited)
  1384609 states, matched
  2236102 transitions (= visited+matched)
   873940 atomic steps

hash factor: 157.626 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
   81.205       equivalent memory usage for states (stored*(State-vector + overhead))
   16.000       memory used for hash array (-w27)
    3.815       memory used for bit stack
   41.962       memory used for DFS stack (-m1000000)
    1.612       other (proc and chan stacks)
    1.513       memory lost to fragmentation
   64.901       total actual memory usage


unreached in proctype netLWF1
        (0 of 136 states)
unreached in proctype netLWF2
        (0 of 107 states)
unreached in proctype netAC
        .\exemplo_mirtha_workflow.pml:4, state 142, "set_priority(_pid, 1)"
        .\exemplo_mirtha_workflow.pml:110, state 143, "-end-"
        (2 of 143 states)
unreached in init
        .\exemplo_mirtha_workflow.pml:141, state 58, "-end-"
        (1 of 58 states)
unreached in claim s
        _spin_nvr.tmp:6, state 6, "-end-"
        (1 of 6 states)

pan: elapsed time 5.14 seconds
pan: rate 165692.35 states/second
```

Figura 5.3: *Resultado da verificação da propriedade* `ltl s {<> (p4>0)}` *no código PROMELA do código PROMELA modelado a partir da rede da Figura 5.1 por Venero e Corrêa da Silva [83]. Assim como o resultado da tradução automática, a condição da propriedade foi satisfeita.*

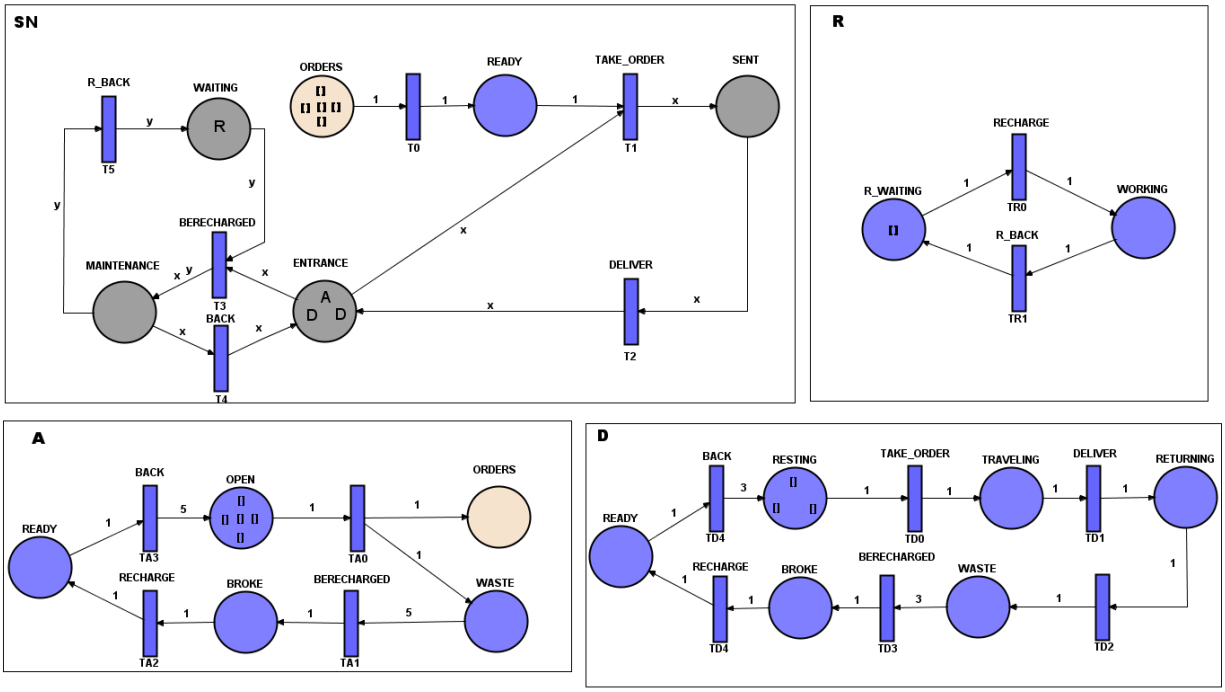

Figura 5.4: *Exemplo de um sistema de entregas modelado por uma Rede de Petri aninhada conforme a Definição 2.6.1. O sistema é composto por quatro redes (SN, R, A e D), onde as transições são etiquetadas de forma a permitir o disparo síncrono destas entre as redes em passos verticais e horizontais. Além disso, os pedidos são controlados por um lugar compartilhado (ORDERS) que está presente em todas as quatro redes mas foi omitido no diagrama daquelas onde o mesmo não possui relevância para o estado das redes.*

principal função a criação de novos pedidos por meio do disparo da transição `TA0`. O agente `D` é o entregador do sistema, que parte de um estado de repouso para um estado de entrega através da retirada de um pedido por meio do disparo da transição `T1` com a etiqueta de sincronização vertical `TAKE_ORDER`. Os agentes `A` e `D` possuem contadores de carga da bateria que são incrementados a cada ação. Ao atingir determinado limiar, ditado pela quantidade de *tokens* nos lugares iniciais `OPEN` e `RESTING`, os agentes ficam retirados de ação pelo estado `BROKE` e precisam de recarga.

O processo de recarga é realizado pelo agente reparador `R` e coordenado pela rede sistema. Para a recarga ocorrer, o agente quebrado e o reparador precisam ser sincronizados através das transições com a etiqueta de sincronização horizontal `RECHARGE` e, para isso, os *tokens* de instância destas redes precisam estar no mesmo lugar `MAINTENANCE` da rede sistema. A ideia deste sistema é modelar um processo autônomo de forma que nenhum agente fique parado e o sistema de pedidos permaneça funcionando ininterruptamente.

O processo de tradução, seguindo a metodologia apresentada no Capítulo 4.3, iniciou com a modelagem deste sistema no formato PNML apresentado na Seção 4.1. Inicialmente o modelo gráfico foi feito através da ferramenta de modelagem de redes por objeto com semântica por referência *Renew* [48]. Cada subrede do sistema foi modelada em um arquivo diferente e através do *Renew* o diagrama foi exportado para o formato PNML criado pelos mantenedores da ferramenta e, por fim, o PNML foi adaptado de acordo com a extensão proposta na Seção 4.1.

O código PNML resultante pode ser conferido no repositório público do projeto[3]. Novamente, estes arquivos PNML foram utilizados como entrada do tradutor para serem convertidos em PRO-

[3]Códigos PNML do exemplo logística de entregas https://github.com/lugoboni/pnmltopml/tree/master/src/Examples/Drone_det

```
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> .\spin649_windows64 .\drone_saida.pml
     SN setting initial marking

             D setting initial marking

                 A setting initial marking

         D setting initial marking

 19: setting priority of proc 0 (:init:) to 3
     Firing transition T0 23: setting priority of proc 0 (:init:) to 1
                     R setting initial marking

             Firing transition TD0 32: setting priority of proc 0 (:init:) to 3
     Firing transition T0 36: setting priority of proc 0 (:init:) to 1
         Firing transition TD0 42: setting priority of proc 0 (:init:) to 3
     Firing transition T0 46: setting priority of proc 0 (:init:) to 1
 50: setting priority of proc 0 (:init:) to 3
     Firing transition T0 54: setting priority of proc 0 (:init:) to 1
 57: setting priority of proc 0 (:init:) to 3
     Firing transition T0 61: setting priority of proc 0 (:init:) to 1
 63: setting priority of proc 3 (A) to 3
                 Firing transition TA0 69: setting priority of proc 3 (A) to 3
                 Firing transition TA0 75: setting priority of proc 3 (A) to 3
                 Firing transition TA0 81: setting priority of proc 3 (A) to 3
                 Firing transition TA0 87: setting priority of proc 3 (A) to 3
                 Firing transition TA0                 Firing transition TA1 97: setting priority of proc 0 (:init:) to 3
     Firing transition T0101: setting priority of proc 0 (:init:) to 1
106: setting priority of proc 4 (R) to 3
                     Firing transition TR0110: setting priority of proc 0 (:init:) to 3
     Firing transition T0114: setting priority of proc 0 (:init:) to 1
117: setting priority of proc 0 (:init:) to 3
     Firing transition T0121: setting priority of proc 0 (:init:) to 1
124: setting priority of proc 0 (:init:) to 3
     Firing transition T0128: setting priority of proc 0 (:init:) to 1
131: setting priority of proc 0 (:init:) to 3
     Firing transition T0135: setting priority of proc 0 (:init:) to 1
138: setting priority of proc 0 (:init:) to 3
145: setting priority of proc 2 (D) to 10
             Firing outer loop transition TD0151: setting priority of proc 2 (D) to 1
     Firing transition T1163: setting priority of proc 0 (:init:) to 1
168: setting priority of proc 0 (:init:) to 3
174: setting priority of proc 3 (A) to 2
     Firing transition T3193: setting priority of proc 0 (:init:) to 1
                 Firing outer loop transition TA1199: setting priority of proc 3 (A) to 1
             Firing transition TD0             Firing transition TD1210: setting priority of proc 0 (:init:) to 3
217: setting priority of proc 1 (D) to 10
         Firing outer loop transition TD0223: setting priority of proc 1 (D) to 1
     Firing transition T1235: setting priority of proc 0 (:init:) to 1
         Firing transition TD0244: setting priority of proc 0 (:init:) to 3
249: setting priority of proc 2 (D) to 10
             Firing outer loop transition TD1255: setting priority of proc 2 (D) to 1
     Firing transition T2267: setting priority of proc 0 (:init:) to 1
270: setting priority of proc 0 (:init:) to 3
277: setting priority of proc 2 (D) to 10
             Firing outer loop transition TD0286: setting priority of proc 2 (D) to 1
     Firing transition T1298: setting priority of proc 0 (:init:) to 1
         Firing transition TD1311: setting priority of proc 4 (R) to 3
                 Firing transition TA2                         Firing outer loop transition TR0321: setting priority of proc 4 (R) to 1
             Firing transition TD0             Firing transition TD1333: setting priority of proc 0 (:init:) to 3
```

Figura 5.5: *Impressão de parte da saída de uma execução aleatória do código PNML da rede da Figura 5.4. Nela é possível observar a ordem dos disparos da execução.*

MELA. Ao rodar o programa PROMELA com o SPIN, este irá executar o programa de maneira aleatória, onde é possível observar a sequência dos disparos do sistema. A Figura 5.5 mostra a parte inicial da saída desta execução. Nela é possível observar os passos executados por meio dos comandos de impressão colocados no código. Esta execução é útil para verificar a ordem dos disparos da rede, contudo, por ser apenas uma execução aleatória nada pode ser concluído a respeito das propriedades desejadas no modelo.

Um ponto que deve ser ressaltado é que em uma execução aleatória a prioridade dada aos processos é crucial. No exemplo de execução contido na Figura 5.5, a saída do tradutor, contida no repositório[4], contém uma modificação para que a prioridade das redes `D` fosse maior nos passos de disparo das transições `TAKE_ORDER` e `DELIVER` pois, sem alteração de prioridade, a execução dos passos da rede `A` acabaram tendo um viés prioritário pela combinação ou ausência dos tipos de sincronização.

Por se tratar de um sistema modelado para funcionar de maneira ininterrupta, a propriedade

[4]Resultado da tradução do exemplo logística de entregas https://github.com/lugoboni/pnmltopml/blob/master/src/Examples/Drone_det/drone_net_translated.pml

desejada nesta rede é a vivacidade. Não seria aceitável nesta rede uma marcação morta, ou seja, uma marcação tal que nenhuma transição esteja habilitada. Porém, como citado na Seção 2.2, a vivacidade é uma propriedade cuja verificação é virtualmente impraticável em sistemas com uma alta quantidade de componentes e grande complexidade. Neste caso, por se tratar de um sistema pequeno, a execução da verificação padrão do SPIN já forneceria um bom indicativo de que a rede não é viva caso um estado final, *deadlock* ou erro fosse encontrado durante a execução.

Uma verificação foi realizada com a versão 6.4.9 do SPIN em um computador com processador Intel Core i5-5200U CPU @ 2.20GHz, 2201 Mhz, 2 Núcleo(s), 4 processadores lógicos e 8 GB de memória RAM, utilizando os parâmetros `-DVECTORSZ=16384 -DBITSTATE -O2` e seu resultado se encontra na Figura 5.8. Nela, é possível observar que mesmo o sistema sendo composto de uma rede aninhada relativamente simples, após atingir uma profundidade de 405 estados a verificação foi concluída. Nenhum erro foi encontrado neste processo e, portanto, nenhum *deadlock* ocorreu. Caso um estado inválido fosse encontrado nesta verificação, a execução seria interrompida e uma mensagem de erro do SPIN seria apresentada [40].

As funcionalidades de verificação de lógica linear temporal do SPIN também podem ser utilizadas neste caso para verificar se o comportamento desejado pelo projetista do sistema é de fato obedecido. Por exemplo, é possível verificar que a rede está se comportando como esperado e removendo os *tokens* do lugar `ORDERS` com a fórmula `ltl s: [] ((ORDERS > 0))`, em que `[]` significa *sempre*. Se esta fórmula for violada, sabemos que eventualmente `ORDERS` ficará sem *tokens*. A Figura 5.6 mostra o resultado da verificação deste teste.

O mesmo teste realizado com o lugar `ORDERS` poderia ser feito com a fórmula `ltl s: <> ((ORDERS == 0))`, em que `<>` significa *eventualmente*. A Figura 5.7 mostra o resultado desta verificação onde é possível observar que a verificação foi concluída com sucesso e a fórmula foi satisfeita, comprovando que de fato, os *tokens* do lugar `ORDERS` estão sendo removidos.

```
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> .\run_drone.exe -a -m10000000
pan:1: assertion violated  !( !((ORDERS>0))) (at depth 355)
pan: wrote .\drone_saida.pml.trail

(Spin Version 6.4.9 -- 17 December 2018)
Warning: Search not completed

Bit statespace search for:
        never claim             + (s)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 10336 byte, depth reached 355, errors: 1
       72 states, stored
        0 states, matched
       72 transitions (= stored+matched)
      224 atomic steps

hash factor: 1.86414e+006 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
    0.711       equivalent memory usage for states (stored*(State-vector + overhead))
   16.000       memory used for hash array (-w27)
   38.147       memory used for bit stack
  419.617       memory used for DFS stack (-m10000000)
    1.602       other (proc and chan stacks)
    1.523       memory lost to fragmentation
  476.889       total actual memory usage



pan: elapsed time 0.011 seconds
pan: rate 6545.4545 states/second
```

Figura 5.6: *Resultado da verificação da fórmula* `ltl s: [] ((ORDERS>0))` *na rede da Figura 5.4. A verificação foi realizada com a versão 6.4.9 do SPIN e mostra a violação da fórmula, o que comprova a tese de que os* tokens *do lugar ORDERS estão sendo removidos.*

```
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> .\spin649_windows64 -a .\drone_saida.pml
ltl s: <> ((ORDERS==0))
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> gcc -DVECTORSZ=16384 -DBITSTATE -O2 -o run_drone pan.c
PS C:\Users\gusta\Documents\UFABC\mestrado\spin\Spin-version-6.5.0\Bin> .\run_drone.exe -a -m10000000
Depth=     373 States=   1e+006 Transitions= 2.16e+006 Memory=   476.889        t=     82.9 R=  1e+004
Depth=     373 States=   2e+006 Transitions= 4.49e+006 Memory=   476.889        t=      166 R=  1e+004

(Spin Version 6.4.9 -- 17 December 2018)

Bit statespace search for:
        never claim             + (s)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 10336 byte, depth reached 373, errors: 0
  1487305 states, stored (2.97464e+006 visited)
  3993640 states, matched
  6968284 transitions (= visited+matched)
 15348275 atomic steps

hash factor: 45.1206 (best if > 100.)

bits set per state: 3 (-k3)

Stats on memory usage (in Megabytes):
14677.651       equivalent memory usage for states (stored*(State-vector + overhead))
   16.000       memory used for hash array (-w27)
   38.147       memory used for bit stack
  419.617       memory used for DFS stack (-m10000000)
    1.602       other (proc and chan stacks)
    1.523       memory lost to fragmentation
  476.889       total actual memory usage
```

Figura 5.7: *Resultado da verificação da fórmula* `ltl s: <> ((ORDERS==0))` *na rede da Figura 5.4. A verificação foi realizada com a versão 6.4.9 do SPIN e mostra que fórmula foi satisfeita.*

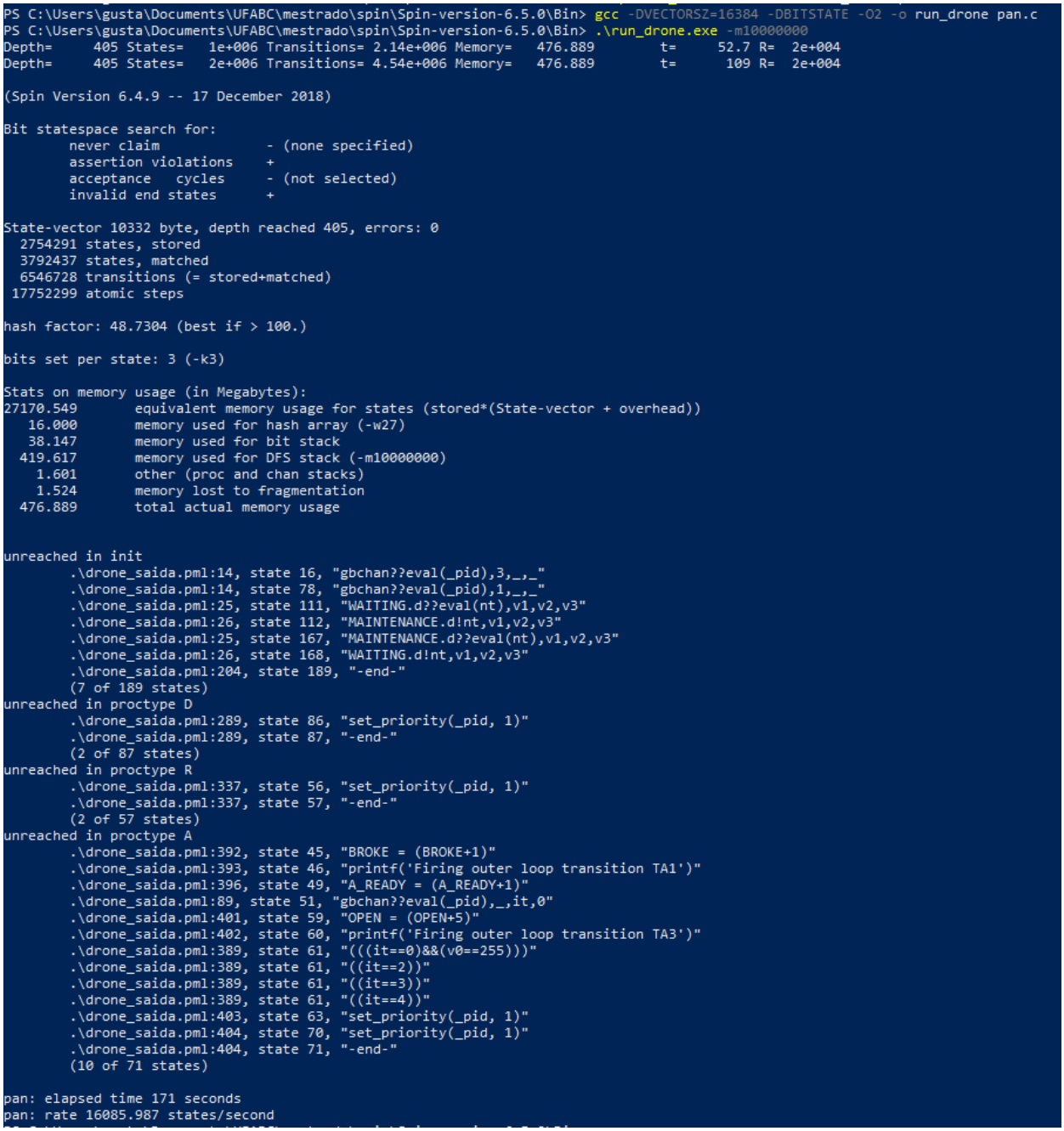

Figura 5.8: *Impressão da saída verificação do SPIN do código PNML da rede da Figura 5.4. A verificação foi realizada com a versão 6.4.9 do SPIN e mostra a ausência de* deadlocks *no modelo.*

Capítulo 6

Considerações finais

Nesta dissertação foram apresentados, inicialmente, conceitos básicos de Redes de Petri com o propósito de apresentar as propriedades básicas e o que pode ser verificado ou não através de técnicas de verificação. Com isso, criamos uma base do que foi fundamentalmente a proposta deste trabalho, que era a formalização de um método para simular e verificar Redes de Petri Aninhadas de maneira automática tendo como ponto de partida as publicações na literatura que delineavam tal método [82, 85].

O Capítulo 2 seguiu esta apresentação com um resumo do histórico do paradigma de *tokens* como redes, partindo das principais definições formais que foram publicadas ao longo do tempo que, ao final, levaram Lomazova a propor as redes aninhadas para simplificar a semântica complexa das redes por objeto. Partindo destas redes de Lomazova, foram definidas as redes aninhadas a serem utilizadas como linguagem formal a serem traduzidas pelo método automático. Estas redes previram as propriedades principais, sincronização horizontal e vertical, lugares compartilhados e, principalmente, a semântica simplificada das redes aninhadas.

A revisão sistemática do Capítulo 3 demonstrou a necessidade de uma ferramenta como a proposta nesta dissertação e no Capítulo 4 uma contribuição visando satisfazer esta necessidade foi proposta com uma expansão para a norma ISO15909-2 [41] contendo o padrão PNML NPN para redes aninhadas. O propósito desta expansão é fornecer uma forma padronizada de representação das redes aninhadas definidas no Capítulo 2 de forma a permitir que qualquer ferramenta possa implementar o uso destas redes. Propusemos então uma forma de modelar as redes aninhadas também na linguagem PROMELA, onde cada elemento do comportamento das redes recebeu uma tradução equivalente.

Por fim, no Capítulo 5 avaliamos o resultado a partir de dois estudos de caso. O primeiro foi uma rede de fluxos de trabalho interorganizacional extraído do artigo de Venero e Corrêa da Silva [83], que serviu como base de comparação entre a nossa metodologia e a metodologia original dos autores. O segundo foi um estudo de caso baseado em um sistema multiagente de um fluxo de entregas ininterrupto.

Apesar do incipiente estado da ferramenta de tradução construída e apresentada neste trabalho, os testes realizados nos estudos de caso mostraram que o método é de fato funcional e pode ser útil para verificar algumas propriedades de acordo com o desejo do projetista da rede. Alguns pontos, entretanto, devem ser levantados. Tanto a Definição 2.6.1 quanto a metodologia de modelagem desta para o PROMELA admitem apenas sincronizações entre dois *tokens* por vez, dificuldade que parece ter sido superada na metodologia de modelagem apresentada no trabalho mais recente de

Venero e Corrêa da Silva [85], cuja implementação permaneceu restrita a apenas um estudo de caso modelado de forma manual.

Além disso, o uso de apenas um canal global para toda a comunicação entre as instâncias das redes sem o uso de uma fila para ordená-lo eleva o viés das sequências de disparo em simulações utilizando a execução padrão do SPIN, de forma que passos mais simples nas redes, como disparo de transições sem sincronização e apenas com operações aritméticas em lugares de tipo básico, são claramente favorecidos e tendem a ocorrer com elevada prioridade. Este viés pôde ser observado principalmente no estudo de caso da logística de entregas na Seção 5.2, e só foi possível corrigi-lo por meio de uma alteração manual nas prioridades. No estudo de caso do *workflow* interorganizacional apresentado na Seção 5.1 isso foi percebido através da verificação do SPIN, que mostrou alguns estados inalcançáveis na rede, a saber, os estados que dependiam do disparo de transições com sincronização horizontal que tinham uma rota alternativa a si na rede através de transições sem a necessidade destas sincronizações e que, ao ocorrerem, desabilitavam as primeiras.

Apesar destas questões, como dito anteriormente, verificações puderam ser feitas nas redes traduzidas e até mesmo o estudo de caso utilizado como comparativo com abordagens manuais anteriores se mostrou bem sucedido. Isso reforça o potencial deste método automatizado de, com um esforço maior de engenharia de *software*, tornar-se uma adição robusta a ser incorporada em alguma ferramenta de modelagem gráfica como motor para verificações do tipo de redes aninhadas aqui apresentado. Para isso, é necessário também melhorar e testar o processamento de redes com lugares e arcos de tipos em multiconjuntos pois, nesta dissertação, foram testadas apenas redes com um único tipo de *token* além dos tipos de redes.

As questões abordadas anteriormente podem ser melhoradas no método automático proposto neste trabalho a partir da incorporação de outras técnicas de estruturas de dados mais complexas e também, em trabalhos futuros, readicionando alguns elementos retirados da proposta mais recente de Venero e Corrêa da Silva. Esses elementos foram removidos da metodologia em nosso projeto pois aplicamos uma abordagem por subtração de elementos não essenciais, focando nos elementos principais do método de modelagem, com o fim de simplificar e automatizar a tradução deste método para simular e verificar as Redes de Petri Aninhadas.

Referências Bibliográficas

[1] Petri Nets Tools Database. Quick Overview. URL http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html. Accessed: 2021-02-03. 31, 34, 36, 37, 39

[2] PNML.ORG. PetriNet Markup Language (PNML). URL http://www.pnml.org. Accessed: 2021-02-03. 46

[3] Spin. SPIN VERIFIER's ROADMAP. URL http://spinroot.com/spin/Man/Roadmap.html. Accessed: 2021-02-03. 50

[4] I. J. Abdullahi and B. Müller. Towards Efficient Verification of Elementary Object Systems. In *Proceedings of the 25th International Workshop on Concurrency, Specification and Programming*, volume 1698 of *CEUR Workshop Proceedings*, pages 86–100, 2016. 40

[5] L. M. Almutairi and S. Shetty. Generalized stochastic Petri Net model based security risk assessment of software defined networks. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 545–550, Oct 2017. doi10.1109/MILCOM.2017.8170813. 36, 39

[6] B. Aristyo, K. Pradityo, T. A. Tamba, Y. Y. Nazaruddin, and A. Widyotriatmo. Model Checking-based Safety Verification of a Petri Net Representation of Train Interlocking Systems. In *2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 392–397, Sept 2018. doi10.23919/SICE.2018.8492661. 36

[7] J. C. Augusto, M. Butler, C. Ferreira, and S. J. Craig. Using SPIN and STeP to Verify Business Processes Specifications. In *Perspectives of System Informatics*, volume 2890 of *LNCS*, pages 207–213. 2003. 2

[8] J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In W. M. P. van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003*, pages 483–505, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-44919-5. 43

[9] Z. Cao, F. Qiao, and Q. Wu. Queueing Generalized Stochastic Colored Timed Petri Nets-based Approach to Modeling for Semiconductor Wafer Fabrication. In *2007 IEEE International Conference on Control and Automation*, pages 2834–2838, May 2007. doi10.1109/ICCA.2007.4376879. 33

[10] L. Chang and X. He. A model transformation approach for verifying multi-agent systems using SPIN. In *Proceedings of the ACM Symposium on Applied Computing*, pages 37–42, 2011. 2, 41

[11] S. Chen, J. Ke, and J. Chang. Knowledge representation using fuzzy Petri nets. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):311–319, Sept 1990. ISSN 1041-4347. doi10.1109/69.60794. 11

[12] S. Community. RELAX NG, "A schema language for XML". URL https://relaxng.org. Accessed: 2021-02-03. 46

[13] R. Davidrajuh. Developing a New Petri Net Tool for Simulation of Discrete Event Systems. In *2008 Second Asia International Conference on Modelling Simulation (AMS)*, pages 861–866, May 2008. doi10.1109/AMS.2008.13. 33, 36

[14] R. Davidrajuh. Representing resources in petri net models: Hardwiring or soft-coding? In *Proceedings of 2011 IEEE International Conference on Service Operations, Logistics and Informatics*, pages 62–67, July 2011. doi10.1109/SOLI.2011.5986529. 33

[15] J. Desel and W. Reisig. Place/transition Petri nets. In *Advanced Course on Petri Nets*, pages 122–173. Springer, 1996. 5

[16] Y. Du, X. Wang, and H. Xu. A PIPE Based System for Checking Temporal Constraints in Service Composition. In *2011 International Conference on Information Management, Innovation Management and Industrial Engineering*, volume 2, pages 522–525, Nov 2011. doi10.1109/ICIII.2011.271. 36, 39

[17] H. T. Dung, B. H. Thang, and Q. T. Tho. Model Checking Control Flow Petri Nets Using PAT. In *2013 13th International Conference on Computational Science and Its Applications*, pages 124–129, June 2013. doi10.1109/ICCSA.2013.26. 36

[18] L. Dworzanski and D. Frumin. NPNtool: Modelling and analysis toolset for Nested Petri Nets. In *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*, number 7, pages 1–6, 2013. 41

[19] L. W. Dworzański and I. A. Lomazova. On Compositionality of Boundedness and Liveness for Nested Petri Nets. *Fundamenta Informaticae*, 120(3–4):275–293, 2012. 40

[20] L. W. Dworzański and I. A. Lomazova. CPN tools-assisted simulation and verification of nested Petri nets. *Automatic Control and Computer Sciences*, 47(7):393–402, 2013. 2, 40, 41

[21] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL Model Checker. In *Proceedings of the Workshop on Rewriting Logic and Its Applications*, volume 71 of *ENTCS*, pages 162–187, 2002. 41

[22] R. Eshuis. Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology*, 15(1):1–38, 2006. 2, 41

[23] B. Farwer and M. Leuschel. Model checking object Petri nets in Prolog. In *Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 20–31, 2004. 41

[24] H. Fleischhack and B. Grahlmann. Towards compositional verification of SDL systems. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume 7, pages 404–414, Jan 1998. doi10.1109/HICSS.1998.649235. 36, 40

[25] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar. Comparison of model checking tools for information systems. In *Proceedings of the 12th International Conference on Formal Engineering Methods and Software Engineering*, pages 581–596, 2010. 41

[26] A. Furfaro, L. Nigro, and F. Pupo. Distributed simulation of timed coloured Petri nets. In *Proceedings. Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pages 159–166, Oct 2002. doi10.1109/DISRTA.2002.1166902. 36, 39, 40

[27] G. C. Gannod and S. Gupta. An Automated Tool for Analyzing Petri Nets Using SPIN. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, pages 404–407. IEEE Computer Society, 2001. 2, 41

[28] G. Gardey, O. H. Roux, and O. F. Roux. State space computation and analysis of Time Petri Nets. *Theory and Practice of Logic Programming*, 6(3):301–320, 2006. doi10.1017/S147106840600264X. 40

[29] D. Gasevic and V. Devedzic. Software support for teaching Petri nets: P3. In *Proceedings 3rd IEEE International Conference on Advanced Technologies*, pages 300–301, July 2003. doi10.1109/ICALT.2003.1215093. 36

[30] G. Genter, S. Bogdan, Z. Kovacic, and I. Grubisic. Software tool for modeling, simulation and real-time implementation of Petri net-based supervisors. In *2007 IEEE International Conference on Control Applications*, pages 664–669, Oct 2007. doi10.1109/CCA.2007.4389308. 36

[31] C. Girault and R. Valk. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. 01 2003. ISBN 978-3-540-41217-5. 8, 11

[32] L. Gomes, J. P. Barros, A. Costa, and R. Nunes. The Input-Output Place-Transition Petri Net Class and Associated Tools. In *2007 5th IEEE International Conference on Industrial Informatics*, volume 1, pages 509–514, June 2007. doi10.1109/INDIN.2007.4384809. 33, 36

[33] B. Grahlmann and C. Pohl. Profiting from Spin in PEP. In *SPIN Workshop*, 1998. 2

[34] Y. Harie, Y. Mitsui, K. Fujimori, A. Batajoo, and K. Wasaki. HiPS: Hierarchical Petri Net design, simulation, verification and model checking tool. In *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, pages 1–5, Oct 2017. doi10.1109/GCCE.2017.8229199. 36

[35] B. J. Haverkort, J. M. Martinez, B. J. Haverkort, and J. M. Martinez. MathMC: A Mathematica-Based Tool for CSL Model Checking of Deterministic and Stochastic Petri Nets. In *Third International Conference on the Quantitative Evaluation of Systems (QEST'06)*, pages 133–134, Sept 2006. doi10.1109/QEST.2006.29. 36

[36] X. He, R. Zeng, S. Liu, Z. Sun, and K. Bae. A Term Rewriting Approach to Analyze High Level Petri Nets. In *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 109–112, July 2016. doi10.1109/TASE.2016.11. 36, 39

[37] L. M. Hillah, E. Kindler, F. Kordon, L. Petrucci, and N. Treves. A primer on the Petri Net Markup Language and ISO/IEC 15909-2. *Petri Net Newsletter*, 76:9–28, 2009. 44, 46

[38] G. J. Holzmann. Tutorial: Design and Validation of Protocols. *Tutorial Computer Networks and ISDN Systems*, 25:981–1017, 1991. 2

[39] G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23 (5):279–295, 1997. ISSN 0098-5589. doi10.1109/32.588521. 43, 49

[40] G. J. Holzmann, E. Najm, and A. Serhrouchni. Spin model checking: An introduction. *STTT*, 2:321–327, 03 2000. doi10.1007/s100090050039. 78

[41] ISO/IEC 15909-2:2011. Systems and software engineering – High-level Petri nets – Part 2: Transfer format. Standard 1, International Organization for Standardization, Geneva, CH, 2011. 2, 3, 43, 81

[42] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer Publishing Company, Incorporated, 2 edition, 1997. 2, 11, 13

[43] K. Jensen. *An introduction to the practical use of coloured Petri Nets*, pages 230–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49441-6. doi10.1007/3-540-65307-4_50. 11

[44] K. Jensen and G. Rozenberg. *High-level Petri nets: theory and application*. Springer Science & Business Media, 2012. 2

[45] B. Kitchenham. Procedures for Performing Systematic Reviews. Keele university. technical report tr/se-0401, Department of Computer Science, Keele University, UK, 2004. 31

[46] I. Koch. Petri nets in systems biology. *Software & Systems Modeling*, pages 1–8, 2014. 2

[47] M. Kuchárik and Z. Balogh. Evaluation of fuzzy Petri nets with the tool TransPlaceSim. In *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–5, Oct 2016. doi10.1109/ICAICT.2016.7991673. 36

[48] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, and R. Valk. An Extensible Editor and Simulation Engine for Petri Nets: Renew. In *ICATPN*, volume 3099 of *LNCS*, 2004. 73, 76

[49] C. A. Lakos. From coloured Petri Nets to Object Petri Nets. In *Application and Theory of Petri Nets 1995*, LNCS, pages 278–297. Springer, 1995. 12, 13, 16, 28, 32

[50] C. A. Lakos. *The object orientation of object Petri nets*. Department of Computer Science, University of Tasmania, 1995. 16

[51] D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of uml statechart diagrams using the spin model-checker. *Formal Aspects of Computing*, 11(6): 637–664, 1999. 2

[52] K. Latha and B. Umamaheswari. Supervisory control of an automated system with ladder logic programming and analysis using Petri nets. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, page 5, Oct 2002. doi10.1109/ICSMC.2002.1176037. 36

[53] N. Leyla, A. S. Mashiyat, H. Wang, and W. MacCaull. Towards workflow verification. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, pages 253–267, 2010. 2, 41

[54] Y. Li and F. Mao. Research of the Verification in Workflow Process Modeling on the Application of Petri Nets. In *2010 International Conference on e-Education, e-Business, e-Management and e-Learning*, pages 21–24, Jan 2010. doi10.1109/IC4E.2010.71. 33

[55] B. Liu and M. Ghazel. Petri Net Diagnosability Analyzer. In *2016 11th International Design Test Symposium (IDT)*, pages 13–18, Dec 2016. doi10.1109/IDT.2016.7843007. 36

[56] G. Liu and C. Jiang. Petri net based model checking for the collaborative-ness of multiple processes systems. In *2016 IEEE 13th International Conference on Networking, Sensing, and Control (ICNSC)*, pages 1–6, April 2016. doi10.1109/ICNSC.2016.7478980. 36

[57] Y. Liu. Game theory semantics for PCTL model checking label-extended probabilistic Petri net. In *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)*, pages 369–374, June 2014. doi10.1109/ICIS.2014.6912160. 36

[58] I. A. Lomazova. Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems. *Fundamenta Informaticae*, 43(1-4):195–214, 2000. 22

[59] I. A. Lomazova. Recursive Nested Petri Nets: Analysis of Semantic Properties and Expessibility. *Programming and Computer Software*, 27(4):183–193, 2001. 2, 28

[60] I. A. Lomazova. Nested Petri Nets: Multi-level and Recursive Systems. *Fundamenta Informaticae*, 47:283–293, 08 2001. 23, 24, 28, 29

[61] I. A. Lomazova. Modeling dynamic objects in distributed systems with nested Petri nets. *Fundamenta Informaticae*, 51(1-2):121–133, 2002. 28

[62] I. A. Lomazova. Nested Petri nets for adaptive process modeling. In *Pillars of Computer Science*, volume 4800 of *LNCS*, pages 460–474, 2008. 2

[63] I. A. Lomazova and V. O. Ermakova. Verification of Nested Petri Nets Using an Unfolding Approach. In *Proceedings of the International Workshop on Petri Nets and Software Engineering*, volume 1591 of *CEUR Workshop Proceedings*, pages 93–112, 2016. 40

[64] I. A. Lomazova and P. Schnoebelen. Some Decidability Results for Nested Petri Nets. In *Perspectives of System Informatics'99, 3rd International Andrei Ershov Memorial Conference*, volume 1755 of *LNCS*, pages 208–220, 2000. 28

[65] E. K. M. Jüngel and M. Weber. The Petri Net Markup Language. *Petri Net Newsletter*, 59: 24–29, 2000. 43

[66] T. Miyamoto and K. Horiguchi. Modular Reachability Analysis of Petri Nets for Multiagent Systems. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 43(6):1411–1423, 2013. 40

[67] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4): 541–580, April 1989. ISSN 0018-9219. doi10.1109/5.24143. 1, 5, 9

[68] F. Pereira, F. Moutinho, and L. Gomes. IOPT-tools — Towards cloud design automation of digital controllers with Petri nets. In *2014 International Conference on Mechatronics and Control (ICMC)*, pages 2414–2419, July 2014. doi10.1109/ICMC.2014.7232002. 33

[69] S. Popa and R. Dobrescu. Petri net models in computational biology. In *2015 E-Health and Bioengineering Conference (EHB)*, pages 1–4, Nov 2015. doi10.1109/EHB.2015.7391598. 36, 39

[70] O. Prisecaru and T. Jucan. Interorganizational Workflow Nets: a Petri Net Based Approach for Modelling and Analyzing Interorganizational Workflows. In *EOMAS*, pages 64–78, 2008. 72

[71] S. Proß, B. Bachmann, S. J. Janowski, and R. Hofestädt. A new object-oriented Petri net simulation environment based on Modelica. In *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–13, Dec 2012. doi10.1109/WSC.2012.6465287. 36

[72] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In *Proceedings of the International Conference on Applications and Theory of Petri Nets*, volume 2679 of *LNCS*, pages 450–462. 2003. 40

[73] G. Regis, N. Ricci, N. M. Aguirre, and T. Maibaum. Specifying and Verifying Declarative Fluent Temporal Logic Properties of Workflows. In *15th Brazilian Symposium on Formal Methods*, volume 7498 of *LNCS*, pages 147–162. 2012. 2, 41

[74] O. R. Ribeiro and J. M. Fernandes. Translating Synchronous Petri Nets into PROMELA for Verifying Behavioural Properties. In *International Symposium on Industrial Embedded Systems*, pages 266–273, 2007. 2, 41

[75] M. Schwarick, M. Heiner, and C. Rohr. MARCIE - Model Checking and Reachability Analysis Done EffiCIEntly. In *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, pages 91–100, Sept 2011. doi10.1109/QEST.2011.19. 36

[76] E. B. Sloane and V. Gelhot. Applications of the Petri net to simulate, test, and validate the performance and safety of complex, heterogeneous, multi-modality patient monitoring alarm systems. In *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 3492–3495, Sept 2004. doi10.1109/IEMBS.2004.1403980. 36

[77] E. Smith. *Principles of high-level net theory*, pages 174–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49442-3. doi10.1007/3-540-65306-6_16. URL https://doi.org/10.1007/3-540-65306-6_16. 2, 11

[78] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *ICATPN*, volume 1420, pages 1–25. 1998. 11

[79] R. Valk. Object Petri Nets – Using the Nets-within-Nets Paradigm. pages 819–848, 01 2003. doi10.1007/b98282. 11, 16, 32

[80] W. M. P. van der Aalst. Interorganizational Workflows: an Approach Based on Message Sequence Charts and Petri Nets. *Systems Analysis – Modelling – Simulation*, 34(3), 1999. 71

[81] W. M. P. van der Aalst. Business process management as the killer app for petri nets. *Software & Systems Modeling*, pages 1–7, 2014. 2

[82] M. L. F. Venero. Verifying Cross-Organizational Workflows Over Multi-Agent Based Environments. In J. Barjis and R. Pergl, editors, *Enterprise and Organizational Modeling and Simulation*, pages 38–58, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44860-1. 2, 81

[83] M. L. F. Venero and F. S. C. da Silva. On the Use of SPIN for Studying the Behavior of Nested Petri Nets. In J. Iyoda and L. de Moura, editors, *Formal Methods: Foundations and Applications*, volume 8195 of *LNCS*, pages 83–98. Springer Berlin Heidelberg, 2013. xii, 2, 22, 23, 24, 29, 41, 51, 52, 53, 58, 71, 72, 73, 75, 81

[84] M. L. F. Venero and F. S. C. da Silva. A general translation from nested Petri nets into PROMELA. *ArXiv e-prints*, 2014. URL https://arxiv.org/abs/1403.7991. 51, 55

[85] M. L. F. Venero and F. S. C. da Silva. Model checking multi-level and recursive nets. *Software & Systems Modeling*, pages 1–28, 2016. 1, 2, 51, 81, 82

[86] F. Vernadat, B. Berthomieu, F. Vernadat, and B. Berthomieu. Time Petri Nets Analysis with TINA. In *Third International Conference on the Quantitative Evaluation of Systems (QEST'06)*, pages 123–124, Sept 2006. doi10.1109/QEST.2006.56. 36

[87] X. Wu, L. Chen, and L. Zhang. Logistics distribution business process simulation and optimization based on Petri nets. In *6th International Conference on Pervasive Computing and Applications*, pages 126–128, Oct 2011. doi10.1109/ICPCA.2011.6106491. 33

[88] S. Yamaguchi, M. Yamaguchi, and M. Tanaka. A Soundness Verification Tool Based on the SPIN Model Checker for Acyclic Workflow Nets. In *Proceedings of the 23rd International Conference on Circuits/Systems, Computers and Communications*, pages 285–288, 2008. 2, 41, 72

[89] H. Yu and X. Wu. A Petri net software for mission reliability evaluation of PMS. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 6040–6044, May 2015. doi10.1109/CCDC.2015.7161894. 36, 39, 40

[90] A. Zimmermann, M. Knoke, A. Zimmermann, and M. Knoke. Distributed Simulation of Colored Stochastic Petri Nets With TimeNET 4.0. In *Third International Conference on the Quantitative Evaluation of Systems (QEST'06)*, pages 117–118, Sept 2006. doi10.1109/QEST.2006.16. 36

[91] W. R. Zuberek. Event-driven simulation of timed Petri net models. In *Proceedings 33rd Annual Simulation Symposium (SS 2000)*, pages 91–98, April 2000. doi10.1109/SIMSYM.2000.844905. 36

Apêndice A

Códigos PNML

A seguir, temos o código referente à rede da Figura 2.1.

```
<pnml xmlns="NPN">
   <net id="netId1599682219094" type="NPN">
      <place id="68">
         <initialMarking> <text>[]</text> </initialMarking>
         <initialMarking> <text>[]</text> </initialMarking>
         <name> <text>P1</text> </name>
      </place>
      <place id="71">
         <name> <text>P2</text> </name>
      </place>
      <place id="72">
         <name> <text>P3</text> </name>
      </place>
      <transition id="73">
         <name> <text>T1</text> </name>
      </transition>
      <arc id="74" source="68" target="73">
         <inscription> <text>[]</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <arc id="75" source="73" target="71">
         <inscription> <text>[]</text> </inscription>
         <inscription> <text>[]</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <arc id="76" source="73" target="72">
         <inscription> <text>[]</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <transition id="77">
         <name> <text>T2</text> </name>
      </transition>
      <arc id="78" source="71" target="77">
         <inscription> <text>[]</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <arc id="79" source="72" target="77">
```

```
38          <inscription> <text>[]</text> </inscription>
39          <type> <text>ordinary</text> </type>
40       </arc>
41       <arc id="80" source="77" target="68">
42          <inscription> <text>[]</text> </inscription>
43          <inscription> <text>[]</text> </inscription>
44          <type> <text>ordinary</text> </type>
45       </arc>
46       <name> <text>rede1</text> </name>
47    </net>
48 </pnml>
```

Algoritmo A.1: *Rede da Figura 2.1 no formato PNML.*

O código a seguir é referente à rede da Figura 2.7.

```
1  <!-- SN.pnml -->
2
3  <pnml xmlns="NPN">
4     <net id="netId1606949699936" type="NPN">
5        <place id="120">
6           <name> <text>ENTRANCE</text> </name>
7           <initialMarking> <text>D</text> </initialMarking>
8           <initialMarking> <text>D</text> </initialMarking>
9           <initialMarking> <text>A</text> </initialMarking>
10       </place>
11       <transition id="121">
12          <downlink_label> <text>TAKE_ORDER</text> </downlink_label>
13          <name> <text>T1</text> </name>
14       </transition>
15       <transition id="122">
16          <downlink_label> <text>DELIVER</text> </downlink_label>
17          <name> <text>T2</text> </name>
18       </transition>
19       <place id="123">
20          <name> <text>SENT</text> </name>
21       </place>
22       <place id="124">
23          <type> <text>shared</text> </type>
24          <initialMarking> <text>[]</text> </initialMarking>
25          <initialMarking> <text>[]</text> </initialMarking>
26          <initialMarking> <text>[]</text> </initialMarking>
27          <initialMarking> <text>[]</text> </initialMarking>
28          <initialMarking> <text>[]</text> </initialMarking>
29          <name> <text>ORDERS</text> </name>
30       </place>
31       <arc id="125" source="121" target="123">
32          <inscription> <text>x</text> </inscription>
33          <type> <text>ordinary</text> </type>
34       </arc>
35       <arc id="126" source="120" target="121">
36          <inscription> <text>x</text> </inscription>
37          <type> <text>ordinary</text> </type>
```

```
38          </arc>
39          <arc id="127" source="122" target="120">
40             <inscription> <text>x</text> </inscription>
41             <type> <text>ordinary</text> </type>
42          </arc>
43          <arc id="128" source="123" target="122">
44             <inscription> <text>x</text> </inscription>
45             <type> <text>ordinary</text> </type>>
46             <name> <text>MAINTENANCE</text> </name>
47          </place>
48          <transition id="141">
49             <name> <text>T3</text> </name>
50             <downlink_label> <text>BERECHARGED</text> </downlink_label>
51          </transition>
52          <transition id="143">
53             <name> <text>T4</text> </name>
54             <downlink_label> <text>BACK</text> </downlink_label>
55          </transition>
56          <arc id="145" source="139" target="143">
57             <inscription> <text>x</text> </inscription>
58             <type> <text>ordinary</text> </type>
59          </arc>
60          <arc id="146" source="143" target="120">
61             <inscription>
62                <text>x</text>
63             </inscription>
64             <type>
65                <text>ordinary</text>
66             </type>
67          </arc>
68          <arc id="147" source="120" target="141">
69             <inscription>
70                <text>x</text>
71             </inscription>
72             <type>
73                <text>ordinary</text>
74             </type>
75          </arc>
76          <arc id="149" source="141" target="139">
77             <inscription> <text>y</text> </inscription>
78             <inscription> <text>x</text> </inscription>
79             <type> <text>ordinary</text> </type>
80          </arc>
81          <place id="154">
82             <name> <text>WAITING</text> </name>
83             <initialMarking> <text>R</text> </initialMarking>
84          </place>
85          <arc id="156" source="154" target="141">
86             <inscription> <text>y</text> </inscription>
87             <type> <text>ordinary</text> </type>
88          <transition id="157">
89             <downlink_label> <text>R_BACK</text> </downlink_label>
90             <name> <text>T5</text> </name>
```

```
91          </transition>
92          <arc id="159" source="139" target="157">
93             <inscription> <text>y</text> </inscription>
94             <type> <text>ordinary</text> </type>
95          <arc id="160" source="157" target="154">
96             <inscription> <text>y</text> </inscription>
97             <type> <text>ordinary</text> </type>
98          </arc>
99          <place id="169">
100            <name> <text>READY</text> </name>
101         </place>
102         <arc id="170" source="169" target="121">
103            <type> <text>ordinary</text> </type>
104         </arc>
105         <transition id="171">
106            <name> <text>T0</text> </name>
107         </transition>
108         <arc id="173" source="124" target="171">
109            <type> <text>ordinary</text> </type>
110         </arc>
111         <arc id="174" source="171" target="169">
112            <type> <text>ordinary</text> </type>
113         </arc>
114         <name> <text>untitled</text> </name>
115      </net>
116   </pnml>
117
118   <!-- A.pnml -->
119
120   <pnml xmlns="NPN">
121      <net id="netId1606949609622" type="NPN">
122         <place id="57">
123            <type> <text>shared</text> </type>
124            <initialMarking> <text>[]</text> </initialMarking>
125            <initialMarking> <text>[]</text> </initialMarking>
126            <initialMarking> <text>[]</text> </initialMarking>
127            <initialMarking> <text>[]</text> </initialMarking>
128            <initialMarking> <text>[]</text> </initialMarking>
129            <name> <text>ORDERS</text> </name>
130         </place>
131         <place id="59">
132            <name> <text>OPEN</text> </name>
133            <initialMarking> <text>[]</text> </initialMarking>
134            <initialMarking> <text>[]</text> </initialMarking>
135            <initialMarking> <text>[]</text> </initialMarking>
136            <initialMarking> <text>[]</text> </initialMarking>
137            <initialMarking> <text>[]</text> </initialMarking>
138         </place>
139         <transition id="61">
140            <name> <text>TA0</text> </name>
141         </transition>
142         <arc id="62" source="59" target="61">
143            <inscription> <text>1</text> </inscription>
```

```
144            <type> <text>ordinary</text> </type>
145         </arc>
146         <arc id="63" source="61" target="57">
147            <inscription> <text>1</text> </inscription>
148            <type> <text>ordinary</text> </type>
149         </arc>
150         <place id="64">
151            <name> <text>WASTE</text> </name>
152         </place>
153         <arc id="66" source="61" target="64">
154            <inscription> <text>1</text> </inscription>
155            <type> <text>ordinary</text> </type>
156         </arc>
157         <place id="67">
158            <name> <text>BROKE</text> </name>
159         </place>
160         <transition id="69">
161            <uplink_label> <text>BERECHARGED</text> </uplink_label>
162            <name> <text>TA1</text> </name>
163         </transition>
164         <arc id="70" source="69" target="67">
165            <inscription> <text>1</text> </inscription>
166            <type> <text>ordinary</text> </type>
167         </arc>
168         <transition id="71">
169            <horizontal_label> <text>RECHARGE</text> </horizontal_label>
170            <name> <text>TA2</text> </name>
171         </transition>
172         <arc id="73" source="67" target="71">
173            <inscription> <text>1</text> </inscription>
174            <type> <text>ordinary</text> </type>
175         </arc>
176         <arc id="75" source="64" target="69">
177            <inscription> <text>5</text> </inscription>
178            <type> <text>ordinary</text> </type>
179         </arc>
180         <place id="89">
181            <name> <text>A_READY</text> </name>
182         </place>
183         <transition id="91">
184            <name> <text>TA3</text> </name>
185            <uplink_label> <text>BACK</text> </uplink_label>
186         </transition>
187         <arc id="93" source="71" target="89">
188            <inscription> <text>1</text> </inscription>
189            <type> <text>ordinary</text> </type>
190         </arc>
191         <arc id="94" source="89" target="91">
192            <inscription> <text>1</text> </inscription>
193            <type> <text>ordinary</text> </type>
194         </arc>
195         <arc id="95" source="91" target="59">
196            <inscription> <text>5</text> </inscription>
```

```
197          <type> <text>ordinary</text> </type>
198       </arc>
199       <name> <text>untitled</text> </name>
200    </net>
201 </pnml>
202 
203 <!-- D.pnml -->
204 
205 <pnml xmlns="NPN">
206    <net id="netId1606949557273" type="NPN">
207       <place id="1">
208          <name> <text>TRAVELING</text> </name>
209       </place>
210       <transition id="2">
211          <uplink_label> <text>TAKE_ORDER</text> </uplink_label>
212          <name> <text>TD0</text> </name>
213       </transition>
214       <transition id="3">
215          <uplink_label> <text>DELIVER</text> </uplink_label>
216          <name> <text>TD1</text> </name>
217       </transition>
218       <arc id="4" source="2" target="1">
219          <inscription> <text>1</text> </inscription>
220          <type> <text>ordinary</text> </type>
221       </arc>
222       <arc id="5" source="1" target="3">
223          <inscription> <text>1</text> </inscription>
224          <type> <text>ordinary</text> </type>
225       </arc>
226       <place id="11">
227          <name> <text>WASTE</text> </name>
228       </place>
229       <transition id="13">
230          <name> <text>TD2</text> </name>
231       </transition>
232       <place id="15">
233          <name> <text>RETURNING</text> </name>
234       </place>
235       <arc id="17" source="3" target="15">
236          <inscription> <text>1</text> </inscription>
237          <type> <text>ordinary</text> </type>
238       </arc>
239       <place id="18">
240          <name> <text>RESTING</text> </name>
241          <initialMarking> <text>[]</text> </initialMarking>
242          <initialMarking> <text>[]</text> </initialMarking>
243          <initialMarking> <text>[]</text> </initialMarking>
244       </place>
245       <arc id="20" source="18" target="2">
246          <inscription> <text>1</text> </inscription>
247          <type> <text>ordinary</text> </type>
248       </arc>
249       <arc id="21" source="13" target="11">
```

```
250           <inscription> <text>1</text> </inscription>
251           <type> <text>ordinary</text> </type>
252        </arc>
253        <place id="22">
254           <name> <text>BROKE</text> </name>
255        </place>
256        <transition id="24">
257           <uplink_label> <text>BERECHARGED</text> </uplink_label>
258           <name> <text>TD3</text> </name>
259        </transition>
260        <arc id="25" source="11" target="24">
261           <inscription> <text>3</text> </inscription>
262           <type> <text>ordinary</text> </type>
263        </arc>
264        <arc id="26" source="24" target="22">
265           <inscription> <text>1</text> </inscription>
266           <type> <text>ordinary</text> </type>
267        </arc>
268        <transition id="27">
269           <name> <text>TD4</text> </name>
270           <horizontal_label> <text>RECHARGE</text> </horizontal_label>
271        </transition>
272        <arc id="28" source="22" target="27">
273           <inscription> <text>1</text> </inscription>
274           <type> <text>ordinary</text> </type>
275        </arc>
276        <arc id="29" source="15" target="13">
277           <inscription> <text>1</text> </inscription>
278           <type> <text>ordinary</text> </type>
279           <name> <text>D_READY</text> </name>
280        </place>
281        <transition id="48">
282           <name> <text>TD5</text> </name>
283           <uplink_label> <text>BACK</text> </uplink_label>
284        </transition>
285        <arc id="51" source="27" target="45">
286           <inscription> <text>1</text> </inscription>
287           <type> <text>ordinary</text> </type>
288        </arc>
289        <arc id="52" source="45" target="48">
290           <inscription> <text>1</text> </inscription>
291           <type> <text>ordinary</text> </type>
292        </arc>
293        <arc id="53" source="48" target="18">
294           <inscription> <text>3</text> </inscription>
295           <type> <text>ordinary</text> </type>
296        </arc>
297        <name> <text>untitled</text> </name>
298     </net>
299  </pnml>
300
301  <!-- R.pnml -->
302
```

```
<pnml xmlns="NPN">
   <net id="netId1606949650735" type="NPN">
      <place id="101">
         <name> <text>R_WAITING</text> </name>
         <initialMarking> <text>[]</text> </initialMarking>
      </place>
      <transition id="104">
         <horizontal_label> <text>RECHARGE</text> </horizontal_label>
         <name> <text>TR0</text> </name>
      </transition>
      <place id="105">
         <name> <text>WORKING</text> </name>
      </place>
      <transition id="107">
         <uplink_label> <text>R_BACK</text> </uplink_label>
         <name> <text>TR1</text> </name>
      </transition>
      <arc id="108" source="101" target="104">
         <inscription> <text>1</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <arc id="109" source="104" target="105">
         <inscription> <text>1</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <arc id="110" source="105" target="107">
         <inscription> <text>1</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <arc id="111" source="107" target="101">
         <inscription> <text>1</text> </inscription>
         <type> <text>ordinary</text> </type>
      </arc>
      <name> <text>untitled</text> </name>
   </net>
</pnml>
```

Algoritmo A.2: *Rede da Figura 2.7 no formato PNML.*