Jonatha Rodrigues da Costa

Modelagem de Sistemas a Eventos Discretos

Maracanaú
2025

Jonatha Rodrigues da Costa

Modelagem de Sistemas a Eventos Discretos

Este livro trata sobre modelagem de sistemas a eventos discreotos aplicados à engenharia incluindo modelagem de sistemas utilizando a ferramenta de análise gráfica CPN IDE® .

Instituto Federal de Educação, Ciência e Tecnologia do Ceará

Maracanaú
2025

Dedico este trabalho a Deus, a minha família, aos colegas acadêmicos e aos discentes, especialmente os discentes dos cursos de engenharia.

Agradecimentos

"Estudar é uma dádiva divina!"

Resumo

Este livro apresenta um percurso abrangente sobre a modelagem de sistemas a eventos discretos, com ênfase nas redes de pétalas coloridas, articulando fundamentos conceituais, formalismos matemáticos e aplicações práticas. A obra inicia-se com uma análise detalhada de sistemas de manufatura, discutindo o processo de fabricação de produtos, os desafios de controle e as motivações para a modelagem. São introduzidos conceitos fundamentais, como percepção fabril, métodos de representação e as principais ferramentas de modelagem, culminando em uma visão formal por meio de máquinas de estados finitos e outros mecanismos de abstração.

A segunda parte conduz o leitor ao universo das Redes de Petri (RP), apresentando desde as representações formais e gráficas até o funcionamento conceitual de uma RP, incluindo sequência, concorrência, conflito, sincronismo, caminhos alternativos, repetição e alocação de recursos. São discutidas propriedades estruturais e comportamentais, como alcance, limitação, vivacidade, bloqueio, marcação morta, reversibilidade, persistência e uso de matrizes de incidência. Essa seção fornece ainda exemplos de modelagem de casos industriais, contemplando a construção e análise de matrizes de pré e pós-incidência, bem como a formulação de modelos que representam processos reais.

Em seguida, o livro aprofunda-se nas extensões temporais e hierárquicas das redes de Petri, explorando redes temporais e temporizadas, comparando suas vantagens e implicações para o desempenho de sistemas complexos. Essa abordagem permite compreender como o tempo e a hierarquia se integram à modelagem, oferecendo ferramentas mais expressivas para a representação de sistemas dinâmicos e distribuídos.

A parte central da obra é dedicada às Redes de Petri Coloridas (CPN), onde a expressividade do formalismo se amplia para lidar com sistemas de maior complexidade. Apresenta-se o ambiente *CPN IDE*®, descrevendo sua interface, funcionalidades, linguagem CPN ML e estruturas de controle, tais como *if-then-else*, *case-of*, funções anônimas e nomeadas, bem como recursos para manipulação de listas, recursividade e multiconjuntos. São detalhadas as técnicas de definição de conjuntos de cores — simples (UNIT, BOOLEAN, INTEGER, STRING, ENUMERATED, INDEXED) e compostos (PRODUCT, RECORD, UNION, LIST) — e sua aplicação em modelos de diferentes escalas. A obra também explora expressões de guarda, temporização de transições, operadores para entrada, saída e ações, além de estratégias de priorização de disparos, tudo integrado a exemplos práticos.

Para consolidar o aprendizado, o livro inclui estudos de caso e aplicações, como o modelo lúdico "Conto da Cinderela", que ilustra de maneira criativa a construção e análise de redes. Links para vídeos complementares oferecem uma experiência multimídia, reforçando a prática da modelagem passo a passo. Por fim, uma ampla lista de exercícios desafiadores permite que o leitor aplique o conhecimento adquirido, favorecendo a compreensão crítica e a autonomia na elaboração de modelos complexos.

Dirigida a estudantes, pesquisadores e profissionais das áreas de engenharia, computação e automação, esta obra oferece uma visão unificada da teoria e da prática na modelagem de sistemas a eventos discretos. Ao integrar fundamentos de manufatura, redes de Petri em suas diversas variantes e a sofisticação das redes de pétalas coloridas com o ambiente *CPN IDE*®, o livro constitui um guia completo para a análise, simulação e desenvolvimento de soluções em sistemas modernos e distribuídos.

Sumário

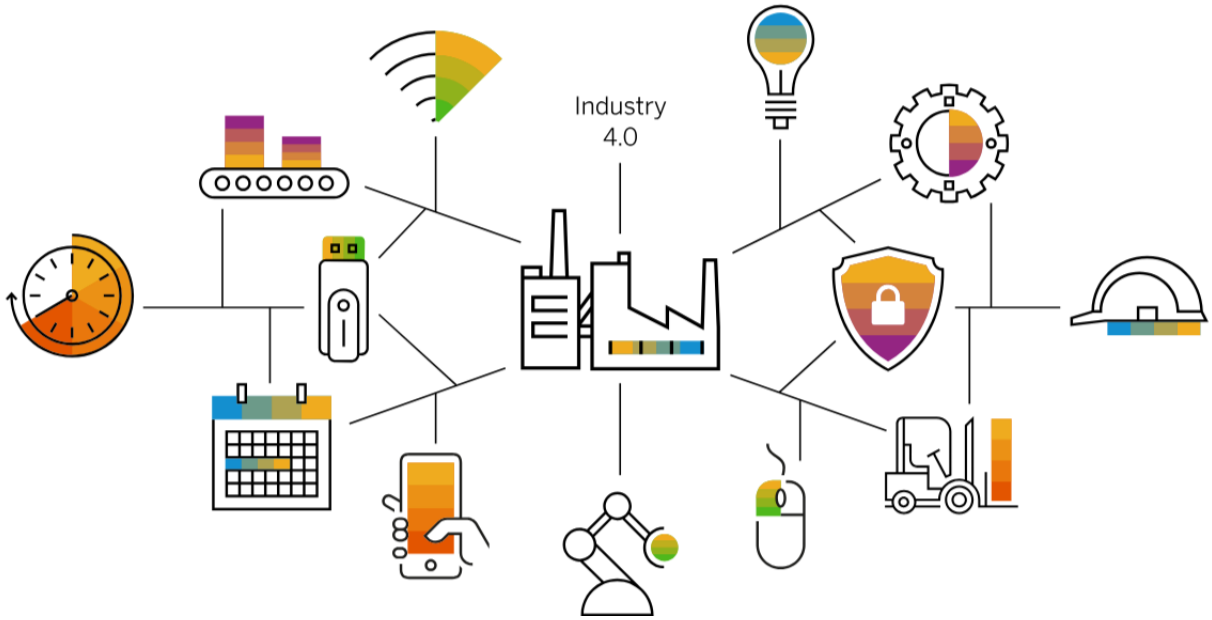
Lista de ilustrações

1 Sistemas de Manufatura: Fabricando um Produto, Modelagem e Problemas de Controle

A manufatura tem desempenhado um papel crucial como um dos pilares fundamentais para o desenvolvimento econômico e tecnológico ao longo de toda a história da humanidade. Desde os primeiros artefatos rudimentares, fabricados manualmente nas antigas civilizações, até os modernos e complexos sistemas automatizados de produção que caracterizam a indústria de ponta nos dias de hoje, a evolução dos processos de manufatura reflete diretamente o avanço da sociedade. Esse progresso se manifesta de forma clara na busca constante pela maior eficiência, qualidade e inovação no modo como os produtos são criados e entregues ao consumidor. Essa manufatura, mais do que uma simples atividade produtiva, se consolidou como uma força propulsora no desenvolvimento econômico, sendo também um reflexo das mudanças sociais e tecnológicas que moldaram o mundo moderno, conforme Smith (2010). No contexto atual, a manufatura está imersa numa revolução, impulsionada pela Indústria 4.0, que, ao incorporar novas tecnologias como a *Artificial Intelligence* (Inteligência Artificial) (AI), a robótica, *Internet of Things* (Internet das coisas) (IoT) e a computação em nuvem, tem transformando as formas de produção e, consequentemente, as bases da economia global, conforme Jones (2018).

Ilustrando esse cenário, na Figura 1 é apresentada uma percepção basilar de elementos distintos que, ao serem integrados em um único sistema, oferecem uma visão abrangente e interconectada de como diferentes componentes podem interagir de maneira síncrona. Esse arranjo dos elementos não só permite uma melhor compreensão da dinâmica interna do sistema, mas também amplia a percepção sobre suas funcionalidades e possibilidades. A integração dos diversos componentes oferece uma abordagem mais holística e eficaz, que facilita a concepção e o entendimento de sistemas complexos, permitindo uma análise mais detalhada e uma visão mais clara de como cada parte contribui para o todo.

Figura 1 – Percepção da indústria 4.0



Fonte:Adaptado de (SAP, 2025)

O conceito de sistemas de manufatura, portanto, pode ser definido como um conjunto organizado de processos físicos e lógicos que interagem para transformar matérias-primas em produtos acabados, atendendo às demandas específicas de consumidores e mercados. Historicamente, pode-se identificar quatro grandes fases no desenvolvimento dos sistemas de manufatura. A primeira fase, a manufatura artesanal, era caracterizada pela produção manual e altamente personalizada de produtos, em que cada item produzido era único, feito sob medida de acordo com as necessidades do cliente. Já a segunda fase, marcada pela revolução industrial e pela introdução da linha de montagem, impulsionou a produção em massa, que permitiu a fabricação de grandes quantidades de produtos padronizados a custos muito mais baixos, representando um marco na história da produção. A terceira fase, a manufatura enxuta, focou na redução de desperdícios e na melhoria contínua dos processos, buscando a máxima eficiência na utilização de recursos e tempo, sem sacrificar a qualidade, conforme Anderson (2015). Essa estrutura sistêmica converge na Indústria 4.0, que integra a digitalização dos processos produtivos, automação avançada e sistemas ciberfísicos, possibilitando um nível de personalização e eficiência que antes era impensável. Essa nova fase tem o potencial de transformar profundamente todos os aspectos da produção, tornando-a mais ágil, inteligente e conectada, segundo Mellor (2014).

1.1 PERCEPTIVA FABRIL

Dentro desse amplo panorama, os sistemas de manufatura podem ser classificados com base na natureza do processo produtivo em três categorias principais: **sistemas de produção por lote**, **sistemas de produção de fluxo contínuo** e **sistemas flexíveis de manufatura**.

Nos *sistemas de produção por lote*, os produtos são fabricados em unidades individualmente identificáveis e contáveis. Cada item é tratado separadamente, permitindo rastreabilidade e controle específicos, como ocorre na produção de automóveis, eletrodomésticos e dispositivos eletrônicos. Já os *sistemas de produção de fluxo contínuo* operam com o processamento ininterrupto de materiais, caracterizando-se por fluxos constantes ao longo do tempo. São típicos desse modelo setores como o químico, petroquímico e alimentício, em que a produção envolve reações ou transformações físicas que ocorrem de forma contínua, sem interrupções entre as etapas. Os *sistemas flexíveis de manufatura*, por sua vez, representam uma abordagem intermediária e adaptável, capaz de lidar com diferentes tipos de produtos e variações de demanda. Esses sistemas são projetados para ajustar-se rapidamente a mudanças nas especificações de produção, incorporando tecnologias como controle numérico computadorizado e automação programável, permitindo assim maior personalização e agilidade no atendimento ao mercado. Nesse contexto, *CAM - Computer Aided Manufacturing*) (Manufatura Assistida por Computador) (CAM) surge como uma tecnologia fundamental para otimizar processos, empregando simulações e controle digital para maximizar a eficiência e minimizar erros durante a produção, conforme apresenta Baskin (2017). Isso implica em produtos manufaturados com ciclos de vida distintos entre si, tais que podem ou não resultar de projetos com variações de qualidade e segurança.

O ciclo de vida de um produto, desde sua concepção até sua disposição final, envolve uma série de etapas interdependentes , as quais podem ser definidas como: fase inicial, projeto, fabricação e reciclagem. A fase inicial, que abrange o projeto e o desenvolvimento do produto, é de extrema importância, pois é nessa fase que são definidos aspectos cruciais como a funcionalidade, os materiais a serem utilizados e os processos de fabricação mais adequados para atingir a qualidade desejada, conforme Harris (2002). Após o projeto, segue-se o planejamento da produção, em que são definidos os recursos necessários,

como equipamentos, materiais e mão de obra, bem como os processos a serem seguidos. A etapa de fabricação é em que ocorre a transformação real da matéria-prima no produto acabado, seguindo-se, então, a distribuição e o suporte, garantindo a entrega do produto e a manutenção adequada ao longo de sua vida útil, conforme Wilson (2016). O ciclo se fecha com a reciclagem ou o descarte adequado do produto, uma fase cada vez mais importante, dada a crescente preocupação com a sustentabilidade e o reaproveitamento de materiais, conforme Brundtland (1987).

Para garantir a eficiência e o controle sobre esses processos, a modelagem de sistemas de manufatura desempenha um papel essencial. Modelos formais, como diagramas de fluxo, máquinas de estados finitos e Redes de Petri, são ferramentas amplamente utilizadas para a análise, otimização e simulação dos fluxos produtivos. Essas ferramentas permitem que engenheiros e gestores analisem de maneira detalhada o comportamento dos sistemas e identifiquem pontos críticos, como gargalos e potenciais falhas. Com isso, é possível antecipar problemas e otimizar a sincronização entre as diversas etapas da produção, resultando em uma maior eficiência operacional, conforme Murata (1989).

No entanto, mesmo com a utilização dessas ferramentas de modelagem, os sistemas de manufatura enfrentam desafios consideráveis no controle de suas operações. A sincronização de processos, o balanceamento da linha de produção e a gestão eficaz de estoques e logística são questões complexas que exigem soluções criativas e eficientes por parte dos engenheiros e gestores. A teoria de controle supervisório, por exemplo, oferece uma abordagem robusta para enfrentar esses desafios. Baseada na modelagem com Redes de Petri, essa teoria permite a supervisão e coordenação eficaz dos sistemas produtivos, possibilitando uma maior flexibilidade e uma resposta rápida às mudanças nas condições de produção, conforme Chase e Aquilano (2006). A integração de diferentes abordagens de controle também é fundamental para a adaptação das fábricas às exigências da produção moderna, cada vez mais dinâmicas e variáveis,conforme Cassandras (2008).

Dessa forma, compreender a estrutura e o funcionamento dos sistemas de manufatura é essencial para os profissionais da engenharia moderna. A modelagem eficiente e o controle adequado desses sistemas não apenas otimizam a produção, mas também impulsionam a inovação, a competitividade e o avanço tecnológico, permitindo que as empresas se destaquem no mercado globalizado (THOMPSON, 2019). As organizações que adotam essas tecnologias e abordagens têm uma vantagem estratégica significativa, pois conseguem produzir com mais eficiência, atender de forma mais precisa às necessidades dos consumidores e se adaptar rapidamente às mudanças do mercado.

1.2 CONCEITOS DE MODELAGEM DE SISTEMAS DE MANUFATURA

A modelagem de sistemas de manufatura é uma técnica essencial para representar, analisar e otimizar processos complexos dentro das fábricas. Modelar um sistema envolve criar uma abstração que descreve seus componentes, interações e comportamentos. Essas representações são fundamentais para entender e prever como o sistema opera, identificar falhas potenciais, otimizar o desempenho e apoiar decisões de planejamento e controle. A modelagem de sistemas pode ser dividida em diferentes abordagens, como modelagem gráfica, matemática e computacional.

Entre os conceitos-chave em modelagem, destacam-se:

- **Abstração**: Reduzir a complexidade de um sistema real, considerando apenas os aspectos mais relevantes para a análise ou melhoria do desempenho.

- **Simulação**: Técnica usada para imitar o comportamento do sistema sob diferentes condições, o que ajuda a prever os resultados de mudanças no processo sem a necessidade de experimentos reais.
- **Otimização**: Processo de encontrar a melhor solução para um problema de manufatura, seja em termos de custo, tempo ou recursos, ajustando variáveis dentro de um modelo matemático.
- **Estudo de Fluxo**: Análise dos caminhos que os materiais percorrem ao longo do sistema, buscando melhorar a fluidez e reduzir os tempos de espera e os gargalos.

1.2.1 PRINCIPAIS FERRAMENTAS DE MODELAGEM DE SISTEMAS

Diversas ferramentas e técnicas são utilizadas para modelar sistemas de manufatura. Cada ferramenta possui características distintas, adequando-se a diferentes tipos de problemas e abordagens de análise. Na Tabela 1 são apresentadas as principais ferramentas de modelagem, seus critérios de classificação e recomendações de utilização.

Tabela 1 – Principais ferramentas de modelagem de sistemas de manufatura.

ID	Ferramenta	Critérios de Classificação	Recomendações de Utilização
1	Diagrama de Fluxo	Simples, direto e visual. Ideal para sequenciamento lógico e entendimento geral do processo.	Indicado para sistemas com baixa complexidade e processos bem definidos.
2	Máquinas de Estados Finitos	Representa sistemas discretos com transições entre estados claramente definidos.	Recomendado para modelar processos sequenciais como linhas de montagem.
3	Redes de Petri	Forte capacidade de modelar simultaneidade, concorrência e sincronização.	Adequada para sistemas complexos e concorrentes, especialmente em controle de produção.
4	Simulação de Monte Carlo	Modelagem baseada em variáveis aleatórias e distribuições de probabilidade.	Utilizada em processos sujeitos a incertezas e variações estocásticas, como falhas ou tempos de ciclo.
5	Programação Linear	Abordagem matemática formal com foco em otimização sob restrições.	Indicada para resolver problemas de alocação de recursos e balanceamento de linhas.
6	Algoritmos Genéticos	Técnicas metaheurísticas baseadas em seleção natural e evolução.	Aplicados em problemas complexos de otimização com múltiplos critérios e grandes espaços de busca.
7	Modelagem Baseada em Agentes	Enfatiza a interação entre múltiplos componentes autônomos.	Ideal para sistemas distribuídos e dinâmicos, como fábricas inteligentes e logística adaptativa.

Fonte: Autor (2025)

A escolha da ferramenta de modelagem adequada depende diretamente do tipo de sistema a ser analisado, das características dos processos envolvidos e dos objetivos da análise. Por exemplo, para sistemas mais simples, em que os processos são claros e as variáveis poucas, diagramas de fluxo podem ser

suficientes. No entanto, quando lidamos com sistemas mais dinâmicos, com muitas variáveis e interações complexas, ferramentas mais sofisticadas, como Redes de Petri (RP) e simulações de Monte Carlo, podem ser mais eficazes. Isso nos leva à questão central de como essas ferramentas nos ajudam a entender e otimizar sistemas de manufatura, especialmente quando se trata de analisar eventos discretos.

Em todos os cenários, a modelagem de sistemas de manufatura permite uma visão mais integrada da operação como um todo. Ao aplicar essas ferramentas, conseguimos identificar pontos críticos, como gargalos e falhas potenciais, além de descobrir oportunidades para melhorias. Essa abordagem, ao mesmo tempo analítica e estratégica, é fundamental para o desenvolvimento de sistemas de produção mais eficientes e adaptáveis, alinhados às necessidades da Indústria 4.0. E, ao falarmos da Indústria 4.0, nos deparamos com uma transformação profunda na forma como os sistemas de manufatura são gerenciados, o que torna a escolha da ferramenta certa ainda mais relevante.

No entanto, quando o foco está na modelagem de sistemas de manufatura baseados em eventos discretos, as RP se destacam como uma ferramenta particularmente poderosa. Sua capacidade de representar a dinâmica de sistemas complexos, em que múltiplos componentes interagem de maneira concorrente e assíncrona, torna-as ideais para ambientes em que a sincronização e a comunicação entre eventos são fundamentais. Ao contrário de abordagens mais tradicionais, as Redes de Petri nos fornecem uma representação visual clara e intuitiva dessas interações, o que facilita o entendimento da complexidade dos sistemas. Isso é especialmente importante em sistemas de manufatura modernos, como os encontrados em fábricas automatizadas ou ambientes de produção flexível, em que a variabilidade e a incerteza desempenham um papel significativo.

Portanto, ao nos aprofundarmos nesse tema, vemos que as RP não são apenas uma ferramenta teórica, mas uma solução prática e altamente adaptável para modelar e otimizar sistemas com eventos discretos. Elas nos permitem simular e analisar o comportamento de sistemas sujeitos a mudanças e falhas, oferecendo insights valiosos para melhorar a eficiência e a produtividade. Por essas razões, no próximo capítulo, adotaremos as RP como a principal ferramenta de estudo. Com isso, seremos capazes de realizar uma análise aprofundada das interações dentro dos sistemas de manufatura, identificando pontos críticos que impactam diretamente seu desempenho, ao mesmo tempo em que propomos melhorias para alcançar um processo de produção mais robusto e eficiente.

1.3 CONCEITOS E TERMOS-CHAVE UTILIZADOS NA MODELAGEM

Para compreender adequadamente o comportamento dos sistemas de manufatura sob diferentes abordagens, é fundamental distinguir algumas classificações relevantes no **domínio da modelagem** e controle de **sistemas dinâmicos**. Conforme Cardoso e Valette (2007), destacam-se três categorias principais referentes a tais sistemas:

(a) **Sistemas discretizados**: são sistemas contínuos observados em instantes discretos, ou seja, sistemas cujo estudo ocorre apenas em momentos específicos. As variáveis de estado evoluem de maneira contínua, sem mudanças bruscas, mas o interesse está restrito aos seus valores em determinados instantes de tempo.

(b) **Sistemas discretos**: nesses sistemas, os valores das variáveis de estado (ou de algumas delas) apresentam variações abruptas em certos instantes, os quais, entretanto, nem sempre podem ser

previstos. Além disso, conhecer o estado atual não é suficiente para determinar diretamente o próximo estado sem algum tipo de cálculo intermediário.

(c) **Sistemas a Eventos Discretos (SED)**: são sistemas modelados de forma que as variáveis de estado sofrem mudanças bruscas em instantes determinados pela ocorrência de eventos. Nesses casos, os valores das variáveis nos estados seguintes podem ser calculados diretamente a partir dos valores anteriores, sem necessidade de considerar o tempo decorrido entre os eventos. Os SEDs são particularmente úteis na modelagem de processos industriais, nos quais a evolução do sistema depende da ocorrência de eventos como o início ou término de operações, a ativação de sensores ou a chegada de materiais.

Para os propósitos de modelagem aplicada, este material trabalhará com SEDs.

1.3.1 CONCEITOS UTILIZADOS NA MODELAGEM

A modelagem de sistemas baseados em eventos discretos fundamenta-se em conceitos essenciais:

Eventos: são instantes nos quais ocorre uma mudança de estado no sistema, representando momentos de observação relevantes.

Atividades: referem-se aos processos ocultos (caixas-pretas) que descrevem a evolução do sistema físico entre dois eventos. Em geral, os eventos correspondem ao início e ao término de uma atividade.

Processos: consistem em sequências de eventos e atividades interdependentes. Por exemplo, um evento pode iniciar uma atividade que, ao ser concluída, gera outro evento, o qual, por sua vez, desencadeia uma nova atividade, e assim sucessivamente.

Paralelismo, cooperação e competição

Os processos em um sistema podem evoluir de maneira simultânea ou sequencial. Quando ocorre de forma simultânea, os processos podem ser completamente independentes ou apenas relativamente independentes.

A independência relativa implica que algumas atividades sejam totalmente independentes entre si, enquanto outras necessitem de pontos de sincronização, isto é, eventos comuns a diferentes evoluções. Existem diferentes formas de interação entre processos:

Cooperação: ocorre quando processos distintos convergem para um objetivo comum. Busca-se, nesse caso, descrever a independência de cada processo antes de atingirem um ponto de sincronização.

Competição: caracteriza-se pela necessidade de múltiplos processos acessarem um mesmo recurso para realizar suas tarefas. Se houvesse recursos ilimitados, os processos seriam totalmente independentes. Entretanto, como normalmente há restrições, é necessário um mecanismo de compartilhamento, geralmente implementado por exclusões mútuas. Assim, descreve-se a exclusão entre processos a partir de pontos de sincronização.

Pseudo-paralelismo: ocorre quando o paralelismo é apenas aparente, pois mesmo que os eventos sejam independentes, eles nunca ocorrem simultaneamente. A execução é ordenada por um relógio comum. Um exemplo é o processamento de várias tarefas por um único processador, que executa apenas uma instrução por vez.

Paralelismo verdadeiro: ocorre quando eventos podem acontecer exatamente ao mesmo tempo, sem que exista uma escala de tempo comum suficientemente precisa para determinar qual evento ocor-

reu antes. Esse tipo de paralelismo é típico em sistemas computacionais paralelos, nos quais tarefas independentes são executadas simultaneamente por diferentes processadores.

1.3.2 MÁQUINA DE ESTADOS FINITOS

Processo sequencial único

A representação clássica de sistemas a eventos discretos com número finito de estados consiste em enumerar todos os estados possíveis e descrever as mudanças de estado (eventos) que os conectam, especificando o próximo estado a partir de cada situação atual.

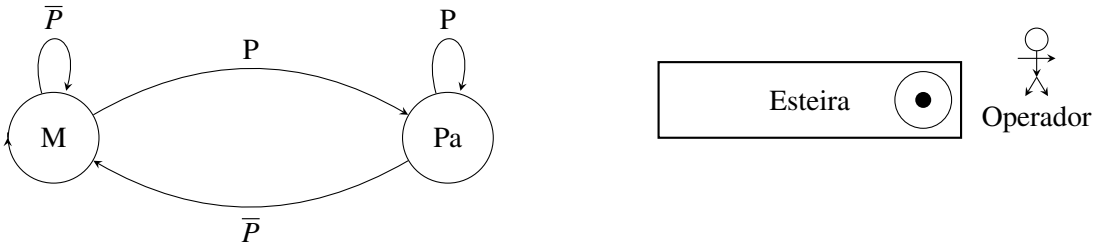
Matematicamente, esse modelo é definido como uma máquina de estados finitos $M = (E, A, \theta, E_0)$, onde:

- E é o conjunto finito de estados;
- E_0 é o estado inicial;
- A é o alfabeto de entrada (conjunto de eventos);
- θ [1]: $E \times A \to E$ é a função de transição, que associa a cada par estado-evento o próximo estado.

Essa máquina pode ser representada por um grafo, no qual os nós correspondem aos estados e os arcos representam as transições de estado, rotulados pelos eventos que as provocam. O modelo de máquina de estados finitos expressa adequadamente a noção de evento, e parcialmente a de atividade (considerando o estado como o período entre dois eventos). No entanto, não representa a noção de processo como evolução simultânea de múltiplos processos paralelos, pois descreve apenas um processo sequencial único.

Exemplo. Considere um sistema de triagem em uma esteira transportadora T. Objetos pesados devem ser removidos por um operador. Um sensor P detecta a presença de um objeto pesado, situação na qual a esteira deve parar para possibilitar a retirada. Após a remoção, a esteira volta a se mover, conforme representado na Figura 2.

Figura 2 – Máquina de estados e representação de processo industrial



(a) (b)

Notas: (a) Máquina de estados do sistema de triagem, indicando os estados de movimento (M) e parada (Pa) com transições baseadas nos eventos P (detecção de objeto pesado) e $\overline{P}$ (ausência do objeto pesado). (b) Representação esquemática da esteira transportadora com lugar P (token indica detecção de objeto pesado) e operador responsável pela remoção.

Fonte: Adaptado de Cardoso e Valette (2007)

[1] θ (teta) é a letra minúscula do alfabeto grego; sua forma maiúscula é Θ.

Este modelo descreve que, enquanto a esteira estiver em movimento (M) e o sensor detectar um objeto pesado (P), o sistema transita para o estado de parada (Pa). Quando o objeto pesado é retirado ($\overline{P}$), a esteira retorna ao movimento (M).

Processo sequencial único

A representação clássica de sistemas a eventos discretos com número finito de estados consiste em enumerar todos os estados possíveis e descrever as mudanças de estado (eventos) que os conectam, especificando o próximo estado a partir de cada situação atual.

Matematicamente, esse modelo é definido como uma máquina de estados finitos $M = (E, A, \theta, E_0)$, onde:

- E é o conjunto finito de estados;
- E_0 é o estado inicial;
- A é o alfabeto de entrada (conjunto de eventos);
- $\theta : E \times A \to E$ é a função de transição, que associa a cada par estado-evento o próximo estado.

Essa máquina pode ser representada por um grafo, no qual os nós correspondem aos estados e os arcos representam as transições de estado, rotulados pelos eventos que as provocam. O modelo de máquina de estados finitos expressa adequadamente a noção de evento, e parcialmente a de atividade (considerando o estado como o período entre dois eventos). No entanto, não representa a noção de processo como evolução simultânea de múltiplos processos paralelos, pois descreve apenas um processo sequencial único.

Adicionalmente, os autômatos finitos (ou máquinas de estados) são amplamente utilizados para modelagem de sistemas discretos devido à sua simplicidade e clareza. Por meio deles, é possível especificar formalmente o comportamento de controladores, sistemas de supervisão, protocolos de comunicação e circuitos digitais, entre outros. Sua força reside na definição explícita de todos os estados possíveis e na descrição detalhada de cada transição provocada por eventos, permitindo análises como verificação de alcançabilidade, consistência e ausência de estados mortos.

Porém, apesar de sua utilidade para processos sequenciais, os autômatos apresentam limitações importantes quando se deseja modelar sistemas complexos, caracterizados por:

- Evolução concorrente de múltiplos subprocessos independentes ou parcialmente dependentes;
- Sincronização entre diferentes componentes do sistema;
- Compartilhamento de recursos ou restrições que envolvem contagem de objetos, tarefas ou permissões;
- Explosão combinatória do número de estados quando há necessidade de representar todas as combinações possíveis de estados parciais.

Diante dessas desvantagens, torna-se necessário um modelo mais robusto e expressivo para representar sistemas a eventos discretos com comportamento paralelo e concorrente de forma natural e compacta. Nesse contexto, as **Redes de Petri** surgem como uma alternativa poderosa, oferecendo vantagens significativas para a modelagem, análise e verificação de sistemas complexos, como será aprofundado na próxima seção.

2 Redes de Petri: conceitos iniciais

As RPs constituem uma ferramenta matemática e gráfica de alta expressividade, empregada na modelagem, análise e simulação de SEDs. Introduzidas por Carl Adam Petri em 1962, essas redes têm sido amplamente adotadas para representar sistemas concorrentes, tanto síncronos quanto assíncronos, além de sistemas distribuídos. Seu emprego se destaca em áreas como automação industrial, protocolos de comunicação, controle de manufatura e gerenciamento de fluxos de trabalho em sistemas de informação.

2.1 REPRESENTAÇÕES FORMAIS DA REDE DE PETRI

Para formalizar a modelagem, a Rede de Petri pode ser descrita por diferentes estruturas matemáticas, dependendo do nível de detalhe necessário. Em literatura, a definição formal de uma Rede de Petri pode variar conforme o nível de detalhamento e os aspectos que se deseja enfatizar na modelagem. Assim, podem ser apresentadas diferentes definições, tais como no conceito de uma n-tupla. O termo n-upla refere-se ao número de elementos considerados na definição da RP, podendo assumir diferentes configurações:

- **Dupla (2 elementos)**: estrutura básica da rede;
- **Trípla (3 elementos)**: inclui conexões entre os componentes;
- **Quádrupla (4 elementos)**: adiciona a função de peso dos arcos;
- **Quíntupla (5 elementos)**: incorpora a marcação inicial da rede;
- **Séxtupla (6 elementos)**: inclui informações complementares, como rótulos e tempos.

RP COMO UMA DUPLA

Para incluir a noção de pesos associados aos arcos, pode-se definir, conforme Equação 1:

$$RP = (P, T) \tag{1}$$

em que:

- P é o conjunto finito de *lugares*;
- T é o conjunto finito de *transições*;

RP COMO UMA TRIPLA

Para incluir a noção de pesos associados aos arcos, pode-se definir, conforme Equação 2:

$$RP = (P, T, F) \tag{2}$$

em que:

- P, T possuem os mesmos significados anteriores;
- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto de *arcos*, que conecta lugares a transições e vice-versa;

RP COMO UMA QUADRUPLA

Para incluir a noção de pesos associados aos arcos, pode-se definir, conforme Equação 3:

$$RP = (P, T, F, W) \tag{3}$$

em que:

- P, T e F possuem os mesmos significados anteriores;
- $W : F \rightarrow \mathbb{N}^+$ é uma função que atribui um *peso* a cada arco, representando, por exemplo, a quantidade mínima de *fichas* necessários para que uma transição seja disparada.

RP COMO UMA QUÍNTUPLA

A definição clássica de uma Rede de Petri integra tanto os pesos quanto a marcação inicial, conforme Equação 4::

$$RP = (P, T, F, W, M_0) \tag{4}$$

- P, T, F e W possuem os mesmos significados anteriores;
- $M_0 : P \rightarrow \mathbb{N}$ é a *marcação inicial*, definindo a distribuição inicial de *fichas* em cada lugar.

RP COMO UMA SÉXTUPLA

Em contextos que demandam uma modelagem mais rica, como em sistemas temporizados ou hierárquicos, pode-se estender a definição, conforme Equação 5, para:

$$RP = (P, T, F, W, M_0, \Lambda) \tag{5}$$

em que:

- P, T, F, W e M_0 : possuem os mesmos significados anteriores;
- Λ[2]: é uma função adicional que pode representar rótulos, restrições, tempos de disparo, prioridades ou quaisquer outras informações complementares necessárias para descrever aspectos específicos do sistema modelado.

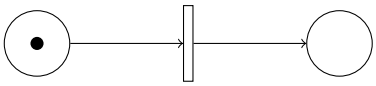
Cada uma dessas definições é útil em contextos distintos, permitindo que o modelador/projetista escolha a abordagem que melhor se adequa à complexidade e aos requisitos do sistema a ser analisado.

2.2 REPRESENTAÇÃO GRÁFICA

Uma RP pode ser representada graficamente conforme a Figura 3. Os lugares são representados por círculos, as transições por retângulos e as fichas (*fichas*) por pontos dentro dos círculos.

[2] Λ (lambda) é a forma maiúscula da letra no alfabeto grego; a forma minúscula é λ.

Figura 3 – Exemplo de Rede de Petri simples



Fonte: Autor, (2025)

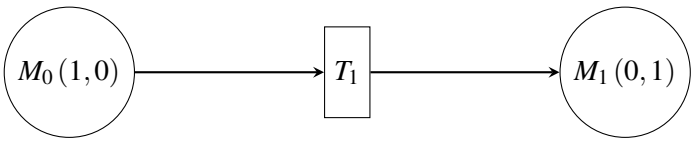
2.2.1 FUNCIONAMENTO CONCEITUAL DE UMA RP

Na Figura 4 ilustra-se a evolução de uma Rede de Petri simples. Considere uma rede com dois lugares (P_1 e P_2) e uma transição (T_1). A marcação inicial possui um ficha em P_1:

1. Estado inicial: $M_0 = (1,0)$ (um ficha em P_1, nenhum em P_2).
2. A transição T_1, estando habilitada, pode disparar, removendo um ficha de P_1 e adicionando um ficha em P_2.
3. Novo estado: $M_1 = (0,1)$ (o ficha é transferido para P_2).

Esse processo ilustra a evolução do sistema conforme os eventos ocorrem, permitindo uma análise dinâmica e detalhada do comportamento da rede.

Figura 4 – Grafo de marcações ilustrando a evolução do sistema



Fonte: Autor, (2025)

Nesse contexto, podem ser realizadas a modelagem e análise de sistemas de manufatura, a simulação de protocolos de comunicação, a modelagem de processos de *workflow* (fluxo de trabalho) e *Business Process Management - BPM* (Gerenciamento de Processos de Negócio), sistemas embarcados e de tempo real, e automação de processos industriais, dentre outros. Esses sistemas contém mais recursos, eventos e condicionantes para que cada evento ocorra.

2.3 PODER DE EXPRESSÃO DAS REDES DE PETRI

As RPs possuem um alto poder de expressão, sendo capazes de modelar diversos comportamentos de sistemas concorrentes. A seguir, ilustramos alguns desses comportamentos utilizando grafos de RPs.

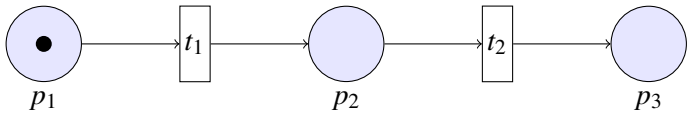
2.3.1 SEQUÊNCIA

A sequência é uma das estruturas fundamentais de um processo modelado em redes de Petri. Nesse tipo de relação, a ocorrência de uma transição depende estritamente da execução de outra anterior, estabelecendo, assim, uma ordem causal entre os eventos.

De forma mais precisa, diz-se que uma transição t_2 está em sequência com uma transição t_1 quando t_2 somente pode ser habilitada após o disparo de t_1. Isso ocorre porque o lugar intermediário, resultante da execução de t_1, deve conter pelo menos uma ficha para que t_2 seja ativada. Esse comportamento

representa, de maneira direta, a noção de fluxo ordenado de atividades, em que cada ação só é possível após a conclusão da anterior. A Figura 5 ilustra esse encadeamento, evidenciando como a rede de Petri capta a estrutura sequencial dos processos. Esse tipo de modelagem é particularmente relevante para sistemas que demandam execução estritamente controlada, como protocolos de comunicação, fluxos de trabalho industriais ou etapas de controle em sistemas embarcados.

Figura 5 – Expressão de processo em sequência numa RP



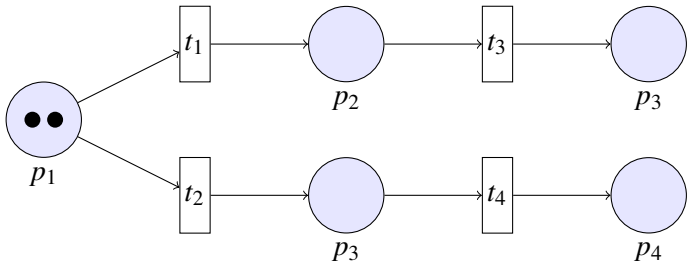
Fonte: Autor, (2025)

2.3.2 CONCORRÊNCIA

A concorrência expressa a possibilidade de duas ou mais atividades ocorrerem de forma independente e, potencialmente, simultânea. Em redes de Petri, isso acontece quando transições distintas não possuem relação direta de conflito ou de causalidade. Mais precisamente, duas transições t_1 e t_2 são concorrentes quando ambas podem ser habilitadas ao mesmo tempo, desde que seus lugares de entrada possuam as fichas necessárias. A execução de uma não interfere na execução da outra.

Na Figura 6 é exemplificado esse comportamento, que é central para a representação de sistemas paralelos e distribuídos. Essa característica torna as redes de Petri especialmente úteis no estudo de arquiteturas multiprocessadas, processos industriais simultâneos e sistemas de controle com múltiplas tarefas independentes.

Figura 6 – Expressão de processo em concorrência numa RP



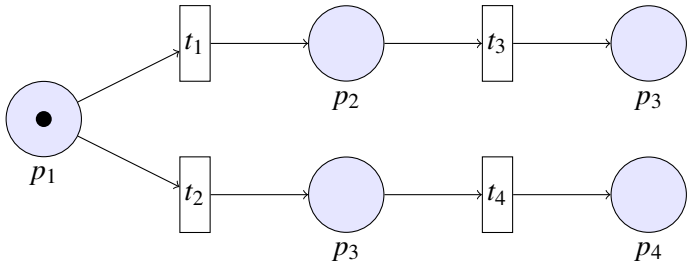
Fonte: Autor, (2025)

2.3.3 CONFLITO

O conflito ocorre quando duas ou mais transições dependem de um mesmo recurso limitado, de modo que apenas uma delas pode ser disparada. Do ponto de vista formal, se duas transições t_1 e t_2 compartilham um mesmo lugar de entrada p contendo apenas uma ficha, o disparo de uma delas consome essa ficha e, portanto, impossibilita o disparo da outra.

Na Figura 7 é ilustrado esse mecanismo de competição. Esse conceito é essencial na análise de sistemas em que há disputa por recursos escassos, como acesso a barramentos de dados, unidades de processamento ou dispositivos de entrada/saída.

Figura 7 – Expressão de processo em conflito numa RP



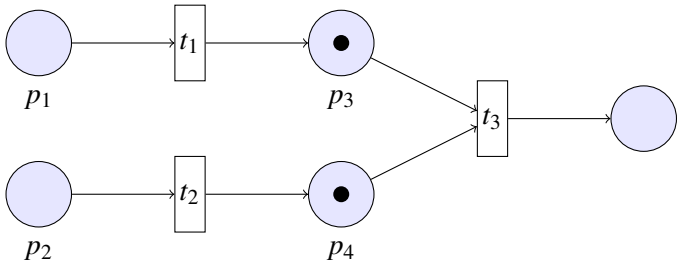
Fonte: Autor, (2025)

2.3.4 SINCRONISMO

O sincronismo ocorre quando duas ou mais condições precisam ser satisfeitas simultaneamente para que uma transição seja disparada. Em termos formais, uma transição t_3 exige fichas em todos os seus lugares de entrada (p_3 e p_4, por exemplo) para ser habilitada. O disparo só acontece quando todas as pré-condições estão atendidas ao mesmo tempo.

Na Figura 8 é apresentado esse tipo de coordenação. Essa propriedade é fundamental em sistemas que requerem junção de fluxos, como sincronização de sinais em sistemas digitais, barreiras de sincronização em computação paralela e processos de manufatura em que diferentes linhas convergem para uma operação conjunta.

Figura 8 – Expressão de processo em sincronismo numa RP



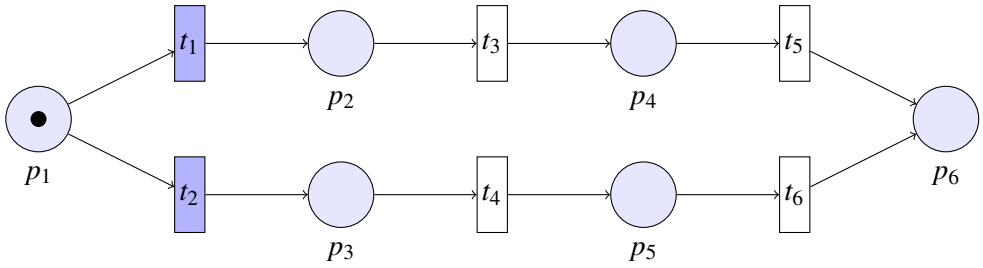
Fonte: Autor, (2025)

2.3.5 CAMINHOS ALTERNATIVOS

Os caminhos alternativos representam situações em que o sistema pode evoluir por diferentes fluxos, dependendo da transição escolhida em um ponto de decisão.vFormalmente, a partir de um mesmo lugar de entrada, duas ou mais transições competem para serem disparadas, mas, diferentemente do conflito estrito, cada caminho leva a uma continuidade distinta do processo. Assim, a escolha de t_1 ou t_2 define qual será a sequência subsequente de eventos.

Na Figura 9 é ilustrado esse comportamento, que é relevante na modelagem de fluxos de trabalho com ramificações condicionais, sistemas de decisão e protocolos que possuem rotas alternativas de execução.

Figura 9 – Expressão de processo em Caminhos Alternativos numa RP



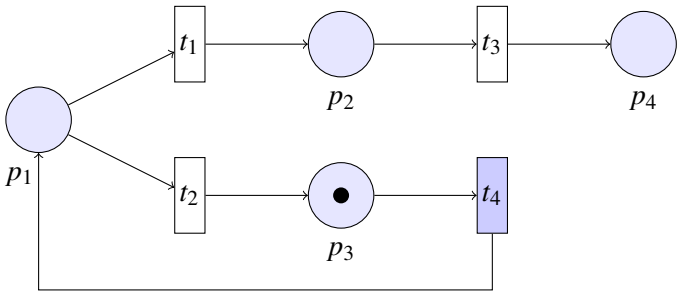
Fonte: Autor, (2025)

2.3.6 REPETIÇÃO

A repetição corresponde à presença de ciclos em uma rede de Petri, permitindo que a execução retorne a um estado anterior e o processo seja reiniciado ou reiterado. Formalmente, ocorre quando uma transição, como t_4, conduz a marcação de volta a um lugar já visitado, mantendo a possibilidade de reiniciar a sequência de disparos. Esse mecanismo possibilita a modelagem de laços de controle.

Na Figura 10 exemplifica-se esse comportamento cíclico. Esse conceito é de grande importância para representar sistemas iterativos, como ciclos de produção, algoritmos com estruturas de repetição ou processos de controle em tempo real que se renovam continuamente.

Figura 10 – Expressão de processo de Repetição numa RP



Fonte: Autor, (2025)

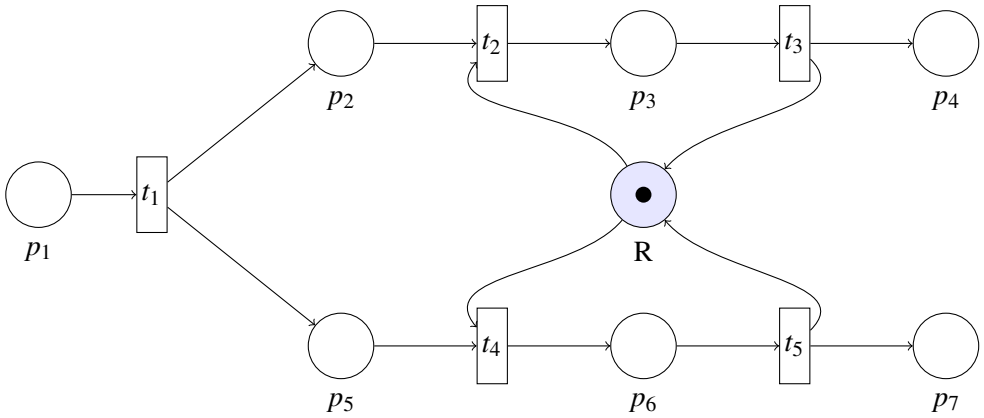
2.3.7 ALOCAÇÃO DE RECURSOS

Um recurso compartilhado, por definição, não pode ser utilizado simultaneamente por múltiplos processos sem um mecanismo adequado de controle de concorrência. Dessa forma, torna-se necessário que, após o uso, o recurso seja devidamente liberado antes de ser novamente alocado.

Na Figura 11, observa-se que as transições t_2 e t_4 representam a alocação do recurso R pelo processo em execução. Em outras palavras, tais transições indicam o momento em que o recurso passa do estado de disponível para o estado de ocupado, tornando-se inacessível a outros processos. A liberação do recurso, por sua vez, ocorre por meio das transições t_3 e t_5, que restauram R à condição de disponível, permitindo que volte a ser utilizado em ciclos subsequentes do sistema.

Esse mecanismo de alocação e liberação garante que não haja conflito no uso do recurso e que a lógica do modelo represente fielmente a disciplina de compartilhamento necessária em sistemas concorrentes.

Figura 11 – Expressão de Alocação de Recursos numa RP



Fonte: Autor, (2025)

2.4 PROPRIEDADES DAS REDES DE PETRI

Conforme descrito anteriormente, as RP são modelos matemáticos poderosos para a descrição e análise de sistemas distribuídos e concorrentes, e suas propriedades fundamentais são essenciais para a compreensão do comportamento dinâmico desses sistemas. A seguir, exploramos as propriedades mais importantes das RPs, ilustrando com exemplos práticos e abordagens analíticas.

2.4.1 ALCANCE

O **alcance**, ou a alcançabilidade, de uma Rede de Petri refere-se ao conjunto de todas as marcações atingíveis a partir da marcação inicial M_0. Em outras palavras, é a coleção de todos os estados do sistema que podem ser alcançados por uma sequência de disparos de transições. Formalmente, diz-se que uma marcação M_n é alcançável a partir de M_0 se existe uma sequência de disparos s, conforme Equação 6:

$$\exists\, s \mid M_0 \xrightarrow{s} M_n \tag{6}$$

O alcance pode ser representado visualmente por meio de um *grafo de marcações*, em que cada nó do grafo representa uma marcação e cada aresta entre os nós indica a transição disparada para alcançar a próxima marcação.

2.4.2 LIMITAÇÃO

A **limitação** em uma Rede de Petri refere-se à capacidade de restringir o número de fichas em certos lugares ao longo de todas as marcações alcançáveis. Um lugar $p \in P$ é dito *limitado* se existe um número natural $k \in \mathbb{N}$ tal que, para toda marcação M alcançável a partir de M_0, tem-se, conforme Equação 7:

$$M(p) \leq k \tag{7}$$

Se todos os lugares de uma rede forem limitados por algum k, a rede é dita *limitada*. Em particular, se $k = 1$, o lugar é chamado de *seguro*, e a rede é dita *segura*. A limitação é uma propriedade importante para garantir que não haja crescimento descontrolado de fichas no sistema.

2.4.3 VIVACIDADE

A **vivacidade** é uma propriedade fundamental das Redes de Petri que expressa o potencial de atividade contínua das transições. Uma transição $t \in T$ é considerada *viva* se, independentemente do caminho de execução escolhido, sempre existe a possibilidade de que t venha a ser habilitada e disparada em algum ponto futuro da evolução da rede.

A vivacidade de uma transição pode ser classificada em cinco níveis, de acordo com o grau de possibilidade de disparo:

- **Nível 0** (Morta): A transição nunca mais poderá ser disparada a partir da marcação inicial. Isto é, não existe nenhuma sequência de disparos que a habilite.
- **Nível 1** (Potencialmente disparável): Existe ao menos uma sequência de disparos a partir da marcação inicial que leva a uma marcação onde a transição está habilitada. No entanto, essa possibilidade pode nunca se realizar dependendo do caminho tomado.
- **Nível 2** (Repetidamente disparável): Para toda sequência de disparos suficientemente longa, a transição ocorrerá ao menos uma vez. Isso significa que, no longo prazo, o disparo da transição é garantido, embora possa haver longos períodos de inatividade.
- **Nível 3** (Infinitamente disparável): Existe uma sequência infinita de disparos em que a transição ocorre infinitamente muitas vezes.
- **Nível 4** (Sempre habilitável): A transição está habilitada em toda marcação alcançável. Isto é, pode ser disparada a qualquer momento da execução do sistema.

Diz-se que uma rede é *viva* se todas as suas transições são vivas no sentido do **nível 1** ou superior. Redes vivas garantem a ausência de situações de paralisação permanente (como *deadlocks*), sendo especialmente desejáveis em sistemas que devem operar continuamente, como sistemas de controle e automação.

2.4.4 BLOQUEIO

O **bloqueio** ocorre quando a Rede de Petri atinge uma marcação da qual nenhuma transição está habilitada. Essa marcação é chamada de *marcação morta* ou *marcação de bloqueio*. Formalmente, uma marcação M_b é bloqueada se:

$$\forall t \in T, \quad t \text{ não está habilitada em } M_b$$

O bloqueio representa uma condição indesejável em muitos sistemas, pois indica que o sistema chegou a um ponto onde não é possível continuar a execução. A análise do bloqueio é essencial em projetos que demandam alta disponibilidade e operação contínua.

2.4.5 MARCAÇÃO MORTA E AUSÊNCIA DE BLOQUEIO

Em Redes de Petri, uma **marcação morta** é uma configuração do sistema na qual nenhuma transição está habilitada, impossibilitando qualquer evolução futura do comportamento da rede. Formalmente, uma marcação M é dita *morta* se:

$$\forall t \in T, \quad t \text{ não está habilitada em } M$$

Nessa condição, o sistema atinge um estado de inatividade permanente, também conhecido como *deadlock*. Esse tipo de marcação é geralmente indesejável, especialmente em aplicações que requerem continuidade operacional, como sistemas automatizados ou de controle em tempo real.

A partir dessa definição, pode-se caracterizar uma importante propriedade estrutural:

Uma Rede de Petri (N, M_0) é dita **sem bloqueio** (ou *livre de deadlock*) se nenhuma marcação morta for alcançável a partir da marcação inicial M_0. Isto é, para toda marcação M tal que:

$$M_0 \xrightarrow{s} M$$

existe pelo menos uma transição $t \in T$ que esteja habilitada em M. Em outras palavras, sempre haverá alguma possibilidade de continuidade na execução do sistema, evitando paralisações definitivas.

A propriedade de ausência de bloqueio está intimamente relacionada à confiabilidade e disponibilidade do sistema modelado. Redes sem bloqueio asseguram que, independentemente da sequência de disparos executada, o sistema nunca ficará preso em um estado sem saída. Portanto, a verificação da existência de marcações mortas no conjunto de marcações alcançáveis é essencial na validação de modelos de sistemas concorrentes.

2.4.6 REVERSIBILIDADE

A **reversibilidade** é uma propriedade importante em Redes de Petri que descreve a capacidade do sistema de retornar à sua marcação inicial M_0 a partir de qualquer marcação alcançável. Em termos práticos, uma rede reversível é capaz de restaurar seu estado original, independentemente da sequência de transições disparadas até um dado momento.

Formalmente, uma Rede de Petri (N, M_0) é dita *reversível* se, para toda marcação $M \in R(N, M_0)$, isto é, para toda marcação alcançável a partir de M_0, existe uma sequência de disparos $s \in T^*$ tal que:

$$M \xrightarrow{s} M_0$$

Em outras palavras, a marcação inicial M_0 permanece sempre acessível ao longo da evolução do sistema.

A reversibilidade é particularmente relevante em sistemas que operam de forma cíclica ou que necessitam de reinicialização segura, como linhas de montagem, controladores de processos ou protocolos de comunicação. Ela também oferece uma garantia de recuperação: mesmo após diversas operações, o sistema pode retornar a um estado conhecido e estável.

É importante destacar que a reversibilidade não implica, por si só, a vivacidade da rede. Uma rede pode ser reversível e ainda assim conter transições mortas. Da mesma forma, uma rede viva pode não ser reversível, caso sua evolução a leve a estados dos quais não se pode retornar à marcação inicial. Por isso, a análise de reversibilidade deve ser feita em conjunto com outras propriedades, como alcance, vivacidade e ausência de bloqueio, a fim de garantir um comportamento robusto e confiável do sistema modelado.

2.4.7 ESTADO DE PASSAGEM

Em Redes de Petri, um **estado de passagem** — também chamado de *home state* — é uma marcação especial que pode ser alcançada a partir de qualquer outra marcação atingível da rede. Trata-se de um conceito que expressa uma forma forte de conectividade e estabilidade dentro do sistema modelado.

Formalmente, uma marcação $M' \in R(N, M_0)$ é dita um *estado de passagem* se, para toda marcação $M \in R(N, M_0)$, existe uma sequência de disparos $s \in T^*$ tal que:

$$M \xrightarrow{s} M'$$

Ou seja, M' é um ponto de convergência no comportamento da rede: independentemente do estado atual do sistema, sempre é possível retornar a essa marcação.

A presença de um estado de passagem confere à rede uma notável propriedade de autorrecuperação e previsibilidade, sendo extremamente valiosa em aplicações que exigem reinicialização periódica, sincronização global ou coordenação centralizada. Isso inclui, por exemplo, sistemas de manufatura cíclicos, protocolos de comunicação, processos distribuídos e controladores industriais.

Diferentemente da *reversibilidade*, que exige a possibilidade de retornar à marcação inicial M_0, o estado de passagem permite que outro estado — possivelmente distinto de M_0 — atue como referência comum para o sistema. Essa distinção amplia a flexibilidade no projeto e na análise de redes que precisam garantir estabilidade mesmo após diversas evoluções.

Do ponto de vista do grafo de marcações, a existência de um estado de passagem implica que este nó específico é acessível a partir de todos os outros nós do grafo. Essa característica favorece propriedades como ausência de bloqueio e controle centralizado do comportamento da rede, tornando-se uma poderosa ferramenta para garantir robustez e consistência operacional em sistemas concorrentes.

No entanto, a verificação da existência de um estado de passagem pode ser computacionalmente complexa, especialmente em redes com muitos ciclos ou estados. Ainda assim, sua identificação é de grande utilidade na validação de sistemas críticos que exigem comportamentos confiáveis e recuperáveis.

2.4.8 COBERTURA

A propriedade de **cobertura** em Redes de Petri está relacionada à dominância entre marcações. Intuitivamente, uma marcação M é considerada *coberta* se, em algum momento da evolução do sistema, o número de fichas em cada lugar da rede pode ser igual ou superior ao de M. Essa propriedade permite avaliar a possibilidade de atingir, ou mesmo superar, certos níveis de recursos ou condições dentro da rede.

Formalmente, uma marcação $M \in \mathbb{N}^{|P|}$ é dita *coberta* se existe uma marcação $M' \in R(N, M_0)$, conforme Equação 8:

$$\forall p \in P, \quad M'(p) \geq M(p) \tag{8}$$

Isto é, existe uma marcação alcançável M' na qual o número de fichas em cada lugar $p \in P$ é maior ou igual ao número de fichas da marcação M. Nesse contexto, diz-se que M' *cobre* M.

A análise de cobertura é útil para determinar se uma determinada situação desejada (ou indesejada) pode, em algum momento, ser alcançada ou ultrapassada pela dinâmica da rede. Por exemplo, é possível

verificar se haverá, eventualmente, fichas suficientes em certos lugares para ativar transições críticas, ou se determinados limites de segurança podem ser violados.

Do ponto de vista prático, a cobertura permite abstrair o comportamento exato da rede em favor de uma análise mais conservadora e segura. Isso é especialmente relevante em sistemas onde o crescimento descontrolado de fichas pode representar falhas, como estouros de buffer, saturações de filas ou violações de capacidade.

Além disso, a propriedade de cobertura está intimamente ligada ao conceito de *quase-alcance* (ou *over-approximation*), sendo amplamente utilizada em algoritmos de verificação formal, como na construção de árvores de cobertura (*coverability trees*) para redes com comportamento potencialmente infinito.

É importante destacar que, ao contrário do alcance, a cobertura não exige que a marcação M seja exatamente atingida, mas apenas que exista uma marcação superior ou igual a ela. Essa característica a torna uma ferramenta poderosa para análise de propriedades de segurança, limitação e viabilidade de recursos.

2.4.9 PERSISTÊNCIA

A **persistência** é uma propriedade comportamental de Redes de Petri que expressa uma forma de não interferência entre transições habilitadas simultaneamente. Ela garante que, uma vez habilitada, uma transição não será desabilitada pelo disparo de outra transição também habilitada.

Formalmente, uma Rede de Petri (N, M_0) é dita *persistente* se, para toda marcação $M \in R(N, M_0)$ e para quaisquer transições $t_1, t_2 \in T$, se ambas estão habilitadas em M, então o disparo de t_1 não impede o disparo subsequente de t_2. Ou seja, conforme Equação 9:

$$\text{Se } M \vdash t_1 \text{ e } M \vdash t_2, \text{ então } M \xrightarrow{t_1} M' \Rightarrow M' \vdash t_2 \tag{9}$$

Essa condição implica que não há competição destrutiva entre transições habilitadas simultaneamente: o disparo de uma não invalida a habilitação da outra.

A persistência é uma propriedade desejável em sistemas concorrentes que exigem previsibilidade e independência entre ações paralelas, como controladores reativos, sistemas de tempo real e protocolos de sincronização. Ela assegura que uma transição, uma vez habilitada, poderá sempre ser disparada posteriormente, desde que nenhuma outra ação externa interfira.

No entanto, a persistência é uma propriedade restritiva. Muitas Redes de Petri modelam cenários de escolha ou competição — por exemplo, quando duas transições representam alternativas mutuamente exclusivas. Nesses casos, a rede não será persistente, mas ainda pode ser correta para o comportamento desejado.

A verificação da persistência pode ser feita analisando o grafo de marcações: se, em alguma marcação, duas transições estão habilitadas e o disparo de uma delas leva a uma marcação onde a outra não está mais habilitada, a rede não é persistente. Essa análise ajuda a identificar possíveis condições de corrida ou perda de oportunidades dentro do sistema.

Em resumo, a persistência favorece a concorrência não conflitiva entre transições e é uma propriedade valiosa na modelagem de sistemas onde a confiabilidade e a ausência de interferência são requisitos críticos.

2.4.10 MATRIZES DE INCIDÊNCIA

As **matrizes de incidência** são ferramentas matemáticas fundamentais para a análise e modelagem de Redes de Petri. Elas representam, de forma matricial, a estrutura da rede e o impacto do disparo das transições sobre as marcações dos lugares.

Dada uma Rede de Petri $N = (P, T, Pre, Post)$, onde $P = \{p_1, p_2, \ldots, p_m\}$ é o conjunto de lugares e $T = \{t_1, t_2, \ldots, t_n\}$ o conjunto de transições, definem-se as matrizes, conforme Equação 10:

$$Pre : \mathbb{N}^{m \times n}, \quad Post : \mathbb{N}^{m \times n} \tag{10}$$

em que $Pre(p_i, t_j)$ indica o número de fichas que a transição t_j consome do lugar p_i para ser habilitada, e $Post(p_i, t_j)$ indica o número de fichas que t_j adiciona ao lugar p_i após seu disparo.

A partir dessas duas matrizes, define-se a *matriz de incidência* $C \in \mathbb{Z}^{m \times n}$, conforme Equação 11:

$$C = Post - Pre \tag{11}$$

Cada elemento $C(p_i, t_j)$ representa a variação líquida do número de fichas no lugar p_i causada pelo disparo da transição t_j. Valores positivos indicam incremento, valores negativos indicam decremento, e zero indica que o disparo da transição não altera a quantidade de fichas naquele lugar.

A dinâmica da rede pode ser formalizada, conforme Equação 12:

$$M' = M + C \cdot \sigma \tag{12}$$

em que $M, M' \in \mathbb{N}^m$ são as marcações antes e depois do disparo, respectivamente, e $\sigma \in \mathbb{N}^n$ é o vetor de disparos, indicando quantas vezes cada transição foi disparada.

As matrizes de incidência permitem realizar análises algébricas importantes, tais como:

- Determinação do *alcance* e propriedades estruturais da rede;
- Verificação de conservação e invariantes de lugar;
- Análise de reversibilidade e vivacidade por meio do estudo do espaço de soluções associadas.

Por meio dessas matrizes, é possível aplicar métodos de álgebra linear para estudar o comportamento dinâmico da Rede de Petri, facilitando a implementação de algoritmos de verificação formal e simulação computacional.

2.4.11 DISPARO DE UMA TRANSIÇÃO

O **disparo de uma transição** é o mecanismo fundamental que rege a evolução dinâmica de uma Rede de Petri. Quando uma transição $t \in T$ está habilitada em uma marcação atual M_{n-1}, o disparo dessa transição produz uma nova marcação M_n, refletindo a mudança no número de fichas em cada lugar da rede.

Formalmente, seja $M_{n-1} \in \mathbb{N}^{|P|}$ a marcação no instante anterior, e $C \in \mathbb{Z}^{|P|\times|T|}$ a matriz de incidência da rede. Definimos o vetor de disparo $T' \in \{0,1\}^{|T|}$ como um vetor unitário, com entrada 1 na posição correspondente à transição t que dispara, e zeros nas demais posições, conforme Equação 13:

$$T' = [0,0,\ldots,1,\ldots,0]^T \tag{13}$$

Dessa forma, a nova marcação M_n é calculada, conforme Equação 14:

$$M_n = M_{n-1} + C \times T' \tag{14}$$

Isso significa que o disparo da transição t altera o número de fichas nos lugares da rede conforme o vetor coluna correspondente na matriz de incidência C, somando ou subtraindo fichas de acordo com a estrutura da rede.

Para que o disparo seja válido, a transição t deve estar habilitada na marcação M_{n-1}, ou seja, todos os lugares que alimentam t devem possuir fichas suficientes para satisfazer os requisitos do vetor *Pre* associado.

O conceito de disparo é central para a simulação, análise e verificação do comportamento dinâmico das Redes de Petri, permitindo modelar sistemas concorrentes, distribuídos e com eventos discretos de maneira precisa e formal.

2.4.12 VETOR CARACTERÍSTICO

O **vetor característico** é uma ferramenta matemática que representa de forma compacta e precisa o comportamento de uma sequência de disparos em uma Rede de Petri.

Seja s uma sequência de disparos composta por transições da rede. Definimos o vetor $u \in \mathbb{N}^{|T|}$, onde cada componente $u(t_i)$ indica o número de vezes que a transição $t_i \in T$ ocorre na sequência s.

Formalmente, o vetor u é denominado *vetor característico* da sequência s, e sua dimensão é igual ao número total de transições da Rede de Petri, conforme Equação 15:

$$u = (u(t_1), u(t_2), \ldots, u(t_n))^T, \quad \text{em que } n = |T| \tag{15}$$

Este vetor resume a frequência de disparos das transições ao longo da execução, independentemente da ordem em que ocorrem.

O vetor característico é fundamental para a análise algébrica da Rede de Petri, pois permite expressar a evolução das marcações, conforme Equação 16:

$$M = M_0 + C \cdot u \tag{16}$$

em que M_0 é a marcação inicial, C é a matriz de incidência e u é o vetor característico da sequência de disparos s. Essa equação conecta diretamente a sequência de eventos com a mudança no estado do sistema, facilitando estudos de alcance, reversibilidade e propriedades estruturais.

Em síntese, o vetor característico fornece uma visão global e quantitativa do comportamento da rede, sendo uma peça-chave na modelagem e verificação formal de sistemas representados por Redes de Petri.

2.4.13 MARCAÇÃO ALCANÇADA

Considere uma Rede de Petri (N, M_0) com matriz de incidência C e uma sequência de disparo $s = t_a t_b \dots t_q$ composta por transições da rede. Suponha que uma marcação M_q seja alcançável a partir da marcação inicial M_0 por meio dessa sequência s.

A evolução das marcações pode ser descrita passo a passo, considerando o efeito do disparo de cada transição sobre a marcação anterior, conforme Equação 17:

$$M_1 = M_0 + C \times t_a, \quad M_2 = M_1 + C \times t_b = M_0 + C \times t_a + C \times t_b, \quad \dots \tag{17}$$

em que $C \times t_i$ representa o vetor coluna da matriz de incidência C correspondente à transição t_i.

Estendendo essa ideia, é possível expressar diretamente a marcação M_q após a sequência s em termos do vetor característico u da sequência, onde cada componente $u(t_i)$ indica o número de disparos da transição t_i na sequência s, conforme Equação 18:

$$M_q = M_0 + C \times u \tag{18}$$

Exemplo de marcação alcançada

Seja a sequência: $s = t_1 t_3 t_4 t_1 t_1$, com conjunto de transições $\{t_1, t_2, t_3, t_4\}$. Seja o vetor característico u é dado por:

$$u = [3,\ 0,\ 1,\ 1]^T,$$

em que t_1 ocorre 3 vezes, t_2 não ocorre, t_3 e t_4 ocorrem uma vez cada.

Dada a matriz de incidência 19 C:

$$C = \begin{bmatrix} -1 & 1 & -1 & 0 \\ 3 & 3 & 3 & 3 \\ -1 & 0 & 1 & -1 \\ 1 & -1 & 0 & 0 \end{bmatrix}. \tag{19}$$

a marcação M_q é calculada como:

$$M_q = M_0 + C \times u.$$

Este cálculo fornece a marcação resultante após o disparo da sequência s, indicando o número de fichas em cada lugar da Rede de Petri.

Alcance: resultado numérico

Considerando a Matriz 19, anteriormente apresentada, e os termos M_0, s e u a seguir:

$$C = \begin{pmatrix} -1 & 1 & -1 & 0 \\ 3 & 3 & 3 & 3 \\ -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix}, \qquad M_0 = \begin{pmatrix} 5 \\ 0 \\ 2 \\ 1 \end{pmatrix}, \qquad s = t_1 t_3 t_4 t_1 t_1, \qquad \mathbf{u} = \begin{pmatrix} 3 \\ 0 \\ 1 \\ 1 \end{pmatrix}.$$

Note que as colunas de C (efeito unitário de cada transição) são:

$$C(s,t_1) = \begin{pmatrix} -1 \\ 3 \\ -1 \\ 1 \end{pmatrix}, \quad C(s,t_2) = \begin{pmatrix} 1 \\ 3 \\ 0 \\ -1 \end{pmatrix}, \quad C(s,t_3) = \begin{pmatrix} -1 \\ 3 \\ -1 \\ 0 \end{pmatrix}, \quad C(s,t_4) = \begin{pmatrix} 0 \\ 3 \\ 0 \\ 0 \end{pmatrix}.$$

Evolução passo a passo (aplicando as colunas na ordem da sequência s):

$$M_0 = \begin{pmatrix} 5 \\ 0 \\ 2 \\ 1 \end{pmatrix},$$

$$M_1 = M_0 + C(:,t_1) = \begin{pmatrix} 5 \\ 0 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 3 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 1 \\ 2 \end{pmatrix},$$

$$M_2 = M_1 + C(:,t_3) = \begin{pmatrix} 4 \\ 3 \\ 1 \\ 2 \end{pmatrix} + \begin{pmatrix} -1 \\ 3 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 0 \\ 2 \end{pmatrix},$$

$$M_3 = M_2 + C(:,t_4) = \begin{pmatrix} 3 \\ 6 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} 0 \\ 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 0 \\ 2 \end{pmatrix},$$

$$M_4 = M_3 + C(:,t_1) = \begin{pmatrix} 3 \\ 9 \\ 0 \\ 2 \end{pmatrix} + \begin{pmatrix} -1 \\ 3 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 12 \\ -1 \\ 3 \end{pmatrix},$$

$$M_5 = M_4 + C(:,t_1) = \begin{pmatrix} 2 \\ 12 \\ -1 \\ 3 \end{pmatrix} + \begin{pmatrix} -1 \\ 3 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 15 \\ -2 \\ 4 \end{pmatrix}.$$

Portanto,

$$M_q = \begin{pmatrix} 1 \\ 15 \\ -2 \\ 4 \end{pmatrix}$$

Cálculo direto:

$$C\mathbf{u} = \begin{bmatrix} -4 \\ 15 \\ -4 \\ 3 \end{bmatrix}, \qquad M_q = M_0 + C\mathbf{u} = \begin{pmatrix} 5 \\ 0 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 & 1 & -1 & 0 \\ 3 & 3 & 3 & 3 \\ -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 3 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 15 \\ -2 \\ 4 \end{pmatrix}.$$

Note que M_q e alguns M_i apresentam componentes negativas (por exemplo, a componente 3 torna-se -2). Isso indica que a sequência s *não* é habilitada a partir da marcação inicial escolhida M_0 (na prática um disparo que geraria fichas negativas é inválido). O cálculo aqui mostra apenas o **efeito aritmético** $M_q = M_0 + C\mathbf{u}$. Para verificar se s é realmente executável (série de disparos permissíveis) deve-se checar, a cada passo, se a transição disparada estava habilitada (ou seja, havia fichas suficientes nos lugares de entrada) — condição que falha neste exemplo numérico.

Esse formalismo mostra como o uso da matriz de incidência combinada com o vetor característico da sequência de disparo permite determinar de maneira algébrica a marcação alcançada, facilitando análises como alcance, vivacidade e reversibilidade.

2.4.14 RESUMO DAS PROPRIEDADES DAS REDES DE PETRI

Na Tabela 2 é sintetizada cada propriedade, destacando sua característica essencial e exemplificando conceitos básicos para facilitar a compreensão. Note que as propriedades fundamentais das Redes de Petri estão apresentadas, abrangendo aspectos estruturais, comportamentais e dinâmicos, tais como *alcance*, *vivacidade*, *reversibilidade*, *persistência*, dentre outras. Tais propriedades são imprescindíveis para a análise formal e para o desenvolvimento rigoroso de modelos em sistemas concorrentes e distribuídos.

Tabela 2 – Resumo das propriedades das Redes de Petri

#	Propriedade	Característica Principal	Exemplo Básico
1	**Alcance**	Conjunto de todas as marcações que podem ser alcançadas a partir da marcação inicial M_0. Representa os possíveis estados do sistema.	Marcações obtidas por sequências válidas de disparos a partir de M_0.
2	**Vivacidade**	Garante que cada transição poderá, em algum momento, ser disparada, prevenindo bloqueios permanentes. Classificada em níveis de 0 a 4.	Transição que pode ficar temporariamente desabilitada, mas sempre poderá ser disparada futuramente.
3	**Marcação Morta**	Estado onde nenhuma transição está habilitada, indicando deadlock ou paralisação.	Sistema travado sem transições disparáveis.
4	**Ausência de Bloqueio**	Nenhuma marcação morta é alcançável; o sistema nunca trava completamente.	Rede em que, de qualquer estado, ao menos uma transição está habilitada.
5	**Reversibilidade**	Capacidade de retornar à marcação inicial M_0 a partir de qualquer marcação alcançável.	Sistema que pode reiniciar seu ciclo, voltando ao estado inicial após qualquer sequência.
6	**Estado de Passagem**	Marca uma marcação M' acessível a partir de qualquer outra marcação; ponto de convergência do sistema.	Estado ao qual o sistema pode sempre retornar, independentemente do trajeto.
7	**Cobertura**	Marcações M são cobertas se existe outra M' com número de fichas maior ou igual em todos os lugares.	Verificar se um estado com recursos iguais ou superiores pode ser atingido.
8	**Persistência**	O disparo de uma transição habilitada não desabilita outras transições habilitadas simultaneamente.	Duas transições que podem disparar independentemente sem bloquear uma à outra.
9	**Matrizes de Incidência**	Representam o impacto das transições sobre os lugares, facilitando análise algébrica da rede.	Matriz $C = Post - Pre$, usada para calcular alterações nas marcações após disparos.
10	**Disparo de Transição**	Atualiza a marcação somando o vetor coluna da matriz de incidência relativo à transição disparada.	Se t está habilitada, $M_n = M_{n-1} + C \times T'$, com T' vetor unitário para t.
11	**Vetor Característico**	Vetor que indica o número de disparos de cada transição em uma sequência, usado para calcular marcações resultantes.	Para $s = t_1t_3t_4t_1t_1$, vetor $u = [3,0,1,1]^T$.
12	**Marcação Alcançada**	Marca a marcação resultante após uma sequência de disparos pelo produto da matriz de incidência pelo vetor característico.	$M_q = M_0 + C \times u$, onde u é o vetor característico da sequência disparada.

Fonte: Autor, (2025)

Dessa forma, consolida-se a percepção das propriedade das RP visando assimilação integrada

desses conceitos é indispensável para o desenvolvimento e a análise de Redes de Petri que sejam robustas, eficazes e confiáveis, encerrando assim este capítulo com uma fundamentação teórica consistente para estudos e aplicações futuras.

2.5 CONCLUSÃO

As Redes de Petri se apresentam como uma ferramenta poderosa para a modelagem de sistemas dinâmicos discretos, permitindo a análise de concorrência, sincronização e recursos compartilhados. O formalismo matemático empregado facilita a verificação de propriedades cruciais para a confiabilidade de sistemas computacionais e industriais. Ao combinar o uso de invariantes de lugar e de transição, por exemplo, é possível realizar uma verificação formal de propriedades como segurança e vivacidade. Este capítulo estabeleceu os fundamentos teóricos das Redes de Petri, explorando diversas definições formais que variam conforme o nível de detalhamento, e apresentou suas principais aplicações e propriedades, servindo como base para estudos mais avançados, incluindo modelos estendidos, como Redes de Petri Coloridas e Temporizadas.

3 Redes de Petri: modelos iniciais

3.1 CASO PRÁTICO INDUSTRIAL: REPRESENTAÇÃO EM RP

Considere uma fábrica de montagem em que o processo produtivo é dividido em cinco etapas, cada uma representada por um *lugar* na rede. As etapas são:

- P_1: Recebimento de matéria-prima;
- P_2: Processamento inicial;
- P_3: Inspeção de qualidade;
- P_4: Montagem final;
- P_5: Expedição do produto acabado.

Os *eventos* (ou transições) que conectam essas etapas são:

- T_1: Inicia o processamento da matéria-prima, direcionando-a tanto para o processamento inicial quanto para a inspeção preliminar (um fluxo concorrente);
- T_2: Conclui o processamento inicial e encaminha o material para uma etapa de inspeção ou preparação adicional;
- T_3: Realiza uma inspeção complementar, garantindo a qualidade antes da montagem;
- T_4: Coordena a montagem final do produto;
- T_5: Finaliza o processo, liberando o produto para expedição e, eventualmente, retornando uma parte dos recursos para reinício do ciclo.

A rede em questão modela, portanto, o fluxo de fichas (representando, por exemplo, lotes ou unidades de produto) através dessas etapas, permitindo identificar gargalos, verificar a eficiência do fluxo e garantir que a qualidade seja mantida ao longo do processo produtivo.

3.1.1 ESTRUTURA DA REDE DE PETRI

A rede que descreve o processo acima é composta por:

- **Lugares:** $P = \{P_1, P_2, P_3, P_4, P_5\}$;
- **Transições:** $T = \{T_1, T_2, T_3, T_4, T_5\}$;
- **Arcos:** A conexão entre os elementos é dada por:

$$\begin{aligned} F \quad = \quad & \{(P_1,T_1),\ (T_1,P_2),\ (T_1,P_3),\\ & (P_2,T_2),\ (T_2,P_4),\ (P_3,T_3),\ (T_3,P_4),\\ & (P_4,T_4),\ (T_4,P_5),\ (P_5,T_5),\ (T_5,P_1)\}. \end{aligned}$$

Note que a transição T_1 distribui o fluxo de matéria-prima simultaneamente para duas rotas: uma para processamento inicial e outra para uma inspeção preliminar.

- **Marcação Inicial:** A fábrica inicia com a chegada de um lote de matéria-prima:

$$M_0: \quad M_0(P_1) = 1, \quad M_0(P_2) = M_0(P_3) = M_0(P_4) = M_0(P_5) = 0.$$

- **Função de Peso:** Assume-se que cada arco possui peso unitário:

$$W : F \to \mathbb{N}^+, \quad \text{com } W(e) = 1 \text{ para todo } e \in F.$$

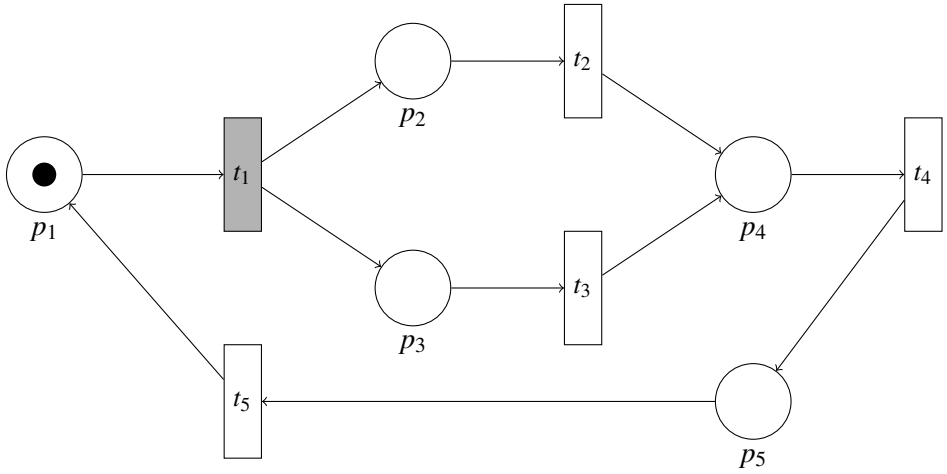
- **Função de Rótulo:** Para complementar a modelagem, associamos rótulos às transições:

$$\Lambda : T \to \{\text{descrições}\}, \quad \text{por exemplo, } \Lambda(T_i) = \text{“Etapa } T_i\text{”}, \; i = 1, \ldots, 5.$$

REPRESENTAÇÃO GRÁFICA DA RP PROPOSTA

Na Figura 12 é apresentada graficamente a RP em questão. Nesta figura, os lugares são representados por círculos e as transições por barras (retângulos):

Figura 12 – Diagrama gráfico da RP representando o fluxo de produção industrial



Fonte: Autor,(2025)

Note, na Figura 12, que os nós (lugares p_i e transições t_i) são conectados por arcos, seguindo fielmente o processo industrial descrito no início desta seção. Além disso, observe a representação da marcação inicial da RP, contendo uma ficha única no lugar p_1(círculo denso de cor preta), a ausência de fichas nos demais lugares p_i. Perceba também que todos os arcos têm peso $W = 1$. A transição t_1 destacada por uma moldura cinza sinaliza sua condição de habilitação para disparo, portanto, de ocorrência do evento.

3.2 MATRIZES DE PRÉ, PÓS E DE INCIDÊNCIA DE UMA REDE DE PETRI

As redes de Petri são uma poderosa ferramenta para modelar sistemas dinâmicos e discretos, como processos de produção, controle de tráfego, e até sistemas computacionais. Elas são compostas por lugares, transições e arcos, em que os lugares representam estados ou recursos, as transições representam eventos ou ações, e os arcos indicam como os lugares e transições se interconectam.

Uma rede de Petri pode ser analisada por meio de suas matrizes de incidência, que fornecem informações cruciais sobre as relações entre lugares e transições. As três matrizes mais importantes em uma rede de Petri são:

- **Matriz de Pré-Incidência** (C^-), que descreve as conexões entre os lugares e as transições para as quais esses lugares servem de entrada.
- **Matriz de Pós-Incidência** (C^+), que descreve as conexões entre as transições e os lugares que recebem fichas após a execução de uma transição.
- **Matriz de Incidência** (C), que é a diferença entre as matrizes de pós e pré-incidência, e descreve de forma completa a interação entre lugares e transições.

Essas matrizes são fundamentais para a análise de comportamento dinâmico de sistemas modelados por redes de Petri, especialmente em termos de controle e verificação de propriedades como alcance de lugares e sincronização de transições.

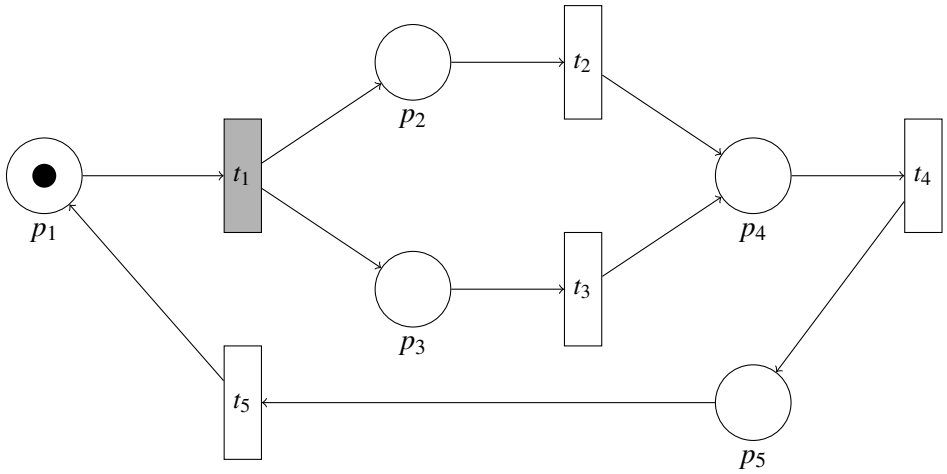
3.2.1 MODELAGEM DAS MATRIZES DE UMA RP

Consideremos a RP da Figura 12, apresentada na Secção 3.1.1, cujas características básicas são:

- p_1 é um lugar de entrada que tem uma ficha inicial.
- p_2 e p_3 são lugares que representam dois caminhos paralelos a partir de T_1.
- p_4 e p_5 são lugares de saída.
- As transições t_1, t_2, t_3, t_4, t_5 são eventos que conectam os lugares.

A referida figura é novamente posta a seguir ,visando facilitar a percepção das matrizes de incidência.

Diagrama gráfico da RP representando o fluxo de produção industrial (Figura 12 da seção 3.1.1.)



Fonte: Autor,(2025)

3.2.2 MATRIZ DE PRÉ-INCIDÊNCIA (C^-)

A matriz de pré-incidência C^- é uma matriz $p \times t$, em que p é o número de lugares e t é o número de transições. Ela descreve a relação entre os lugares e as transições na direção dos arcos que vão dos lugares para as transições. Em outras palavras, cada elemento C^-_{ij} indica o número de fichas que um lugar P_i envia para a transição T_j.

Para a RP apresentada acima, a matriz de pré-incidência C^- seria apresentado conforme (20):

$$C^- = \begin{pmatrix} & t_1 & t_2 & t_3 & t_4 & t_5 \\ p_1 & 1 & 0 & 0 & 0 & 0 \\ p_2 & 0 & 1 & 0 & 0 & 0 \\ p_3 & 0 & 0 & 1 & 0 & 0 \\ p_4 & 0 & 0 & 0 & 1 & 0 \\ p_5 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{20}$$

Cada linha representa um lugar, e cada coluna representa uma transição. Os valores 1 indicam que há um arco conectando o lugar à transição correspem quente.

3.2.3 MATRIZ DE PÓS-INCIDÊNCIA (C^+)

A matriz de pós-incidência C^+ também é uma matriz $p \times t$, mas ela descreve as conexões entre as transições e os lugares que recebem fichas após a execução de uma transição. Em outras palavras, cada elemento C^+_{ij} indica o número de fichas que a transição T_j envia para o lugar P_i.

Para a rede apresentada, a matriz de pós-incidência C^+ seria apresentado conforme (21):

$$C^+ = \begin{pmatrix} & t_1 & t_2 & t_3 & t_4 & t_5 \\ p_1 & 0 & 0 & 0 & 0 & 1 \\ p_2 & 1 & 0 & 0 & 1 & 0 \\ p_3 & 1 & 0 & 0 & 0 & 0 \\ p_4 & 0 & 1 & 1 & 0 & 0 \\ p_5 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{21}$$

Assim como na matriz de pré-incidência, os valores 1 indicam a direção de fluxo de fichas das transições para os lugares.

3.2.4 MATRIZ DE INCIDÊNCIA (C)

A matriz de incidência C é dada pela diferença entre as matrizes de pós-incidência e pré-incidência. Ela descreve a relação completa entre lugares e transições, considerando tanto os arcos de entrada (pré-incidência) quanto os de saída (pós-incidência), conforme apresentado conforme (22):

$$C = C^+ - C^- = \begin{pmatrix} & t_1 & t_2 & t_3 & t_4 & t_5 \\ p_1 & -1 & 0 & 0 & 0 & 1 \\ p_2 & 1 & -1 & 0 & 0 & 0 \\ p_3 & 1 & 0 & -1 & 0 & 0 \\ p_4 & 0 & 1 & 1 & -1 & 0 \\ p_5 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \tag{22}$$

Na matriz de incidência, o valor -1 indica um arco de entrada, o valor 1 indica um arco de saída, e o valor 0 indica que não há arco entre o lugar e a transição.

As matrizes de pré, pós e de incidência são fundamentais para a análise quantitativa de RPs. Elas permitem a descrição das interações entre os lugares e transições e são usadas em diversas aplicações, como análise de alcançabilidade, verificações de segurança e determinação de ciclos de eventos em sistemas dinâmicos. Por meio dessas matrizes, é possível modelar e compreender o comportamento de sistemas complexos de maneira eficaz e matemática.

4 RP temporizadas

Conforme já definido anteriormente, as RPs são uma poderosa ferramenta para modelagem e análise de sistemas concorrentes e distribuídos. No entanto, em muitas aplicações práticas, a dimensão temporal do sistema é crucial, como em sistemas de controle industrial, redes de comunicação e sistemas de produção. Para incorporar aspectos temporais às RP clássicas, surgiram extensões como as *Redes de Petri Temporais* e as *Redes de Petri Temporalizadas*.

4.1 REDES DE PETRI TEMPORAIS

4.1.1 DEFINIÇÃO

Uma Rede de Petri Temporal (RP-T) é uma extensão das RPs clássicas em que são atribuídos intervalos de tempo mínimo e/ou máximo para a condição de disparo das transições. Isso significa que uma transição, uma vez habilitada, deve esperar um tempo mínimo antes de disparar, mas não pode ultrapassar um tempo máximo.

Formalmente, uma RP-T é definida como um par $T : T \to \mathbb{R}^+$ em que, para cada transição $t \in T$, há um tempo associado (α, β), em que:

- α é o tempo mínimo que deve decorrer antes que a transição possa disparar.
- β é o tempo máximo permitido para que a transição dispare.

EXEMPLO DE REDE DE PETRI TEMPORAL

Considere um sistema de produção com duas etapas: **montagem** e **inspeção**. A montagem leva entre 3 e 6 segundos, e a inspeção entre 2 e 4 segundos.

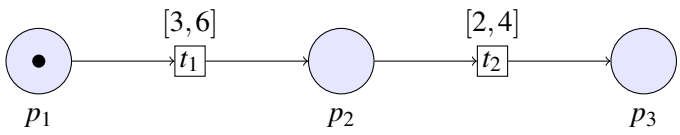
Representamos essa dinâmica por uma RP-T:

- t_1: Montagem, com intervalo de tempo $[3,6]$.
- t_2: Inspeção, com intervalo de tempo $[2,4]$.

Isso significa que, após a habilitação de t_1, deve-se esperar pelo menos 3 segundos antes de dispará-la, mas não mais que 6 segundos. O mesmo ocorre para t_2.

O processo acima descrito é representado graficamente conforme Figura 13.

Figura 13 – Representação gráfica de um RP-T.



Fonte: Autor,(2025)

4.1.2 PROPRIEDADES DAS REDES DE PETRI TEMPORAIS

As RP-Ts preservam muitas propriedades das RP clássicas, mas também introduzem novos desafios, como:

- **Bloqueio Temporal**: Se uma transição perder seu intervalo de disparo, o sistema pode ficar preso.
- **Concorrência Temporizada**: O tempo influencia quais transições disparam primeiro em cenários concorrentes.

4.2 REDES DE PETRI TEMPORALIZADAS

4.2.1 DEFINIÇÃO

As Rede de Petri Temporizadas (RP-Tzs) diferem das RP-Ts por associarem tempos **não às transições**, mas às fichas (*tokens*). Isso permite modelar fenômenos em que o tempo é uma propriedade das entidades que fluem no sistema, como processos industriais ou redes de transporte.

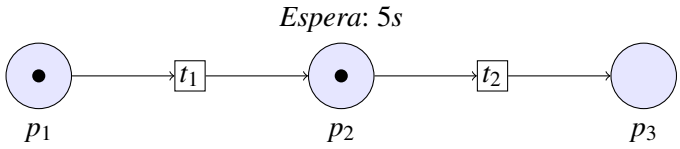
EXEMPLO DE REDE DE PETRI TEMPORALIZADA

Seja um sistema de entrega de pacotes no qual cada pacote tem um tempo de processamento antes de ser enviado. Podemos modelar isso com fichas que carregam *timestamps*(marca temporal)[3]:

- p_1: Local de coleta
- p_2: Processamento (com tempo de retenção de 5 segundos)
- p_3: Entrega

Aqui, quando uma ficha chega a p_2, ela deve aguardar 5 segundos antes que t_2 possa movê-la para p_3. Esse processo descrito é representado graficamente na Figura 14.

Figura 14 – Representação gráfica de um RP-Tz.



Fonte: Autor,(2025)

4.2.2 VANTAGENS DAS REDES DE PETRI TEMPORALIZADAS

- Modelam sistemas em que a temporalidade é inerente às entidades.
- Evitam bloqueios causados por tempos fixos nas transições.

[3] Um *timestamp* (marca temporal) representa um valor de tempo associado a um evento ou elemento do sistema, indicando quando algo ocorreu ou quando será processado.

4.3 COMPARATIVO ENTRE RP-T E RP-TZ

Característica	RP-T	RP-Tz
Tempo associado a	Transições (t)	Fichas (*tokens*)
Controle de fluxo	Baseado em intervalos fixos	Baseado no tempo de vida da ficha
Aplicabilidade	Processos industriais, workflow	Redes de comunicação, transporte

Tabela 3 – Comparativo entre RPT e RPTz

4.4 CONCLUSÃO

Redes de Petri Temporais e Temporalizadas são ferramentas fundamentais para modelagem de sistemas dinâmicos com restrições temporais. Enquanto as primeiras são mais indicadas para processos de controle e produção, as segundas são essenciais para modelar sistemas onde o tempo é uma propriedade das entidades que fluem.

No próximo capítulo, exploraremos algoritmos de análise para essas redes e suas aplicações na otimização de sistemas reais.

5 Redes de Petri Hierárquicas

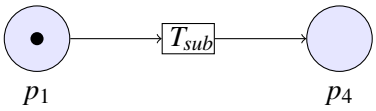
As Redes de Petri Hierárquicas (RPH) estendem as Redes de Petri convencionais permitindo a modelagem em diferentes níveis de abstração. Essa abordagem facilita a decomposição de sistemas complexos em componentes mais gerenciáveis.

5.1 DEFINIÇÃO

Uma RPH consiste em uma estrutura em que certos componentes (sub-redes) podem ser representados como transições abstratas em um nível superior. Isso permite uma visão modular e organizada do sistema modelado.

Sejam as figuras 15 e 16 a seguir. Na Figura 15 é ilustrada uma RP em que é representado um processo em nível supeiror como entrada e saída de um sub-processo que ocorre num nível de sub-rede. Na Figura 16, por sua vez, representa-se esse processo no sub-nível da RP.

Figura 15 – Rede de Petri Hierárquica - Nível Superior

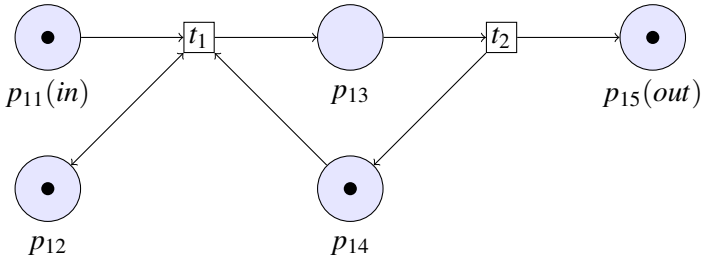


Fonte: Autor,(2025)

Aqui, a transição T_{sub} representa uma sub-rede detalhada em outro nível, cuja a estrutura interna da transição T_{sub} exibe sua funcionalidade interna.

Perceba, na Figura 16, que os lugares p_{11} e p_{15} estão referenciados com as inscrições *in* e *out*, respectivamente. Essas inscrições referem-se aos lugares de entrada e saída da transição T_{sub} que estão representados na Figura 15 por p_1 e p_4. Isso significa que para cada lugar de entrada e saída p_i na transição T_{sub}, haverá um lugar p_j correspondente dentro da sub-rede modelada ($\forall\, i, j \in \mathbb{N}$).

Figura 16 – Rede de Petri Hierárquica - Nível de sub-rede



Fonte: Autor,(2025)

As Redes de Petri Hierárquicas permitem uma organização mais clara e a reutilização de componentes em sistemas de grande porte.

6 CPN IDE®

Com o avanço das tecnologias de modelagem e simulação, a análise de sistemas complexos tornou-se mais acessível e eficiente. Nesse cenário, o CPN Tools® consolidou-se como uma ferramenta amplamente reconhecida na comunidade de Redes de Petri, oferecendo um ambiente robusto para a construção, simulação e análise de modelos de Redes de Petri Coloridas (RPC) (*Coloured Petri Nets* – CPN). No entanto, sua implementação na linguagem BETA, bem como a dependência do sistema de desenvolvimento Mjølner — ambos atualmente em desuso — comprometeram sua manutenção e expansão, conforme divulgado no sítio oficial do desenvolvedor (<https://cpnide.org/>).

Visando superar tais limitações e assegurar a continuidade evolutiva da ferramenta, foi desenvolvido o CPN IDE®, que substitui o CPN Tools® como solução moderna para edição e simulação de modelos de RPCs. Sua principal vantagem reside na extensibilidade e compatibilidade com sistemas operacionais contemporâneos. Diferentemente de seu antecessor, que utilizava um editor baseado em BETA, o CPN IDE® adota uma abordagem modular, integrando um editor em JavaScript® que se comunica com um controlador Java por meio de uma interface REST[4]. Esse controlador utiliza o Access/CPN, que encapsula o simulador ML do CPN Tools® em uma implementação Java.

Segundo seus desenvolvedores, a transição para o CPN IDE foi motivada pela necessidade de garantir a longevidade da ferramenta, sobretudo em aplicações como a mineração de processos. No Departamento de Matemática e Ciência da Computação da Eindhoven University of Technology (TU/e), onde o CPN Tools era amplamente empregado no ensino e na pesquisa, a obsolescência tecnológica motivou o desenvolvimento do CPN IDE como alternativa sustentável e expansível. Além de preservar as funcionalidades do CPN Tools, o CPN IDE permite integração com ferramentas como o **ProM**[5], possibilitando a descoberta de uma Rede de Petri a partir de *logs*, sua edição e posterior análise de conformidade por meio de simulações.

6.1 INTERFACE, FUNCIONALIDADES E LINGUAGEM DO CPN IDE®

6.1.1 INTERFACE

Na Figura 17 é apresentada a interface principal do ambiente de desenvolvimento do CPN IDE®. No centro, observa-se a área de modelagem, destacando-se três elementos: (a) o lugar p_1; (b) a transição t_1; e (c) o lugar p_2.

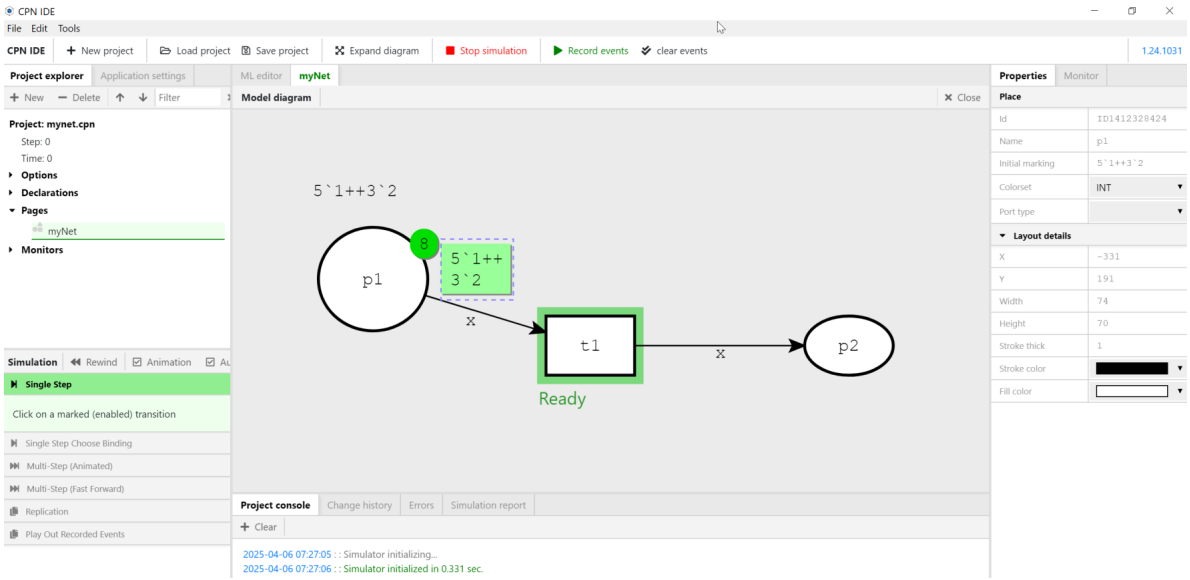
A transição t_1 está envolta por uma moldura verde, indicando que está habilitada para disparo — indicação padrão utilizada pelo software no modo de animação. Verifica-se que o lugar p_1 contém oito fichas: cinco com valor 1 e três com valor 2.

Os arcos que ligam p_1 a t_1 e t_1 a p_2 utilizam a variável x, representando o valor de cada ficha presente em p_1. Isso é viabilizado pela definição de conjuntos de cores, ou seja, os tipos de dados atribuídos a cada lugar. Neste exemplo, p_1 e p_2 foram definidos com o conjunto de cores **INT** (inteiros).

[4] REST (*Representational State Transfer*) é uma arquitetura para desenvolvimento de sistemas distribuídos baseada em operações HTTP como GET, POST, PUT e DELETE.

[5] ProM é uma plataforma de mineração de processos que oferece uma coleção extensível de algoritmos para análise de *logs* de eventos. Disponível em: <https://www.promtools.org/>.

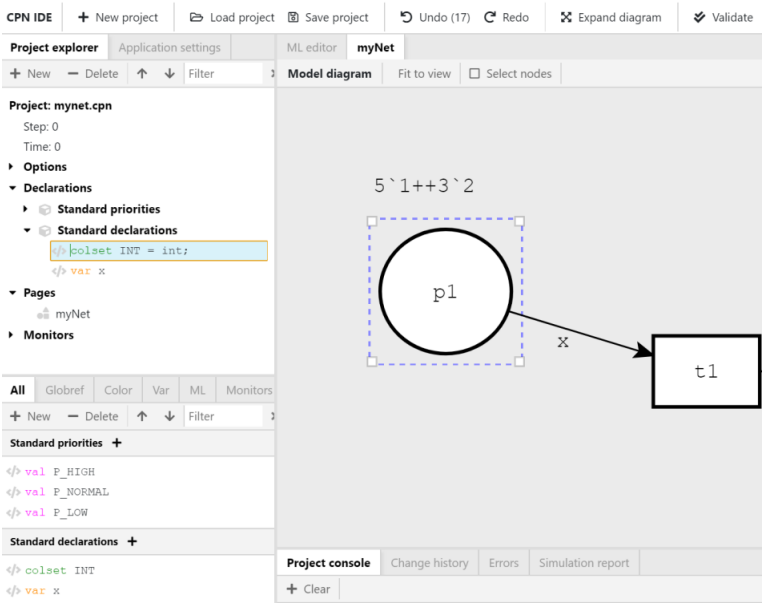
Figura 17 – Interface de desenvolvimento do CPN IDE®



Fonte: Autor (2025)

A inserção dessas declarações ocorre por meio do painel *Project Explorer*, conforme ilustrado na Figura 18. Este será detalhado na Seção 7.

Figura 18 – Área de declarações no CPN IDE®



Fonte: Autor (2025)

6.1.2 FUNCIONALIDADES PRINCIPAIS DO CPN IDE®

O CPN IDE® oferece recursos essenciais para a modelagem, simulação e análise de Redes de Petri Coloridas, dentre os quais se destacam:

- **Criação e edição de modelos**: por meio de um editor gráfico que permite desenhar redes, configurar atributos e inserir declarações em **CPN ML**. Os modelos podem ser organizados em páginas

hierárquicas.

- **Simulação do modelo**: permite a análise dinâmica e depuração, com suporte à simulação passo a passo ou automática. Também é possível realizar simulações de longo prazo para estimativa de métricas como taxa de transferência e qualidade de serviço (**QoS**)[6].

- **Geração e análise do espaço de estados**: possibilita verificar propriedades como limitação (*boundedness*) e vivacidade (*liveness*) por meio de consultas escritas em **CPN ML**. Em modelos complexos, a análise pode ser comprometida pela explosão do espaço de estados.

6.1.3 LINGUAGEM DO CPN IDE®: CPN ML

O CPN IDE® utiliza a linguagem **CPN ML**, uma extensão da linguagem ML voltada à definição de declarações e inscrições em RPCs. Nela, é possível declarar conjuntos de cores (tipos de dados), variáveis, funções e constantes.

Cada lugar da rede associa-se a um conjunto de cores, que define os tipos de fichas permitidos. Variáveis e funções são utilizadas nas inscrições dos arcos e nas guardas das transições.

As declarações são inseridas na área denominada **índice**. O CPN ML disponibiliza os seguintes conjuntos de cores predefinidos:

- **E** – Elementar;
- **INT** – Inteiro;
- **BOOL** – Booleano;
- **STRING** – Cadeia de caracteres.

O usuário pode, ainda, definir conjuntos de cores personalizados, variáveis e estruturas, por meio do **menu sensível ao contexto**, ou importar declarações de arquivos externos — recurso útil em modelos complexos.

O sistema realiza verificação automática de sintaxe ao criar ou carregar uma rede. O status dessa verificação é indicado por **auras coloridas** nos elementos e **sublinhados coloridos** nas declarações, conforme descrito a seguir:

- **Aura laranja**: elemento ainda não verificado;
- **Aura amarela**: verificação em andamento;
- **Sem aura**: verificação concluída com sucesso;
- **Aura vermelha**: erro de sintaxe detectado.

O tempo necessário para a verificação depende da complexidade do modelo. Declarações são avaliadas sequencialmente, de cima para baixo, e dependências não resolvidas geram erros. Alterações em uma declaração provocam a reavaliação automática das dependentes.

[6] **QoS**: métricas como latência, jitter, largura de banda e taxa de perda de pacotes.

Erros são sinalizados com sublinhado vermelho tanto no nome da declaração quanto na página correspondente. Quando um elemento apresenta aura vermelha, o sistema exibe uma **janela explicativa** descrevendo o erro. Elementos conectados a componentes com erro não são verificados até a resolução completa do problema.

6.2 FUNÇÕES E ESTRUTURAS DE CONTROLE NA LINGUAGEM CPN ML

A linguagem utilizada no CPN IDE é uma variação da ML (MetaLanguage), conhecida como Standard ML (SML), adaptada para suportar a modelagem de Redes de Petri Coloridas (CPNs). Um dos principais recursos dessa linguagem são as **funções**, que permitem expressar lógica condicional, iteração e decisões complexas de forma funcional e elegante. As estruturas mais comuns são: `if`, `case`, `let`, `fn`, `fun`, e estruturas de iteração como `List.map`, `List.foldl`, entre outras.

6.2.1 ESTRUTURA `IF-THEN-ELSE`

A estrutura `if-then-else` permite realizar decisões condicionais simples. Sua sintaxe é semelhante a outras linguagens de programação:

```
fun eMaiorQueZero x =
if x > 0 then true
else false;
```

O exemplo acima define uma função que retorna `true` se o número for maior que zero; caso contrário, retorna `false`. Essa estrutura é útil em guardas de transições ou em expressões associadas às variáveis de arco.

6.2.2 ESTRUTURA `CASE-OF`

A estrutura `case-of` permite o tratamento de diferentes padrões de dados, o que é extremamente útil para modelagem com tipos algébricos e listas:

```
fun classificarNota n =
case n of
10 => "Excelente"
| 7 => "Bom"
| 5 => "Regular"
| _ => "Insuficiente";
```

Neste exemplo, a função retorna uma classificação textual conforme a nota fornecida. O caractere _ representa qualquer outro valor que não tenha sido previamente tratado.

6.2.3 FUNÇÕES ANÔNIMAS: `FN`

Funções anônimas são expressões de função que não precisam ser nomeadas. São úteis para serem passadas como argumento a outras funções:

```
val dobrar = fn x => x * 2;
```

A função acima cria uma expressão que dobra o valor de entrada. Pode ser usada diretamente em expressões maiores, como em mapeamentos sobre listas.

6.2.4 FUNÇÕES NOMEADAS: `FUN`

A palavra-chave `fun` permite declarar funções nomeadas, ideais para expressar lógica reutilizável e modular em modelos:

```
fun fatorial n =
if n = 0 then 1
else n * fatorial(n - 1);
```

Esse exemplo define a função recursiva do fatorial. Essa abordagem é muito comum em cálculos dentro de expressões de arco ou ações de transição.

6.2.5 BLOCO `LET-IN-END`

A estrutura `let-in-end` permite declarar variáveis locais temporárias para compor uma lógica mais complexa:

```
fun media (a, b, c) =
let
val soma = a + b + c
in
soma / 3
end;
```

Neste exemplo, uma variável intermediária `soma` é utilizada dentro da função `media`, tornando o código mais organizado.

6.2.6 ITERAÇÃO FUNCIONAL: `MAP` E `FOLDL`

Como SML é uma linguagem funcional, a repetição é feita com funções de ordem superior. Por exemplo:

```
val quadrados = List.map (fn x => x * x) [1, 2, 3, 4];
```

A função `List.map` aplica a função anônima a cada elemento da lista, retornando uma nova lista com os quadrados dos números originais.

Outra função poderosa é o `foldl`, que agrega valores da esquerda para a direita:

```
val soma = List.foldl (fn (x, acc) => x + acc) 0 [1, 2, 3, 4];
```

6.2.7 RESUMO DAS ESTRUTURAS DE CONTROLE

Na Tabela 4 é apresentada um resumo das estruturas de controle na Linguagem CPN ML.

Tabela 4 – Principais Estruturas de Controle na Linguagem CPN ML

Comando	**Descrição**
`if-then-else`	Avaliação condicional simples.
`case-of`	Avaliação de múltiplos padrões.
`fn`	Funções anônimas para uso pontual.
`fun`	Definição de funções nomeadas e recursivas.
`let-in-end`	Declaração de variáveis locais.
`List.map`	Iteração funcional para transformar listas.
`List.foldl`	Redução de lista com acumulador.

6.3 FUNÇÕES E RECURSOS ASSOCIADOS ÀS TRANSIÇÕES E ARCOS

Em redes de Petri coloridas (CPNs), as transições representam eventos que modificam o estado do sistema ao consumirem e produzirem tokens. No *CPN Tools*, essas transições podem ser enriquecidas com expressões escritas em uma linguagem funcional baseada em Standard ML (Meta Language), oferecendo um elevado nível de expressividade e controle sobre o comportamento da rede.

6.3.1 GUARDS: RESTRIÇÕES AO DISPARO DE TRANSIÇÕES

Os *guards* são expressões booleanas associadas às transições que funcionam como condições de habilitação. Uma transição somente poderá ser disparada se seu guard for avaliado como verdadeiro, dado o estado atual das fichas nos lugares de entrada.

```
x > 0 andalso x < 10
```

Esse guard restringe o disparo da transição para valores de `x` estritamente entre 1 e 9, promovendo maior controle e segurança lógica sobre o fluxo de eventos no modelo.

6.3.2 EXPRESSÕES EM ARCOS: MANIPULAÇÃO DE TOKENS

Os arcos de entrada e saída podem conter expressões que descrevem como os tokens devem ser lidos ou produzidos. Essas expressões são avaliadas no momento do disparo da transição e podem utilizar operadores, funções ou estruturas compostas como tuplas e listas.

Listing 6.1 – Expressão de arco

```
(x, y)
```

A expressão acima indica que dois valores, `x` e `y`, são consumidos ou produzidos como um par ordenado.

6.3.3 ESTRUTURAS DE CONTROLE: `IF`, `CASE`, `LET`

A linguagem de expressões suporta estruturas de controle típicas de linguagens funcionais, permitindo o uso de lógica condicional, correspondência de padrões e definições locais.

- **`if ... then ... else ...`**: permite avaliação condicional simples.
- **`case ... of ...`**: realiza análise estrutural por padrão, ideal para listas e tipos compostos.
- **`let ... in ... end`**: declara variáveis locais e organiza expressões complexas.

```
if x mod 2 = 0 then "par" else "impar"
```

```
case lista of
[] => 0
| x::xs => x + List.length(xs)
```

```
let
val soma = x + y
in
soma * 2
end
```

Essas estruturas tornam o comportamento das transições mais robusto e adaptável a diferentes condições do sistema.

6.3.4 FUNÇÕES PADRÃO E DEFINIDAS PELO USUÁRIO

O *CPN IDE* oferece suporte tanto a funções nativas de ML quanto à criação de funções personalizadas. As funções podem ser utilizadas em guards, expressões de arco ou inicializações de lugares.

Listing 6.2 – Função definida pelo usuário

```
fun dobro x = x * 2;
```

Tais funções podem ser escritas na aba de código global (`Declarations`), permitindo reutilização e modularidade no modelo.

6.3.5 FUNÇÕES DE LISTA E RECURSIVIDADE

A linguagem suporta operações sobre listas, como `List.map`, `List.foldl`, `List.length`, entre outras. A recursão também é amplamente utilizada para implementar comportamentos iterativos e transformações complexas.

```
List.foldl (fn (x, acc) => x + acc) 0 [1,2,3,4]
```

```
fun fatorial n =
if n = 0 then 1
else n * fatorial(n - 1);
```

Esses recursos tornam possível simular algoritmos computacionais diretamente dentro das transições.

6.4 DISTRIBUIÇÕES DE PROBABILIDADE NO CPN IDE

O *CPN IDE*, por meio do módulo `Random`, fornece funções para gerar números aleatórios com base em distribuições estatísticas amplamente utilizadas em modelagem estocástica. A seguir, apresentam-se as principais distribuições disponíveis, com sintaxe, parâmetros e principais propriedades.

DISTRIBUIÇÕES DISPONÍVEIS

- **Bernoulli** – `bernoulli(p)`: $0 \leq p \leq 1$, Média: p, Variância: $p(1-p)$
- **Binomial** – `binomial(n, p)`: $n \in \mathbb{N}^+$, $0 \leq p \leq 1$, Média: np, Variância: $np(1-p)$
- **Qui-quadrado (Chi-sq)** – `chisq(n)`: $n \in \mathbb{N}^+$, Média: n, Variância: $2n$
- **Erlang** – `erlang(n, r)`: $n \in \mathbb{N}^+$, $r > 0$, Média: nr, Variância: nr^2

- **Exponencial** – `exponential(r)`: $r > 0$, Média: $\frac{1}{r}$, Variância: $\frac{1}{r^2}$
- **Normal (Gaussiana)** – `normal(`μ`, `σ^2`)`: $\mu, \sigma^2 \in \mathbb{R}$, Média: μ, Variância: σ^2
- **Poisson** – `poisson(m)`: $m > 0$, Média = Variância = m
- **Student (t)** – `student(n)`: $n > 1$, Média: 0, Variância: $\frac{n}{n-2}$ (se $n > 2$)
- **Uniforme** – `uniform(a, b)`: $a \leq b$, Média: $\frac{a+b}{2}$, Variância: $\frac{(b-a)^2}{12}$

6.5 MULTI-CONJUNTOS (MULTI-SETS)

Os **multi-conjuntos** são amplamente utilizados no *CPN IDE* para representar marcações em lugares, além de outros propósitos. Também conhecidos como *bolsas* (*bags*), diferem dos conjuntos tradicionais por permitirem múltiplas cópias do mesmo elemento.

O operador ` (acento grave) é utilizado para construir multi-conjuntos. Por exemplo, `7`4` representa o multi-conjunto com sete cópias do elemento de cor `4`. A sintaxe geral é:

- `i`c` – onde `i` é um inteiro não negativo que representa a multiplicidade do elemento `c`.

O operador de construção de multi-conjuntos pode ser combinado com os operadores de adição (++) e subtração (−), permitindo uma especificação simples e clara. Por exemplo, a marcação inicial do lugar `saco com grãos`, mostrada na Figura **??**, na Seção 2.2, é:

```
250`arroz ++ 300`aveia ++ 270`trigo
```

Isso significa que o lugar contém 250 grãos de arroz, 300 de aveia e 270 de trigo.

Atenção: o operador correto de construção de multi-conjuntos é o acento grave (`), e não o apóstrofo comum (').

OPERAÇÕES E FUNÇÕES DISPONÍVEIS

Os multi-conjuntos oferecem diversas operações úteis, como segue:

- `empty` – multi-conjunto vazio (compatível com qualquer tipo de cor).
- `ms1 == ms2` – verifica igualdade entre `ms1` e `ms2`.
- `ms1 <><> ms2` – verifica desigualdade.
- `ms1 » ms2` – `ms1` é estritamente maior que `ms2`.
- `ms1 »== ms2` – `ms1` é maior ou igual a `ms2`.
- `ms1 « ms2` – `ms1` é estritamente menor que `ms2`.
- `ms1 «== ms2` – `ms1` é menor ou igual a `ms2`.
- `ms1 ++ ms2` – união de multi-conjuntos (soma das multiplicidades).

- `ms1 - ms2` – subtração de multiplicidades.
- `i ** ms` – multiplicação escalar do multi-conjunto `ms` por `i`.
- `size ms` – retorna a cardinalidade total do multi-conjunto `ms`.
- `random ms` – seleciona pseudo-aleatoriamente uma cor de `ms`.
- `cf(c, ms)` – retorna o número de cópias da cor `c` em `ms`.
- `filter p ms` – retorna um novo multi-conjunto com elementos que satisfazem o predicado `p`.

EXEMPLO PRÁTICO

Sejam os multi-conjuntos:

```
m1 = 3`7 ++ 5`2 ++ 8`14;
m2 = 1`7 ++ 2`2 ++ 5`14;
```

Então, temos:

- `m1 ++ m2 = 4`7 ++ 7`2 ++ 13`14`
- `m1 - m2 = 2`7 ++ 3`2 ++ 3`14`
- `m1 » m2` é `verdadeiro`
- `size m1 = 16`
- `cf(14, m2) = 5`

USO NO *CPN IDE*

No *CPN IDE*, assim como no *CPN Tools*, qualquer marcação — seja inicial ou corrente — de um lugar é representada por um multi-conjunto do conjunto de cores associado a esse lugar.

No momento do disparo de uma transição, é realizada uma escolha aleatória de uma ficha, de acordo com a variável associada ao arco de saída do lugar, respeitando a multiplicidade definida pelo multi-conjunto.

CONSIDERAÇÕES

As funções são avaliadas no estilo ML e podem ser utilizadas diretamente em marcações de tempo, guardas ou ações dentro das transições. A escolha da distribuição deve refletir o comportamento probabilístico do sistema modelado.

6.5.1 BOAS PRÁTICAS E CONSIDERAÇÕES DE TIPAGEM

Por ser uma linguagem fortemente tipada, o CPN IDE exige que todas as expressões estejam de acordo com os tipos definidos nos *colsets* dos lugares. Incompatibilidades de tipo impedem o disparo da transição e devem ser corrigidas antes da simulação.

Além disso, é recomendável:

- Escrever funções pequenas e bem nomeadas;
- Evitar duplicação de lógica usando o bloco `let`;
- Utilizar `case` para tratamento seguro de listas e estruturas opcionais;
- Testar funções na aba de código antes de usá-las nos arcos ou guards.

Desse modo, observa-se que compreender as estruturas de controle da linguagem CPN ML é fundamental para criar modelos eficientes, expressivos e modulares no CPN IDE. Tais estruturas permitem incorporar lógica complexa de maneira clara e rigorosa, promovendo maior confiabilidade na simulação de sistemas concorrentes e distribuídos.

7 Modelagem de Redes de Petri Coloridas com Conjuntos de Cores

O ambiente *CPN IDE*, assim como o *CPN Tools*, disponibiliza um sistema robusto de definição de conjuntos de cores, fundamental para a modelagem de Redes de Petri Coloridas (RPC). Esses conjuntos de cores são categorizados em dois grupos principais: os conjuntos de cores simples, abordados nesta seção, e os conjuntos de cores compostas, que serão tratados na Seção 7.2.

7.1 CONJUNTOS DE CORES SIMPLES

Os conjuntos de cores simples constituem os blocos básicos para a construção de modelos no *CPN Tools*. Eles representam domínios de dados fundamentais e são amplamente utilizados em marcações, expressões e transições. Os tipos simples disponíveis são:

```
Unit, Boolean, Integer, String, Enumerated, Index
```

Cada um desses conjuntos será apresentado individualmente, destacando suas características, uso típico em modelagem e operações disponíveis no ambiente.

7.1.1 CONJUNTO DE CORES UNIT

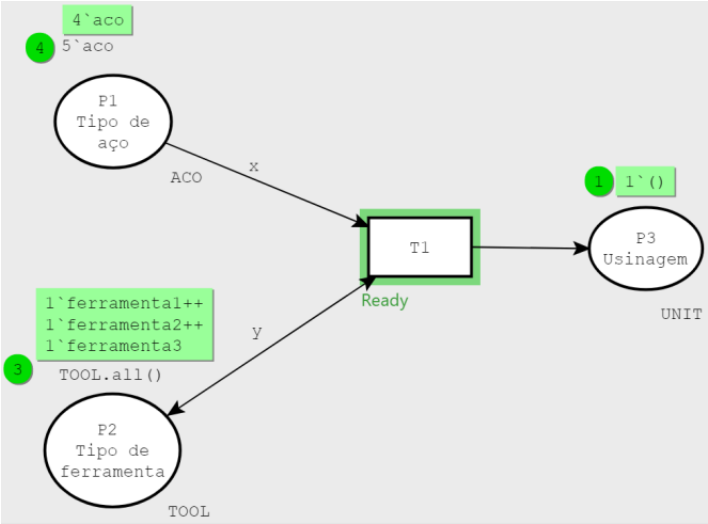
O conjunto de cores `unit` compreende um único elemento. Esse conjunto é utilizado quando se deseja modelar um lugar que simplesmente indica presença de uma ficha (*token*). Sua declaração possui a seguinte sintaxe:

```
colset name = unit [with new unit];
```

Se a opção (*new unit*) não for utilizada, o nome da ficha coincide com o nome do conjunto de cores.

Na Figura 19 é apresentada uma rede que modela uma etapa de usinagem utilizando o conjunto de cores `unit`. O lugar p_1 representa a modelagem de material a ser usinado, o lugar p_2 representa o recurso de ferramental necessário à usinagem, e o lugar p_3 representa o material já usinado. Perceba que: as fichas em p_1 são do tipo `aco`, pertencentes ao `colset` ACO; as fichas em p_2 são do tipo `ferramenta1,ferramenta2,ferramenta3`, pertencentes ao `colset` TOOL; as fichas em p_3 são do tipo indefinido, porém pertencentes ao `colset` UNIT.

Figura 19 – Modelagem do conjunto de cores `unit` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset ACO = unit with aco;
colset TOOL = with ferramenta1 | ferramenta2 | ferramenta3 ;
colset UNIT = unit;
var x : ACO;
var y : TOOL;
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.1.2 CONJUNTO DE CORES BOOLEAN

O conjunto de cores boolean compreende dois valores: `true` e `false`. Sua declaração possui a seguinte sintaxe:

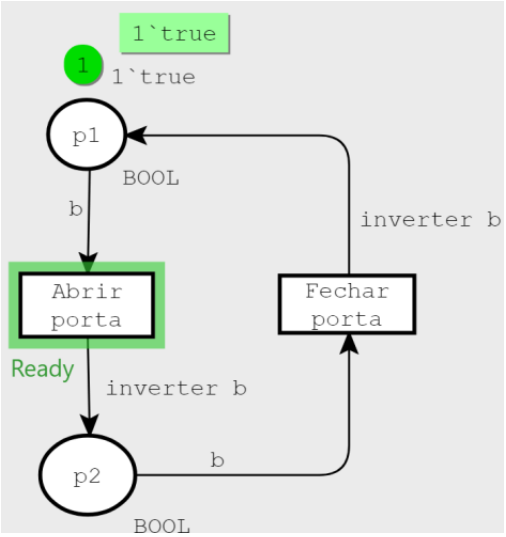
```
colset name = bool [with new false, new true];
```

As opções (*new false, new true*) permitem novos nomes para *false* e *true*. Por exemplo, não e sim. `colset Pergunta = bool with (nao, sim);` As seguintes operações podem ser aplicadas às variáveis booleanas:

```
not b : negação do valor booleano de b;
b1 andalso b2 : conjunção booleana and;
b1 orelse b2 :  disjunção booleana or.
```

Na Figura 20, é apresentada uma rede de Petri colorida que modela o processo de abertura e fechamento automático de uma porta com base em um sensor de presença. O lugar p_1 representa o estado inicial do sensor, indicando que há uma pessoa próxima à porta (valor booleano `true`). Já o lugar p_2 representa o estado final do sensor, indicando que a pessoa se afastou (valor `false`). A função `inverter b` é responsável por realizar a operação booleana de negação, alternando entre os estados `true` e `false`, conforme o comportamento esperado do sensor.

Figura 20 – Modelagem do conjunto de cores `bool` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset BOOL = bool;
var b : BOOL;
fun inverter b = not b;
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.1.3 CONJUNTO DE CORES INTEGER

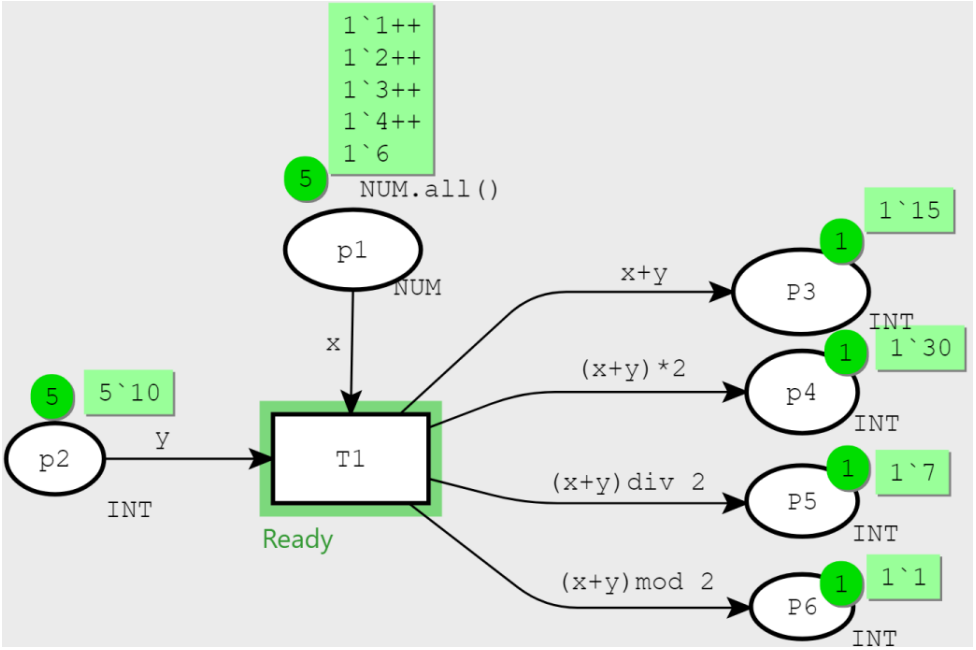
O conjunto de cores Integer compreende os valores inteiros. Sua declaração possui a seguinte sintaxe:

```
colset name = int with int-exp1 .. int-exp2;
```

Adicionalmente, a opção **with** permite restringir o conjunto de cores inteiro pelo *intervalo* determinado pelas duas expressões `int-exp1` e `int-exp2`.

Na Figura 21, é apresentada uma rede de Petri colorida que modela uma etapa de contagem de produtos, utilizando o conjunto de cores `int`. O lugar p_1 armazena os valores individuais dos produtos, enquanto o lugar p_2 representa um peso fixo de 10 unidades associado a cada operação de contagem.

Figura 21 – Modelagem do conjunto de cores `int` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset INT = int;
colset NUM = int with 1..6;
var x : NUM;
var y : INT;
```

Na marcação atual da rede, observam-se cinco fichas em p_1, com os valores $[1,2,3,4,5]$; cinco fichas de valor 10 em p_2; e uma ficha no lugar p_3, com valor 15. Essa ficha em p_3 resulta da soma valor 5, anteriormente presente em uma ficha de p_1, com o valor 10, proveniente de uma ficha de p_2. Essa operação é descrita no arco que liga T_1 a p_3 pela expressão $x+y$. O modo processo aplica-se às expressões dos demais arcos de modo que: p_4 recebe o produto de $(x+y)\times 2$, p_5 recebe o valor da divisão de $\frac{(x+y)}{2}$, e p_5 recebe o valor do resto de divisão $(x+y)\times 2$. Assim, a rede executa uma operação de adição entre os valores de entrada, simulando o processo de contagem ponderada de produtos.

As seguintes operações podem ser aplicadas às variáveis inteiras: `+, -, div, mod, abs, Int.min, Int.max`. A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.1.4 CONJUNTO DE CORES STRING

O conjunto de cores **String** é especificado por uma sequência de caracteres ASCII entre aspas. Sua declaração possui a seguinte sintaxe:

```
colset name = string [with string-exp1 ..string-exp2
[and int-exp1..int-exp2]];
```

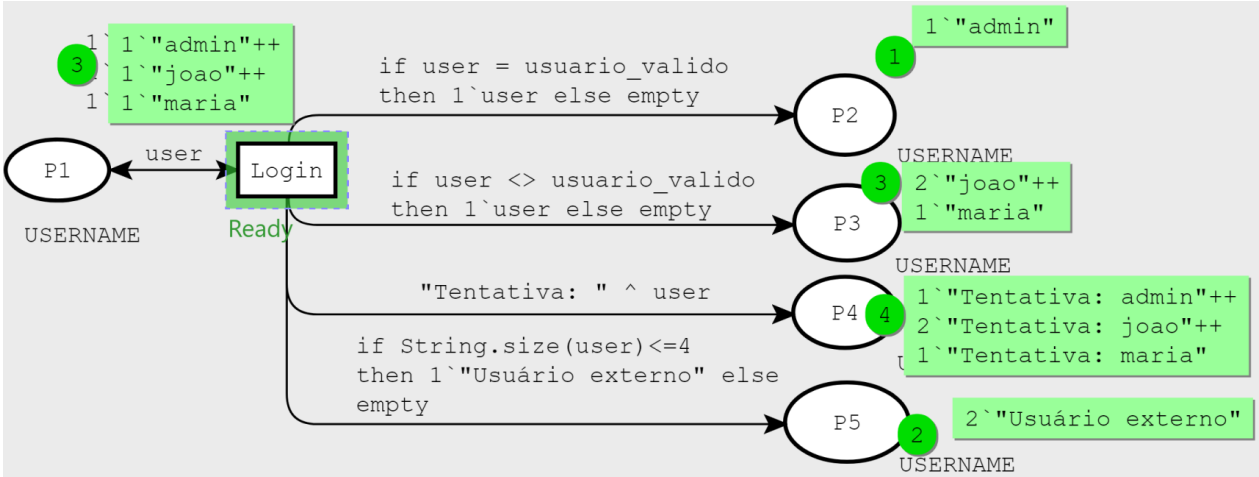
A opção `with` especifica o intervalo de caracteres válidos, por exemplo:

```
colset minusculas = string with "a".."z";
```

Na Figura 22, é apresentada uma rede de Petri colorida que modela uma etapa de acesso (*login*) de um sistema digital, utilizando o conjunto de cores `string`. O lugar p_1 armazena os valores individuais dos usuários que tentam acessar o sistema, o lugar p_2 representa os acessos de usuários autorizados *admin*, o lugar p_3 armazena o usuários não autorizados, o lugar p_4 armazena todas as tentativas de acesso, e o lugar p_5 armazena as fichas cujo valor tamanho da `string` seja menor ou igual a 4, identificando como `usuário externo`.

A distinção entre os usuários é realizada por uma função *if* sobre a variável *user*, tendo *admin* como *usuário válido*.

Figura 22 – Modelagem do conjunto de cores `string` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição.

As seguintes declarações foram utilizadas na rede em questão.

```
colset USERNAME = string;
var user : USERNAME;
val usuario_valido = "admin";
```

As seguintes operações podem ser aplicadas às variáveis string: `^(concatenação), String.size, substring`.

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.1.5 CONJUNTO DE CORES ENUMERATED

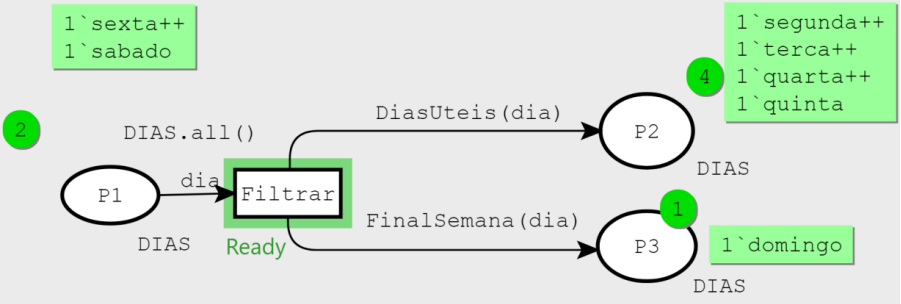
O conjunto de cores **Enumerated** explicita todos os identificadores na sua declaração. Sua sintaxe é assim definida:

```
colset name = with id0 | id1 | ...| idn;
```

Na Figura 23, é apresentada uma Rede de Petri Colorida que modela uma etapa relacionada aos dias da semana, utilizando um conjunto de cores do tipo `enumerated`. O lugar p_1 armazena fichas re-

presentando os dias da semana; o lugar p_2 armazena os dias úteis, filtrados pela função `DiaUtil(dia)`; e o lugar p_3 contém os dias de sábado e domingo, identificados pela função `FimdeSemana(dia)`.

Figura 23 – Modelagem do conjunto de cores `enumerated` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset DIAS = with segunda | terca | quarta | quinta | sexta | sabado | domingo;
var dia : DIAS;
fun DiaUtil dia =  if dia <> sabado andalso dia <> domingo then 1‘dia else empty;
fun FimdeSemana dia =  if dia = sabado orelse dia = domingo then 1‘dia else empty;
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.1.6 CONJUNTO DE CORES INDEXED

Os conjuntos de cores **Indexed** são sequências de valores que contêm um inteiro e um índice especificador. Sua declaração possui a seguinte sintaxe:

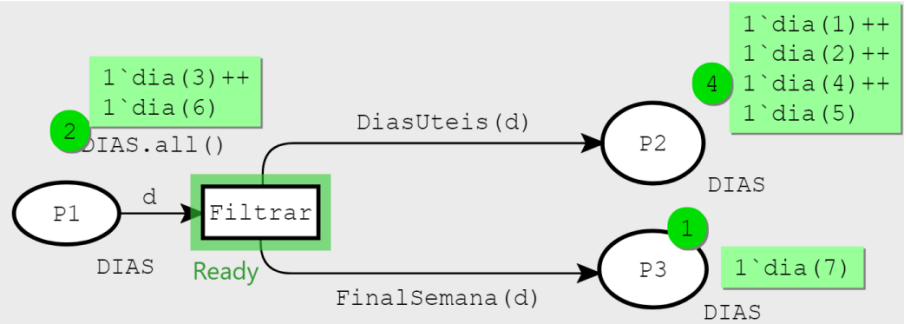
`colset name = index id with int-exp1 ..int-exp2;`

Valores indexados possuem o seguinte formato: `id i` ou `id(i)`, em que

`i` é um inteiro e `int-exp1` $\leq$ `i` $\leq$ `int-exp2`.

Na Figura 24, é apresentada uma Rede de Petri Colorida que modela uma etapa relacionada aos dias da semana, utilizando um conjunto de cores do tipo `indexed`. O lugar p_1 armazena fichas representando os dias da semana; o lugar p_2 armazena os dias úteis, filtrados pela função `DiasUteis(d)`; e o lugar p_3 contém os dias de sábado e domingo, identificados pela função `FinalSemana(d)`.

Figura 24 – Modelagem do conjunto de cores `indexed` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset DIAS = index dia with 1..7;
var d: DIAS;
fun DiasUties (dia(i)) =  if i <=5 then 1`dia(i) else empty;
fun FinalSemana (dia(i)) =  if i >5 then 1`dia(i) else empty;
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.1.7 RESUMO: CONJUNTO DE CORES SIMPLES

Na Tabela 5 é apresentado um resumo do conjunto de cores simples, detalhado nesta secção.

Tabela 5 – Resumo dos conjuntos de cores simples no CPN Tools

Conjunto	Sintaxe	Uso típico	Operações
UNIT	`colset X = unit;`	Presença de ficha	–
BOOLEAN	`colset X = bool;`	Estados lógicos	`not`, `andalso`, `orelse`
INTEGER	`colset X = int [a..b];`	Contagem, pesos, índices	`+`, `-`, `div`, `mod`, `abs`
STRING	`colset X = string;`	Identificadores, nomes	`^`, `size`, `substring`
ENUM.	`colset X = with id1|...;`	Domínios discretos	Guardas, arcos, marcações
INDEXED	`colset X = index id with a..b;`	Elementos com índice	Funções condicionais

7.2 CONJUNTOS DE CORES COMPOSTOS

Os *conjuntos de cores compostos* são construídos a partir da combinação de conjuntos de cores simples. A linguagem CPN ML oferece suporte aos seguintes tipos de conjuntos compostos: `product`, `record`, `union`, `list`, `subset` e `alias`.

Dentre esses, os conjuntos `product` e `record` representam dados estruturados formados pelo produto cartesiano entre os elementos de outros conjuntos. A principal diferença entre eles está na nomeação dos componentes: enquanto os elementos do conjunto `product` são referenciados apenas por

sua posição (sem nomes explícitos), os componentes do conjunto `record` são identificados por nomes, o que favorece a legibilidade e organização dos dados. Essa distinção é análoga à encontrada entre tipos de dados em linguagens de programação tradicionais, como o tipo `record` na linguagem Pascal e a estrutura `struct` na linguagem C.

7.2.1 CONJUNTO DE CORES PRODUCT

O conjunto de cores do tipo `product` segue a seguinte sintaxe:

```
colset <nome> = product <nome_1> * <nome_2> * ...* <nome_n>;
```

Os valores pertencentes a esse conjunto possuem a forma:

$$(v_1, v_2, \ldots, v_n),$$

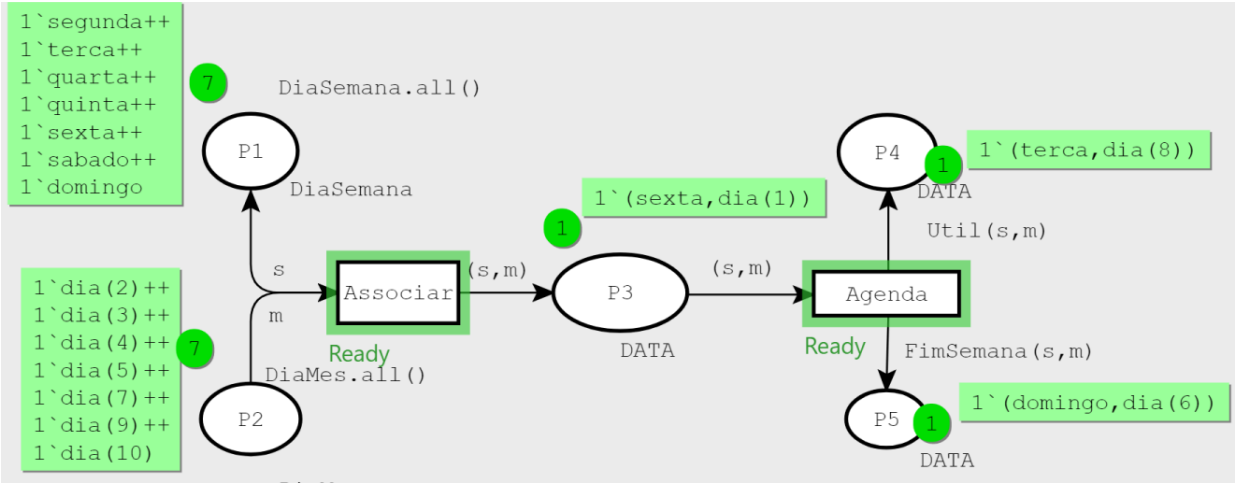
em que cada v_i é do tipo correspondente $< nome_i >$, para $1 \leq i \leq n$.

Para acessar o i-ésimo elemento de uma variável do tipo `product`, utiliza-se a seguinte notação:

```
#<i> <nome>;
```

Na Figura 25, apresenta-se uma Rede de Petri Colorida que modela uma etapa relacionada aos dias da semana. O lugar p_1 armazena fichas representando os dias da semana, enquanto o lugar p_2 contém fichas correspondentes a 10 dias úteis. O lugar p_3 representa a associação entre as fichas provenientes de p_1 e p_2, utilizando um conjunto de cores do tipo `product`. Por fim, os lugares p_4 e p_5 armazenam, respectivamente, as fichas classificadas como dias úteis e fins de semana, com base nas funções `Util` e `FimSemana` definidas nos arcos correspondentes.

Figura 25 – Modelagem do conjunto de cores `product` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset DiaSemana = with segunda | terca | quarta | quinta | sexta | sabado |domingo;
colset DiaMes = index dia with 1 .. 10;
colset DATA = product DiaSemana * DiaMes;
```

```
var s : DiaSemana;
var m : DiaMes;
fun Util(s,m) = if s<>sabado andalso s<>domingo then 1`(s,m) else empty;
fun FimSemana(s,m) = if s=sabado orelse s=domingo then 1`(s,m) else empty;
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.2.2 CONJUNTO DE CORES RECORD

O conjunto de cores do tipo **record** possui a seguinte sintaxe:

```
colset nome = record id1:name1 * id2:name2 * ... * idn:namen;
```

Os valores pertencentes a esse conjunto de cores têm a forma:

$$(id1 = v1, id2 = v2, \ldots, idn = vn)$$

em que cada v_i é um valor do tipo $name_i$, para $1 \leq i \leq n$.

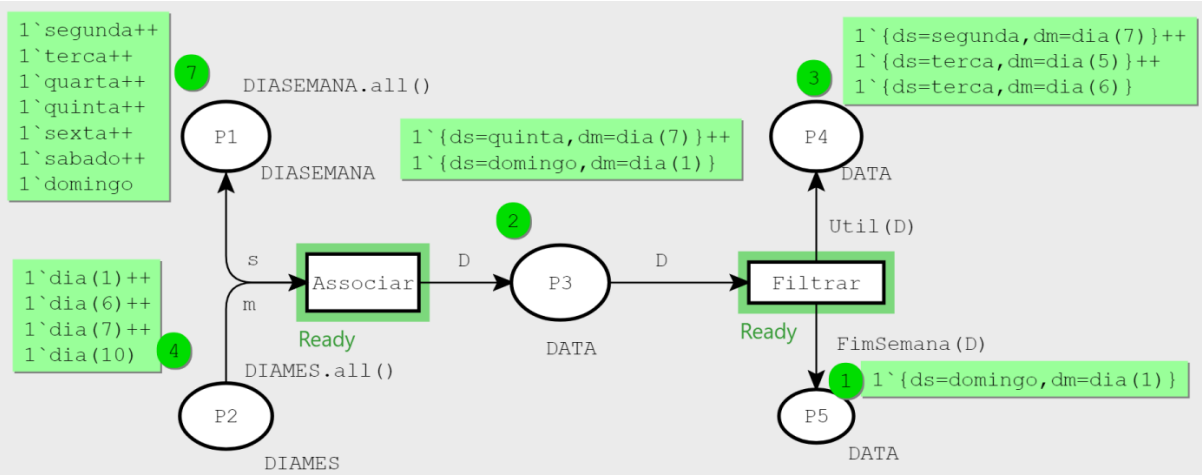
Para acessar o i-ésimo campo de um **record**, utiliza-se a seguinte operação:

```
#idi nome;
```

Considere o mesmo exemplo apresentado para o conjunto de cores `product` representado na Figura 25. Perceba a mesma rede agora modificada para o tipo `record` e representado na Figura 26.

Perceba que a Rede de Petri Colorida que modela, **de modo símile**, uma etapa relacionada aos dias da semana. O lugar p_1 armazena fichas representando os dias da semana, enquanto o lugar p_2 contém fichas correspondentes a 10 dias úteis. O lugar p_3 representa a associação entre as fichas provenientes de p_1 e p_2, utilizando um conjunto de cores do tipo `record`. Por fim, os lugares p_4 e p_5 armazenam, respectivamente, as fichas classificadas como dias úteis e fins de semana, com base nas funções `Util` e `FimSemana` definidas nos arcos correspondentes.

Figura 26 – Modelagem do conjunto de cores `record` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição.

As seguintes declarações foram utilizadas na rede em questão:

```
colset DIASEMANA = with segunda | terca | quarta | quinta | sexta | sabado |domingo;
colset DIAMES = index dia 1 .. 10;
colset DATA = record ds:DIASEMANA * dm:DIAMES;
var s : DIASEMANA;
var m : DIAMES;
var D : DATA;
fun Util(D: DATA) =
      if #ds D <>sabado andalso #ds D<>domingo
      then 1`D else empty;
fun FimSemana(D: DATA) =
      if #ds D=sabado orelse #ds D=domingo
      then 1`D else empty;
```

Observe que o `colset record` permite a declaração de identificadores nomeados diretamente associados aos tipos de dados do conjunto. Isso pode ser verificado no trecho destacado abaixo, extraído das declarações utilizadas na modelagem da rede apresentada anteriormente, em que os identificadores `ds` e `dm` são definidos como parte integrante da estrutura do conjunto de cores:

```
colset DATA = record ds:DIASEMANA * dm:DIAMES;
```

A principal vantagem dessa abordagem é a melhoria na legibilidade e na clareza semântica do modelo. Ao utilizar identificadores nomeados, como `ds` e `dm`, torna-se mais intuitivo compreender o significado de cada componente do *token*, reduzindo a ambiguidade e a necessidade de memorizar posições específicas, como ocorre nos conjuntos de cores do tipo `product`. Essa estrutura favorece a manutenção e a escalabilidade do modelo, especialmente em redes complexas, em que o número de campos e a reutilização de estruturas são frequentes.

Além disso, embora os conjuntos de cores `product` e `record` apresentem diversas semelhanças estruturais, a forma de acesso aos seus elementos é distinta. No caso de `record`, o acesso é mais intuitivo e se assemelha à utilização de `structs` na linguagem de programação C, onde os campos são acessados por identificadores nomeados, utilizando-se a notação `#`. Em contraste, no tipo `product`, o acesso é posicional, geralmente realizado por meio de funções como `proj_i`.

A modelagem desta rede no `CPN Tools`® está disponível no repositório: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.2.3 CONJUNTO DE CORES `UNION`

O conjunto de cores `union` permite combinar dois ou mais conjuntos de cores distintos em um único tipo. Em condições normais, cada lugar da rede aceita fichas de apenas um conjunto de cores. No entanto, a utilização de `union` supera essa limitação ao possibilitar que diferentes tipos de dados sejam associados a um mesmo lugar, proporcionando maior flexibilidade na modelagem de comportamentos heterogêneos.

A sintaxe geral para a declaração de um conjunto `union` é a seguinte:

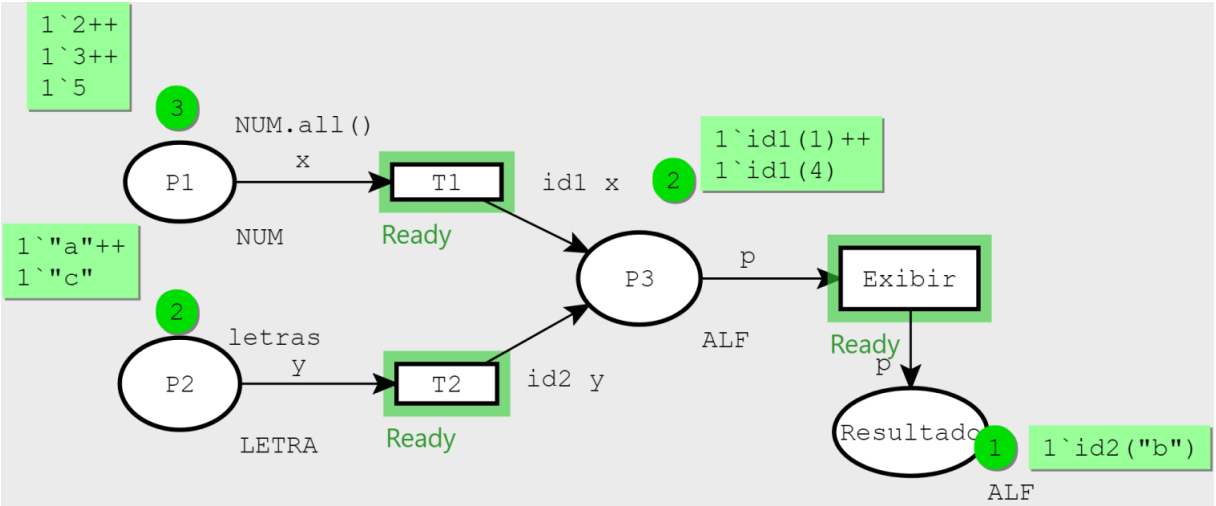
```
colset Nome = union id1[:Tipo1] + id2[:Tipo2] + ... + idn[:Tipon];
```

Cada `id` atua como um construtor que identifica qual tipo está sendo utilizado em uma determinada ficha. Caso o tipo (`tipo`) seja omitido, o identificador correspondente é tratado como um valor simbólico (constante), podendo ser referenciado diretamente por seu nome.

Operações de correspondência de padrões (*pattern matching*) podem ser utilizadas para identificar e extrair valores das fichas conforme o construtor utilizado, facilitando o tratamento seletivo de informações dentro das transições.

Na Figura 27 apresenta-se uma Rede de Petri Colorida que ilustra o evento de união entre dois conjuntos de fichas distintas. Perceba que a rede modela, na etapa de entrada, a união de fichas provenientes de tipos diferentes e, na etapa de saída, uma operação sobre fichas sem restrição de tipo. O lugar p_1 armazena fichas correspondentes a valores inteiros, enquanto o lugar p_2 contém fichas que representam caracteres. O lugar p_3 associa as fichas recebidas de p_1 e p_2, utilizando um conjunto de cores definido como `union`. Por fim, o lugar p_4 armazena as fichas unificadas em p_3, independentemente dos tipos originais das entradas em p_1 e p_2.

Figura 27 – Modelagem do conjunto de cores `union` no CPN IDE®



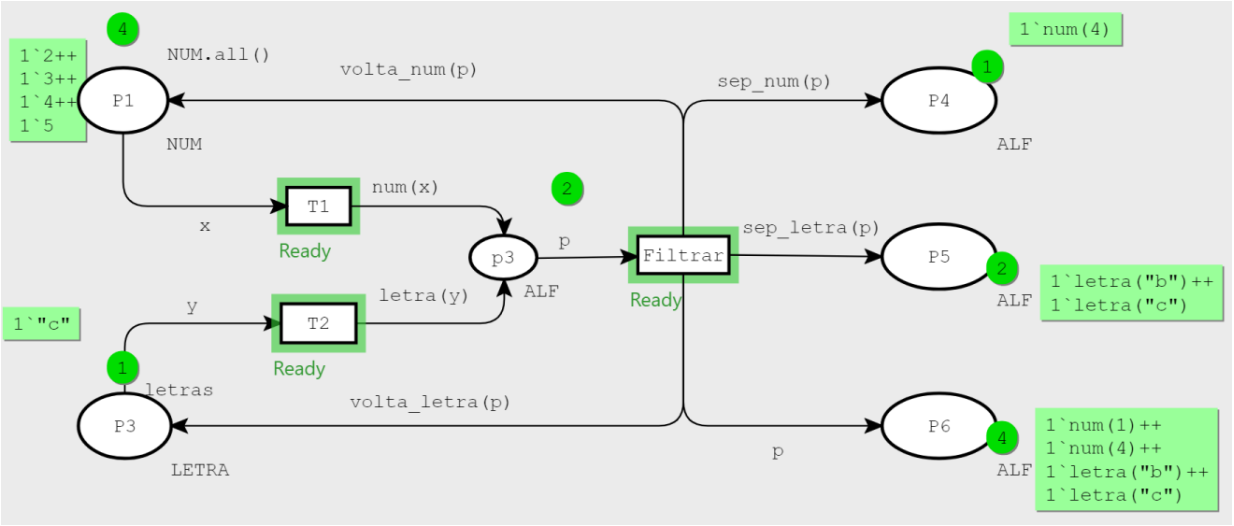
Fonte: Autor, (2025)

As seguintes declarações foram utilizadas na rede em questão.

```
colset NUM = int with 1..5;
colset LETRA = string;
colset ALF = union id1 : NUM + id2: LETRA;
var p : ALF;
var x : NUM;
var y : LETRA;
val letras = 1`"a"++ 1`"b"++1`"c";
```

Opcionalmente, funções de separação das fichas contidas em p_3 podem ser aplicadas utilizando-se uma sintaxe complementar. Desse modo a rede acima pode ser reconstruída, conforme apresentado na Figura 28.

Figura 28 – Modelagem do conjunto de cores `union` no CPN IDE® com funções complementares



Fonte: Autor, (2025)

Note que a nova rede inicia o processo com todas as fichas em p_1 e recebe retorno apenas das fichas com valores maiores que 2, enquanto p_2 recebe fichas apenas com valores diferentes da letra `"a"`.

As seguintes declarações foram utilizadas na rede em questão.

```
colset NUM = int with 1..5;
colset LETRA = string;
colset ALF = union num : NUM + letra: LETRA;
var p : ALF;
var x : NUM;
var y : LETRA;
val letras = 1`"a"++ 1`"b"++1`"c";
fun sep_num (num(x)) =
      if x>2 then 1`num(x) else empty | sep_num (letra(y)) = empty;
fun volta_num (num(x)) =
      if x>2 then 1`x else empty | volta_num (letra(y)) = empty;
fun sep_letra(letra(y)) =
      if y<>"a" then 1`letra(y) else empty | sep_letra(num(x)) = empty;
fun volta_letra(letra(y)) =
      if y<>"a" then 1`y else empty | sep_letra(num(x)) = empty;
```

Isto posto, considere que:

- O operador " | "entre as definições indica alternativas de *pattern matching*. Ou seja, cada cláusula define o comportamento da função para um formato específico de entrada.
- No caso da função `sep_num`, temos duas cláusulas:
  - Se o parâmetro é do tipo `id1(x)`, aplica-se a condição `if x > 2`, retornando a ficha `1`` id1(x)` quando verdadeira, ou `empty` quando falsa.
  - Se o parâmetro é do tipo `id2(y)`, a função simplesmente retorna `empty`.
- A função `sep_letra` é análoga, mas atua sobre o construtor `id2(y)` associado às letras. Se a letra for diferente de “a”, retorna `1`` id2(y)`; caso contrário, retorna `empty`.

- As funções `volta_num` e `volta_letra` realizam o processo inverso: a partir de um valor do tipo `ALF`, retornam diretamente o valor `x` (no caso de números) ou `y` (no caso de letras), respeitando as mesmas condições lógicas de filtragem.

A modelagem destas redes no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.2.4 CONJUNTO DE CORES `LIST`

Uma lista pode ser entendida como uma sequência de elementos pertencentes a um conjunto de cores previamente definido. As funções padrão oferecem meios simples de acessar diretamente o primeiro e o último elemento dessa lista. Entretanto, quando se deseja acessar elementos localizados no interior da sequência, é necessário utilizar funções recursivas. Esse processo consiste em percorrer a lista passo a passo, até encontrar a posição desejada, o que reforça a importância da recursão no tratamento de listas.

O conjunto de cores do tipo `list` é definido pela seguinte sintaxe:

```
colset nome = list nome0 [with int-exp1 .. int-exp2];
```

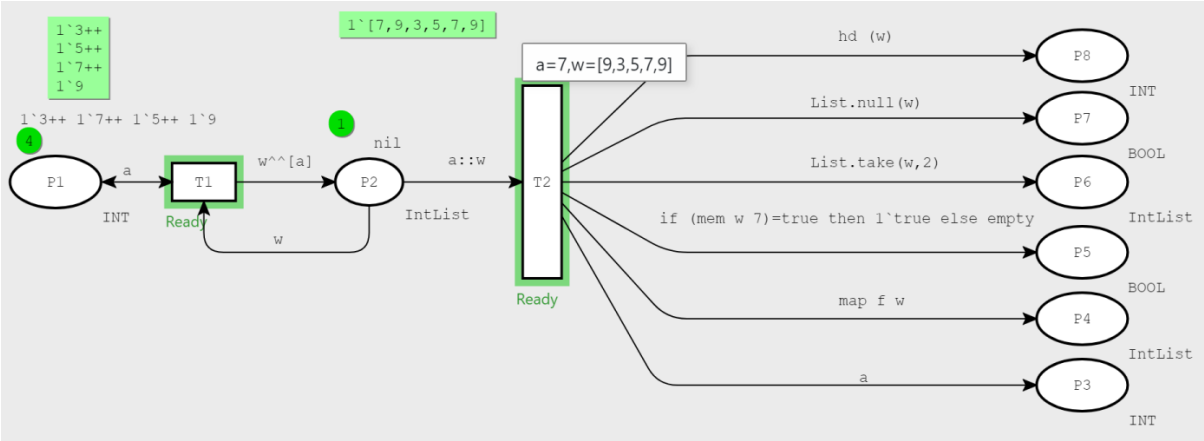
A cláusula `with` estabelece o tamanho mínimo (`int-exp1`) e o tamanho máximo (`int-exp2`) da lista. Os elementos de uma lista são organizados sequencialmente, obedecendo a essas restrições de comprimento. Além disso, existem diversas funções e operadores úteis para manipulação de listas, conforme apresentado a seguir:

- `nil` - lista vazia (o mesmo que `[]`)
- `e::l` - insere o elemento `e` no início da lista `l`
- `l1^^l2` - concatena as duas listas `l1` e `l2`
- `hd l` - primeiro elemento da lista `l`
- `tl l` - toda a lista `l`, exceto o primeiro elemento
- `length l` - retorna o tamanho da lista `l`
- `rev l` - retorna uma lista inversa à lista `l`
- `map f l` - aplica a função `f` em todos os elementos da lista `l`
- `mem l x` - retorna verdadeiro se `x` pertence à lista `l`
- `List.nth(l,n)` - retorna o n-ésimo elemento da lista `l`, onde $0 \leq n <$ `length l`
- `List.take(l,n)` - retorna os primeiros `n` elementos da lista `l`
- `List.drop(l,n)` - retorna o que resta na lista após remover os primeiros `n` elementos
- `List.exists p l` - retorna verdadeiro se `p` for verdadeiro para algum elemento da lista `l`
- `List.null l` - retorna verdadeiro se a lista `l` é vazia

O domínio dessas operações sobre listas é essencial, pois muitas vezes o comportamento de sistemas modelados por RPC depende diretamente da ordem de chegada, remoção ou manipulação de fichas, que podem ser representadas como elementos de listas. Normalmente, no disparo de uma transição em uma RPC, retiram-se fichas dos lugares de entrada da transição de forma aleatória, ou seja, a ligação entre as variáveis dos arcos e as fichas é feita de forma aleatória, desde que essa ligação habilite a transição. Entretanto, em alguns setores do conhecimento, tais como telecomunicações, linha de montagem, etc., existe um ordenamento de chegada e saída de informação ou produtos. Surge, então, a necessidade de políticas de prioridade. Algumas destas políticas são, por exemplo, a FIFO (*First In First Out*), ou primeira que chega é a primeira que sai; e a LIFO (*Last In First Out*), ou última que chega é a primeira que sai.

Isto posto, na Figura 29 é apresentado um exemplo de FIFO. Primeiro, valores inteiros são retirados do lugar p_1 de forma aleatória e armazenados por ordem de chegada em uma lista no lugar p_2 `(1`[ ])`, definido como uma lista vazia. Quando $t_2$ estiver habilitada, o disparo da mesma vai retirar sempre a ficha que chegou primeiro ao lugar $p_2$. Note, na figura em questão, após $t_2$ disparar duas vezes, as fichas `3` e `5` são retiradas consecutivamente de $p_2$, restando apenas a ficha `9`.

Figura 29 – Modelagem do conjunto de cores `list` no CPN IDE®



Fonte: Autor, (2025) adaptado de (BARROSO, 2006)

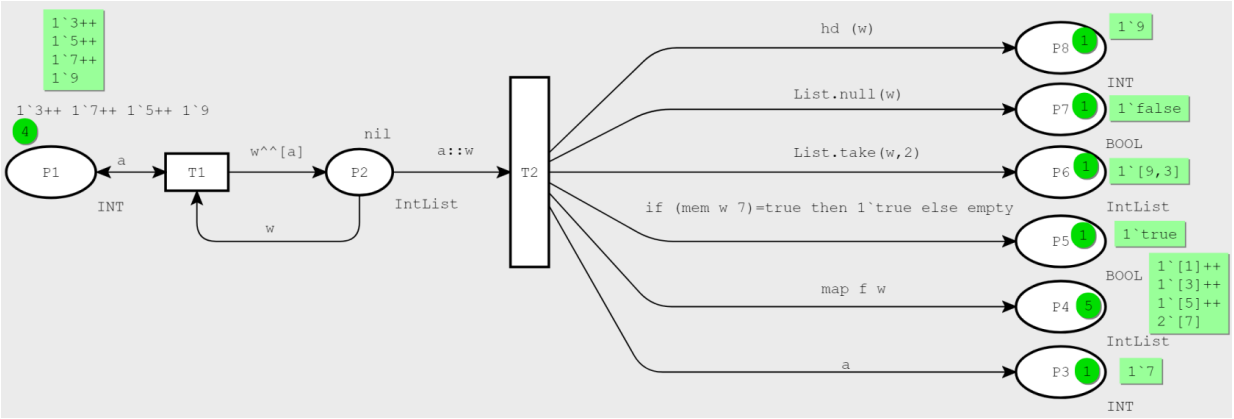
As seguintes declarações foram utilizadas na rede em questão.

```
colset INT = int ;
colset IntList = list INT with 1`[];
var a : INT;
var w : IntList;
```

Perceba, ainda na mesma figura, a existência uma moldura na transição t_2 contendo a seguinte informação `a=3,w=[5,9]`. Significa que ao disparar t_2 a ficha de valor 3 será removida para p_2, depois a ficha de valor 5 e assim sucessivamente, exatamente mantendo a ordem de chegada.

Adicionalmente, uma expansão da rede apresentada anteriormente é ilustrada na Figura 30, com o objetivo de explorar funcionalidades adicionais do conjunto de cores `list`.

Figura 30 – Modelagem do conjunto de cores `list` no CPN IDE®: rede complementar



Fonte: Autor (2025)

Perceba que os resultados seguem as definições contidas nos arcos de t_2 para p_i, com $i \in [3,7]$, a partir da marcação: `1`[7,9,3,5,7,9]`, contida em p_2, conforme Figura 29.

A modelagem destas redes no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.2.5 RESUMO: CONJUNTOS DE CORES COMPOSTAS

A Tabela 6 apresenta uma síntese dos conjuntos de cores compostas abordados nesta seção, proporcionando uma visão consolidada de suas características e aplicações práticas na modelagem de Redes de Petri Coloridas.

Tabela 6 – Resumo dos conjuntos de cores compostos

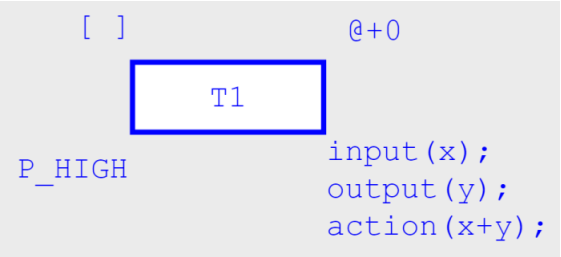
Tipo	**Definição / Sintaxe**	**Características Principais**
`product`	`colset N = product T1 * T2 * ... * Tn;`	Produto cartesiano de tipos; acesso posicional (`#i`); simples mas menos legível.
`record`	`colset N = record id1:T1 * id2:T2 * ... * idn:Tn;`	Campos nomeados; acesso por identificadores (`#id`); mais legibilidade, análogo a `struct` em C.
`union`	`colset N = union id1:T1 + id2:T2 + ...;`	Combina tipos heterogêneos; uso de construtores; permite *pattern matching* para filtragem seletiva.
`list`	`colset N = list T [with m..n];`	Sequência ordenada; acesso recursivo; suporte a funções padrão (`hd`, `tl`, `map`, etc.); modelagem de políticas FIFO/LIFO.

7.3 RECURSOS DAS TRANSIÇÕES NUMA REDE DE PETRI

Numa rede de Petri as transições modelagem um evento que ao ocorrer implica a mudança de estado de um sistema.

Na Figura 31 estão ilustradas as posições primárias dos elementos recursivos de uma transição.

Figura 31 – Elementos recursivos de uma transição no CPN IDE®



Fonte: Autor (2025)

Os elementos recursivos de uma transição são:

- Expressões de guarda - `[ ]`;
- Temporização da transição - `@+0`;
- Operadores `input()`, `output()`, `action()`;
- Prioridades de Transição - `P_HIGH`;

Tais elementos serão apresentados em modo detalhada a seguir.

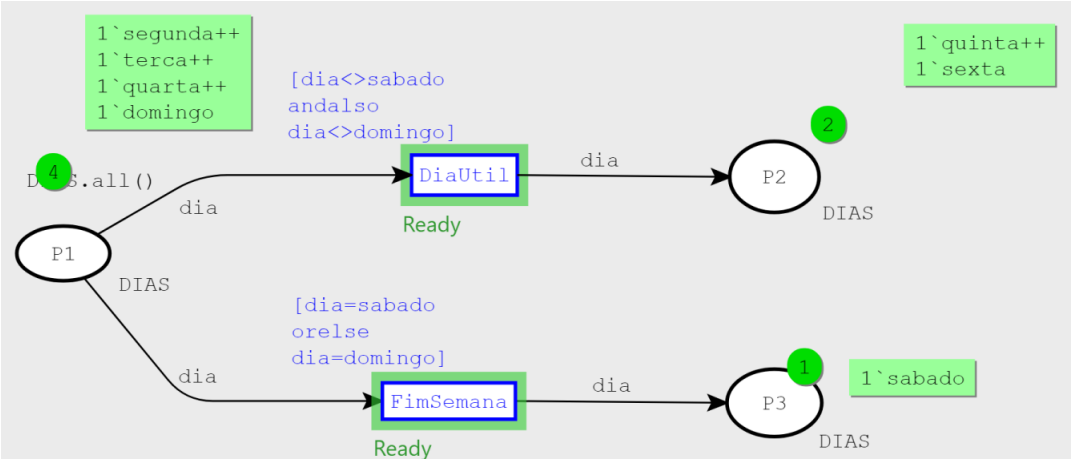
7.3.1 EXPRESSÃO DE GUARDA

A expressão de guarda é uma condição booleana associada a uma transição. Ela deve ser satisfeita para que a transição possa ser habilitada, além da presença suficiente de tokens nos lugares de entrada. No CPN IDE, a guarda é escrita após uma barra vertical (`|`) e pode utilizar variáveis vinculadas pelas expressões de arco.

Exemplo: `x > 0`

Na Figura 32 é utilizado diretamente uma expressão condicional do tipo `if` no campo de expressão de guarda da transição. Esse recurso permite realizar o filtro lógico no próprio corpo da transição, eliminando a necessidade de funções externas nos arcos, como `DiaUtil` e `FimSemana`. (Um modelo semelhante foi apresentado na Figura 23.) Desse modo, cada transição somente estará habilitada para disparado se satisfeita a condição da expressão de guarda em destaque na figura em questão.

Figura 32 – Modelagem com expressão de guarda no CPN IDE®



Fonte: Autor (2025)

Observe que a marcação atual da rede representa um estado arbitrário que pode ou não habilitar o disparo da transição, dependendo da avaliação da expressão de guarda.

As seguintes declarações foram utilizadas na rede em questão.

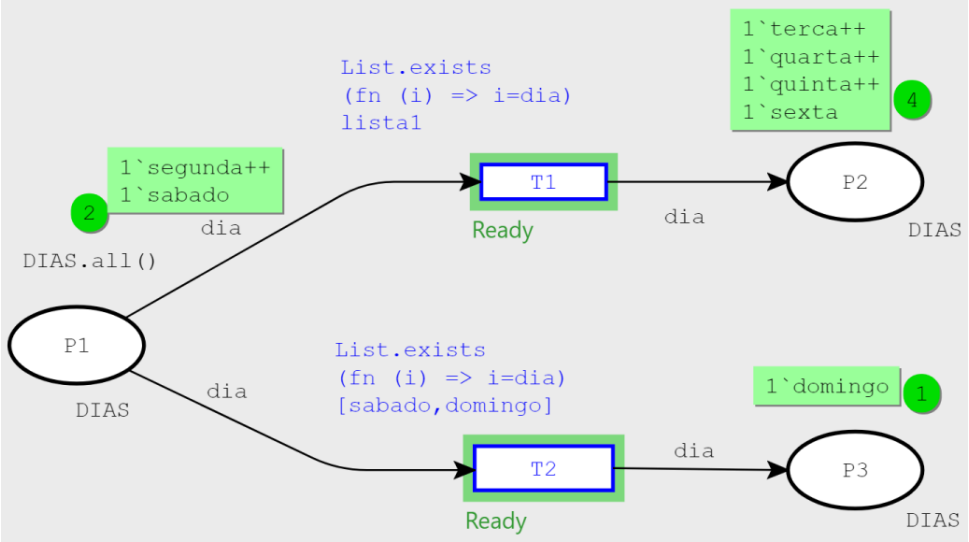
```
colset DIAS = with segunda | terca | quarta | quinta | sexta | sabado | domingo;
var dia : DIAS;
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

Expressão de Guarda com utilização de lista

A mesma rede apresentada anteriormente (Figura 32) é reproduzida na Figura 33, agora com uma construção alternativa da expressão de guarda. Nesta versão, utilizam-se estruturas do tipo **lista**, com o objetivo de ampliar a flexibilidade e expressividade do modelo.

Figura 33 – Modelagem com expressão de guarda no CPN IDE® com **lista**



Fonte: Autor (2025)

Perceba a expressão de guarda utilizada na transição $T1$:

```
List.exists (fn i => i = dia) lista1
```

Perceba que essa mesma expressão é utilizada na transição $T2$; entretanto, nesta última, a lista `lista1` foi declarada diretamente dentro da própria expressão de guarda, com o objetivo de demonstrar essa possibilidade como uma opção válida de modelagem. Essa construção alternativa se baseia na função de ordem superior `List.exists`, pertencente à biblioteca padrão da linguagem SML. A função `List.exists` recebe dois argumentos:

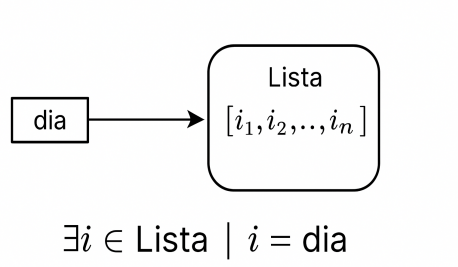
- uma função booleana (também chamada de predicado), que é aplicada a cada elemento da lista;
- e a própria lista de elementos a ser percorrida.

Neste caso, o predicado `fn i => i = dia` avalia, para cada elemento `i` da lista, se ele é igual ao valor atual da variável `dia`. A função `List.exists` retorna `true` se ao menos um elemento da lista satisfizer a condição $i = \text{dia}$, e `false` caso nenhum elemento corresponda. Formalmente, o predicado pode ser representado pela expressão da Equação 23:

$$\exists i \in \text{Lista} \mid i = \text{dia} \tag{23}$$

em que Lista representa o conjunto de elementos da lista $[i_1, i_2, \ldots, i_n]$ e `dia` é o valor atual da variável a ser verificada. A transição será habilitada sempre que existir pelo menos um elemento i na lista que satisfaça $i = \text{dia}$. Essa relação pode ainda ser visualizada de forma gráfica na Figura 34.

Figura 34 – Diagrama ilustrando a verificação de pertencimento de uma variável a uma lista



Fonte: Autor (2025)

Dessa forma, a expressão de guarda retorna verdadeiro sempre que o valor de `dia` estiver *contido* na lista especificada. Isso torna o modelo mais dinâmico, permitindo que diferentes condições de habilitação da transição sejam expressas de maneira compacta e reutilizável — seja referenciando uma lista declarada previamente (como em T_1), seja definindo a lista diretamente na expressão (como em T_2).

As seguintes declarações foram utilizadas na rede em questão.

```
colset DIAS = with segunda | terca | quarta | quinta | sexta | sabado | domingo;
var dia : DIAS;
val lista1 = [segunda, terca, quarta, quinta,sexta];
```

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.3.2 TEMPORIZAÇÃO EM REDES DE PETRI

A temporização é um recurso fundamental na modelagem de sistemas que envolvem restrições de tempo. No contexto das Redes de Petri Coloridas (RPC), existem duas abordagens distintas para lidar com aspectos temporais: as **Redes de Petri Temporais** e as **Redes de Petri Temporizadas**. Ambas podem ser modeladas no ambiente *CPN IDE*, porém com semânticas e objetivos diferentes.

7.3.2.1 Redes de Petri Temporais

Nas **Redes de Petri Temporais**, o tempo é associado diretamente às **transições**. Formalmente, cada transição $t \in T$ recebe um intervalo de disparo $I(t) = [t_{min}(t), t_{max}(t)]$, em que:

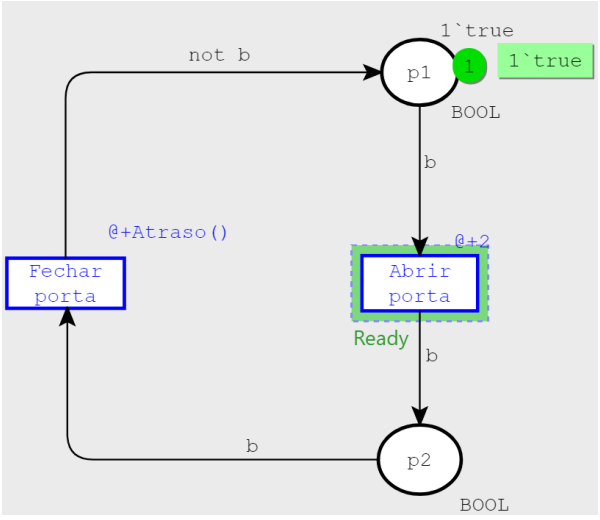
- $t_{min}(t)$ representa o tempo mínimo que deve transcorrer após a habilitação da transição antes que o disparo seja permitido;
- $t_{max}(t)$ representa o limite superior do intervalo de disparo. Se a transição permanecer habilitada até esse instante, o disparo pode ser forçado a ocorrer (semantica forte), ou apenas não ocorrer (semântica fraca)[7].

Esse formalismo é adequado para modelar sistemas com restrições temporais explícitas, como prazos, atrasos e *deadlines* associados ao disparo de eventos. Portanto, perceba que:

1. O disparo da transição só é concluído dentro da janela $[t_{min}, t_{max}]$ após sua habilitação.
2. As fichas permanecem nos lugares de entrada durante a espera temporal da transição.

Na Figura 35, apresenta-se uma RPC Temporal, construída no CPN IDE®. A modelagem baseia-se na rede funcional da Figura 20 apresentada anteriormente junto ao `colset bool`; contudo agora estendida com aspectos temporais a fim de representar com maior realismo o comportamento dinâmico do sistema de abertura e fechamento de uma porta.

Figura 35 – Modelagem de uma rede **temporal** no CPN IDE®



Fonte: Autor, (2025)

[7] A distinção é relevante em análises formais: a semântica forte assegura o cumprimento de prazos, enquanto a fraca modela situações em que a habilitação expira.

No exemplo da figura acima, a transição `Fechar porta` possui atraso definido por uma função aleatória, enquanto a transição `Abrir porta` possui inscrição temporal fixa `@+2`. As declarações a seguir configuram esse comportamento:

```
colset BOOL = bool;
colset Atraso = int with 1..10;
var b : BOOL;
fun Atraso() = Atraso.ran();
```

Note que, nesse modelo, as fichas são **atemporais** — não carregam carimbos de tempo (*timestamps*); toda a lógica temporal é incorporada às transições.

Conforme supracitado, em uma RPC Temporal cada transição $t \in T$ deve possuir um intervalo de disparo $I(t) = [t_{min}(t), t_{max}(t)]$, no qual t_{min} representa o atraso mínimo antes do disparo e t_{max} o limite máximo. A interpretação desse intervalo depende da semântica forte (forçada a disparar em t_{max}), ou fraca (disparo pode não ocorrer). Contudo, o *CPN IDE* não implementa de forma nativa esse formalismo completo com intervalos $[t_{min}, t_{max}]$. Essa ferramenta limita-se a inscrições temporais fixas (como `@+`τ, em que τ é um inteiro) ou a funções de atraso, aproximando apenas o conceito de t_{min}, mas sem suporte direto ao limite superior t_{max}. Na prática, isso significa que o ambiente aproxima a **semântica fraca**, já que não há mecanismo para forçar o disparo de uma transição no instante t_{max}.

Para contornar essa limitação, empregam-se técnicas alternativas que simulam a semântica temporal desejada, tais como:

- **Guardas condicionais temporais:** adiciona-se uma condição extra que verifica se o tempo atual atingiu t_{max}, forçando o disparo quando necessário.
- **Fichas de controle (*tokens* especiais):** utiliza-se *tokens* com atributos de tempo que indicam prazos máximos, permitindo que transições sejam habilitadas ou desabilitadas conforme a lógica temporal desejada.
- **Programação de atrasos escalonados:** modela-se o intervalo $[t_{min}, t_{max}]$ como uma sequência de transições intermediárias, cada uma com atraso incremental, de modo que o disparo final ocorra aproximadamente no instante t_{max}.
- **Eventos externos de sincronização:** integra-se o modelo com *triggers* (gatilhos) externos ou relógios globais que simulam *deadlines*(prazos), ativando transições no momento correto mesmo sem suporte nativo.
- **Funções de atraso adaptativas:** combina-se funções de atraso aleatórias com verificações de tempo em cada ciclo, ajustando dinamicamente a habilitação das transições conforme o limite superior desejado.

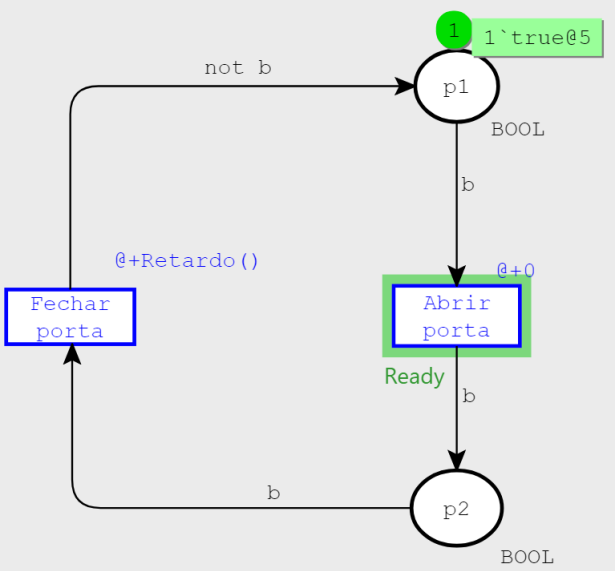
7.3.2.2 Redes de Petri Temporizadas

Diferentemente das redes **temporais**, nas quais o tempo está vinculado ao disparo das **transições**, as redes **temporizadas** associam o tempo diretamente às **fichas**. Nessa abordagem, cada ficha possui um carimbo temporal (*timestamp*) que indica o instante a partir do qual ela pode ser utilizada para habilitar uma transição. Em consequência, uma transição só poderá ocorrer se todas as fichas necessárias estiverem

presentes no tempo atual da simulação, e seus respectivos carimbos temporais forem menores ou iguais a esse tempo.

A Figura 36 ilustra um exemplo de rede de Petri colorida **temporizada**, modelada no ambiente CPN IDE®. O cenário representa a lógica de abertura e fechamento de uma porta, sendo que o comportamento temporizado é implementado por meio de fichas que carregam a informação sobre o tempo de validade. Essa modelagem permite representar eventos que ocorrem apenas após certo intervalo de tempo, sendo especialmente útil em contextos que envolvem atrasos de processamento, latência de comunicação ou ciclos internos de espera.

Figura 36 – Modelagem de uma rede temporizada no CPN IDE®



Fonte: Autor, (2025)

No modelo da Figura 36, a temporização é implementada a partir da definição de tipos de dados com suporte a tempo. A seguir, apresentam-se as declarações utilizadas:

```
colset BOOL = bool timed;
var b: BOOL;
fun Retardo() = round(normal(10.0 , 2.0 ));
```

A linha `colset BOOL = bool timed;` define o tipo de cor `BOOL` como temporizado, o que significa que todas as fichas desse tipo conterão um *timestamp*.

A anotação `(true, @+5)`, no lugar p_1, representa uma ficha com o valor lógico `true`, que se tornará válida apenas cinco unidades de tempo após ser produzida. A função `T_atraso()`, utilizada na rede anterior, foi aqui substituída pela função `Retardo() = round(normal(10.0,2.0))`[8] a fim de explorar o recurso de distribuição gaussiana.

Esse mecanismo de temporização nas fichas proporciona uma modelagem mais precisa de sistemas nos quais os eventos não ocorrem de forma imediata, mas dependem de intervalos temporais definidos. Trata-se de uma solução eficaz para representar atrasos inerentes a operações reais, sem a necessidade de manipular diretamente o tempo nas transições.

[8] **normal(10.0 , 2.0)**: A função $norma(\mu, \sigma)$ gera um número aleatório segundo uma distribuição normal (ou Gaussiana), com média $\mu = 10.0$ e desvio padrão $\sigma = 2.0$. Os valores gerados estarão, na maioria das vezes, próximos de 10.0, geralmente variando dentro do intervalo $(8, 12)$.

O modelo completo dessa rede está disponível no repositório: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

Resumo Comparativo

Na Tabela 7, apresenta-se uma comparação entre as redes de Petri **temporais** e **temporizadas**, destacando suas principais características. Essa distinção é essencial para a escolha do modelo mais adequado de acordo com a natureza temporal do sistema a ser representado. Enquanto as redes temporais vinculam o tempo ao comportamento das transições, as redes temporizadas incorporam a noção de tempo diretamente nas fichas, permitindo um controle mais refinado sobre a disponibilidade de informações ao longo da simulação.

Tabela 7 – Comparação entre redes temporais e temporizadas

Aspecto	**Redes Temporais**	**Redes Temporizadas**
Unidade temporizada	O tempo é associado às transições, que possuem atrasos definidos	O tempo é associado às fichas, que possuem carimbos temporais (*timestamps*)
Disparo da transição	Ocorre após o tempo mínimo definido para a transição, considerando o tempo global	Ocorre quando todas as fichas requeridas estão disponíveis no tempo atual da simulação
Comportamento das fichas	Fichas são atemporais; o atraso está no disparo da transição	Fichas permanecem nos lugares, mas só podem ser consumidas após seu *timestamp*
Aplicações típicas	Modelagem de tarefas com duração específica (ex: processos industriais ou *workflows*)	Modelagem de sistemas com disponibilidade futura de recursos ou eventos (ex: filas, redes logísticas)

7.3.2.3 Multiconjuntos com Restrições de Tempo

A noção de multiconjuntos com restrições de tempo amplia significativamente a expressividade das Redes de Petri Coloridas (RPC), ao permitir que cada ficha carregue não apenas um valor lógico ou estrutural, mas também um *timestamp* que regula sua disponibilidade no decorrer da simulação. Esse recurso é nativo do ambiente *CPN IDE*® e constitui a base formal para a modelagem de sistemas em que a ordem de ocorrência dos eventos depende explicitamente do tempo, como em arquiteturas embarcadas, protocolos de comunicação ou linhas de produção industriais.

Formalmente, um multiconjunto temporizado é definido como uma coleção de pares (c,t), em que $c \in C$ é uma cor pertencente ao conjunto de tipos e $t \in \mathbb{N}$ é o instante temporal associado à ficha. Durante a execução do modelo, uma ficha (c,t) permanece **invisível** para as transições até que o tempo global do simulador atinja o valor t. Apenas a partir desse instante a ficha torna-se elegível para compor a habilitação de uma transição.

Para declarar que um tipo de cor suporta tempo, utiliza-se o modificador `timed`. O exemplo a seguir define um conjunto de cores booleanas temporizadas:

```
colset TBool = bool timed;
```

Com isso, toda ficha do tipo `TBool` será automaticamente associada a um carimbo temporal. Uma ficha anotada como `1`‘`true@25` indica que o valor lógico `true` só poderá ser consumido a partir do instante 25 da simulação. Esse mecanismo permite que o tempo se propague de forma natural pelo

fluxo da rede, sem que seja necessário inserir atrasos diretamente nas transições, como ocorre nas redes temporais.

Operadores Temporais

O *CPN ML*, linguagem interna do *CPN IDE*, fornece operadores específicos para manipulação de multiconjuntos temporizados. Os mais relevantes são:

- `c@t` – associa explicitamente um tempo absoluto `t` (do tipo `Time.time`) à cor `c`;
- `ms @+ i` – adiciona um atraso relativo de `i` unidades de tempo a todas as fichas do multiconjunto `ms`;
- `tms1 +++ tms2` – realiza a união de dois multiconjuntos temporizados, preservando os respectivos carimbos de tempo.

Esses operadores tornam possível tanto a criação de fichas futuras, como também o escalonamento de eventos em diferentes instantes, característica crucial para modelagens que precisam representar paralelismo realista.

Exemplo de Uso

```
var f1, f2 : TBool;
f1 = 1`true@25;
f2 = 2`false@10;
```

Nesse exemplo:

- `f1` representa uma ficha com valor `true`, válida apenas a partir do tempo 25;
- `f2` representa duas fichas com valor `false`, válidas a partir do tempo 10.

Enquanto o tempo global da simulação for inferior aos valores especificados, as fichas permanecem "ocultas" no lugar correspondente, não contribuindo para a habilitação de transições. Essa característica é particularmente relevante para evitar disparos prematuros e sincronizar processos em cenários distribuídos.

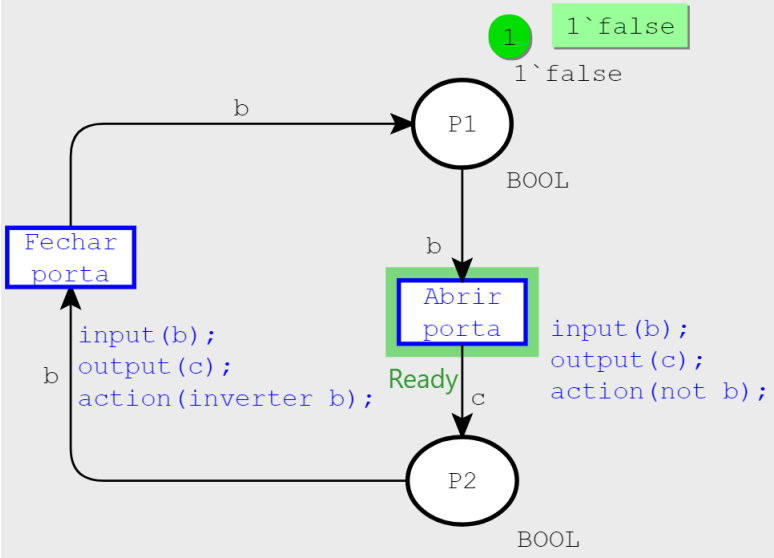
7.3.3 OPERADORES: `INPUT()`, `OUTPUT()`, `ACTION()`

Em CPN IDE®, o comportamento das transições é definido pelas seguintes categorias de operadores:

- `input()`: especifica as fichas que são consumidos dos lugares de entrada.
- `output()`: determina as fichas que serão produzidos nos lugares de saída.
- `action()`: define ações executadas no momento do disparo da transição, como imprimir dados ou atualizar variáveis internas. É executada após a retirada das fichas de entrada e antes da colocação das fichas de saída.

De forma semelhante à rede apresentada anteriormente na Figura 20, o modelo ilustrado na Figura 37, a seguir, utiliza diretamente os operadores `input()`, `output()` e `action()` na declaração da transição. Esse recurso substitui a necessidade de uma função externa, como `inverter b`, centralizando a lógica de manipulação de fichas na própria transição.

Figura 37 – Modelagem basilar com uso de expressão na transição `bool` no CPN IDE®



Fonte: Autor, (2025)

Perceba que a marcação atual da rede corresponde a um estado aleatório de disparo da transição. As seguintes declarações foram utilizadas na rede em questão.

```
colset BOOL = bool;
var b ,c: BOOL;
fun inverter b = not b;
```

Observe que a transição `Abrir porta` ainda faz uso da função `inverter b`, previamente definida de forma externa. Em contraste, a transição `Fechar porta` realiza a operação de inversão diretamente no campo `action()`, incorporando a lógica ao próprio corpo da transição. Essa abordagem demonstra maior flexibilidade e expressividade na modelagem, ao permitir encapsular comportamentos diretamente na estrutura da rede, favorecendo a reutilização e a recursividade.

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

7.3.3.1 Prioridade de Transições no CPN IDE®

O *CPN IDE*® permite definir prioridades entre transições para controlar qual deve disparar primeiro quando múltiplas estiverem habilitadas simultaneamente. Essa prioridade é expressa por um número inteiro associado à transição: **quanto menor o valor, maior a prioridade**.

Por padrão, todas as transições têm prioridade `0`. Para definir outra prioridade, basta clicar na transição, acessar as propriedades e alterar o campo `Priority`.

Regras de Disparo com Prioridade

- Apenas transições habilitadas com a **maior prioridade** (menor valor) são consideradas para disparo.
- Transições com mesma prioridade competem de forma *nondeterminística*.
- Transições com prioridade inferior são ignoradas enquanto houver outras de maior prioridade habilitadas.

Desse modo, considere o contexto das redes anteriores (figuras 32 - 36) que modelam um processo de abrir e fechar porta. Se a transição `fechar_porta` tiver prioridade `-1` e `abrir_porta` prioridade `0`, a primeira será sempre disparada primeiro quando ambas estiverem habilitadas. Contudo, cabe destacar que a utilização de prioridades deve ser feita com cautela, pois pode bloquear transições de menor prioridade, afetando o comportamento esperado da rede.

7.3.4 RESUMO DE RECURSOS DE TRANSIÇÕES

As transições em Redes de Petri no ambiente CPN IDE® possuem diversos recursos que ampliam a expressividade do modelo, permitindo a definição de condições lógicas, temporizações, operadores internos e prioridades de disparo. Na Tabela 8 é apresentado um resumo conciso desses principais recursos, servindo como referência rápida para a modelagem e análise de sistemas.

Tabela 8 – Resumo dos Recursos de Transição em Redes de Petri (CPN IDE®)

Recurso	**Descrição**
Expressão de Guarda	Condição booleana que habilita a transição apenas quando satisfeita, podendo usar variáveis de arco e funções SML (`x>0`, `List.exists`).
Temporização	Define comportamento temporal da rede: **Temporal** – atraso associado à transição (`@+n`); **Temporizada** – fichas com *timestamp* que habilitam disparo somente após tempo válido.
Operadores `input()`, `output()`, `action()`	`input()`: fichas consumidas; `output()`: fichas produzidas; `action()`: ações no disparo (ex.: inversão lógica, impressão).
Prioridade de Transição	Inteiro que define ordem de disparo; menor valor = maior prioridade. Transições com mesma prioridade disparam de forma não determinística.

7.4 REDES HIERÁRQUICAS NO CPN IDE®

No CPN IDE®, a hierarquia de uma RPC é estabelecida através de *sub-páginas*, também chamadas de *subnets*. Cada transição em uma página de nível superior pode ser associada a uma sub-página que detalha o comportamento interno daquela etapa do processo. Essa estrutura modular facilita a:

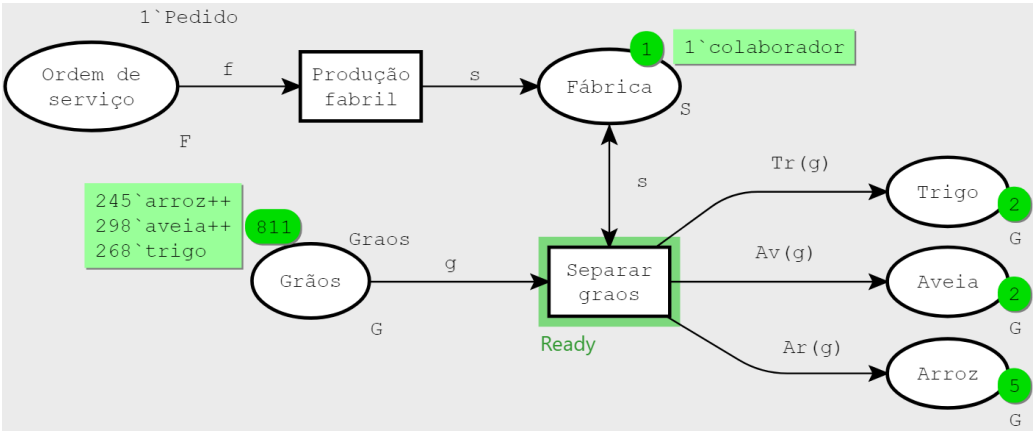
- **Organização**: separando subsistemas em páginas independentes, com interfaces de entrada e saída bem definidas.
- **Reutilização**: permitindo replicar sub-redes em diferentes partes do modelo sem redefinir a lógica interna.
- **Análise Formal**: cada sub-página pode ser verificada isoladamente antes de ser integrada ao modelo completo.

A interface do CPN IDE apresenta um navegador hierárquico em árvore, no qual cada nível corresponde a uma página ou sub-página. A ligação entre uma transição e a sua sub-rede é feita por meio de *substitution transitions*, que funcionam como portas de entrada para níveis mais detalhados. Lugares de entrada e saída da sub-página devem ser mapeados para os lugares correspondentes da transição de nível superior, garantindo consistência de *tokens* e tipos de cores (dados).

7.4.1 REDE HIERÁRQUICA SIMPLES

Considere a RPC ilustrada na Figura 38.

Figura 38 – Rede simples sem hierarquia no CPN IDE®

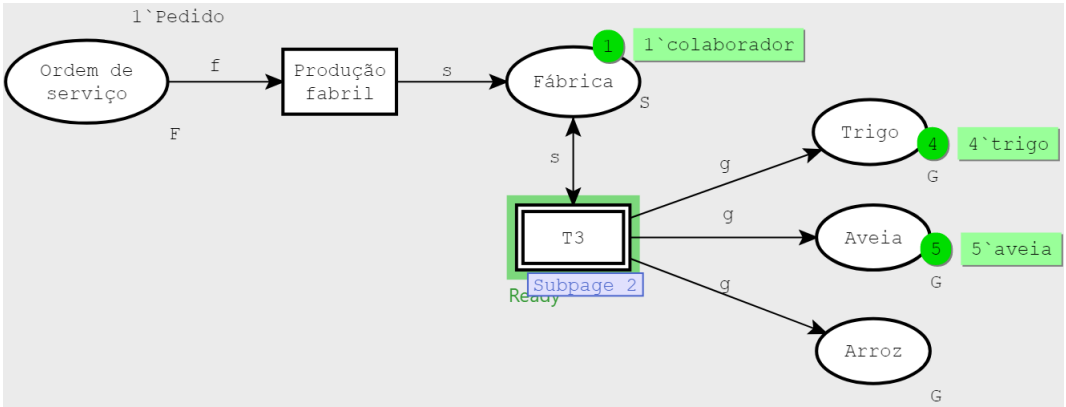


Fonte: Autor (2025)

A rede modela uma etapa de processamento fabril dedicada à separação de 820 grãos inicialmente armazenados no lugar `Grãos`. A operação é disparada por uma ordem de serviço da fábrica, e o fluxo de transições garante que os lugares `Trigo`, `Aveia` e `Arroz` recebam exclusivamente os grãos correspondentes a cada tipo.

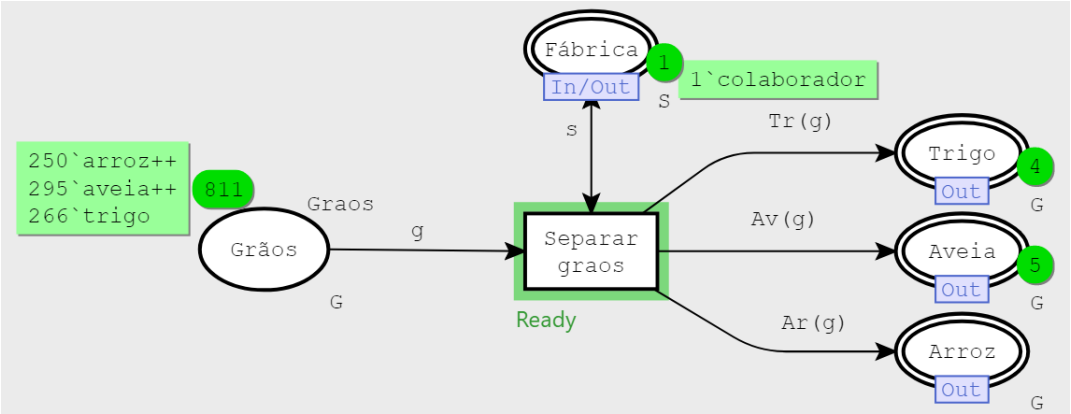
Embora funcional, este modelo pode ser reorganizado de forma mais modular, encapsulando o processo de separação em uma sub-rede sem perda de comportamento ou rastreabilidade. Na Figura 39 é apresentado o nível principal da rede hierárquica, enquanto na Figura 40 é detalhada a subpágina correspondente.

Figura 39 – Rede hierárquica no CPN IDE®: nível principal



Fonte: Autor (2025)

Figura 40 – Rede hierárquica no CPN IDE®: nível de subpágina



Fonte: Autor (2025)

A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

Configuração da Sub-página Hierárquica no CPN IDE

A organização hierárquica em redes de Petri coloridas requer a definição explícita dos pontos de interação entre diferentes níveis de modelagem por página.

Na sub-página, Figura 40, os lugares que estabelecem comunicação com a página principal são caracterizados por tipos de porta (`Port type`) que especificam a direção do fluxo de fichas. Essas opções de porta tornam-se disponíveis no menu de propriedades sempre que um lugar ou arco é selecionado, permitindo a definição individual de cada elemento. As portas podem ser classificadas como `in`, `out` ou `in/out`, correspondendo, respectivamente, a entradas, saídas ou conexões bidirecionais de fichas. Essa categorização confere ao modelo a capacidade de modularizar subprocessos, garantindo a coerência do intercâmbio de informações entre os níveis hierárquicos.

No nível superior da rede, Figura 39, a correspondência entre os arcos da página principal e as portas da sub-rede é formalizada por meio do mecanismo `Port Bind`. Essa vinculação estabelece o mapeamento semântico entre os elementos das duas camadas, assegurando a consistência do comportamento global do sistema e preservando as propriedades de análise típicas das redes de Petri. Dessa forma, a hierarquização não apenas organiza o modelo, mas também reforça a possibilidade de verificação formal e reutilização de componentes em sistemas de maior complexidade. Isto posto, cumpre destacar que a sub-rede pode assumir desde a simplicidade do exemplo apresentado até graus elevados de sofisticação, conforme as especificidades do subprocesso a ser representado.

8 Aplicações e proposições de Modelagem

As Redes de Petri são apropriadas para modelar sistemas a eventos discretos por suas capacidades de representar concorrência, sincronização e conflitos; modelar causalidade entre eventos; simular e verificar propriedades como vivacidade e ausência de deadlocks; serem estendidas com cores e temporização; além de oferecerem suporte à modularidade e à análise formal de sistemas.

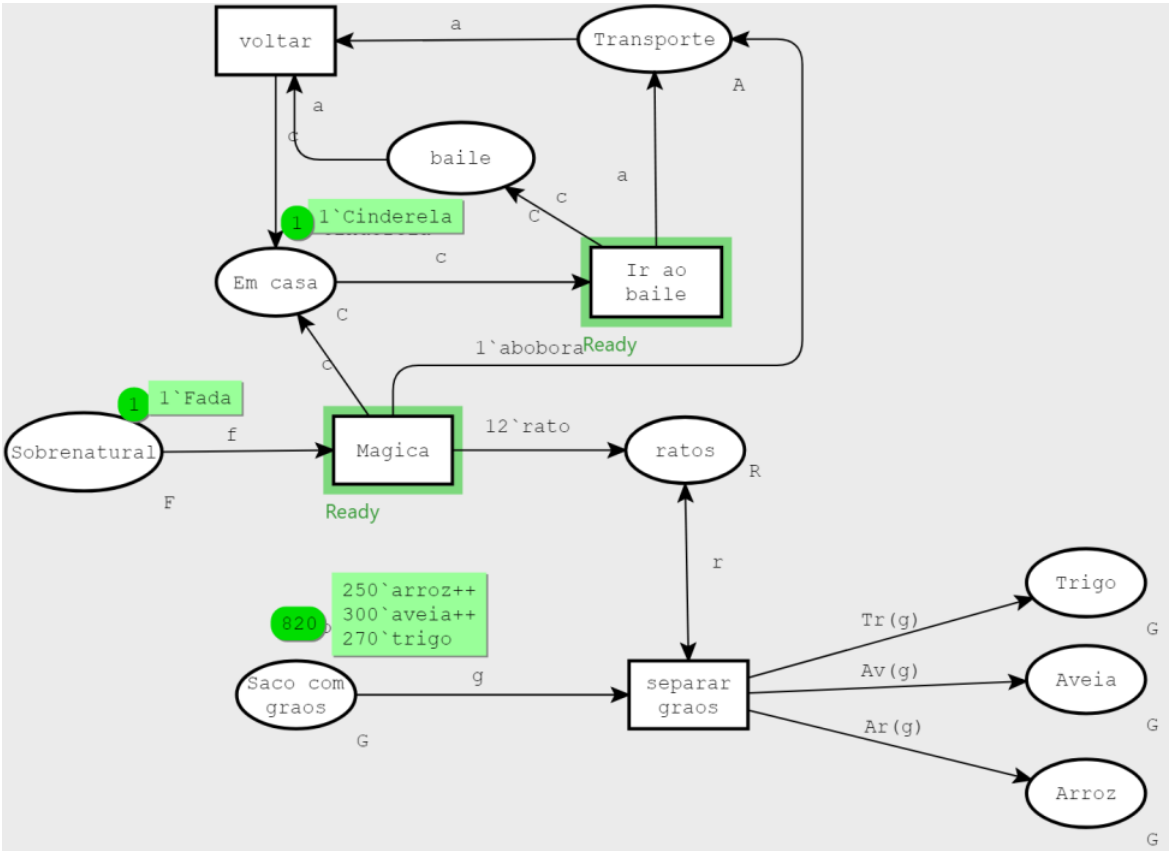
8.1 MODELO DE REDE: CONTO DA CINDERELA

O exemplo a seguir apresenta a modelagem de uma situação clássica da literatura infantil em RP, adaptada de Zaitsev (2006, *apud* (BARROSO, 2006)).

O CASO CINDERELA

A madrasta de Cinderela mandou que ela separasse grãos de diferentes tipos, mas no exemplo, camundongos amigos de Cinderela separam os grãos enquanto ela vai ao baile. Este exemplo está ilustrada na Figura 41.

Figura 41 – Modelagem de caso Cinderela no CPN IDE®



Fonte: Autor, (2025) adaptado de (BARROSO, 2006)

Neste modelo de rede de Petri, as seguintes declarações de conjuntos de cores e variáveis foram utilizadas:

```
colset A = unit with abobora;
colset C = unit with Cinderela;
colset G = with arroz | aveia | trigo;
colset R = unit with rato;
colset F = unit with Fada;
var a : A;
var c : C;
var g : G;
var r : R;
var f : F;
fun Tr(g) = if (g=trigo) then 1`trigo else empty;
fun Av(g) = if (g=aveia) then 1`aveia else empty;
fun Ar(g) = if (g=arroz) then 1`arroz else empty;
val graos = 250`arroz++300`aveia++270`trigo;
```

Nesta modelagem, foram definidos cinco conjuntos de cores: **A**, contendo a ficha *abóbora*; **C**, com a ficha *Cinderela*; **G**, composto por três fichas (*arroz*, *aveia* e *trigo*); **R**, com a ficha *rato*; e **F**, com a ficha *Fada*. Note que inicialmente, apenas a transição *Mágica* está habilitada, destacada por uma moldura na cor verde. Ao conversar com *Cinderela*, a fada cria doze ratos, uma abóbora e desaparece, acionando a transição *Mágica*. Note também que a viagem de *Cinderela* até o baile e a separação dos grãos ocorrem de forma concorrente, podendo acontecer simultaneamente e em qualquer ordem. A abóbora funciona como um recurso essencial para habilitar e disparar as transições *Ir ao baile* e *Voltar*, que levam *Cinderela* ao baile e a trazem de volta. Para isso, utiliza-se um auto-laço, representado por um arco bidirecional conectando o lugar *Transporte* à transição *Ir ao baile* (Figura 41). Os ratos, por sua vez, servem como recursos para habilitar e disparar a transição *Separar* os tipos de grãos.

Considerando o sentido dos arcos e suas respectivas inscrições, perceba que o disparo da transição *Mágica* não altera a marcação do lugar *Em casa*, que contém uma ficha de cor *Cinderela*. Da mesma forma, o disparo da transição *Ir ao baile* não modifica a marcação do lugar *Transporte* (ficha de cor *Abóbora*), pois esse mesmo recurso é utilizado para trazer *Cinderela* de volta para casa ao acionar a transição *Voltar*.

Note que os demais arcos são unidirecionais. Desse modo, um arco que conecta um lugar a uma transição indica que, no momento do disparo, uma ou mais fichas serão removidas do lugar, conforme definido pela inscrição do arco de entrada da transição. No exemplo, todas as inscrições são variáveis pertencentes aos respectivos conjuntos de cores.

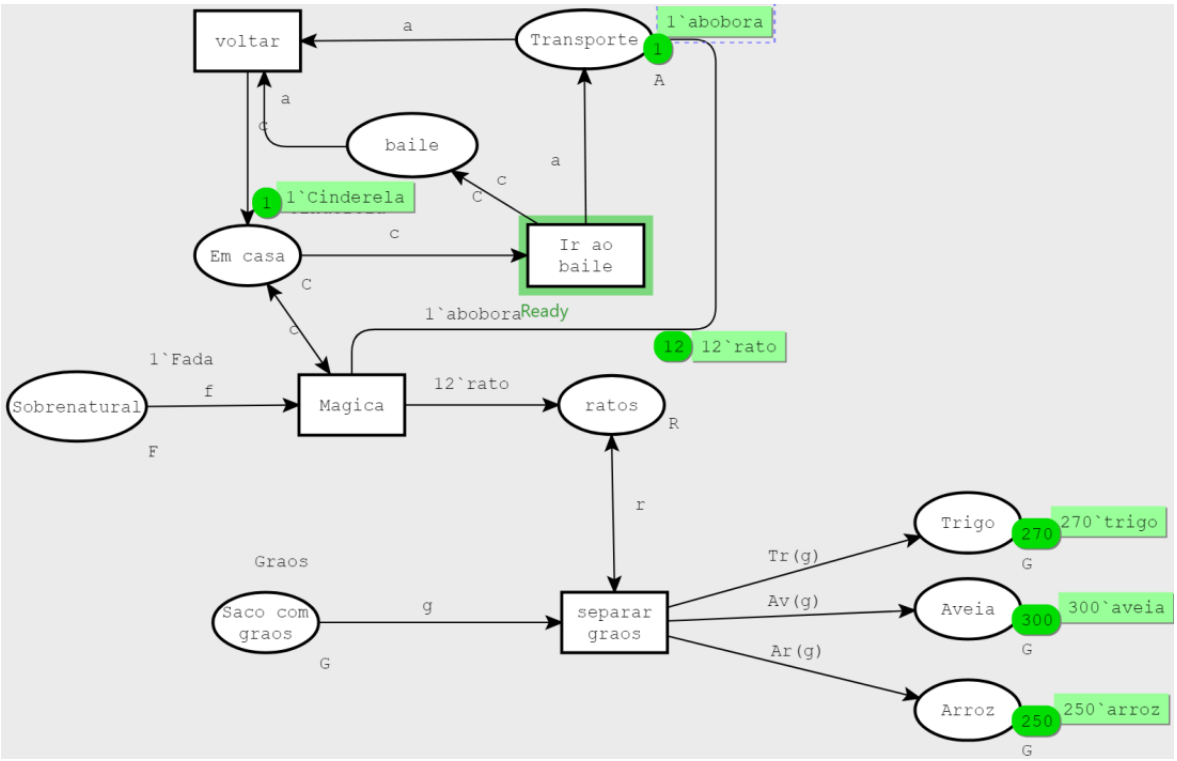
Como ilustração disso, perceba que o arco de entrada da transição *Separar grão* possui a inscrição *g*, que representa uma variável do conjunto de cores **G**. Assim, quando a transição *Separar* é disparada, um grão é retirado aleatoriamente do lugar *Saco com grãos*. Inscrições de arco mais complexas serão exploradas nas próximas seções. Os arcos de saída das transições, ao serem disparadas, geram novas fichas, que podem ou não coincidir com as retiradas dos lugares de entrada. Isso significa que novas fichas podem ser criadas no sistema. Outro modo, no disparo da transição *Ir ao baile*, a ficha *Cinderela* é removida do lugar *Em casa* por meio da variável *c*, associada ao arco que conecta esse lugar à transição.

Em seguida, uma ficha de mesmo valor (*Cinderela*) é inserida no lugar *Baile*, conforme a inscrição *c* no arco que liga a transição *Ir ao baile* a esse lugar.

Finalmente, a transição *Mágica*, por sua vez, possui uma lógica mais complexa. Ao ser disparada, a ficha *Fada* é removida do lugar *Sobrenatural* pela variável f e desaparece, pois essa variável não está associada a nenhum arco de saída da transição. Entretanto, a marcação do lugar *Em casa* permanece inalterada devido ao auto-laço (arco bidirecional) associado à variável *c*.

Como resultado do disparo dessa transição, são criadas doze fichas *Rato* e uma ficha *Abóbora*, conforme as inscrições especificadas nos arcos de saída. Veja na Figura 42 a marcação final do modelo após 823 passos de simulação.

Figura 42 – Modelagem de caso Cinderela - Resultado da simulação no CPN IDE®



Fonte: Autor, (2025) adaptado de (BARROSO, 2006)

Nesta marcação, Cinderela já retornou do baile e está em casa. A fada desapareceu, e seus amigos ratos finalizaram a separação dos grãos, distribuindo 270 grãos de trigo, 300 de arroz e 250 de aveia em seus respectivos lugares. Note que o disparo **separar grão** retira aleatoriamente uma ficha do saco de grãos por meio da variável *g*. No entanto, esse grão só pode ser colocado em um único destino entre os lugares de saída da transição (arroz, aveia ou trigo). Os arcos de saída da transição possuem funções associadas que garantem a seleção apenas do grão correspondente ou de uma ficha especial chamada *empty*. Quando uma ficha *empty* é escolhida, isso significa que nenhum grão será colocado no lugar de saída da transição.

O modelo RPC estudado não contempla diversas características do conto de fadas. Por exemplo, o tempo não é considerado, o que significa que o aviso da Fada para Cinderela sobre a meia-noite não está representado no modelo. Por isso, recomenda-se uma análise detalhada para identificar outras características relevantes do conto. Após um estudo mais aprofundado do CPN IDE®, pode-se construir

um modelo que represente de forma mais fiel a história original. A modelagem desta rede no CPN IDE® está disponível no endereço: <https://github.com/JonathaCosta-IA/MDES/tree/main/B-MDES_Nets/Nets_handbook>.

8.2 EXERCÍCIOS: SISTEMAS A SEREM MODELADOS

A seguir, apresenta-se uma lista de vinte exemplos de Sistemas a Eventos Discretos (MSED) que podem ser modelados com Redes de Petri no ambiente CPN IDE. Para cada sistema, é indicada a abordagem mais adequada: Rede de Petri Lugar-Transiçao (RP-LT) ou Rede de Petri Colorida (RPC), considerando a complexidade e a necessidade de manipulação de dados, contudo ambos podem ser ajustados para a serem modelados em ambos as abordagens de Rede de Petri, incluindo funções avançados.

1. **Semáforo veicular com controle de tráfego** — RP-LT: permite modelar a alternância entre luzes e sincronização entre cruzamentos.
2. **Linha de produção com múltiplas estações** — RPC: modela tipos de peças, rotas e estados de máquinas.
3. **Sistema de filas em banco ou hospital** — RPC: representa clientes com atributos como prioridade ou tipo de atendimento.
4. **Elevador com múltiplos andares** — RPC: manipula requisições por andar e direção com tokens estruturados.
5. **Máquina de venda automática** — RP-LT: simples lógica de inserção de moeda, escolha e entrega.
6. **Controle de acesso com cartão magnético** — RPC: uso de tokens com campos de usuário, permissões e horários.
7. **Robô industrial em célula de manufatura** — RPC: modela tarefas específicas por tipo de peça ou operação.
8. **Protocolo de comunicação entre dois processos** — RP-LT: sincronização de envio e recebimento de mensagens.
9. **Impressora compartilhada por usuários** — RPC: representa trabalhos com atributos como usuário, páginas, prioridade.
10. **Sistema de embarque em metrô** — RP-LT: controle de abertura/fechamento de portas e fluxo de passageiros.
11. **Processo de login e autenticação de usuário** — RPC: modela tentativas, credenciais e tempo de bloqueio.
12. **Distribuição de tarefas em servidores** — RPC: alocação dinâmica de tarefas com características distintas.
13. **Estoque e reposição em almoxarifado** — RPC: itens com quantidade, validade e localização.

14. **Controle de tráfego aéreo em aeroporto** — RPC: voos com atributos como origem, destino, prioridade e pista.
15. **Sistema de transporte público com baldeações** — RPC: passageiros com rotas distintas e pontos de troca.
16. **Lavanderia industrial com múltiplas etapas** — RP-LT: sequência fixa de lavagens, secagens e dobras.
17. **Atendimento em call center com prioridades** — RPC: chamadas com níveis de urgência e tempo de espera.
18. **Gerenciamento de pedidos em restaurante** — RPC: pedidos com itens, mesa e tempo de preparo.
19. **Caixa eletrônico com múltiplas operações** — RPC: operações distintas por cliente (saque, saldo, depósito).
20. **Sistema de alarme com múltiplos sensores** — RP-LT: combinação de sinais para ativação ou desativação do alarme.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDERSON, D. **Produção em Massa e a Evolução da Manufatura**. [S.l.]: Editora XY, 2015.

BARROSO, J. M. S. G. C. **Introdução às Redes de Petri Coloridas usando a ferramenta CPN Tools**. 2006. Acesso em: 11 mar. 2025. Disponível em: <https://www.scribd.com/document/496849470/redespetritutorialcpntools-160625125824>.

BASKIN, D. **Sistemas Flexíveis de Manufatura**. [S.l.]: Editora Flex, 2017.

BRUNDTLAND, G. H. **Nosso Futuro Comum**. [S.l.]: Editora Mundial, 1987.

CARDOSO, J.; VALETTE, R. **Redes de Petri**. 1. ed. São Paulo: EdUSP, 2007. Tradução de Valette, Robert. Original: Les Réseaux de Petri. ISBN 9788531409685.

CASSANDRAS, C. **Introduction to Discrete Event Systems**. [S.l.]: Springer, 2008.

CHASE, R. B.; AQUILANO, N. J. **Gestão de Operações: Produção e Cadeia de Suprimentos**. [S.l.]: Editora McGraw-Hill, 2006.

HARRIS, F. **Gestão da Produção e Operações**. [S.l.]: Editora Pearson, 2002.

JONES, M. Indústria 4.0: O futuro da manufatura. **Revista de Tecnologia**, v. 15, n. 2, p. 123–135, 2018.

MELLOR, P. **Indústria 4.0: A Revolução Digital**. [S.l.]: Editora Tech, 2014.

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

SAP. **What is Industry 4.0?** 2025. Acesso em: 11 mar. 2025. Disponível em: <https://www.sap.com/products/scm/industry-4-0/what-is-industry-4-0.html>.

SMITH, J. **História da Manufatura**. [S.l.]: Editora ABC, 2010.

THOMPSON, D. **Inovação e Competitividade na Manufatura Global**. [S.l.]: Springer, 2019.

WILSON, J. M. **Planejamento e Controle da Produção**. [S.l.]: Editora Cengage Learning, 2016.