DEPTO. DE ENG. DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

UNIVERSIDADE ESTADUAL DE CAMPINAS

Relatório Técnico
DCA - 001/2000

CAV (*C*oordenação em *A*mbientes *V*irtuais)

Uma biblioteca para o planejamento de animações e interações em ambientes virtuais colaborativos

Alberto Barbosa Raposo
Léo Pini Magalhães
Ivan L. M. Ricarte

Universidade Estadual de Campinas (UNICAMP)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Depto. de Engenharia de Computação e Automação Industrial (DCA)
C.P. 6101 - 13083-970 - Campinas, SP, Brasil
Tel.: +55 - 19 - 788-3700 - Fax: +55 - 19 - 289-1395
alberto, leopini, ricarte@dca.fee.unicamp.br

Abril 2000

Abstract

Virtual environments are interactive simulations of real or imaginary worlds. The development of virtual environments has been essentially directed to leisure activities, enabling basically the communication among remote users. In order to be effectively used as collaborative work tools, the developers of these environments should invest, among other aspects, in the coordination of the activities. The goal of this work is to model a library of coordination mechanisms (CAV – Coordination in Virtual Environments) that may be reused in different implementations of collaborative virtual environments. The proposed coordination mechanisms are modeled using an approach based on Petri nets and workflow nets.

Resumo

Os ambientes virtuais são simulações interativas de mundos reais ou imaginários. O desenvolvimento de ambientes virtuais tem sido essencialmente voltado para atividades de "lazer", permitindo basicamente que usuários se comuniquem remotamente. Para que estes ambientes possam ser efetivamente utilizados como ferramenta de trabalho colaborativo é necessário, dentre outros aspectos, investir na coordenação das atividades. O objetivo deste trabalho é modelar uma biblioteca de mecanismos de coordenação (CAV - *C*oordenação em *A*mbientes *V*irtuais) reutilizáveis em diferentes implementações de ambientes virtuais colaborativos. Para modelar os mecanismos de coordenação propostos, é usada uma abordagem baseada em redes de Petri e redes de *workflow.*

1. Introdução

O avanço da capacidade de processamento dos computadores e o aumento da capacidade de transmissão das redes permitiram a criação de ambientes virtuais (VEs – *Virtual Environments*), onde vários usuários podem estar simultaneamente presentes em uma simulação de um mundo tridimensional.

No entanto, a maior parte das aplicações de VEs se enquadra na categoria de "relações sociais", regidas unicamente pelo protocolo social, que é caracterizado pela ausência de qualquer mecanismo de coordenação entre as atividades, confiando na capacidade dos participantes de mediarem as interações. Há ainda poucos avanços no sentido de prover ambientes virtuais para dar suporte ao trabalho colaborativo (os chamados CVEs - *Collaborative Virtual Environments*). Exemplos de atividades auxiliadas por computador que utilizam o protocolo social são os *chats* e as videoconferências. Assim como no mundo real, os participantes destas atividades conseguem interagir eficientemente sem o auxílio de qualquer mecanismo de planejamento ou coordenação das atividades (cada um sabe quando deve falar, ficar em silêncio, etc). Entretanto, a realização em CVEs de tarefas que exigem algum grau de planejamento e coordenação ainda é bastante difícil, pois exige um grande esforço de programação por parte do projetista do ambiente.

O objetivo deste trabalho é modelar uma biblioteca de mecanismos de coordenação (CAV - *C*oordenação em *A*mbientes *V*irtuais) reutilizáveis em diferentes implementações de CVEs. A idéia é separar os mecanismos de coordenação dos elementos que constituem o ambiente. Este paradigma é utilizado na SYNOPSIS [Dellarocas 96], que é uma linguagem para a descrição de arquitetura de sistemas provendo duas abstrações ortogonais: atividades, que representam as peças funcionais de uma aplicação, e dependências, que descrevem as relações de interconexão entre as atividades. Este paradigma é aqui adaptado aos CVEs, considerando as peças funcionais como sendo as tarefas da colaboração e criando mecanismos de coordenação apropriados ao modelo de interação dos CVEs. A separação entre atividades e dependências permite o reuso de arquiteturas de software e de componentes em nível de código (na SYNOPIS). No caso de CVEs, esta separação permite várias formas de colaboração em um mesmo ambiente, variando apenas os mecanismos de coordenação (flexibilidade).

No presente trabalho é usada uma abordagem baseada em redes de Petri e redes de *workflow* [van der Aalst 94] para modelar os mecanismos de coordenação propostos. A representação gráfica das redes de Petri, além de ser simples, permite encapsular detalhes e oferece um modelo de descrição hierárquico adequado para modelar os diferentes níveis de coordenação desejados. Além disso, as redes de Petri oferecem um forte suporte teórico para a análise do comportamento do ambiente e técnicas de simulação complementares. Com isso, é possível prever e testar o comportamento de processos antes de sua efetivação real. Um exemplo de utilização deste mecanismo pode ser encontrado em [Magalhães 98b] para animações e aqui ele será estudado para a análise do comportamento de um ambiente virtual antes de implementá-lo.

A seção seguinte apresenta o conceito de CVE, e a Seção 3 analisa, sob a óptica da animação por computador, o impacto dos ambientes virtuais, dando uma idéia dos campos de aplicação da biblioteca proposta. A Seção 4 introduz as redes de Petri. A Seção 5 faz uma revisão dos trabalhos relacionados e apresenta o conceito de redes de *workflow*, usado nos mecanismos de coordenação da CAV, que são apresentados na

Seção 6. Alguns exemplos de utilização da CAV são vistos na Seção 7 e as contribuições do trabalho são resumidas na Seção 8. A Seção 9 apresenta a conclusão.

2. CVEs - *Collaborative Virtual Environments*

Os CVEs são definidos como "simulações em tempo real de mundos reais ou imaginários, onde os usuários estão simultaneamente presentes e podem navegar e interagir com objetos e outros usuários" [Hagsand 96]. As seguintes características definem um CVE [Waters 97]:

- Permite que um grupo de usuários separados geograficamente interaja em tempo real.
- É tridimensional para os olhos e ouvidos, ou seja, movimentos no ambiente mudam a perspectiva visual e auditiva do usuário.
- Usuários são representados como objetos do ambiente (a representação do usuário no ambiente é chamada de avatar).
- Muda continuamente em todos os aspectos (usuários entrando e saindo do ambiente, movendo-se, mudando estados dos objetos, etc).
- Usuários, além de interagirem entre si, podem interagir com simulações computacionais. CVEs, portanto, podem ir além da imitação da realidade, permitindo situações que não existem no mundo real.

Alguns possíveis cenários de aplicação dos CVEs descritos na literatura [Waters 97] são: arquitetos em locais diferentes "visitando" prédio que estão projetando, turistas "visitando" local de destino antes de comprar as passagens, simulações militares para situações de combate, e lojas virtuais simulando compras no mundo real (carrinho de compras, produtos nas prateleiras e interação com vendedores e outros clientes da loja).

Até o presente momento, no entanto, o desenvolvimento de CVEs tem sido essencialmente voltado para atividades de "lazer", permitindo que usuários possam se comunicar remotamente [Frécon 98]. Para que os CVEs possam ser efetivamente utilizados como ferramenta de trabalho colaborativo é necessário, dentre outros aspectos, investir no trabalho de articulação, definido como o "conjunto de atividades necessárias para gerenciar a natureza distribuída do trabalho cooperativo" [Schmidt 92]. O trabalho de articulação é o esforço extra, necessário para que a colaboração seja obtida a partir da soma dos trabalhos individuais. Fazem parte do trabalho de articulação a identificação dos objetivos, o mapeamento destes objetivos em tarefas, a seleção dos "atores", a distribuição de tarefas entre eles e a coordenação da realização das atividades.

Particularmente importante entre as atividades do trabalho de articulação é a coordenação, definida em um sentido restrito como "o ato de gerenciar interdependências entre as atividades realizadas para se atingir um objetivo" [Malone 90]. A coordenação representa o aspecto dinâmico do trabalho de articulação, pois precisa ser "renegociada" de maneira quase contínua ao longo de todo o tempo que durar a colaboração.

O grande desafio ao se propor mecanismos de coordenação para controlar o trabalho colaborativo consiste em obter destes mecanismos a flexibilidade necessária para se adequar ao dinamismo da interação entre os participantes [Edwards 96]. Em outras palavras, a política de coordenação varia entre as colaborações e pode variar até mesmo durante a evolução de uma mesma colaboração. Portanto, é essencial que os sistemas de suporte ao trabalho colaborativo sejam suficientemente flexíveis para suportar estas variações [Li 98].

O objetivo deste trabalho é modelar uma biblioteca de mecanismos de coordenação reutilizáveis em diferentes implementações de CVEs. A idéia é separar

os mecanismos de coordenação dos elementos que constituem o ambiente. Para modelar os mecanismos de coordenação propostos, é usada uma abordagem baseada em redes de Petri, que serão introduzidas na Seção 4. Antes, no entanto, será dada uma visão geral de como os CVEs se enquadram dentro de uma visão estendida de animação por computador.

3. Animação por computador

Animação é definida como o "processo no qual é gerada dinamicamente uma série de quadros (...), onde cada quadro é uma alteração do quadro anterior" [Thalmann 85]. Em outras palavras, animação é a alteração no tempo de parâmetros (cor, posição, etc) de uma cena estática.

Quando o computador auxilia todo o processo de criação de uma animação, desde a modelagem do ambiente tridimensional, dos atores e de seus movimentos, até o controle da animação como um todo, diz-se que a animação gerada é *modelada por computador*, contrastando com a animação *auxiliada por computador*, na qual o computador participa apenas de algumas etapas do processo de geração (por exemplo, edição e sincronização) [Raposo 96].

O processo de criação de uma animação por computador envolve três etapas distintas: modelagem geométrica, controle e *rendering*. A fase de controle da animação é a que será tratada mais diretamente neste trabalho, pois é ela que define os movimentos dos atores e as interações entre eles e o animador.

E possível distinguir dois níveis para o controle de movimentos em animação por computador [Magalhães 98a]. No nível *concreto*, as técnicas de controle são consideradas apenas segundo suas características matemáticas de modelar os movimentos. Neste nível estão enquadradas as técnicas de interpolação de quadros-chave, de simulação cinemática e dinâmica (tanto direta quanto inversa), algoritmos genéticos, etc. No nível de intenção, o objetivo não é definir como o movimento é realizado, mas a seqüência de eventos que causa ou é afetada pelo movimento. Modelos baseados em comportamento e sistemas reativos são exemplos de técnicas representativas de controle no nível de intenção. Neste nível de controle, a animação é definida através de assertivas do tipo: "ande depressa até o centro" e "escolha uma expressão (rir, ignorar, etc) para representar o sentimento em relação ao ator com o qual está interagindo" [Perlin 96].

A Figura 1 ilustra a relação entre estes dois níveis de controle. O nível de intenção pode ser visto como uma camada acima do nível concreto, pois as regras de comportamento são mapeadas em alguma forma matemática de definição dos movimentos.

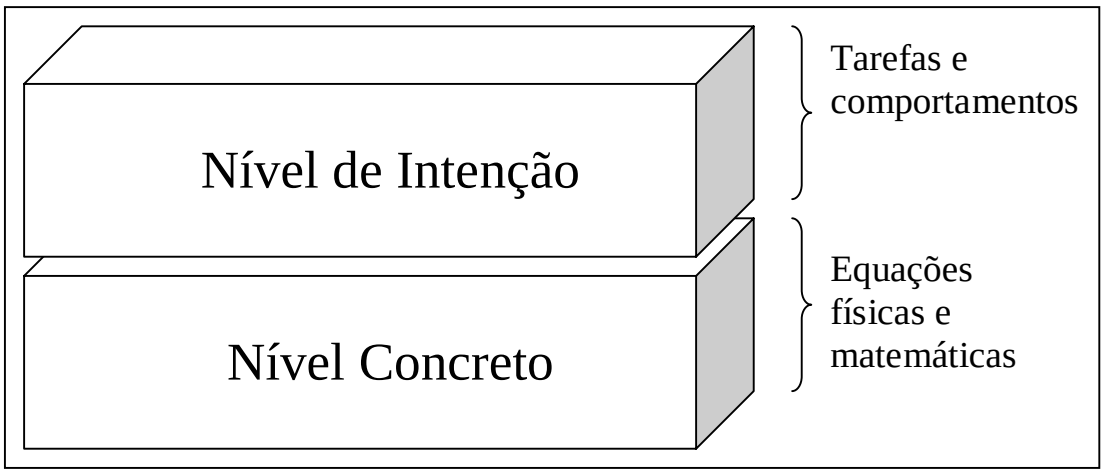

Figura 1: Níveis de controle de movimento em animação por computador.

De acordo com a visão "clássica" de animação por computador, o controle tanto no nível concreto quanto no de intenção ocorre no momento em que a animação está

sendo criada. O objetivo destas técnicas é gerar uma animação final que pode ser visualizada repetidamente, mas não pode mais ser modificada (a não ser para a criação de uma outra animação).

O conceito de modificação em tempo de execução da animação foi trazido inicialmente pelos jogos de computador, e depois pela realidade virtual, que colocam o usuário/animador como participante da cena, podendo interagir com elementos da cena e alterar todo o comportamento do ambiente.

Além disso, é cada vez mais comum a utilização de ambientes distribuídos em sistemas de animação. A distribuição, por sua vez, pode envolver conceitos complexos, como o de colaboração (vários animadores construindo a mesma animação, ou interagindo com ela) que também não são tratados pela visão clássica de animação por computador. Por estas razões, serão apresentadas na próxima seção extensões desta visão clássica, buscando englobar uma série de novos paradigmas e técnicas para o desenvolvimento e interação em animações modeladas por computador.

3.1. Uma visão estendida de animação por computador

Com o avanço das tecnologia de rede, da realidade virtual e dos estudos sobre interação multiusuário, a animação por computador passou a englobar novos conceitos e dispor de novos recursos que alteram sensivelmente a visão clássica comentada anteriormente. Assim, os próximos tópicos procuram estabelecer um *framework* que englobe as várias dimensões que estendem a visão clássica, buscando inicialmente um ordenamento e o estabelecimento de inter-relacionamentos destes novos conceitos e recursos.

3.1.1. Alteração em tempo de execução

A primeira noção que tem sido estendida hoje em dia é a de que animação é algo que, uma vez finalizado, pode apenas ser visualizado, e não mais modificado. Os jogos por computador e a realidade virtual alteraram radicalmente esta idéia, pois permitiram que o usuário “entrasse” na animação, provocando alterações ao longo da execução da animação. Este é o paradigma utilizado em ambientes virtuais (passeios por museus, universidades virtuais, etc), em jogos e simulações.

Recentemente, a especificação do MPEG-4 [MPEG-4 99] confirmou esta tendência, definindo um modelo de interação onde, dependendo do grau de liberdade permitido pelo autor, o usuário pode navegar pela cena criada, mudar a posição de objetos da cena, iniciar uma seqüência de eventos a partir da interação com um objeto, dentre outras possibilidades. Este é um modelo de interação bastante semelhante ao de linguagens típicas de realidade virtual (por exemplo, a VRML - *Virtual Reality Modeling Language* [VRML 97]).

Pode-se concluir, portanto, que as necessidades da animação por computador e da realidade virtual tendem a convergir em um futuro próximo [Green 96].

3.1.2. Processamento distribuído

A visão clássica de animação por computador também não faz nenhuma consideração a respeito da possibilidade de uma animação ser construída ou executada a partir de várias máquinas diferentes. A realização de animações distribuídas tem se

tornado uma realidade cada vez mais presente nos dias de hoje, e é essencial para o desenvolvimento de animações complexas e altamente interativas.

O conceito de animação distribuída pode ser entendido de duas maneiras diferentes [MacIntyre 98]: do ponto de vista "tradicional", onde uma única aplicação gráfica possui componentes distribuídos em várias máquinas, ou do ponto de vista dos ambientes virtuais distribuídos, onde cada máquina "representa" um usuário que compartilha o ambiente gráfico interativo multiusuário. O que há em comum nestas duas situações é que a aplicação gráfica e os dados estão distribuídos em várias máquinas, o que caracteriza a distribuição.

3.1.3. Colaboração

A colaboração traz um novo grau de liberdade aos pontos citados, ao permitir que uma animação seja utilizada por vários usuários simultaneamente. No caso da realidade virtual, existem os CVEs. No caso da animação propriamente dita, existe a possibilidade de vários usuários estarem trabalhando colaborativamente nas várias etapas da construção de uma animação.

3.1.4. A visão estendida

As seções anteriores introduziram as três dimensões ortogonais por onde o conceito clássico de animação por computador tem sido estendido. Estas três dimensões e as aplicações envolvendo animação por computador que se enquadram em cada uma delas (ou em combinações delas) podem ser visualizadas no gráfico tridimensional da Figura 2.

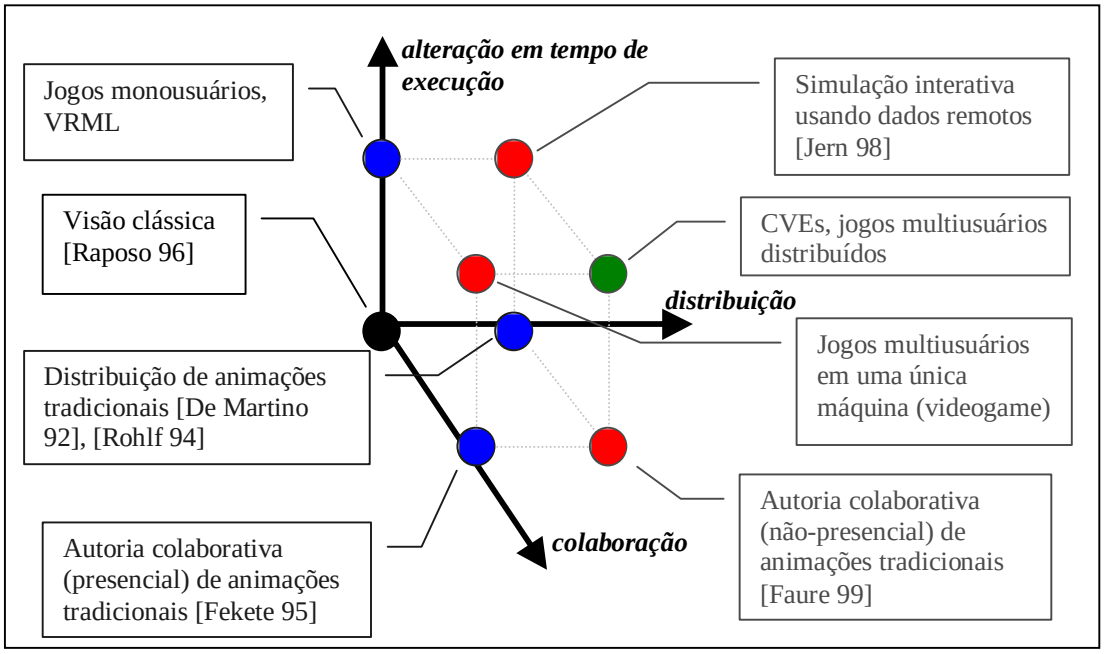

Figura 2: Visão estendida de animação por computador.

Dentro desta visão estendida de animação por computador, o caso que interessa mais diretamente a este trabalho é aquele que apresenta extensão nas três dimensões. Nesta categoria se enquadram os CVEs e os jogos multiusuários executados em máquinas diferentes.

4. Redes de Petri

4.1. Modelagem

Rede de Petri (PN - *Petri Net*) é uma ferramenta de modelagem aplicável a uma série de sistemas, especialmente aqueles com eventos concorrentes. Formalmente, uma PN é definida como uma quíntupla (P, T, F, w, M_0) onde:

$P = \{P_1, ..., P_m\}$ é um conjunto finito de lugares (p*laces*).
$T = \{t_1, ..., t_n\}$ é um conjunto finito de transições.
$F \subseteq (PxT) \cup (TxP)$ é um conjunto de arcos.
$w: F \rightarrow \{1, 2, ...\}$ é uma função que dá peso aos arcos.
$M_0: P \rightarrow \{1, 2, ...\}$ é a marcação inicial da rede (número de *tokens* em cada lugar).
Com $(P \cap T) = \varnothing$ e $(P \cup T) \neq \varnothing$.

No modelo de PN, os estados estão associados aos lugares e suas marcações, e os eventos às transições. Uma transição *t* está habilitada se cada um de seus lugares de entrada $P_i \in \bullet t$ possuir pelo menos $w(P_i, t)$ tokens, onde $w(P_i, t)$ é o peso do arco ligando P_i a *t*. Estando habilitada, uma transição pode ser disparada quando o evento associado a ela ocorrer. O disparo de *t* remove $w(P_i, t)$ tokens de cada um de seus lugares de entrada P_i e adiciona $w(t, P_o)$ tokens a cada lugar de saída $P_o \in t\bullet$[1].

A notação gráfica de PNs é também muito usada. Nesta notação, os lugares são representados por círculos, as transições por barras ou retângulos, os tokens por pontos, e os arcos por setas com os pesos escritos em cima (por definição, um arco não marcado tem peso 1).

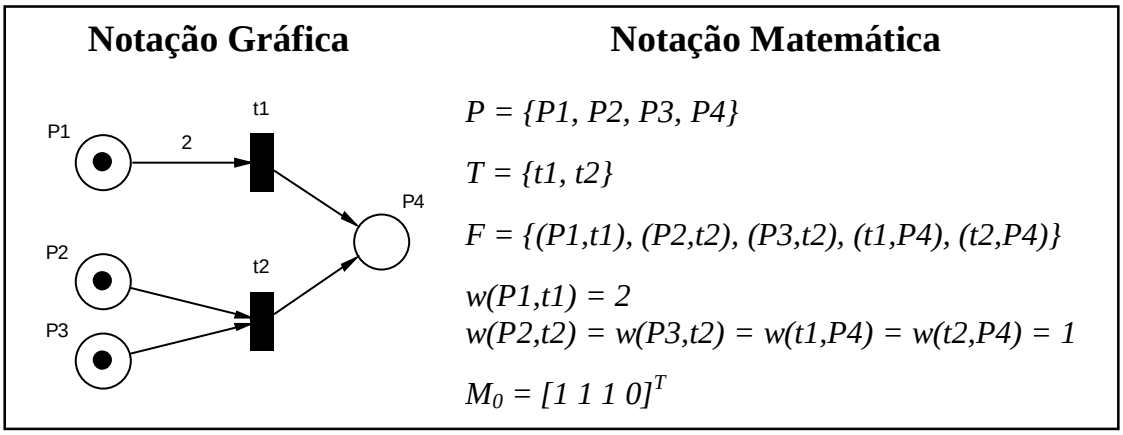

Figura 3: Notação gráfica e notação matemática de PNs.

Na PN dada como exemplo na Figura 3, apenas a transição t_2 está habilitada; t_1 não está habilitada porque seriam necessários dois tokens em P_1 para dispará-la, já que $w(P_1, t_1) = 2$. Quando t_2 for disparada, os tokens em P_2 e P_3 são retirados e P_4 recebe um token (o número de tokens não é necessariamente conservado).

Além das notações apresentadas, existe também uma notação matricial para indicar as possíveis mudanças de estado em uma PN. O estado seguinte ao disparo da transição t_j é dado por $M_{i+1} = M_i + C \times e_j$, onde e_j é um vetor coluna com 1 na

[1] $\bullet t$ é o conjunto de lugares de entrada da transição t e $t\bullet$ o conjunto de lugares de saída de t. Similarmente, $\bullet P$ e $P\bullet$ são os conjuntos de transições de entrada e saída, respectivamente, do lugar P.

posição j e 0 nas demais posições e $M_i = [q_1\ q_2\ ...\ q_m]^T$, onde q_a indica a quantidade de tokens no lugar P_a. A matriz C representa a topografia da rede, tem dimensões $m \times n$ (m é o número de lugares e n o número de transições), e o elemento c_{ij} indica quantos tokens o lugar P_i vai receber (valor positivo) ou perder (valor negativo) quando a transição t_j disparar.

Além do modelo básico, várias extensões de PN existem na literatura [Murata 89]. As extensões utilizadas neste trabalho são: arco inibidor, redes com tempo, redes de predicado/transição e redes coloridas.

O arco inibidor liga um lugar P a uma transição t e funciona de maneira oposta aos arcos comuns. Ele habilita a transição t apenas se P estiver vazio. Na notação gráfica, arcos inibidores são representados com um círculo na extremidade.

O modelo básico de PN não faz nenhum tipo de consideração quanto ao tempo de disparo das transições, ou seja, a partir do momento que estão habilitadas, as transições podem ser disparadas. Uma maneira de incluir a noção de tempo em uma PN é estabelecer um tempo de espera para o token em um lugar, antes dele habilitar as transições de saída [Ramamoorthy 80]. Também é possível estabelecer funções de probabilidade para o tempo de disparo de uma transição [Bause 96]. Neste tipo de PN (chamada PN estocástica), a transição dispara algum tempo depois de habilitada, tempo este determinado pela função de probabilidade associada à transição. O tempo também pode estar associado à "execução" do disparo das transições. Neste caso, os tokens não ficam nos lugares de entrada esperando o disparo da transição, mas são retirados deles e algum tempo depois (tempo de disparo) são entregues aos lugares de saída. Este tipo de disparo não-instantâneo é também chamado de disparo com reserva de tokens.

As PNs de predicado/transição [Genrich 86] e as coloridas [Jensen 86] permitem a diferenciação entre os tokens, definindo tipos para eles. Os arcos possuem funções ou expressões que determinam como será feita a remoção e adição de tokens durante o disparo de uma transição. Estes tipos de extensões serão tratados no Apêndice A.

Em resumo, o comportamento de um sistema modelado por PN é descrito em termos de seus estados e suas mudanças [Murata 89]. Os estados são representados por lugares e tokens, que definem o estado atual do sistema. Transições (regras de disparo) modelam o comportamento dinâmico do sistema. Os arcos indicam as seqüências de possíveis transições entre os estados.

4.2. Análise

Além das interessantes características de modelagem, tais como simplicidade da notação gráfica, formalidade da notação matemática e modelo de descrição hierárquico (encapsulamento de detalhes), as redes de Petri também oferecem importantes ferramentas de análise do sistema modelado. Há três tipos possíveis de análise: verificação, validação e desempenho [van der Aalst 98].

As análises de verificação são realizadas para garantir que a rede esteja corretamente definida e corresponda com exatidão ao sistema modelado. Neste tipo de análise é verificado se a rede apresenta *deadlocks*, se atinge algum estado não permitido, se há transições mortas, etc. As análises de verificação são baseadas em propriedades das PNs, dentre as quais se destacam:

Reachability: há alguma seqüência de disparos que leva a um estado específico? Para a verificação desta propriedade pode-se utilizar a *coverability tree*, que oferece uma visão completa da seqüência de transições e estados de uma PN. Como exemplo, considere a PN mostrada na Figura 4, com $P = \{P1, P2, P3, P4\}$, estado inicial $M_0 = [\ 1\ 1\ 0\ 0\ ]^T$, e estados subseqüentes $M_1 = [\ 0\ 1\ 1\ 0\ ]^T$ e $M_2 = [\ 1\ 0\ 0\ 1\ ]^T$. A *coverability tree* para este exemplo é vista na Figura 5.

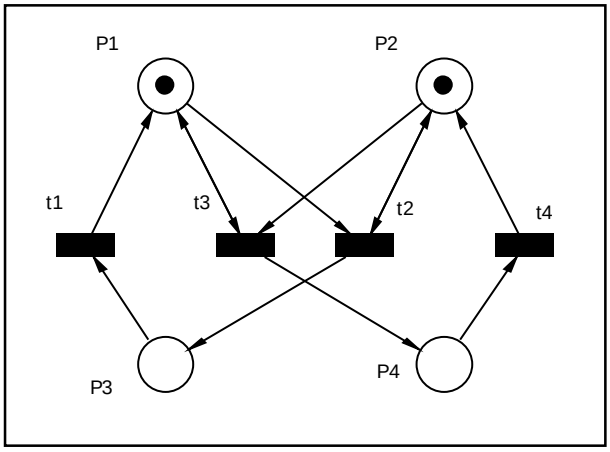

Figura 4: Exemplo de PN.

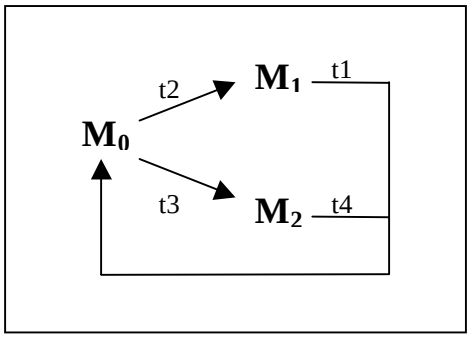

Figura 5: *Coverability tree* para o exemplo da Figura 4.

Liveness: há algum estado ou seqüência de estados que não será mais alcaçado, indicando um possível *deadlock*?

Reversibilidade: é possível retornar ao estado inicial?

Boundness: no máximo quantos tokens permanecerão em um lugar? Uma PN é dita *k-bounded* se o número de tokens em cada lugar nunca exceder *k*. Para o caso de $k = 1$, a PN é chamada de segura (*safe*).

Persistência: o disparo de duas transições habilitadas é independente, ou seja, o disparo de uma desabilita a outra? Duas transições com disparos dependentes indicam um "conflito", pois apenas uma das duas será disparada (OR lógico).

Distância síncrona: indica o nível de dependência mútua entre duas transições. Considerando σ uma seqüência de disparos a partir de qualquer marcação M e $\sigma(t_i)$ o número de vezes que a transição t_i dispara em σ, a distância síncrona entre as transições t_i e t_j é dada por $d_{i,j} = max\ |\ \sigma(t_i) - \sigma(t_j)\ |$. No exemplo da Figura 4, $d_{1,2} = d_{3,4} = 1$, indicando que estes pares de transições são interdependentes. Por outro lado, $d_{1,4} = d_{2,3} = \infty$, indicando que estes pares de transições estão associados a eventos independentes.

A análise de validação testa se a rede funciona como esperado. Os testes são feitos por meio de simulação interativa de situações fictícias para verificar se a rede as trata corretamente.

A análise de desempenho avalia a capacidade do sistema atingir certos requisitos, tais como tempo médio de espera, número médio de casos pendentes, uso de recursos, *throughput times*, etc. Análises de desempenho podem ser feitas por meio de simulação, cadeias de Markov, e outras técnicas [Magalhães 98b], [Marsan 84], [Molloy 82].

Em resumo, PNs apresentam um forte suporte teórico para a análise e um grande número de técnicas de simulação, o que, juntamente com as já comentadas características de modelagem, as tornam ferramentas adequadas para o planejamento de animações interativas [Magalhães 98b] e sistemas de *workflow* [van der Aalst 98].

5. Trabalhos relacionados

Em 1985, Anatol Holt usou redes de Petri para a coordenação de atividades em ambientes de trabalho por computador [Holt 85]. Neste importante trabalho, Holt criou uma nova interpretação para as PNs, propondo uma conexão entre a estrutura formal das mesmas e a "estrutura natural" do trabalho humano (ou computacional). Além disso, neste trabalho também foram identificados pontos essenciais da tecnologia de coordenação, que na época era bastante promissora para a criação de ambientes de trabalho eletrônicos. Dentre estes pontos, o mais importante diz respeito à flexibilidade dos mecanismos de coordenação. Segundo Holt, para serem úteis, os padrões de coordenação devem ser feitos de "maneira flexível (...), com bastante margem para a imprevisibilidade da vida real". Ainda segundo ele, "em cada ambiente de trabalho, deve ser feito um balanço entre a 'firmeza' e a adaptabilidade do suporte estrutural – uma vez que estes são antagonistas inevitáveis".

O trabalho de Holt evoluiu para a criação da *Diplan*, uma linguagem gráfica formal para o planejamento de atividades envolvendo múltiplos agentes colaboradores [Holt 88]. Esta linguagem é baseada em PNs de Predicado/Transição, com algumas pequenas alterações, como por exemplo a determinação de um tempo para as mudanças de estado (transições não instantâneas) e a referência explícita ao papel humano na execução das tarefas.

O *CHAOS* (*Commitment Handling Active Office System*) é outra proposta para a coordenação de atividades em automação de escritórios baseado em PNs [De Cindio 88]. Ele tem uso mais restrito que a Diplan, pois é voltado apenas para a automatização e coordenação das "redes de conversação" que ocorrem em um escritório.

Redes de Petri também constituem a base do *Trellis*, um modelo para a prototipação de protocolos de interação em sistemas colaborativos [Stotts 89], [Furuta 94]. O protocolo criado determina como um controlador central (servidor) deve processar as requisições dos clientes. O Trellis usa uma extensão de PNs, chamada *colored timed Petri nets* [Jensen 86], na qual os *tokens* têm um tipo e carregam informação (PNs coloridas). A noção de tempo aparece na forma de atraso e *timeout* para o disparo das transições. O atraso determina o tempo mínimo que deve ocorrer entre a habilitação e o disparo de uma transição. O *timeout* é o tempo máximo que uma transição pode ficar habilitada antes de ser disparada automaticamente. A funcionalidade de *Trellis* se diferencia da de *Diplan* porque o primeiro vai além dos aspectos de coordenação das atividades, criando "hiperprogramas", que misturam a navegação hipermídia ao suporte à colaboração.

No campo de animação por computador e realidade virtual, PNs já foram usadas para modelar padrões de reação (estímulo/resposta), no suporte à programação visual, por meio de exemplos, de animações de agentes inteligentes em ambientes de realidade virtual [Del Bimbo 96]. PNs também foram usadas para prever e testar o comportamento de animações antes de gerar os quadros das mesmas [Magalhães 98b].

As idéias relacionadas à automação e coordenação de tarefas resultaram em sistemas de *workflow*, com o objetivo de prover suporte a grupos de pessoas na execução de tarefas. Um sistema de *workflow* é definido como um "tipo particular de *groupware* para auxiliar grupos de pessoas na execução de procedimentos de trabalho; ele possui o conhecimento de como o trabalho normalmente flui em uma organização"[Ellis 93] e também como um sistema "que ajuda organizações a especificar, executar, monitorar, e coordena o fluxo de trabalho em um ambiente de trabalho distribuído" [Bull 92]. Redes de Petri e suas variações também são bastante

usadas para a modelagem deste tipo de sistema. Um exemplo são as ICNs (*Information Control Nets*), uma variação de PNs para modelagem de *workflows* [Ellis 93]. Outro exemplo são as *Workflow Nets* (WF-Nets), uma classe de PNs adequada para a representação, validação e verificação de conjuntos de tarefas interdependentes [van der Aalst 94].

Como as WF-Nets constituem o ponto de partida para os mecanismos de coordenação desenvolvidos neste trabalho, elas serão estudadas mais detalhadamente na Seção 5.2. Antes disso, na Seção 5.1, serão apresentadas algumas estruturas básicas de *workflow* e seus mapeamentos em PNs.

5.1. Estruturas básicas de workflows

Independentemente do modelo específico utilizado (ICN, WF-Net, etc), os *workflows* apresentam alguns tipos de conexões básicas e estruturas lógicas que são mapeados diretamente em PNs [van der Aalst 98].

As conexões servem para determinar o seqüenciamento das tarefas. As conexões básicas são: *AND-split*, *AND-join*, *OR-split* e *OR-join*. O *AND-split* inicia um roteamento paralelo, e é composto de uma transição com dois (ou mais) lugares de saída, de modo que estes lugares iniciem trilhas paralelas de tarefas. O *AND-join* encerra o roteamento paralelo, e é representado por uma transição com dois (ou mais) lugares de entrada. O *OR-split* inicia um roteamento condicional, ou seja, apenas uma das trilhas possíveis será seguida. Em PNs, o *OR-split* é representado por uma situação de conflito, ou seja, um lugar com mais de uma transição de saída (apenas uma delas será disparada a cada *token* que chega no lugar). O *OR-join* encerra um roteamento condicional, sendo representado por um lugar com mais de uma transição de entrada. A Figura 6 mostra o modelo de PNs para as conexões básicas de *workflow*.

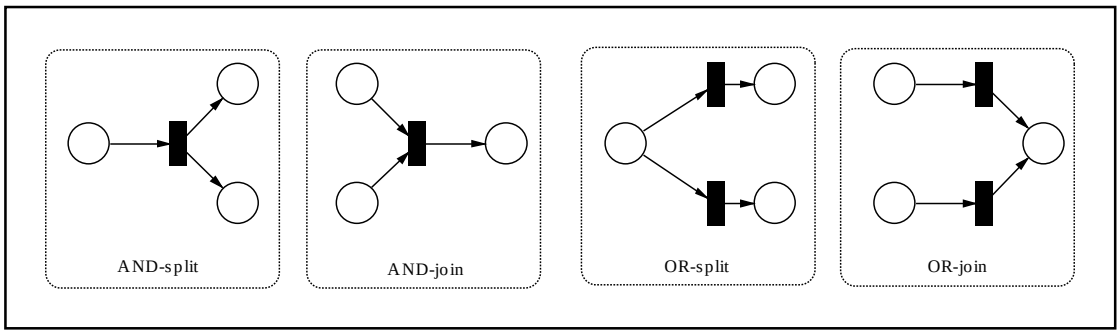

Figura 6: Conexões básicas de *workflow*.

Com relação ao *OR-split*, o modelo apresentado na Figura 6 deixa a decisão sobre qual tarefa será executada para o momento em que uma delas se inicia (uma das duas transições dispara). É possível criar uma variação do modelo acima de modo que a decisão seja tomada antes do início de uma das tarefas, no momento do término de uma tarefa anterior (t_a). Esta variação é chamada *OR-split* explícito, e sua representação é mostrada na Figura 7.

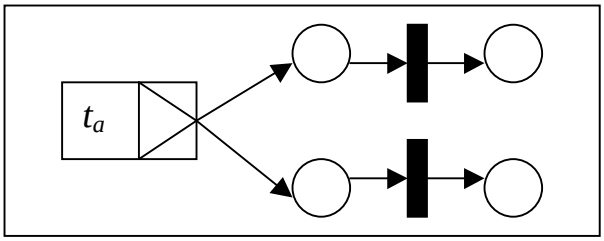

Figura 7: *OR-split* explícito

A partir das conexões básicas, é possível construir algumas estruturas lógicas típicas de *workflows*: roteamento paralelo, condicional exclusivo, condicional não exclusivo, e iteração. O roteamento paralelo é um trecho de PN iniciado por um *AND-split* e finalizado por um *AND-join*, e representa trilhas de tarefas executadas em paralelo. O condicional exclusivo é um trecho iniciado por um *OR-split* e finalizado por um *OR-join*, e representa caminhos alternativos para o fluxo de trabalho. O condicional não exclusivo representa a possibilidade de seguir por apenas um dos caminhos ou por ambos, e é modelado por uma combinação das quatro conexões básicas. A iteração é a possibilidade de repetição de uma seqüência de tarefas e é modelada por um *OR-split* com uma saída retornando a algum lugar da PN. Estas quatro estruturas lógicas são mostradas na Figura 8.

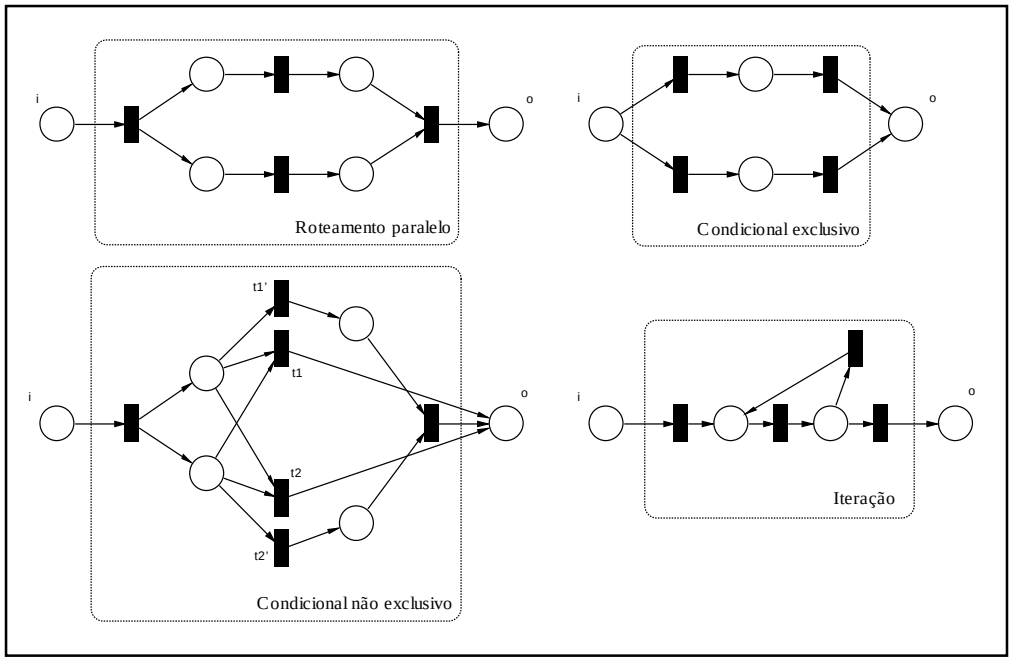

Figura 8: Estruturas lógicas típicas de *workflows*.

5.2. Workflow Nets

Uma WF-Net é uma PN com algumas propriedades especiais para a modelagem de processos de *workflow*. Antes de analisar estas propriedades, é necessário estabelecer alguns conceitos usados no estudo de *workflows* e seu mapeamento em PNs [van der Aalst 94]:

- *Tarefa*: parte do trabalho a ser realizado. Normalmente, uma tarefa é atômica, i.e., não pode ser subdividida em tarefa menores. Em uma WF-Net, as tarefas são representadas por transições.
- *Recurso*: quem realiza a tarefa, podendo ser humano ou não (impressora, software, etc). Um recurso fica ocupado durante o tempo que estiver realizando uma tarefa.
- *Classe de recursos*: conjunto de recursos. Em geral, o sistema de *workflow* não determina o recurso que vai realizar a tarefa, mas a classe de recursos (por exemplo, a tarefa imprimir vai ser realizada por uma impressora, não especificando qual delas – se houver mais de uma disponível).
- *Gerenciador de recursos*: controla a alocação de recursos para as tarefas. Em uma WF-Net, o gerenciador de recursos é modelado por uma sub-rede ligada às tarefas, responsável pela alocação dos recursos (representados por *tokens*).
- *Procedimento*: conjunto (parcialmente) ordenado de tarefas, classes de recursos, atividades de controle e sub-procedimentos. O procedimento é uma PN, composta de transições (tarefas ou atividades de controle) e lugares (condições) ligando estas transições.
- *Atividades de controle*: fazem parte de um procedimento e servem para especificar o roteamento do trabalho dentro do procedimento e a sincronização entre as tarefas. Também são representadas como transições em WF-Nets e fazem parte das conexões básicas apresentadas na Figura 6.
- *Caso (job)*: processo que modela a execução do trabalho em um procedimento. Em uma WF-Net, um caso é representado pelo fluxo de um *token* pela rede. O *token* que representa o caso é chamado *job token* (há outros tipos de *tokens*, como os que representam os recursos). O estado de um caso é dado pela marcação da rede em um determinado instante. É possível que um procedimento tenha mais de um caso simultaneamente (mais de um *job token* fluindo pela rede).

Em resumo, uma WF-Net é uma PN representando um procedimento, onde as transições modelam tarefas ou atividades de controle e o fluxo dos *tokens* por esta rede representa os casos executados. Além disso, quando há a necessidade de gerenciamento de recursos, uma sub-rede deve ser anexada a ela.

O modelo formal exige que a PN satisfaça dois requisitos para ser classificada como WF-Net [van der Aalst 98]. Em primeiro lugar, a rede deve possuir um lugar de entrada (*i*) e um lugar de saída (*o*). Um *token* em *i* corresponde a um caso a ser iniciado, e um *token* em *o* corresponde a um caso já terminado. O segundo requisito impõe que não haja tarefas ou condições pendentes, i.e., todas as tarefas (transições) e condições (lugares) devem contribuir para o processamento dos casos.

Formalmente, uma PN é uma WF-Net se e somente se:

(i) a PN possuir dois lugares especiais: *i* e *o*, onde *i* é um *source place*: $\bullet i = \emptyset$ e *o* é um *sink place*: $o\bullet = \emptyset$.

(ii) ao se adicionar uma transição t^* conectando o lugar de saída *o* ao lugar de entrada *i* (i.e., $\bullet t^* = \{o\}$ e $t^*\bullet = \{i\}$), a PN resultante será fortemente conectada.

A condição *(ii)* corresponde ao segundo requisito descrito acima, e exige uma definição adicional [van der Aalst 97]:

(iii) uma PN é fortemente conectada se e somente se, para cada par de nós (lugares e transições) *x* e *y*, existir um caminho direto ligando *x* a *y*.

A Figura 9 ilustra uma WF-Net, onde as transições t_1, t_2 e t_3 representam tarefas e c_1 e c_2 representam atividades de controle (*AND-split* e *AND-join*, respectivamente). Além disso, a capacidade de hierarquização das PNs, permite o encapsulamento de detalhes, pois qualquer uma das tarefas poderia estar representando uma outra WF-Net (sub-procedimento).

Outro aspecto interessante do modelo de WF-Nets é a decomposição de cada tarefa em uma rede com quatro lugares e cinco transições, que interagem com o recurso e o gerenciador de recursos (Figura 10) [van der Aalst 94].

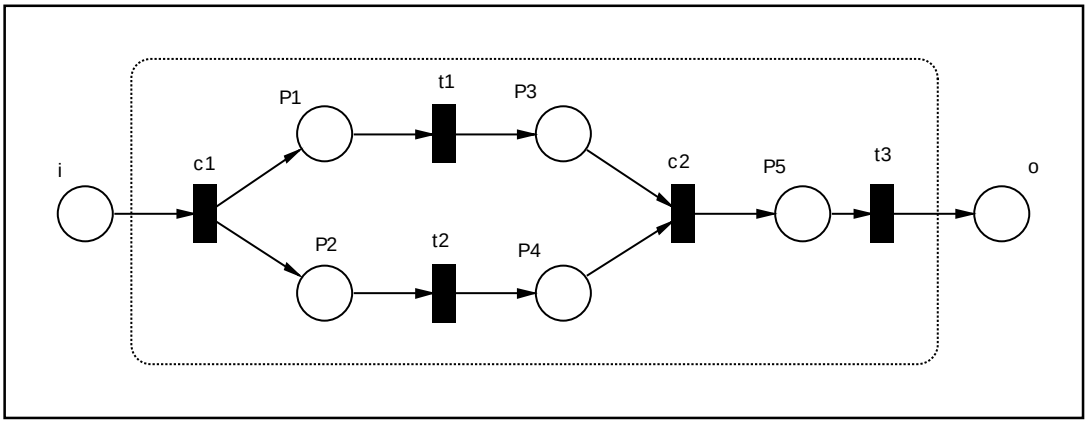

Figura 9: Exemplo de WF-Net

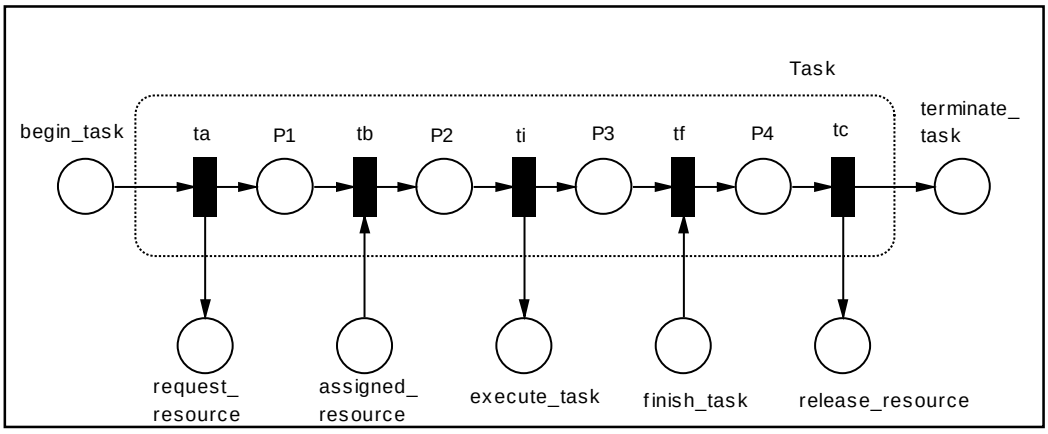

Figura 10: Estrutura de uma tarefa.

No modelo da Figura 10, os cinco lugares que aparecem associados a cada uma das transições (*request_resource*, *assigned_resource*, *execute_task*, *finish_task* e *release_resource*) representam a interação com o gerenciador de recursos e com o recurso (quem realiza a tarefa). O *request_resource* indica ao gerenciador que a tarefa deseja um determinado recurso. Após a alocação deste recurso, o gerenciador coloca um *token* em *assigned_resource*, para dar prosseguimento à tarefa. Os lugares *execute_task* e *finish_task* marcam, respectivamente, o início e o final da tarefa (interação com o recurso). Finalmente, o *release_resource* indica ao gerenciador que a tarefa foi encerrada e o recurso está novamente liberado. Portanto, uma tarefa está ligada a duas subredes, uma representando o gerenciador de recursos (que tem

request_resource e *release_resource* como lugares de entrada e *assigned_resource* como lugar de saída) e outra representando a "lógica" da tarefa executada pelo recurso (tem *execute_task* como lugar de entrada e *finish_task* como lugar de saída).

O modelo de WF-Nets também permite distinguir quatro formas de execução das tarefas (disparo de transições): automática (transição disparada assim que habilitada), pelo usuário, por um evento externo (mensagem) e por tempo.

A noção de WF-Nets e a decomposição de cada tarefa de acordo com o modelo da Figura 10 constituem a base do modelo proposto em seguida para a representação e coordenação de processos em ambientes virtuais colaborativos.

6. CAV – Uma biblioteca de mecanismos de coordenação

O aspecto tratado neste trabalho, como já comentado, diz respeito à inserção de mecanismos de coordenação em ambientes virtuais para gerenciar interdependências entre tarefas. Para isso, será modelada, usando redes de Petri, uma biblioteca de mecanismos de coordenação (CAV – *C*oordenação em *A*mbientes *V*irtuais).

A CAV constrói várias redes padrões que modelam gerenciadores de recursos e a sincronização entre as tarefas para vários tipos de interdependências, usando modelo de tarefa apresentado na seção anterior (Figura 10). A idéia é fazer com que o projetista do ambiente virtual se preocupe apenas com a definição da WF-Net que modela o seqüenciamento das tarefas e com a definição das interdependências entre estas tarefas e não mais com os mecanismos para gerenciar estas dependências, pois eles seriam fornecidos pela CAV.

No esquema proposto, o ambiente virtual é modelado em três níveis distintos: nível de *workflow*, nível de coordenação e nível de especificação das tarefas. No nível de *workflow*, são definidos o seqüenciamento das tarefas e as possíveis dependências entre elas. Neste nível, o sistema é representado por redes como a da Figura 9, onde as tarefas são modeladas por transições. No nível de coordenação, as tarefas interdependentes são expandidas de acordo com o modelo da Figura 10 e os elementos da CAV (mecanismos de coordenação) são inseridos entre elas. O nível de especificação das tarefas expande a tarefa propriamente dita (que ocorre entre *execute_task* e *finish_task* – Figura 10) em uma PN que modela sua execução. Este último nível não será tratado neste trabalho, pois ele se enquadra no nível concreto de controle (Figura 1). É no nível de especificação das tarefas que são modeladas as equações que regem os movimentos para a execução de tarefas (partindo do princípio que todas as tarefas exigem algum tipo de movimento da parte do ator que a está realizando). A Figura 11 ilustra a relação entre estes três níveis.

O objetivo da CAV é facilitar a construção do nível de coordenação a partir do nível de *workflow*, pois uma vez definidas as interdependências entre as tarefas, a expansão ocorre utilizando o modelo da Figura 10 e os mecanismos de coordenação reutilizáveis da biblioteca.

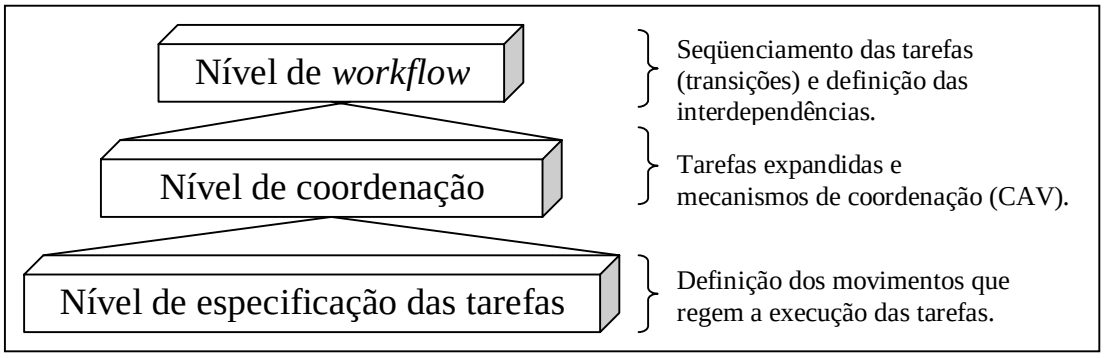

Figura 11: Os níveis do modelo de animações/ambientes virtuais.

Antes de se iniciar o estudo dos mecanismos de coordenação da CAV, é necessário fazer algumas distinções entre o modelo a ser utilizado e o modelo de *workflow* apresentado na seção anterior [van der Aalst 94], [van der Aalst 97], [van der Aalst 98].

1. O modelo de *workflow* engloba toda a lógica de funcionamento do sistema em uma única WF-Net. No caso de ambientes virtuais, o sistema passa a ser

multiusuário e cada ator tem seu comportamento modelado por uma WF-Net, de modo que o sistema como um todo é um conjunto de WF-Nets com tarefas interdependentes não só dentro de uma mesma WF-Net como também entre WF-Nets diferentes.

2. No modelo aqui utilizado não há muito rigor quanto à necessidade de cada ator ter seu comportamento modelado por uma WF-Net formal. Em outras palavras, as PNs para os atores no nível de *workflow* não precisam forçosamente atender ao requisito *(ii)* apresentado na seção 5.2, pois o comportamento projetado para um ator pode aceitar tarefas "pendentes", o que seria inaceitável em *workflows*[2].
3. A tarefa em *workflows* é necessariamente atômica. No modelo de ambientes virtuais, uma tarefa pode encapsular uma série de sub-tarefas que só serão expandidas no nível de especificação.

A CAV reconhece duas grandes classes de interdependências entre as tarefas: dependências temporais e de gerenciamento de recursos. As dependências temporais são tratadas por mecanismos (PNs) inseridas entre os lugares *execute_task* e *finish_task* das tarefas (Figura 10). As dependências de gerenciamento de recurso são tratadas por mecanismos inseridos entre *request_resource*, *assigned_resource* e *release_resource*. A criação destes mecanismos de coordenação entre as tarefas é uma das contribuições deste trabalho.

6.1. Dependências temporais

As dependências temporais servem para estabelecer o ordenamento no processo de execução de tarefas. Através dos mecanismos de coordenação propostos para as dependências temporais, é possível estabelecer se uma tarefa deve ser executada antes, durante ou depois de alguma outra tarefa.

Os mecanismos propostos são baseados nas relações temporais definidas por James F. Allen em um artigo clássico de lógica temporal [Allen 84]. Segundo Allen, há um conjunto de relações primitivas e mutuamente exclusivas que podem ser aplicadas sobre intervalos de tempo:

- **t1 igual a t2** (*t1 equal t2*): t1 e t2 são o mesmo intervalo de tempo.
- **t1 inicia t2** (*t1 starts t2*): t1 e t2 começam juntos, mas t1 termina antes de t2.
- **t1 finaliza t2** (*t1 finishes t2*): t1 e t2 terminam juntos, mas t1 começa depois de t2.
- **t1 antes de t2** (*t1 before t2*): t1 ocorre antes de t2, e eles não se sobrepõem.
- **t1 encontra t2** (*t1 meets t2*): t1 ocorre antes de t2, que começa imediatamente após o término de t1 (não há intervalo entre t1 e t2).
- **t1 sobrepõe t2** (*t1 overlaps t2*): t1 inicia antes de t2, que começa antes de t1 terminar.
- **t1 durante t2** (*t1 during t2*): t1 está totalmente contido em t2.

A Figura 12 ilustra graficamente as relações descritas acima.

[2] As redes que representam o comportamento do ambiente no nível de *workflow* continuarão sendo chamadas de WF-Nets, já feita a ressalva de que nem sempre elas estarão de acordo com os requisitos estabelecidos em [van der Aalst 97].

Figura 12: Relações temporais [Allen 84].

As relações da Figura 12 estabelecem algumas possíveis dependências temporais entre as tarefas. Mecanismos de coordenação da CAV foram modelados para garantir estas relações temporais (e algumas variações delas) entre as tarefas. Estes mecanismos são apresentados nas próximas seções.

6.1.1. Tarefa 1 igual a Tarefa 2

Esta relação estabelece que uma tarefa deve ser executada no mesmo intervalo de tempo que a outra. Para garantir esta relação no nível de coordenação, é necessário apenas garantir que as tarefas só se iniciarão quando ambas estiverem prontas para serem executadas (*tokens* nos lugares *execute_task* de ambas) e que elas terminarão juntas (*tokens* enviados simultaneamente aos lugares *finish_task*). A subrede mostrada na Figura 13 representa o modelo proposto para a coordenação deste tipo de dependência (para simplificação da figura, apenas os lugares *execute_task* e *finish_task* do modelo da Figura 10 estão representados).

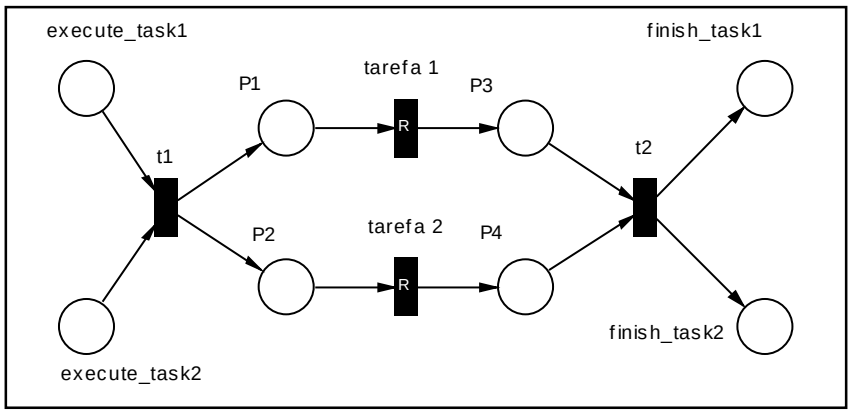

Figura 13: Mecanismo de coordenação para a relação *tarefa 1 igual a tarefa 2*

Na Figura 13, a transição *t1* é uma atividade de controle que constitui ao mesmo tempo um *AND-join* e um *AND-split*, garantindo o início simultâneo da execução das tarefas. A transição *t2* também constitui um *AND-join* e um *AND-split*, que garantem a conclusão simultânea de ambas as tarefas. As transições denominadas *tarefa 1* e

tarefa 2 representam as lógicas das tarefas a serem executadas. Estas transições devem ser expandidas no nível de especificação de tarefas. A letra R na representação gráfica destas transições indica que elas são transições com reserva de *tokens* (i.e., os *tokens* são retirados dos lugares de entrada no início do disparo e enviados para os lugares de saída somente após o tempo de disparo – o disparo não é instantâneo). Este tipo de transição é adequado para encapsular detalhes da lógica das tarefas no nível de coordenação, pois ela é corretamente substituída por uma subrede no nível de especificação de tarefas.

A análise da *coverability tree* para o modelo proposto indica que ele funciona conforme esperado e não há *deadlocks* (desde que as duas tarefas sejam executadas em algum momento).

Do ponto de vista da lógica temporal, o modelo da Figura 13 é suficiente. No entanto, como o objetivo é lidar com relações entre tarefas que fazem parte de procedimentos muitas vezes complexos, este mecanismo foi expandido para acrescentar ao modelo elementos que ajudem a evitar *deadlocks*. Isso porque as tarefas 1 e 2 podem, por exemplo, pertencer a caminhos alternativos (condicionais) de atores diferentes. Desse modo, se o caso da WF-Net 1 optar pelo caminho que passe pela tarefa 1, e o caso da WF-Net 2 optar por um caminho que não passe pela tarefa 2, o primeiro ficaria bloqueado (*deadlock*). Dependendo da rigidez do modelo, o *deadlock* pode ser inevitável, mas é possível apresentar alternativas para prover flexibilidade ao modelo. Uma alternativa seria acrescentar um mecanismo de *time-out*, de modo que uma tarefa possa ser executada se a outra não se iniciar após um certo tempo de espera. O modelo completo do mecanismo de coordenação proposto, incluindo este tipo de *time-out* e a representação completa das tarefas (de acordo com o modelo da Figura 10) é mostrado na Figura 14.

Na Figura 14 as transições denominadas *time_out1* e *time_out2* são transições temporais, disparadas após um certo tempo de habilitadas. As transições *t1'* e *t2'* representam execuções alternativas das tarefas 1 e 2, respectivamente, no caso de ocorridos os *time-outs*.

O modelo da Figura 14 permite a execução das tarefas após um certo tempo, mesmo contrariando a relação de dependência. Uma alternativa que não contraria a relação, propõe um *time-out* ligando o *execute_task* ao lugar de entrada *i* da tarefa, de modo que ela volte ao seu estado inicial decorrido um certo tempo de espera. É necessário também colocar um *token* em *release_resource*, para que o gerenciador de recursos continue funcionando corretamente. Esta alternativa é mostrada na Figura 15.

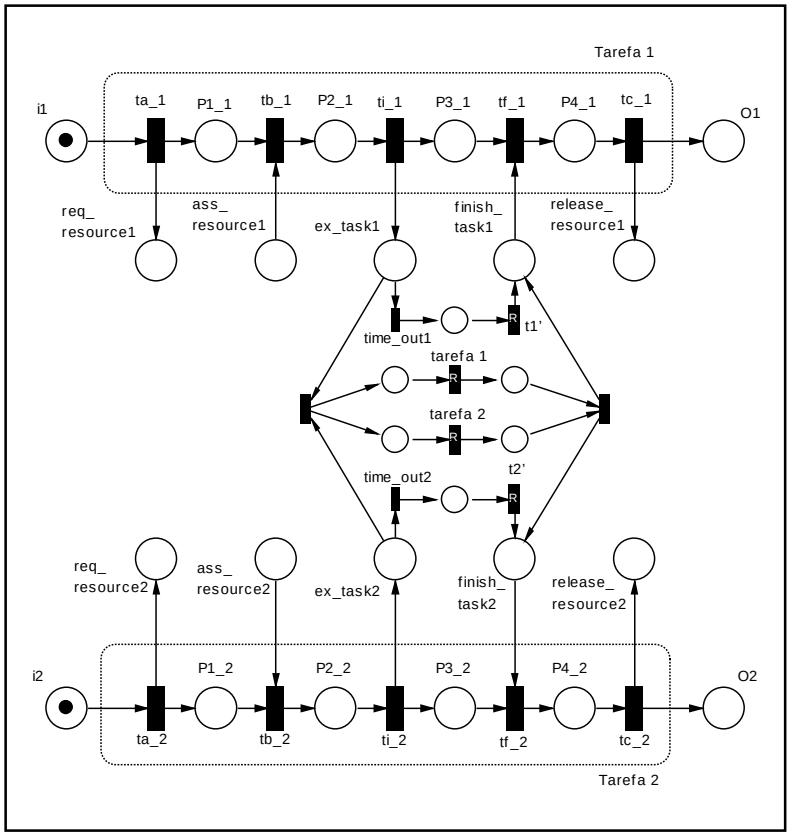

Figura 14: Mecanismo de coordenação para a relação *tarefa 1 igual a tarefa 2* com *time-out*.

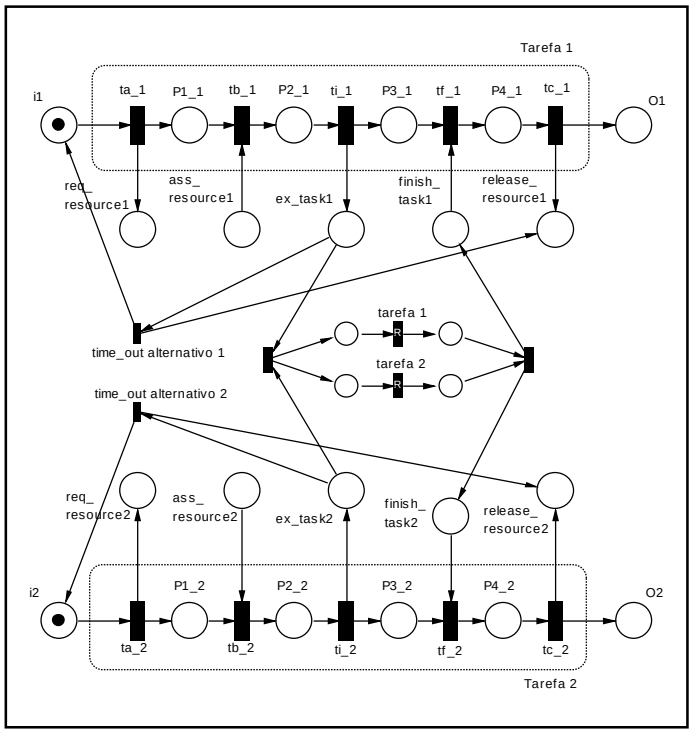

Figura 15: Mecanismo de coordenação para a relação *tarefa 1 igual a tarefa 2* com *time-out* alternativo.

6.1.2. Tarefa 1 inicia Tarefa 2

Esta relação estabelece que as duas tarefas começam juntas, e a primeira deve acabar antes [Allen 84]. No entanto, a ausência desta última restrição também resulta em uma relação interessante: duas tarefas começam juntas, não importa qual termina primeiro. Dessa maneira, a relação *tarefa 1 inicia tarefa 2* será modelada destas duas formas.

A forma sem a restrição de uma tarefa terminar primeiro é na verdade um subconjunto da relação anterior (*tarefa 1 igual a tarefa 2*), onde só a primeira metade da rede é utilizada (a que garante que as duas tarefas começam juntas). Esta relação é modelada na Figura 16, onde só estão representados os lugares *execute_task* e *finish_task* do modelo completo das tarefas (compare com a Figura 13).

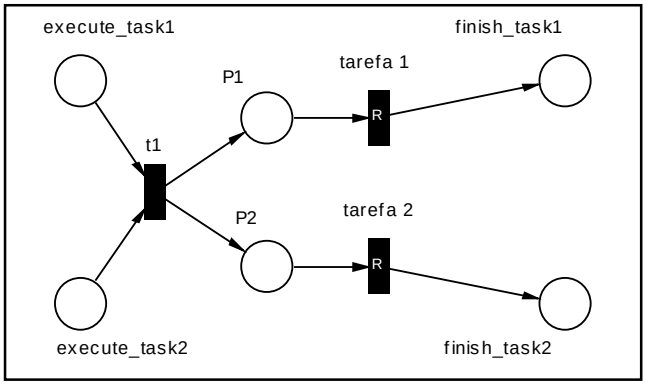

Figura 16: Mecanismo de coordenação para a relação *tarefa 1 inicia tarefa 2*, sem a restrição da tarefa 1 terminar antes.

Para garantir que a tarefa 1 termine antes da tarefa 2, é necessário acrescentar um *AND-join* após as tarefas, de forma a garantir que o *token* só seja enviado para o *finish_task* da tarefa 2 após o término da tarefa 1. A Figura 17 mostra o modelo completo para esta relação, já incluindo o *time-out* que permite a realização de uma tarefa caso a outra não se inicie após um certo tempo de espera. Repare que a única diferença com relação à rede da relação *tarefa 1 igual a tarefa 2* (Figura 14) é que a tarefa 1 não espera a tarefa 2 terminar (o *finish_task1* recebe o *token* logo após a realização da tarefa 1). É possível também utilizar um *time-out* alternativo que retorna a tarefa ao seu estado inicial, sem sua execução (similar ao da Figura 15).

6.1.3. Tarefa 1 finaliza Tarefa 2

Assim como no caso anterior (*tarefa 1 inicia tarefa 2*), é possível modelar esta relação de duas maneiras. A primeira não estabelece restrições sobre qual das duas tarefas deve começar antes, apenas exige que as duas terminem juntas. Esta forma da relação também é um subconjunto da relação *tarefa 1 igual a tarefa 2*, onde só a segunda metade da rede é utilizada (a que garante que as duas tarefas terminam juntas). Esta relação é modelada na Figura 18, onde só aparecem os lugares *execute_task* e *finish_task* do modelo completo das tarefas (repare o *time-out* para evitar que uma tarefa fique esperando infinitamente o término da outra).

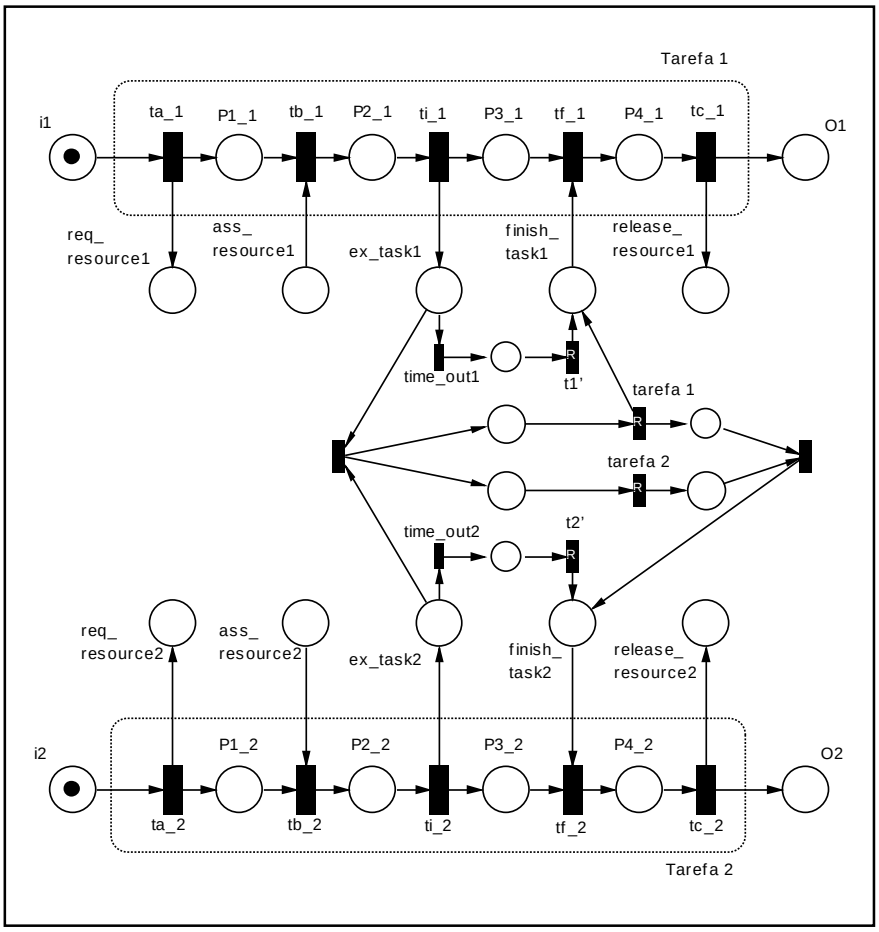

Figura 17: Mecanismo de coordenação para a relação *tarefa 1 inicia tarefa 2*.

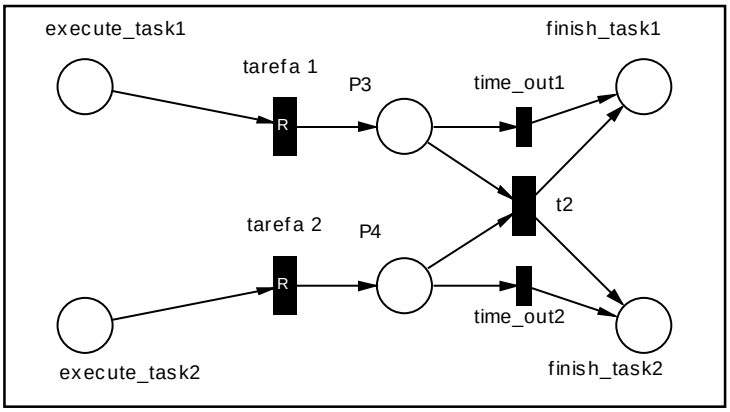

Figura 18: Mecanismo de coordenação para a relação *tarefa 1 finaliza tarefa 2*, sem restrições sobre qual tarefa deve começar antes .

A definição original desta relação [Allen 84] exige que a tarefa 1 comece após a tarefa 2. Para modelá-la em sua forma original (Figura 19), é necessário utilizar uma transição e um lugar (*t1* e *P1*) para garantir que a tarefa 1 só comece depois da tarefa 2. A chegada de um *token* em *P1* indica que a tarefa 2 já está começando e, portanto, a tarefa 1 pode também começar. A transição *t2* é uma atividade de controle que constitui um *AND-join* para garantir que as duas tarefas terminem juntas. No modelo proposto, apenas a tarefa 2 possui *time-out*, pois a tarefa 1 só começa se a outra estiver sendo executada, não existindo a possibilidade dela ficar esperando a tarefa 2.

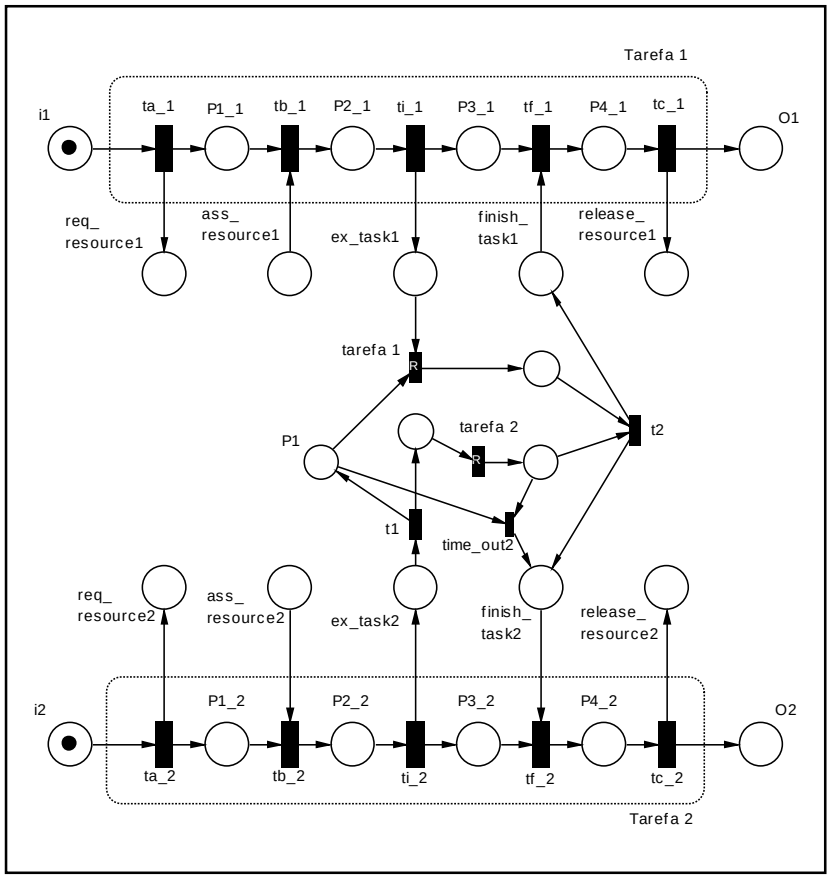

Figura 19: Mecanismo de coordenação para a relação *tarefa 1 finaliza tarefa 2*.

Repare que a transição *time_out2* retira um *token* de *P1*, impedindo a execução da tarefa 1 após o término da tarefa 2. Neste caso, há também a possibilidade de se usar um *time-out* alternativo (como o da Figura 15) para evitar que a tarefa 1 fique esperando indefinidamente pelo início da tarefa 2.

6.1.4. Tarefa 2 depois de Tarefa 1

Esta é uma relação que derivou da relação *antes de* do modelo de Allen [Allen 84]. A relação *tarefa 2 depois de tarefa 1* impõe uma restrição sobre a execução da tarefa 2, que só pode ser executada após a tarefa 1. A tarefa 1 não tem nenhuma restrição. A relação *tarefa 1 antes de tarefa 2* é diferente, pois impõe restrição sobre a execução da tarefa 1, que não pode mais ser executada se a tarefa 2 já foi executada (nesse caso, a tarefa 2 não tem restrições, e não precisa ficar esperando a execução da tarefa 1). Do ponto de vista da lógica temporal, esta diferença pode não ser significativa, mas no caso dos mecanismos de coordenação, esta distinção gera modelos completamente diferentes.

A relação *tarefa 2 depois de tarefa 1* está associada ao conceito de pré-requisito, freqüentemente usado em *workflows* (a tarefa 1 é pré-requisito para a tarefa 2). O modelo é bastante simples, sendo apenas um roteamento seqüencial, onde a tarefa 1 tem um lugar de saída (*P1*) que é um lugar de entrada da tarefa 2 (Figura 20a).

No modelo da Figura 20a, cada execução da tarefa 1 dá direito a uma execução da tarefa 2 de maneira cumulativa (i.e., se a tarefa 1 for executada n vezes seguidas, será possível realizar até n vezes a tarefa 2). Uma possível alternativa é estabelecer que uma única execução da tarefa 1 dá direito a inúmeras execuções da tarefa 2. Para

isso, a única alteração necessária no modelo é adicionar um arco ligando a transição *tarefa 2* ao lugar *P1* (Figura 20b). Repare que em ambos os casos a rede não é *k-bounded* (a não ser que haja alguma restrição no nível de *workflow* sobre o número de vezes que a tarefa 1 deve ser executada antes da tarefa 2).

Uma terceira situação poderia permitir que a tarefa 2 fosse executada um número específico de vezes a cada execução da tarefa 1. Para isso, bastaria colocar um peso no arco saindo da transição tarefa 1 para P1 (Figura 20a). Este valor indicaria o número de vezes que a tarefa 2 poderia ocorrer (número de *tokens* em P1).

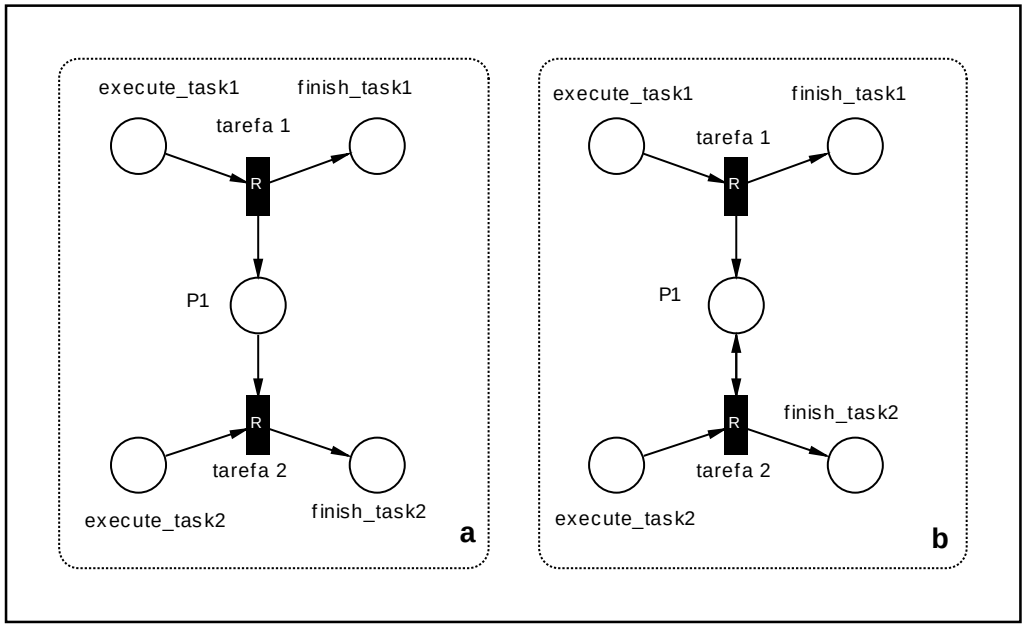

Figura 20: Mecanismos de coordenação para a relação *tarefa 2 depois de tarefa 1*.

Para evitar *deadlocks*, o modelo apresentado pode ser acrescido dos dois tipos de *time-out* discutidos anteriormente.

6.1.5. Tarefa 1 antes de Tarefa 2

Como já comentado, a restrição nesta relação ocorre sobre a tarefa 1, que não poderá mais ser executada após a execução da tarefa 2. A única restrição imposta sobre a tarefa 2 é que ela deve esperar o término da tarefa 1, caso esta já tenha iniciado sua execução. Se a tarefa 1 ainda não estiver pronta para a execução, a tarefa 2 não tem a obrigação de ficar esperando, podendo ser executada e bloquear futuras execuções da tarefa 1.

Este modelo utiliza arcos inibidores (representados com círculos na extremidade). O núcleo do modelo criado para esta relação é a transição *t1* (Figura 21). Esta transição determina a execução da tarefa 2, e fica inibida se houver *tokens* em *execute_task1* ou em *P3* (tarefa 1 pronta para executar ou em execução, respectivamente). O disparo de *t1* coloca um *token* em *P1*, que inibe o disparo de *t2* e, consequentemente, a execução da tarefa 1. Se a tarefa 2 ainda não tiver sido executada (*token* em *P1*), *t2* pode ser disparada, enviando *tokens* para *P2* (habilita a tarefa 2) e *P3* (inibe *t1*). Ao final da tarefa 1, *t4* é disparada retirando o *token* de *P3*. A transição *t3* serve para possibilitar as demais execuções da tarefa 2 (após a primeira, que coloca o *token* em *P1*), que ocorrerão independentes da presença de *tokens* em *execute_task1* (a tarefa 1 não poderá ocorrer mais). É aconselhável colocar um *time-out* entre

execute_task1 e *i1* para que a tarefa 1 sempre volte ao seu estado inicia quando ela não puder mais ser executada (como comentado na Seção 6.1.1 e mostrado na Figura 15, a transição *time_out1* também deve estar ligada ao lugar *release_resource1*; este arco não está mostrado na Figura 21 por uma questão de legibilidade).

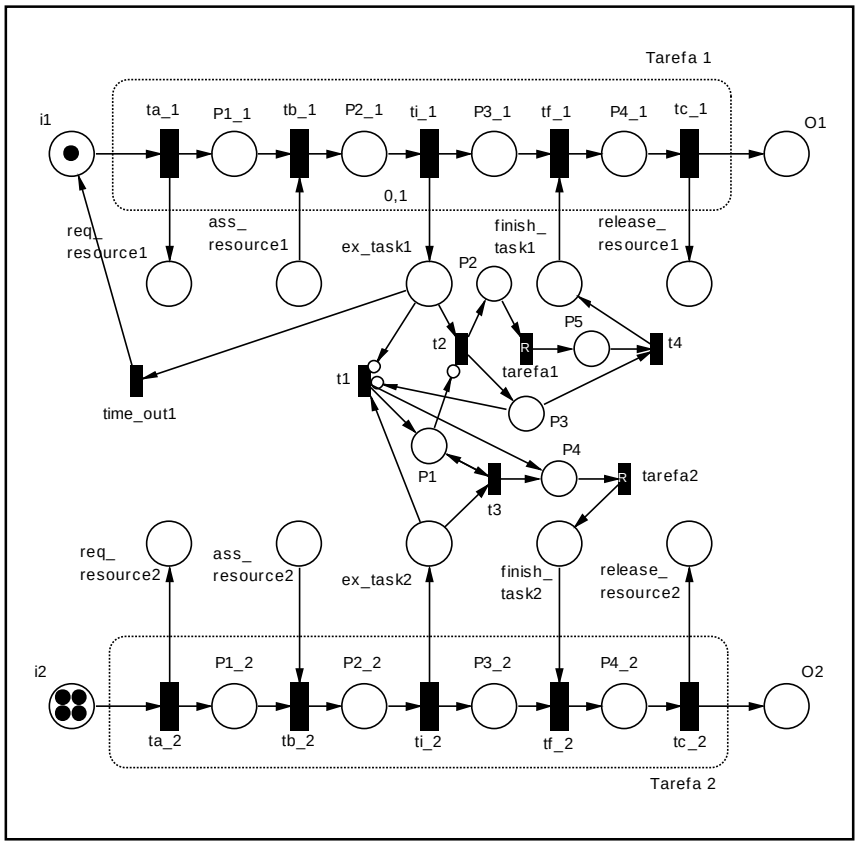

Figura 21: Mecanismo de coordenação para a relação *tarefa 1 antes de tarefa 2*.

6.1.6. Tarefa 1 encontra Tarefa 2

De acordo com esta relação, a tarefa 2 deve começar imediatamente após a tarefa 1. No nível de coordenação, esta relação é garantida bloqueando a conclusão da tarefa 1 enquanto a tarefa 2 não estiver pronta.

No modelo da Figura 22, esta relação é satisfeita colocando o lugar *execute_task2* como entrada da transição que representa a tarefa 1. Dessa forma, a tarefa 1 só será executada se a tarefa 2 estiver pronta para começar logo depois. Para manter a tarefa 2 habilitada, a tarefa 1 deve devolver o *token* ao lugar *execute_task2* após seu término. A conclusão da tarefa 1, habilita a tarefa 2 (token em *P1*). O modelo apresentado possui um *time-out*, para evitar que a tarefa 1 espere indefinidamente a habilitação da tarefa 2. A tarefa 2 também pode possuir um *time-out* no estilo do da Figura 14 (contrariando a relação e executando a tarefa) ou do da Figura 15 (retornando ao estado inicial).

6.1.7. Tarefa 1 sobrepõe Tarefa 2

Esta relação também permite a criação de dois modelos diferentes. De acordo com a definição original [Allen 84], a tarefa 2 deve começar antes do término da

tarefa 1, e esta deve terminar antes da tarefa 2. O primeiro modelo apresentado, no entanto, relaxa a segunda parte da definição, não se preocupando com qual das duas tarefas termina primeiro. Este modelo é apresentado na Figura 23.

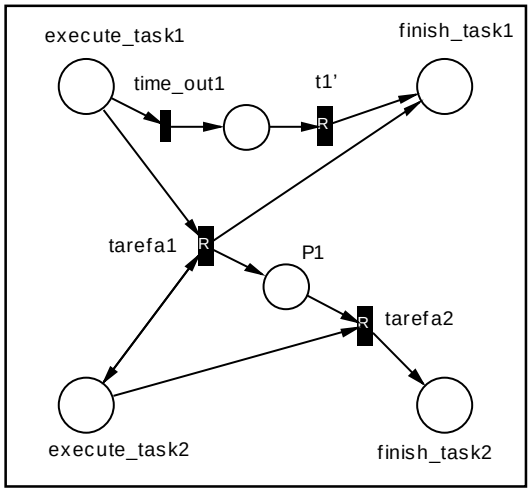

Figura 22: Mecanismo de coordenação para a relação *tarefa 1 encontra tarefa 2*.

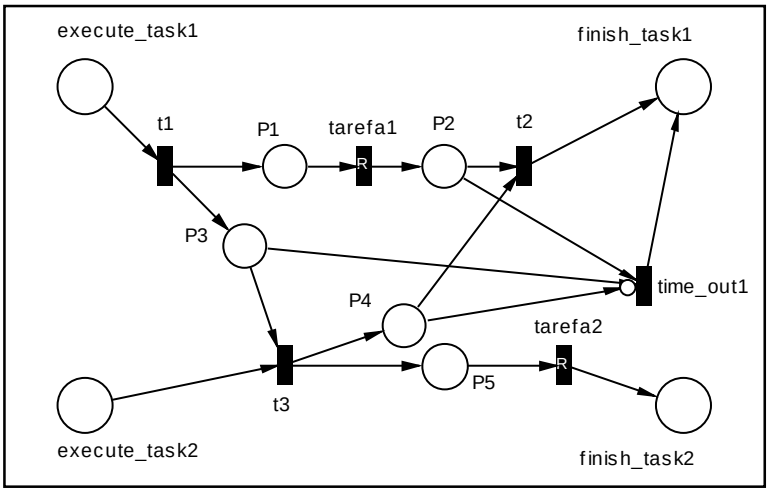

Figura 23: Mecanismo de coordenação para a relação *tarefa 1 sobrepõe tarefa 2*, sem restrição sobre qual tarefa deve terminar antes.

Pelo mecanismo proposto na Figura 23, o disparo da transição *t1* coloca *tokens* em *P1* e *P3*, sendo que este último indica à tarefa 2 que a tarefa 1 já iniciou, permitindo o disparo de *t3*. O disparo da transição t3, por sua vez, coloca *tokens* em *P5* e *P4*, este último indicando à tarefa 1 que a tarefa 2 já iniciou e, portanto, a tarefa 1 pode encerrar (disparo de *t2*). A tarefa 1 também possui um *time-out* para não ficar esperando indefinidamente o início da tarefa 2. O *time-out* retira o *token* de P3 impedindo a execução da tarefa 2, e é inibido pela presença de um *token* em *P4*, indicando que a tarefa 2 está em execução.

Para o modelo da relação em sua forma original, é necessário acrescentar um *AND-join* (constituído por *P6*, *P7* e *t4* – Figura 24) para garantir que a tarefa 2 não termine antes da tarefa 1.

Em ambos os casos, a tarefa 2 pode possuir um *time-out* que a retorna ao seu estado inicial (*token* em *i2* e recurso liberado) após um certo tempo de espera ou um *time-out* que permite a execução da tarefa 2 mesmo contrariando a relação.

6.1.8. Tarefa 2 durante Tarefa 1

Esta relação estabelece que a tarefa 2 deve ocorrer durante o tempo de execução da tarefa 1. Mais uma vez, duas interpretações são possíveis, levando a dois modelos diferentes de mecanismos de coordenação. No primeiro caso, a tarefa 2 pode ser executada apenas uma vez a cada execução da tarefa 1. No segundo caso, a tarefa 2 pode ser executada quantas vezes forem necessárias durante a execução da tarefa 1.

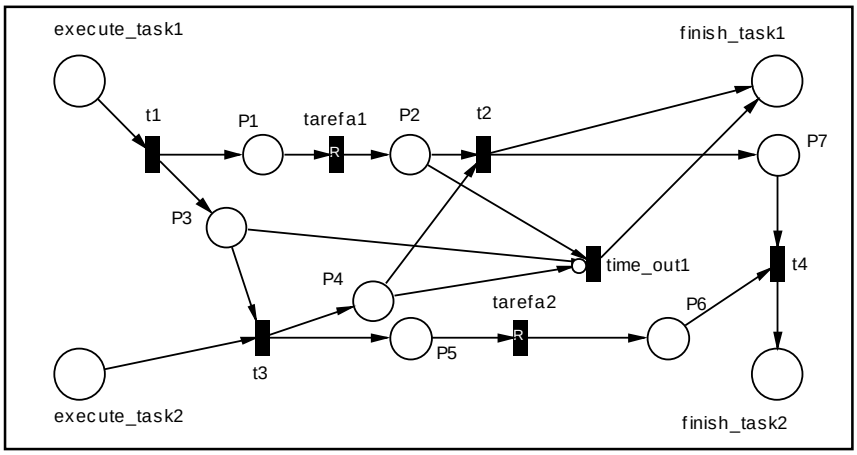

Figura 24: Mecanismo de coordenação para a relação *tarefa 1 sobrepõe tarefa 2*, com a restrição de que a tarefa 1 deve terminar antes.

No modelo da Figura 25, o disparo de *t1* coloca *tokens* em *P1* e *P2*, sendo que este último indica à tarefa 2 que a tarefa 1 já iniciou (habilita o disparo de *tarefa2* uma única vez). Após o disparo de *tarefa 1* (*token* em *P3*), o *token* só será enviado ao *finish_task1* (disparo de *t2*) ao final da tarefa 2 (*token* em *P4*). Para evitar que a tarefa 1 espere indefinidamente, há um *time-out*, inibido pela presença de um *token* em *execute_task2* (i.e., a tarefa 1 não encerra se a tarefa 2 estiver pronta para começar).

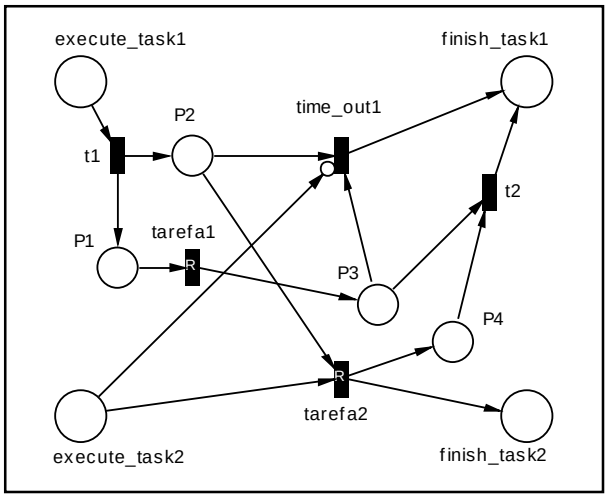

Figura 25: Mecanismo de coordenação para a relação *tarefa 2 durante tarefa 1*, onde a tarefa 2 pode ocorrer apenas uma vez a cada execução da tarefa 1.

Para permitir que a tarefa 2 seja executada mais de uma vez durante a execução da tarefa 1 são necessárias algumas modificações no modelo (Figura 26). A primeira delas é adicionar um arco de retorno da transição *tarefa2* ao lugar *P2*, permitindo futuros disparos de *tarefa2*. Além disso, a transição *t2* passa a ter *P2* como lugar de entrada ao invés de *P4*, que não existe mais. Isso porque, ao finalizar a tarefa 1 (disparo de *t2*) o *token* deve ser retirado de *P2* para impedir novas ocorrências da tarefa 2. Também é necessário fazer com que *t2* seja inibida pela presença de *tokens* em *execute_task2*, impedindo que a tarefa 1 se encerre enquanto a tarefa 2 estiver pronta para ser executada.

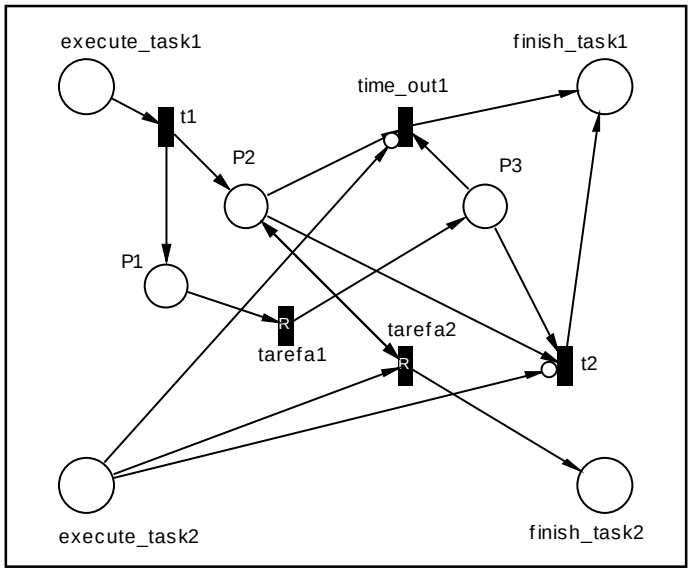

Figura 26: Mecanismo de coordenação para a relação *tarefa 2 durante tarefa 1*, onde a tarefa 2 pode ocorrer várias vezes a cada execução da tarefa 1.

Em ambos os casos, a tarefa 2 pode possuir um *time-out* que a retorna ao seu estado inicial após certo tempo de espera ou um *time-out* que permite a execução da tarefa 2 mesmo contrariando a relação.

6.2. Gerenciamento de recursos

Os mecanismos de coordenação aqui propostos para o gerenciamento de recursos são complementares às dependências temporais e lidam com a distribuição dos recursos entre as tarefas. Há três mecanismos básicos:

- **Divisão de recursos**: um número limitado de recursos precisa ser dividido entre várias tarefas. É o caso mais comum que ocorre, por exemplo, quando vários computadores compartilham uma impressora, uma área de memória, etc.
- **Simultaneidade no uso de recursos**: o recurso só fica disponível se um determinado número de tarefas desejar utilizá-lo simultaneamente. É o caso de uma máquina que precisa de mais de um operador, por exemplo.
- **Volatilidade de recursos**: indica se após o uso, o recurso volta a estar disponível. A impressora é um recurso não volátil, mas uma folha de papel para a impressão é.

Neste contexto, o termo “recurso” está sendo usado de maneira mais ampla que o normalmente usado em *workflows* (Seção 5.2), se referindo não apenas ao agente que

realiza a tarefa, mas também a qualquer artefato necessário à realização da tarefa (folha de papel, por exemplo).

Os mecanismos de gerenciamento de recursos são modelados de forma independente dos mecanismos para as relações temporais, pois eles utilizam os lugares *request_resource*, *assigned_resource* e *release_resource* do modelo expandido de tarefa (Figura 10), e não os lugares *execute_task* e *finish_task*, utilizados nos mecanismos da seção anterior.

As seções seguintes apresentam os componentes da CAV modelados para definir o gerenciamento de recursos entre as tarefas.

6.2.1. Divisão por N

O gerenciador de recursos para a *divisão por N* estabelece que há N instâncias de um recurso disponíveis, de modo que até N tarefas poderão compartilhá-lo simultaneamente. Para o caso particular em que N = 1, estabelece-se a situação bastante comum de exclusão mútua, onde apenas uma tarefa pode utilizar o recurso de cada vez.

O modelo para este tipo de gerenciador é bastante simples, e consiste em um lugar (*Pn*) com N *tokens* representando as instâncias do recurso. Este lugar serve de entrada para uma transição ligando *request_resource* a *assigned_resource*, definindo se há recursos disponíveis ou não para a execução da tarefa. Ao final da tarefa, uma transição saindo de *release_resource* devolve o *token* a *Pn*. A Figura 27 apresenta o modelo para o caso de duas tarefas compartilhando três instâncias do recurso (N = 3).

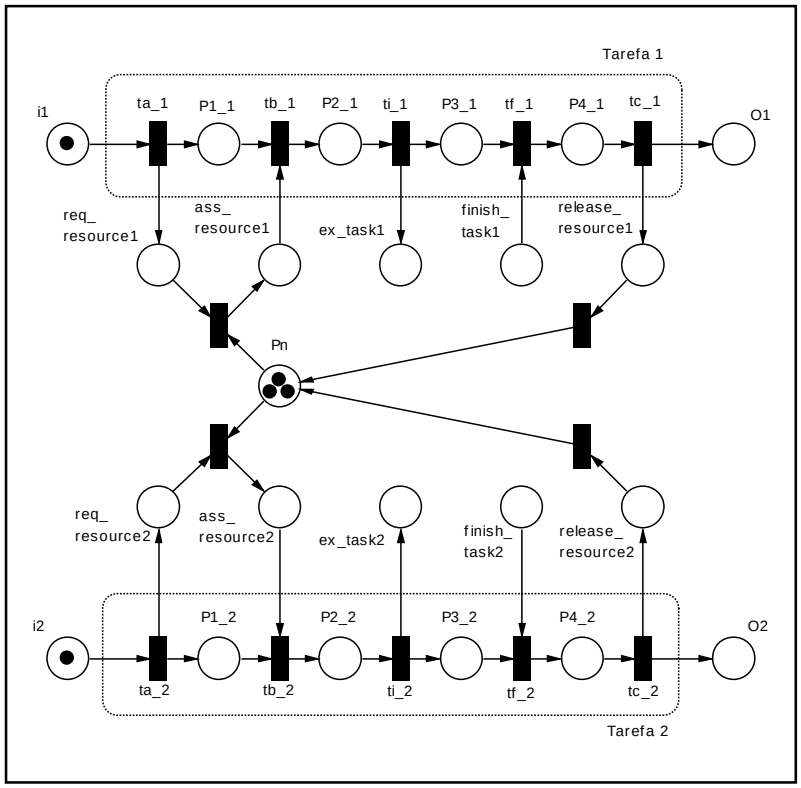

Figura 27: Gerenciador de recursos – *divisão por 3*.

É possível também estabelecer que o recurso será utilizado por duas tarefas consecutivas, de modo que ele seja requisitado pela primeira e liberado pela segunda que, necessariamente, vai ocorrer depois. Este caso é mostrado na Figura 28.

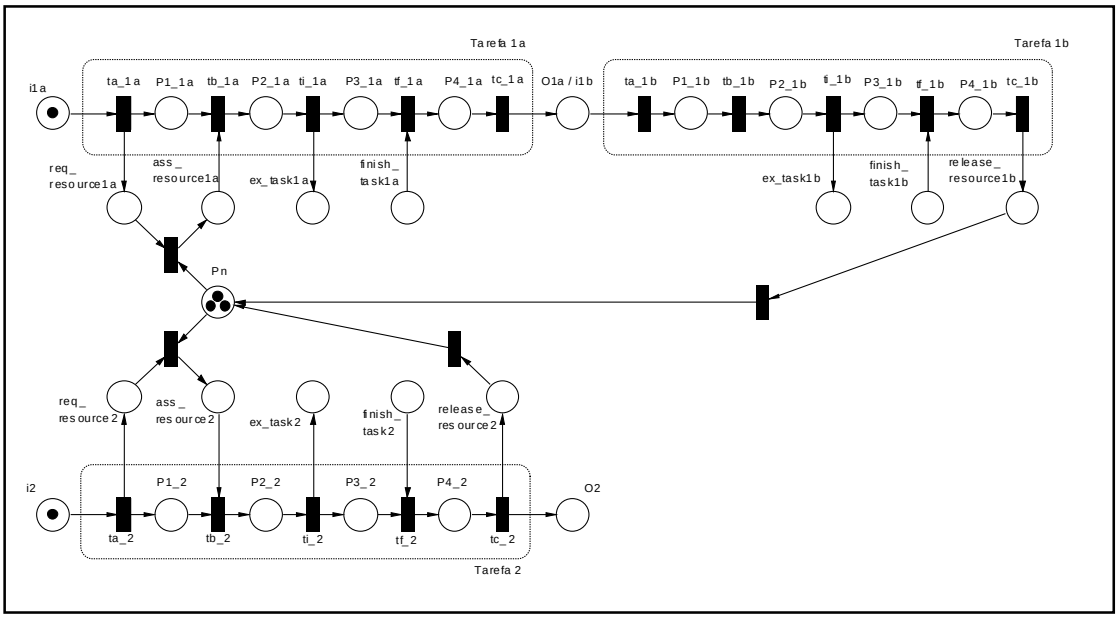

Figura 28: Gerenciador de recursos – *divisão por 3*, com o recurso sendo utilizado por duas tarefas consecutivas.

Para os gerenciadores de recurso, é possível definir *time-outs* partindo de *request_resource* de volta ao lugar de entrada *i*, evitando que uma tarefa fique indefinidamente esperando a alocação de recursos.

6.2.2. Simultaneidade N

Este tipo de gerenciador estabelece que um recurso só é alocado para N tarefas simultaneamente. O modelo (Figura 29) possui um lugar (*Pn*) com N *tokens* que retornam a este lugar ao final das tarefas. *Pn* está ligado a uma transição *t1* por um arco com peso N (no exemplo da Figura 29, N = 2). A transição *t1* só dispara quando houver N *tokens* em *Pn* (indicando N instâncias do recurso) e N *tokens* em *P1* (indicando que N tarefas já requisitaram o recurso). Ao disparar, *t1* envia N *tokens* para *P2*, que os distribui para os N *assigned_resources*. Os lugares *P3* e *P4* servem para impedir que uma mesma tarefa requisite e receba mais de uma vez o recurso (o recurso deve ser alocado para N tarefas diferentes).

No modelo da Figura 29, há apenas duas tarefas exigindo simultaneidade 2, de modo que só há como utilizar o recurso se ambas o estiverem requisitando. Na Figura 30 é mostrado um exemplo onde três tarefas exigem simultaneidade 2. Nesse caso, quando duas das três tarefas requisitar o recurso, ele será alocado a elas. A montagem deste modelo segue o da Figura 29, no qual a tarefa deve possuir um caminho entre *request_resource* e *assigned_resource* (repare a similaridade entre $t2 \rightarrow P3 \rightarrow t3$, $t4 \rightarrow P4 \rightarrow t5$ e $t6 \rightarrow P5 \rightarrow t7$), onde a primeira transição (*t2*, *t4* e *t6*) deve estar ligada a *P1*, indicando que a tarefa requisitou o recurso e o lugar *P2* deve estar ligado à última transição (*t3*, *t5* e *t7*), indicando que o recurso foi alocado. A tarefa também deve devolver o *token* a *Pn* quando ele estiver disponível em *assigned_resource*.

No caso da *simultaneidade N*, também é possível definir um *time-out* devolvendo o *token* de *request_resource* para o lugar de entrada *i*, evitando que uma tarefa espere indefinidamente para que as outras N-1 tarefas requisitem o recurso.

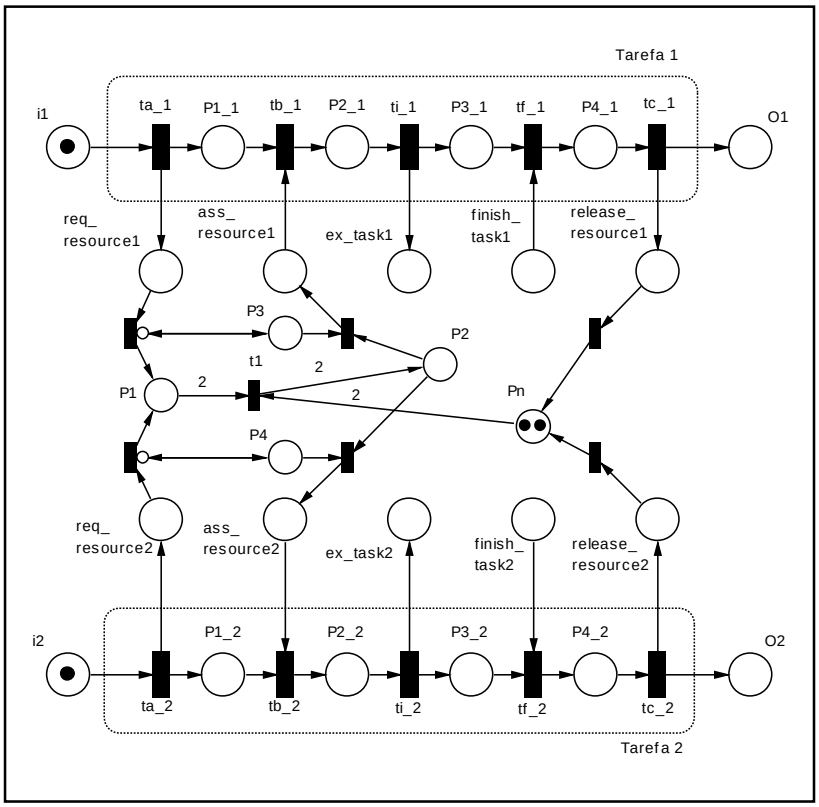

Figura 29: Gerenciador de recursos – *simultaneidade 2*, com duas tarefas podendo requisitar o recurso.

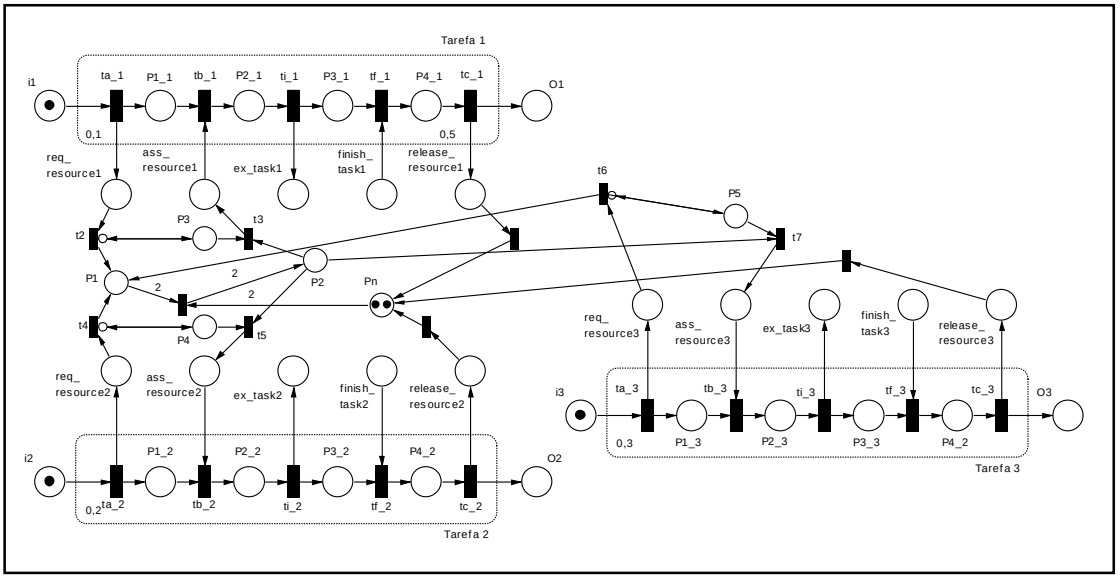

Figura 30: Gerenciador de recursos – *simultaneidade 2*, com três tarefas podendo requisitar o recurso.

6.2.3. Volatilidade N

A volatilidade N de um recurso indica que ele não pode ser reutilizado mais de N vezes por uma tarefa. O modelo é bastante simples, diferenciando-se do modelo da

divisão por N apenas apenas pelo fato do *token* não retornar ao lugar *Pn*. Na Figura 31, este modelo é apresentado, para o caso de N = 2. O lugar *P1*, inicialmente com um *token*, serve para impedir que um novo recurso seja alocado à tarefa antes do término da mesma. O *time-out* só estará habilitado após o término dos recursos (arco inibidor).

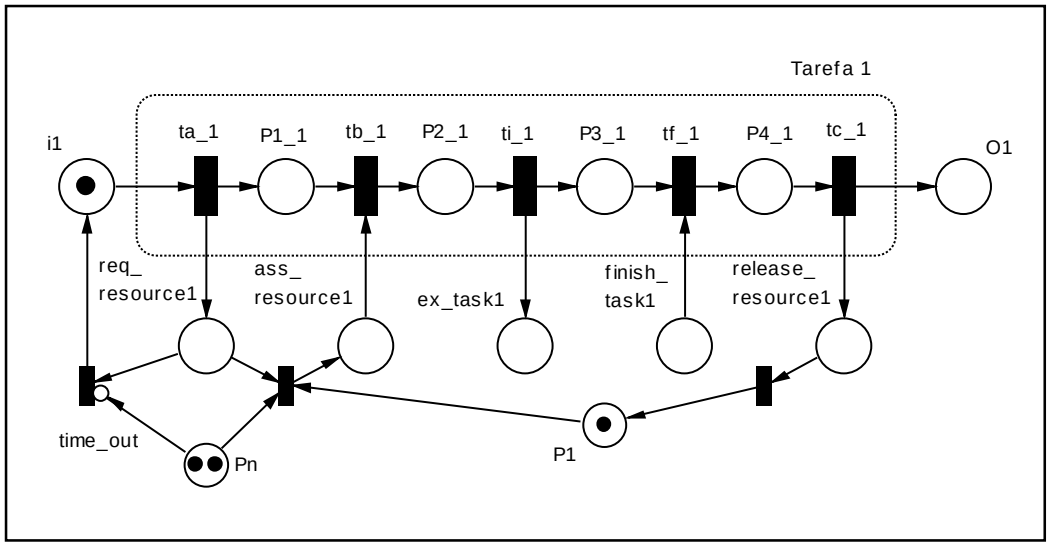

Figura 31: Gerenciador de recursos – *volatilidade 2* (em cada tarefa).

O modelo anterior pode ser modificado de modo a garantir que o recurso não seja utilizado mais de N vezes por qualquer tarefa (e não mais por uma tarefa). Para isto, basta que o lugar *Pn* seja "compartilhado" por todas as tarefas, como no exemplo da Figura 32.

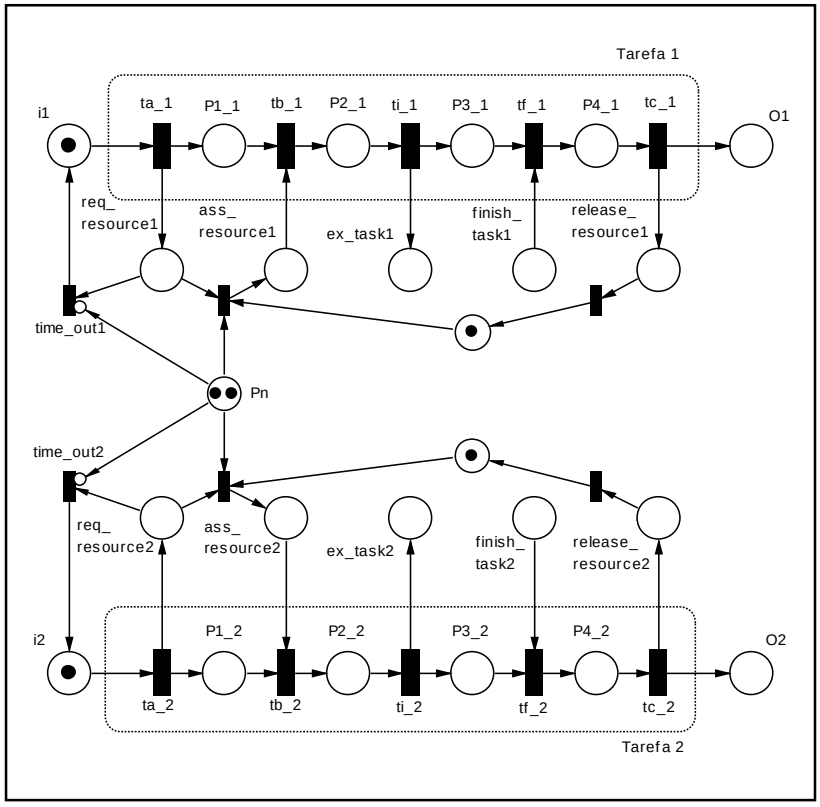

Figura 32: Gerenciador de recursos – *volatilidade 2* (para todas as tarefas).

A partir dos três modelos básicos apresentados (divisão, simultaneidade e volatilidade) é possível criar gerenciadores derivados para as possíveis combinações destes modelos. Isto vai requerer apenas algumas pequenas mudanças nos modelos apresentados, como será visto nas seções seguintes.

6.2.4. Divisão por M + Simultaneidade N

Esta situação estabelece que até M grupos de N tarefas compartilhem um determinado recurso. O modelo é praticamente idêntico ao da *simultaneidade N* (Figuras 29 e 30), apenas colocando $N \times M$ *tokens* no lugar *Pn*. A Figura 33 mostra o modelo para M = 3 e N = 2 ($2 \times 3 = 6$ *tokens* em *Pn*).

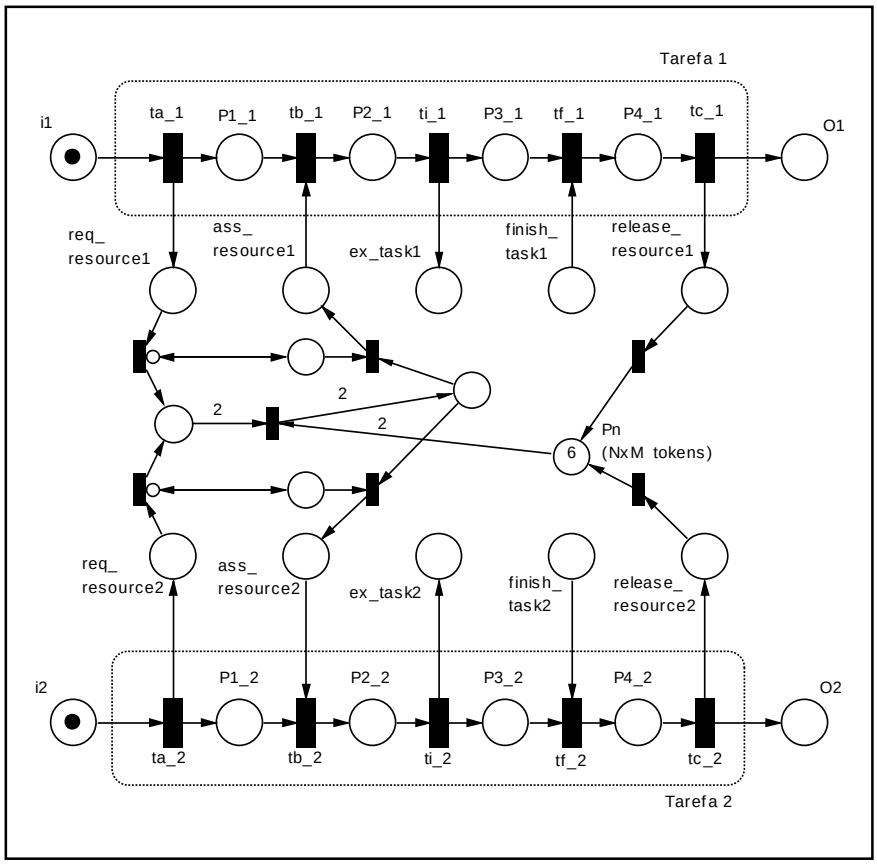

Figura 33: *Divisão por 3 + Simultaneidade 2.*

A análise do modelo através de simulações detectou uma possível situação incorreta no modelo acima. Esta situação ocorre quando uma das duas tarefas que requisitaram o recurso simultaneamente libera o recurso muito antes da outra. Se houver ocorrências sucessivas das tarefas, a tarefa mais rápida pode liberar o recurso duas vezes antes da outra liberar o primeiro. O sistema nesse caso acabaria permitindo, por exemplo, quatro execuções de uma tarefa e duas da outra, e não três de cada, como seria de esperar. Este problema pode ser solucionado por meio de PNs de alto nível (ver Apêndice A).

6.2.5. Divisão por M + Volatilidade N

Nesta situação, até M tarefas podem compartilhar o recurso simultaneamente, e o recurso só pode ser usado até N vezes. O modelo é praticamente o mesmo da divisão

por M (Figura 27), apenas acrescentando o lugar *Pn*, representando a volatilidade do recurso, e os respectivos *time-outs*. Na Figura 34, o modelo está representado para o caso de M = 2 e N = 4.

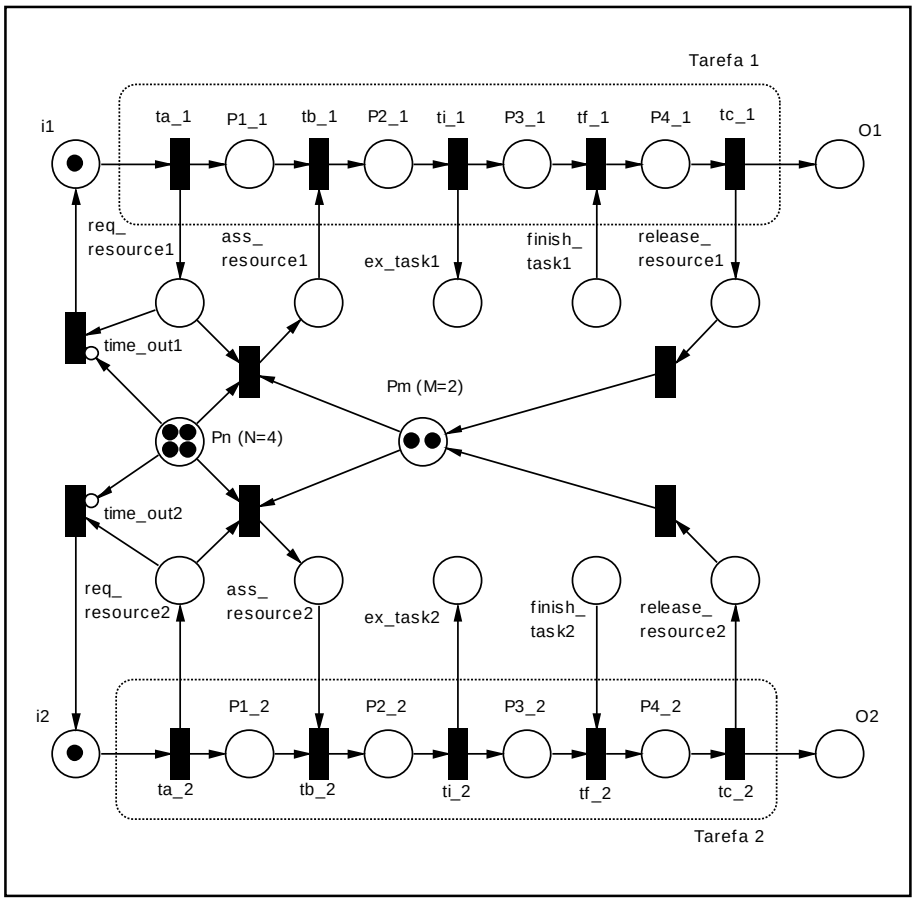

Figura 34: *Divisão por 2 + Volatilidade 4.*

Se a volatilidade for por tarefa, isto é, cada tarefa puder utilizar N vezes o recurso, o modelo deve ser modificado adicionando um lugar *Pn* com N *tokens* para cada tarefa (como na Figura 31).

6.2.6. Simultaneidade N + Volatilidade M

Esta situação define que grupos de N tarefas podem compartilhar um recurso, que pode ser usado até M vezes. O modelo é similar ao da *simultaneidade N* (Figura 29), apenas acrescentando o lugar *Pm*, relativo à volatilidade do recurso, e os respectivos *time-outs*. Na Figura 35, o modelo é apresentado para o caso de N = 2 e M = 4.

M é o número de vezes que o recurso vai ser utilizado, e não o número de grupos de N tarefas que vai utilizá-lo. Se fosse esse o caso, deveria haver M × N *tokens* em *Pm*.

Se a volatilidade for por tarefa, deve haver um lugar *Pm* para cada tarefa.

6.2.7. Divisão por Q + Simultaneidade N + Volatilidade M

Esta situação se difere da anterior porque agora até Q grupos de N tarefas podem compartilhar um recurso simultaneamente. O recurso pode ser utilizado até M vezes. Seguindo a lógica dos dois anteriores, o modelo é similar ao *da divisão por Q +*

simultaneidade N (Figura 33), apenas acrescentando o lugar *Pm* e (opcionalmente) os *time-outs*. O modelo para o caso de Q = 2, N = 2 e M = 6 é mostrado na Figura 36.

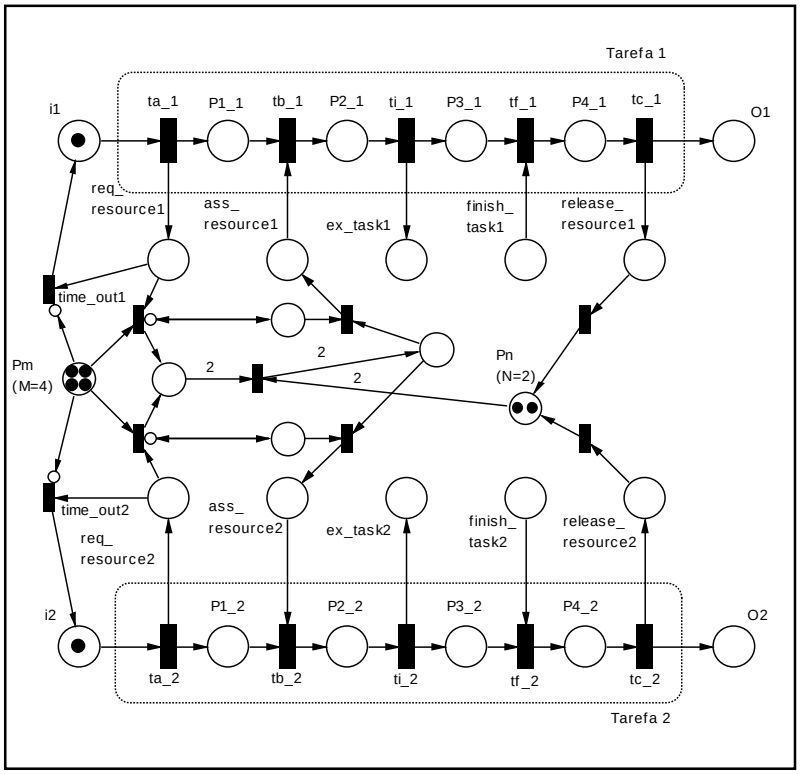

Figura 35: *Simultaneidade 2 + Volatilidade 4.*

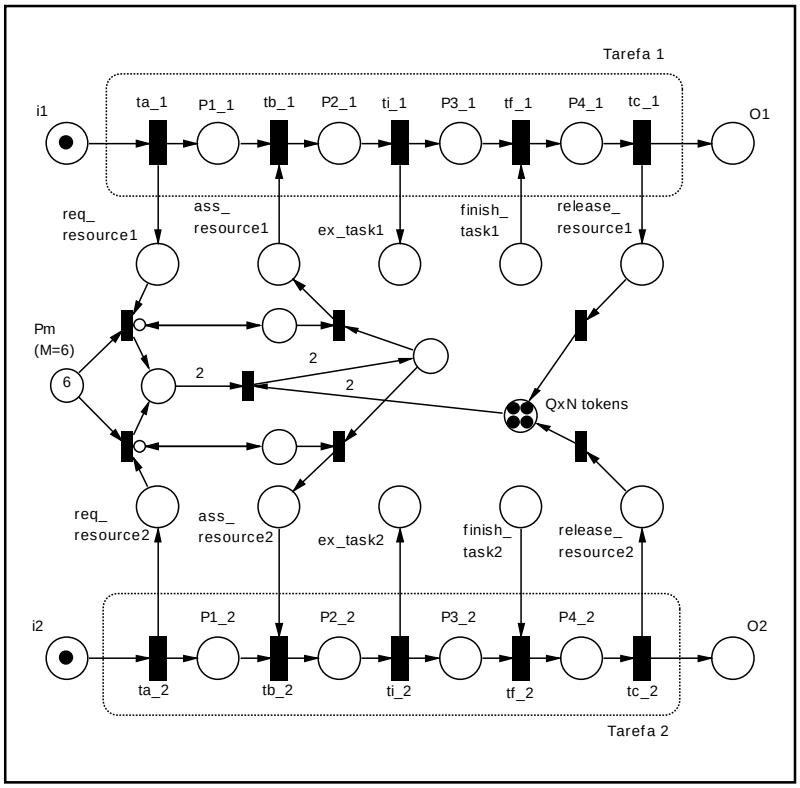

Figura 36: *Divisão por 2 + Simultaneidade 2 + Volatilidade 6.*

6.3. Combinações

Para ilustrar possíveis usos das relações temporais e dos gerenciadores de recursos, serão mostrados nesta seção alguns exemplos de combinações típicas que envolvem ao mesmo tempo dependências temporais e de gerenciamento de recursos. Como poderá ser comprovado nos modelos apresentados, as combinações são realizadas simplesmente pela justaposição das duas subredes (a da relação temporal e a do gerenciador de recurso) no modelo estendido de tarefa.

O primeiro exemplo combina a relação temporal *tarefa 1 igual a tarefa 2* e a *simultaneidade 2*, definindo que duas tarefas devem compartilhar o recurso simultaneamente e também serem realizadas no mesmo intervalo de tempo. Este modelo, mostrado na Figura 37, tem duas subredes distintas entre as tarefas: a da relação temporal entre os lugares *execute_task* e *finish_task* (compare com a Figura 14) e a do gerenciador de recursos, entre os lugares *request_resource*, *assigned_resource* e *release_resource* (ver Figura 29).

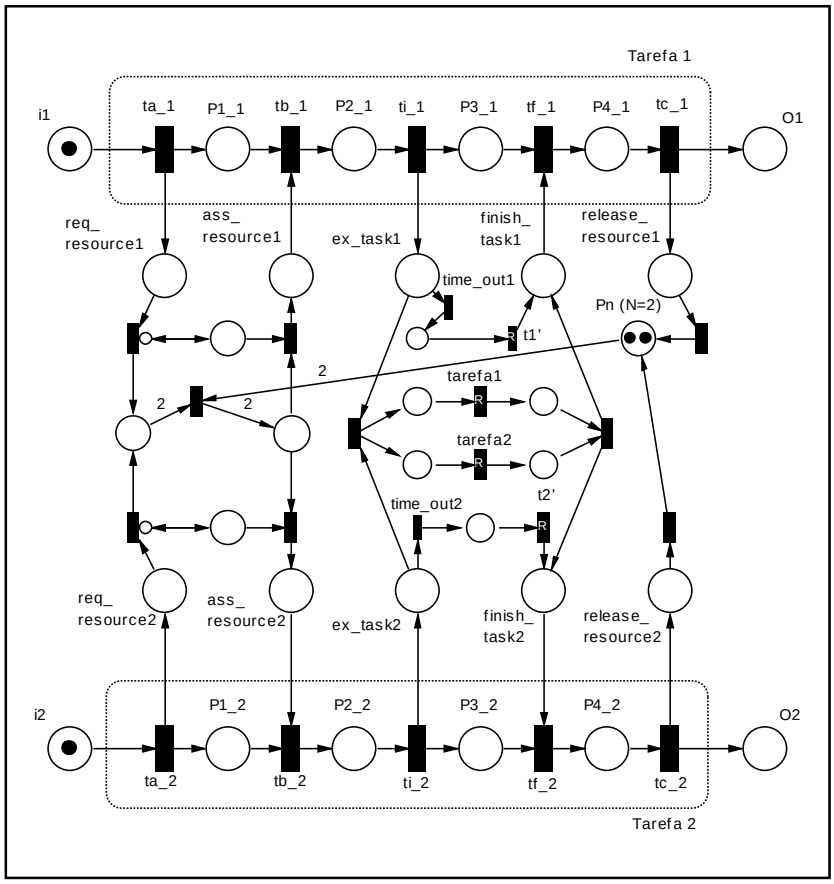

Figura 37: *Tarefa 1 igual a Tarefa 2 + Simultaneidade 2.*

O segundo exemplo combina a relação temporal *tarefa 1 durante tarefa 2* com a *divisão por 2*, estabelecendo que o recurso pode ser compartilhado por duas tarefas, mas uma deve ocorrer durante a execução da outra. Mais uma vez, o modelo para este exemplo (Figura 38) é uma combinação das subredes para a relação temporal (Figura 26) e para o gerenciador de recursos (Figura 27).

Simulações realizadas com o modelo da Figura 38 detectaram uma possível situação indesejada, que ocorre quando a tarefa 2 adquire o recurso e a tarefa 1 não ocorre mais. Como a tarefa 2 só pode ocorrer durante a execução da tarefa 1, ela ficaria bloqueada com o *token* em *execute_task2*. Uma solução parcial, seria colocar o

já comentado *time-out* entre o *execute_task2* e o lugar de entrada *i2*, também liberando o *token* para o *release_resource2* (ver Figura 15).

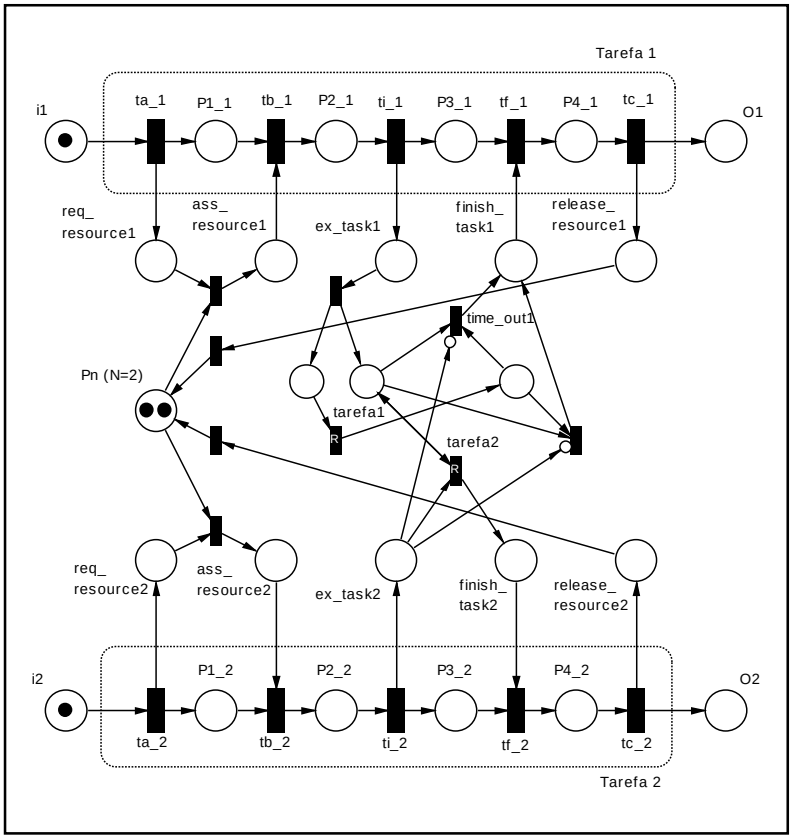

Figura 38: *Tarefa 2 durante Tarefa 1 + Divisão por 2*.

Como último exemplo de possíveis combinações, será mostrado um caso em que as dependências estabelecidas não permitem que nenhuma das tarefas sejam executadas. É o caso em que se combina a relação *tarefa 1 igual a tarefa 2* com a exclusão mútua no acesso ao recurso (*divisão por 1*). Como só há uma instância do recurso disponível, apenas a primeira tarefa que o requisitar conseguirá obtê-lo e fazer o *job token* chegar a *execute_task*. Como a outra tarefa não conseguirá colocar um *job token* em *execute_task*, a transição *t1* (Figura 39) nunca estará habilitada, causando um *deadlock* inevitável. Mais uma vez, a solução passa pelo estabelecimento de *time-outs*, como os das Figuras 14 e 15.

O problema desta última combinação pode ser intuitivamente detectado, mas o modelo em PNs permite a detecção de erros menos previsíveis através da simulação do funcionamento da rede.

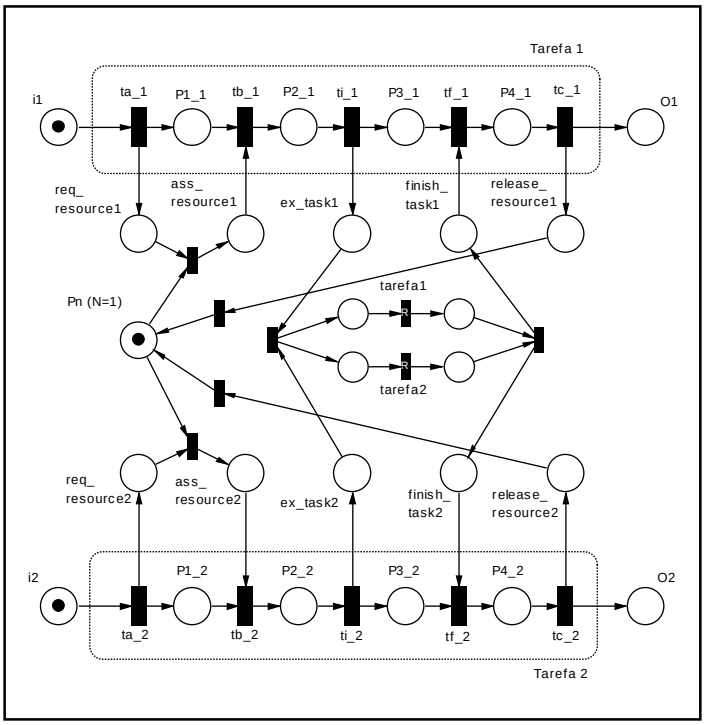

Figura 39: *Tarefa 1 igual a Tarefa 2 + Divisão por 1*. Situação de *deadlock*.

6.4. Implementação

A análise do comportamento de um ambiente virtual modelado por meio dos mecanismos de coordenação da CAV requer a utilização de alguma ferramenta de software. Para automatizar a passagem do modelo no nível de *workflow* para o nível de coordenação, com a expansão das tarefas e a inserção dos mecanismos da CAV é proposto um esquema com três componentes: a ferramenta de simulação de PNs, uma linguagem para a definição das dependências entre as tarefas e um programa capaz de criar uma nova PN para o nível de coordenação a partir da PN do nível de *workflow* e do arquivo com as dependências. A Figura 40 ilustra este esquema.

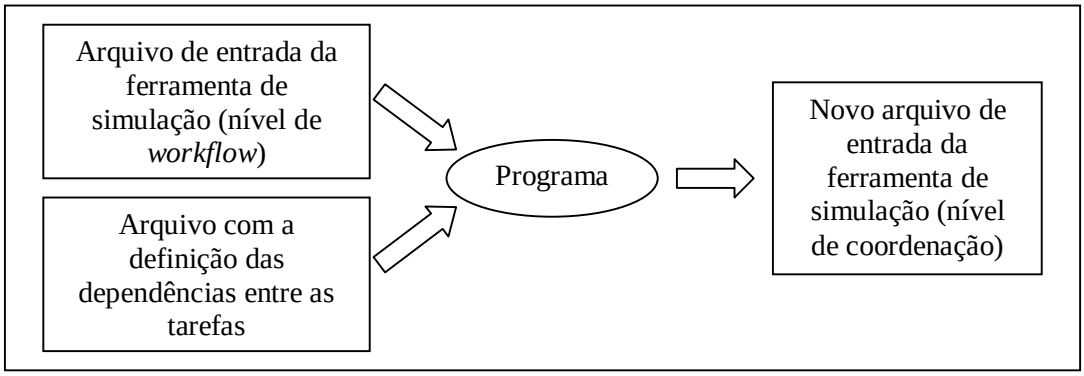

Figura 40: Esquema para a utilização da CAV com um simulador de PNs.

Como os arquivos de entrada variam de acordo com a ferramenta utilizada, o programa deve ser adaptado de acordo a ferramenta desejada. No entanto, a linguagem que define as dependências entre as tarefas independe do simulador usado.

Esta linguagem será apresentada na seção seguinte. Na Seção 6.4.2 será mostrada uma implementação do esquema acima para uma ferramenta de simulação específica.

6.4.1. A linguagem para a definição das dependências

Seguindo o esquema proposto na Figura 40, a utilização da CAV requer a utilização em paralelo de um simulador de PNs. Do ponto de vista do usuário, ele só precisa saber utilizar este simulador (que pode ter uma interface gráfica para facilitar a construção das redes) e escrever um arquivo que define as dependências entre as PNs criadas no simulador. Este arquivo é escrito em uma linguagem bastante simples que será apresentada a seguir.

A linguagem da CAV define, uma a cada linha, todas as dependências existentes entre as tarefas de uma PN previamente criada. Para isso, as tarefas descritas na linguagem devem possuir os mesmos nomes das tarefas (transições) da PN criada. Caso contrário, a dependência será ignorada, pois o programa não terá como saber onde colocar o mecanismo de coordenação para tal dependência. As dependências podem ser listadas no arquivo (uma a cada linha) em qualquer ordem.

A seguir são listadas as possíveis descrições das dependências:

```
equals “<tarefa1>” “<tarefa2>” [<time_out>] [<time_out>]
```

A linha acima define a relação *<tarefa1> igual a <tarefa 2>*, onde <tarefa1> e <tarefa 2>, como já comentado, devem ser os nomes das tarefas (transições) na PN criada no simulador. Os *time-outs* são opcionais e representam, respectivamente, os *time-outs* aplicados sobre a <tarefa1> e a <tarefa2>. O <time_out> pode assumir, nesse caso, três valores: time_outA, time_outB e no_time_out. O time_outA indica o *time-out* que permite a execução das tarefas após um certo tempo, contrariando a relação de dependência (Figura 14). O time_outB indica o *time-out* que retorna a tarefa ao seu estado inicial (Figura 15). Assim, a dependência mostrada na Figura 13, onde nenhum *time-out* é aplicado, é definida como `equals “<tarefa1>” “<tarefa2>”` ou `equals “<tarefa1>” “<tarefa2>” no_time_out no_time_out`. A dependência da Figura 14 é definida como `equals “<tarefa1>” “<tarefa2>” time_outA time_outA`, e a da Figura 15 como `equals “<tarefa1>” “<tarefa2>” time_outB time_outB`. Se fosse desejado colocar um *time-out* do primeiro tipo apenas na primeira tarefa, a definição seria `equals “<tarefa1>” “<tarefa2>” time_outA` ou `equals “<tarefa1>” “<tarefa2>” time_outA no_time_out`. Se fosse desejado colocar o mesmo *time-out* apenas na segunda tarefa, a definição seria `equals “<tarefa1>” “<tarefa2>” no_time_out time_outA`.

A relação *<tarefa1> inicia <tarefa2>* segue o mesmo esquema da relação anterior, mas ela pode existir de duas formas:

```
startsA “<tarefa1>” “<tarefa2>” [<time_out>] [<time_out>]
ou
startsB “<tarefa1>” “<tarefa2>” [<time_out>] [<time_out>]
```

A relação startsA não impõe restrição sobre qual tarefa deve terminar antes (Figura 16), já a relação startsB exige que a <tarefa1> acabe antes da <tarefa2> (Figura 17). Assim como na relação anterior, os *time-outs* são opcionais e podem assumir os três valores anteriormente definidos. A dependência da Figura 16,

portanto, é definida como `startsA “<tarefa1>” “<tarefa2>”` ou `startsA “<tarefa1>” “<tarefa2>” no_time_out no_time_out`. A da Figura 17, por sua vez, é definida como `startsB “<tarefa1>” “<tarefa2>” time_outA time_outA`.

A relação *<tarefa1> finaliza <tarefa2>* também possui duas variantes, a primeira sem a restrição sobre qual tarefa deve começar antes e a segunda impondo que a <tarefa1> só comece após o início da <tarefa2>.

```
finishesA “<tarefa1>” “<tarefa2>” [<time_out>] [<time_out>]
ou
finishesB “<tarefa1>” “<tarefa2>”
```

Os *time-outs* continuam sendo opcionais para o primeiro caso (finishesA), mas só podem ser do tipo que realiza a tarefa mesmo contrariando a relação (Figura 18). Por isso, <time_out> pode assumir apenas os valores time_out ou no_time_out (time_outA e time_outB também podem ser aceitos pelo programa para diminuir a possibilidade de erro dos usuários, mas devem ser entendidos como idênticos a time_out nesse caso). Assim, a Figura 18 é definida como `finishesA “<tarefa1>” “<tarefa2>” time_out time_out`. A segunda variação da relação (finishesB) não aceita nenhum tipo de *time-out* (qualquer *time-out* definido após esta relação deve ser ignorado). A Figura 19 é, portanto, definida como `finishesB “<tarefa1>” “<tarefa2>”`.

A relação *<tarefa2> depois de <tarefa1>* também possui duas variantes. A primeira delas (afterA) permite apenas uma execução da <tarefa2> a cada execução da <tarefa1> (Figura 20a), a segunda variante (afterB) permite várias execuções da <tarefa2> (Figura 20b).

```
afterA “<tarefa2>” “<tarefa1>” [<time_out>]
ou
afterB “<tarefa2>” “<tarefa1>” [<time_out>]
```

Como a <tarefa1> (que ocorre antes) nunca será bloqueada, ela não tem necessidade de *time-out*. Portanto, o único *time-out* que pode existir nesta relação se refere à <tarefa2>, e assume os valores time_outA, time_outB ou no_time_out (se houver um segundo *time-out* na definição da relação, ele deve ser ignorado).

De maneira semelhante, a relação *<tarefa1> antes de <tarefa2>* só possui o *time-out* referente à <tarefa1> (que pode assumir os mesmos valores da relação anterior), pois a <tarefa2> nunca é bloqueada.

```
before “<tarefa1>” “<tarefa2>” [<time_out>]
```

Esta relação não tem uma segunda variante. A Figura 21 é definida como `before “<tarefa1>” “<tarefa2>” time_outB`, pois a <tarefa1> utiliza o *time-out* que a retorna ao seu estado inicial.

A relação *<tarefa1> encontra <tarefa2>* também não possui variantes e pode ter apenas o time-out referente à <tarefa2>, pois o da <tarefa1> já está pré-definido no modelo da Figura 22.

```
meets “<tarefa1>” “<tarefa2>” [<time_out>]
```

Este *time_out* pode assumir os mesmos três valores dos casos anteriores e, para diminuir a possibilidade de erro na definição da dependência, sempre que houver dois *time-outs* definidos, só o segundo será considerado (os casos anteriores só consideram o primeiro porque eles se referem à primeira tarefa; neste caso, ele se refere à segunda tarefa). A Figura 22 é definida como `meets “<tarefa1>” “<tarefa2>”`.

A relação <*tarefa1*> *sobrepõe* <*tarefa2*> tem duas variantes, a primeira não impondo restrições sobre qual tarefa deve terminar antes, e a segunda exigindo que a <tarefa1> termine antes da <tarefa2>.

```
overlapsA “<tarefa1>” “<tarefa2>” [<time_out>]
ou
overlapsB “<tarefa1>” “<tarefa2>” [<time_out>]
```

Assim como no caso anterior, o modelo para estes mecanismos de coordenação (Figuras 23 e 24) já definem um *time-out* para a <tarefa1>. Portanto, apenas o *time-out* para a segunda tarefa deve ser considerado (e pode assumir os mesmos três valores dos casos anteriores).

A relação <*tarefa2*> *durante* <*tarefa1*> também tem duas variantes, a primeira permitindo que a <tarefa2> seja executa uma única vez a cada execução da <tarefa1>, e a segunda permitindo várias execuções da <tarefa2> em cada execução da <tarefa1>.

```
duringA “<tarefa2>” “<tarefa1>” [<time_out>]
ou
duringB “<tarefa2>” “<tarefa1>” [<time_out>]
```

O *time-out* para a <tarefa1> já está pré-definido no modelo (Figuras 25 e 26). Portanto, o único *time-out* que pode aparecer na definição desta relação se refere à <tarefa2> (um segundo *time-out* seria ignorado). Mais uma vez, o <time-out> pode assumir os valores time_outA, time_outB ou no_time_out.

As dependências envolvendo recursos também são definidas de maneira semelhante, mas só podem possuir um tipo de *time-out* (o que retorna as tarefas ao seu estado inicial – time_outB). Para diminuir a possibilidade de erros, tanto time_outA, quanto time_outB e time_out devem ser entendidos como este tipo de *time-out* nos gerenciadores de recursos.

A relação *divisão por N* possui, em primeiro lugar, o valor de N (maior que zero), seguido de uma lista de tarefas que compartilham o recurso. Depois, opcionalmente, pode haver uma lista com os *time-outs* associados a cada uma delas.

```
div <N> “<tarefa1>” [“<tarefaX>”]^k [<time_out>] [<time_out>]^k
```

A Figura 27 (duas tarefas compartilhando três recursos, sem *time-outs*) é definida como `div 3 “<tarefa1>” “<tarefa2>”`. Para definir, por exemplo, uma situação onde três tarefas compartilham dois recursos, onde apenas a última tarefa possui *time-out*, se escreveria `div 2 “<tarefa1>” “<tarefa2>” “<tarefa3>” no_time_out no_time_out time_out`.

Este gerenciador de recursos pode apresentar uma variação (Figura 28), permitindo que o recurso seja utilizado por duas tarefas consecutivas, sendo

requisitado pela primeira e liberado pela segunda (cabe ao projetista da PN no nível de *workflow* garantir que a segunda tarefa vai ocorrer em algum momento depois da primeira; caso contrário o recurso se perderá). Este tipo de relação é indicado colocando um "&" entre estas duas tarefas. Para o caso da Figura 28, a definição é `div 3 "<tarefa1a>"&"<tarefa1b>" "<tarefa2>"`.

A *simultaneidade N* é defnida de maneira semelhante à divisão por N.

```
sim <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
```

A Figura 29, onde ocorre simultaneidade 2 entre duas tarefas é definida como `sim 2 "<tarefa1>" "<tarefa2>"`. A Figura 30 (simultaneidade 2 entre três tarefas) é definida como `sim 2 "<tarefa1>" "<tarefa2>" "<tarefa3>"`.

A *volatilidade* de recursos apresenta duas variações. No primeiro caso (volA), a volatilidade se refere a cada tarefa (Figura 31) e no segundo caso (volB), a todas as tarefas (Figura 32).

```
volA <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
ou
volB <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
```

Para o mecanismo da Figura 31, a definição é `volA 2 "<tarefa1>" time_out` e para o da Figura 32, `volB 2 "<tarefa1>" "<tarefa2>" time_out time_out`.

As combinações entre estes três mecanismos básicos são definidas colocando um "_" entre os nomes de cada um deles, e definindo os dois parâmetros necessários (M e N). Por exemplo, a divisão por M + simultaneidade N é:

```
div_sim <M> <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
```

A Figura 33 (divisão por 3 + simultaneidade 2 entre duas tarefas, sem *time-outs*) é definida como `div_sim 3 2 "<tarefa1>" "<tarefa2>"`.

Similarmente, podem ser definidas as seguintes relações:

```
div_volA <M> <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
ou
div_volB <M> <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k

sim_volA <M> <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
ou
sim_volB <M> <N> "<tarefa1>" ["<tarefaX>"]^k [<time_out>] [<time_out>]^k
```

Para o exemplo da Figura 34 (divisão por 2 + volatilidade 4 entre 2 tarefas, com *time-outs*), a definição é `div_volB 2 4 “<tarefa1>” “<tarefa2>” time_out time_out`. Para a Figura 35 (simultaneidade 2 + volatilidade 4 entre 2 tarefas, com *time-outs*), a definição é `sim_volB 2 4 “<tarefa1>” “<tarefa2>” time_out time_out`.

Seguindo a lógica, a combinação das três relações básicas (*divisão por Q* + *simultaneidade N* + *volatilidade M*) é definida como:

```
div_sim_volA <Q> <N> <M> “<tarefa1>” [“<tarefaX>”]^k [<time_out>] [<time_out>]^k
```

ou

```
div_sim_volB <Q> <N> <M> “<tarefa1>” [“<tarefaX>”]^k [<time_out>] [<time_out>]^k
```

O exemplo da Figura 36 (divisão por 2 + simultaneidade 2 + volatilidade 6, sem *time-outs*) é definido como `div_sim_volB 2 2 6 “<tarefa1>” “<tarefa2>”`.

Os exemplos apresentados na Seção 6.3, envolvendo combinações de relações temporais e de gerenciamento de recursos, exigem duas linhas no arquivo de definições de dependências, como mostrado a seguir.

O exemplo da Figura 37, que combina as relações *igual a* e *simultaneidade 2* entre duas tarefas tem o seguinte arquivo de definição das dependências (substituindo <tarefa1> e <tarefa2> pelo nome das transições que representam as tarefas na PN do nível de *workflow*):

```
equals “<tarefa1>” “<tarefa2>” time_outA time_outA
sim 2 “<tarefa1>” “<tarefa2>”
```

Para o exemplo que combina *durante* e *divisão por 2* (Figura 38), o arquivo é:

```
duringB “<tarefa1>” “<tarefa2>”
div 2 “<tarefa1>” “<tarefa2>”
```

Finalmente, para o exemplo que combina *igual a* e *divisão por 1* (Figura 39) tem-se:

```
equals “<tarefa1>” “<tarefa2>”
div 1 “<tarefa1>” “<tarefa2>”
```

A BNF a seguir define formalmente a linguagem.

```
<dependency> ::= <temporal_dep> | <resource_dep>;

<temporal_dep> ::= <temp_name> “<t1>” “<t2>”
                     [<temp_time_out>] [<temp_time_out>];
<temp_name> ::= equals | startsA | startsB | finishesA |
                finishesB | afterA | afterB | before |
                meets | overlapsA | overlapsB |
                duringA | duringB;
<t1> ::= <string>;
<t2> ::= <string>;
```

```
<temp_time_out> :== time_outA | time_outB | no_time_out;

<resource_dep> ::= <1_dep> | <2_dep> | <3_dep>;
<1_dep> ::= <1_dep_name> <parameter> “<t1>” [{“<tn>”}^n]
            [<resource_time_out>] [{<resource_time_out>}^n];
<1_dep_name> ::= div | sim | volA | volB;
<parameter> ::= <integer>;
<t1> ::= <string>;
<tn> ::= <string>;
<resource_time_out> ::= time_out | no_time_out;
<2_dep> ::= <2_dep_name> {<parameter>}^2 “<t1>” [{“<tn>”}^n]
           [<resource_time_out>] [{<resource_time_out>}^n];
<2_dep_name> ::= div_sim | div_volA | div_volB |
                 sim_volA | sim_volB;
<3_dep> ::= <3_dep_name> {<parameter>}^3 “<t1>” [{“<tn>”}^n]
           [<resource_time_out>] [{<resource_time_out>}^n];
<3_dep_name> ::= div_sim_volA | div_sim_volB;
```

A linguagem aqui apresentada é independente da ferramenta de simulação e extensível, no sentido de que, quando novas dependências forem determinadas, e novos mecanismos de coordenação modelados, novas definições podem ser agregadas a ela. Para automatizar a criação das novas dependências, o programa que converterá o arquivo inicial do simulador em um arquivo com as dependências também precisa ser estendido. Este programa será estudado a seguir.

6.4.2. O programa conversor

Este programa é responsável por converter o arquivo de entrada do simulador de PNs no nível de *workflow* para um arquivo no nível de coordenação, inserindo os mecanismos de coordenação da CAV. Para tanto, o programa é dividido em cinco etapas:

1. Leitura do arquivo de dependências, na linguagem descrita na seção anterior e criação de estruturas de dados que representem as dependências.
2. Leitura do arquivo de entrada da ferramenta de simulação e criação de estruturas de dados que representem a PN.
3. Expansão das transições envolvidas. Isto é, cada transição que possui uma dependência será transformada numa estrutura como a da Figura 10.
4. Inserção dos mecanismos de coordenação entre as transições envolvidas.
5. Criação do novo arquivo de entrada do simulador.

A única etapa totalmente independente da ferramenta de simulação utilizada é a primeira. Ela consiste em interpretar o arquivo de dependências para a criação de duas estruturas: lTE e lRel. A lTE é a lista das transições envolvidas, e serve para definir quais transições serão expandidas na etapa 3. A lRel é a lista completa das relações, e serve para determinar que mecanismos serão utilizados na etapa 4. Esta primeira etapa do programa está implementada (em Java) pela classe Dep_File.class.

As demais etapas são dependentes da ferramenta de simulação. Como exemplo, a próxima seção mostra a implementação da CAV para a ferramenta Visual Simnet.

6.4.3. Utilizando a ferramenta Visual Simnet

O Visual Simnet [Garbe 97] é uma ferramenta *freeware*, com capacidade de modelar e analisar PNs convencionais e estocásticas (disparos das transições determinados por funções de probabilidade). Ele possui editor gráfico para a modelagem das PNs e vários recursos para análise, incluindo simulação animada, análise estrutural, de desempenho, de distribuição (redes estocásticas), *coverability tree* e análise de estruturas mortas.

Além do fato de ser gratuito e possuir boa capacidade de análise, o Visual Simnet foi escolhido porque tem um formato textual bastante simples para a definição das PNs (MoDeL – *Model Description Language* [Garbe 97]) e permite a exportação para outras ferramentas mais poderosas (por exemplo o INA – *Integrated Net Analyzer* [Starke 99]).

A segunda etapa do programa (descrita na seção anterior), lê um arquivo em MoDeL e constrói uma estrutura de dados que representa a PN. Esta estrutura (classe Inp_File), possui uma lista de lugares (lPlaces), uma lista de transições (lTrans) e uma lista com outros elementos do arquivo (lOthers), tais como comentários, *labels*, etc. Cada lugar da lista possui, dentre outras propriedades, a lista de *tokens*, definindo a marcação inicial da rede. Cada transição possui, dentre outras características, uma lista com os arcos de entrada (preArcs) e outra com os de saída (posArcs).

Com a lista de transições envolvidas (lTE, definida na etapa 1) e a estrutura definida na etapa 2, é possível realizar a etapa 3, que consiste em expandir as transições envolvidas. Para simplificar as PNs resultantes, apenas as transições envolvidas tanto em dependências temporais quanto de gerenciamento de recursos são expandidas conforme o modelo da Figura 10. É possível simplificar o modelo da Figura 10 se as transições estão apenas envolvidas em dependências temporais ou de gerenciamento de recursos (Figura 41). Para isso, a lTE também possui informação sobre o tipo de relação em que cada transição está envolvida.

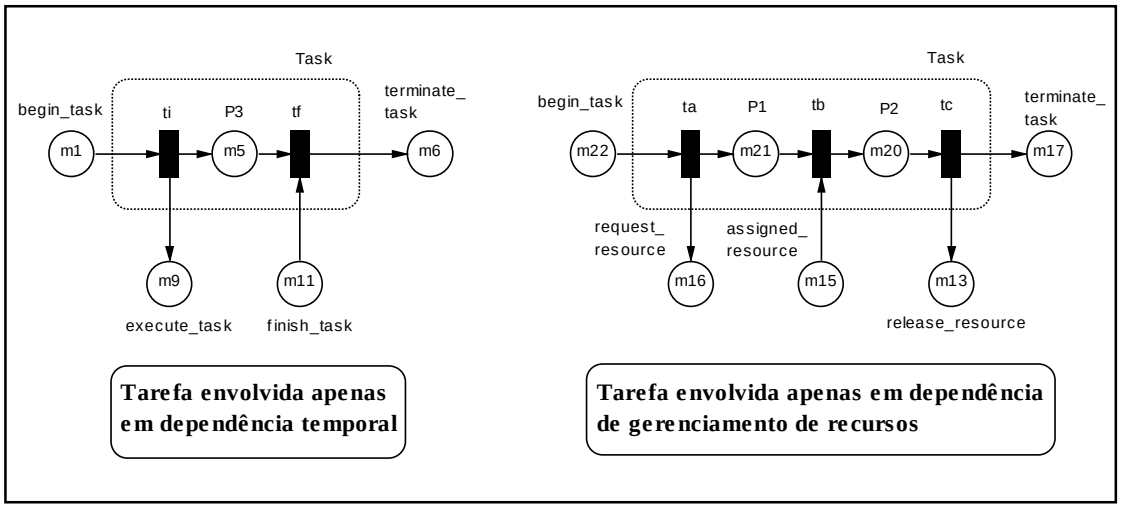

Figura 41: Simplificações do modelo de tarefa expandida.

A classe Expand_Tasks é responsável por esta parte do algoritmo. Ela não gera novas estruturas de dados, apenas altera as já existentes (lPlaces e lTrans) para a expansão das tarefas e adequação das coordenadas x e y dos elementos da rede expandida.

Uma vez expandidas as transições, o programa inicia a etapa 4 a partir da lista completa das relações (lRel), definida na etapa 1. A classe Insert_Coordination,

responsável por esta etapa do programa, conhece a estrutura dos mecanismos de coordenação e as insere entre as transições expandidas, alterando as estruturas lPlaces e lTrans.

Após as alterações nas estruturas (etapas 3 e 4), o programa escreve, a partir das mesmas, um arquivo de saída na linguagem MoDeL que representa o nível de coordenação do modelo, com as tarefas expandidas e os mecanismos de coordenação inseridos (a classe Out_File realiza esta última etapa do programa). Este arquivo de saída pode ser usado no Visual Simnet para a simulação e análise do ambiente virtual.

Longe de abranger todas as possíveis dependências entre as tarefas, a biblioteca CAV pretende apenas fornecer um conjunto de mecanismos de coordenação modelados em PNs (e apresentados nesta seção) para uma série de dependências típicas que podem acontecer freqüentemente entre as tarefas. A idéia principal é, por enquanto, prover um número limitado de mecanismos que serão testados em situações de interação entre atores em animações e entre participantes de CVEs. A partir do uso, surgirão naturalmente novas dependências e mecanismos de coordenação que poderão ser agregados a esta biblioteca. O projeto da CAV optou pela extensibilidade e não pela completude da mesma.

A próxima seção mostrará exemplos de uso dos componentes da CAV, partindo do nível de *workflow* para o nível de coordenação (Figura 11).

7. Exemplos de utilização da CAV

Esta seção apresentará alguns exemplos mostrando a utilização dos componentes da CAV em diversas situações modeladas por PNs.

Em todos os casos, o "animador" (ou projetista do ambiente virtual) constrói as WF-Nets que definem as possíveis seqüências de tarefas para cada "ator" (ou participante do ambiente virtual) e estabelece as dependências existentes entre as tarefas. Este é o modelo no nível de *workflow*, onde o animador trabalha.

A partir das dependências estabelecidas no nível de *workflow*, as tarefas interdependentes são expandidas conforme o modelo da Figura 10, e os mecanismos da CAV são utilizados para coordenar as dependências entre elas. As tarefas expandidas juntamente com os mecanismos da CAV colocados entre elas constituem o modelo no nível de coordenação, que pode ser submetido às diversas formas de análise (verificação, validação e desempenho - ver Seção 4.2).

O modelo no nível de especificação das tarefas implica na expansão das transições que representam a lógica das tarefas. No entanto, como já comentado, este trabalho se restringe aos dois níveis mais altos.

7.1. Exemplo genérico

O primeiro exemplo não se preocupa em modelar uma situação específica de animações ou ambientes virtuais. Ele parte de WF-Nets desprovidas de qualquer interpretação real para mostrar como utilizar os mecanismos da CAV.

A Figura 42 mostra o nível de *workflow* do modelo. Existem duas WF-Nets (que poderiam representar dois atores em uma animação, por exemplo), a primeira delas (WF-Net A) se inicia com um roteamento condicional exclusivo (*OR-split* e depois *OR-join*) e a segunda (WF-Net B) se inicia com um roteamento paralelo (*AND-split* e depois *AND-join*). Ambas terminam com roteamentos seqüenciais. As dependências entre as tarefas também estão representadas na figura. Repare que as dependências podem ocorrer entre tarefas de WF-Nets diferentes (*T1a sobrepõe T1b*, por exemplo) ou entre tarefas de uma mesma WF-Net (*T3b durante T2b*).

O arquivo que define estas dependências é mostrado a seguir:

```
overlapsA "T1a" "T1b"
duringB "T3b" "T2b"
div 1 "T5a" "T4b"
sim 2 "T6a" "T5b"
```

Com o uso da CAV, o trabalho do projetista termina no nível de workflow. A passagem para o nível de coordenação é "imediata", inserindo os mecanismos de coordenação entre as tarefas interdependentes. O modelo no nível de coordenação é mostrado na Figura 43. Apesar de parecer complexa, a rede mostrada na Figura 43 é bastante modular e facilmente montada a partir da rede anterior (Figura 42) e dos componentes reutilizáveis da CAV.

A partir do modelo, foi possível fazer uma série de análises pertinentes. A análise de verificação através da *coverability tree* revelou duas situações possíveis de *deadlock*.

A primeira delas ocorre quando a WF-Net A opta por seguir o caminho passando por T2a. Nesse caso, T1a não será executada e, consequentemente, T1b também não, pois ela precisa se sobrepor a T1a. A WF-Net B, portanto, fica bloqueada antes da

realização de T1b e a WF-Net A será bloqueada mais à frente, por ocasião da tarefa T6a, que exige simultaneidade 2 com a T5b. O *time-out* fazendo com que T1b volte ao seu estado inicial não resolveria, pois a WF-Net B não oferece um caminho alternativo que não passe por T1b (se a tarefa bloqueada fosse a T1a, isto resolveria, já que há o caminho passando por T2a). A única solução para evitar a possibilidade deste *deadlock* é alterar o modelo no nível de *workflow* criando, por exemplo, uma dependência adicional *T2a sobrepõe T1b*, de modo que T1b seja executada independente do caminho seguido na WF-Net.

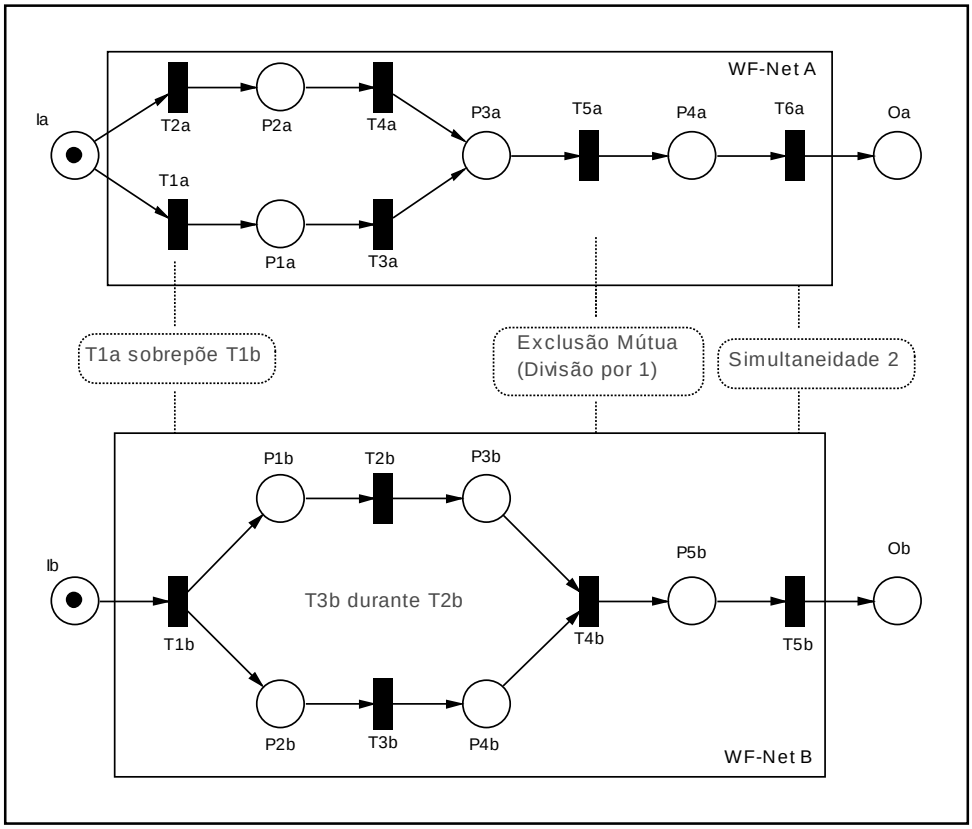

Figura 42: Exemplo 1, nível de *workflow.*

A segunda situação de *deadlock* detectada ocorre na WF-Net B, quando a tarefa T2b ativa o *time-out* , não esperando por T3b. Nesse caso, T4b não será executada (*AND-join*) e a WF-Net B fica bloqueada. Como conseqüência, a WF-Net também ficará bloqueada antes de T6a, que exige simultaneidade 2 com T5b. A análise de validação, feita com simulações de casos fictícios, revelou, no entanto, que se o *time-out* de T2b tiver um valor suficientemente alto, ele nunca será disparado antes da execução de T3b, evitando este *deadlock.*

A análise de desempenho não foi realizada para este exemplo, pois não há variáveis a serem medidas, já que este modelo não foi baseado em um exemplo concreto.

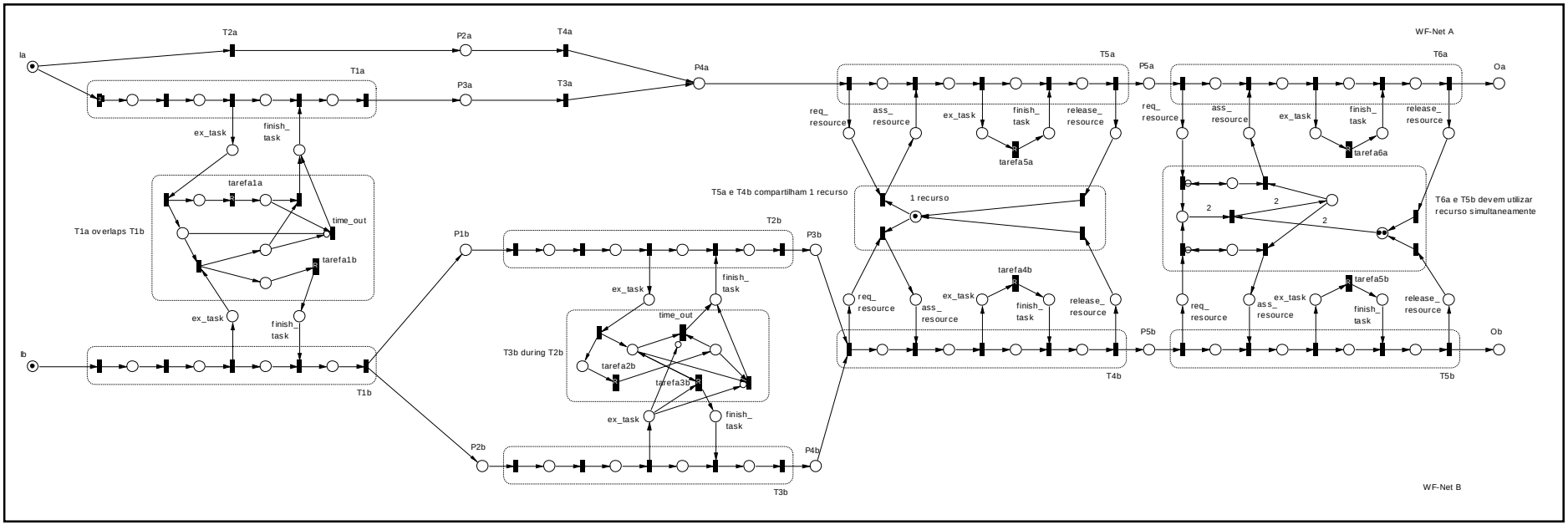

Figura 43: Exemplo 1, nível de coordenação.

7.2. Exemplo para uma situação típica de workflow

O exemplo mostrado nesta seção ainda não trata de ambientes virtuais, mas ilustra uma situação típica dos chamados workflows interorganizacionais [Anderson 99], que ultrapassam os limites de uma única organização, sendo compostos de várias organizações trabalhando de maneira cooperativa.

No exemplo a seguir é representada a situação em que um consumidor contacta um loja virtual para comprar alguma coisa. O ambiente retratado é composto de três "organizações" com workflows independentes: o consumidor, a loja e o produtor (a loja virtual é apenas um intermediário, e precisa adquirir os produtos dos fabricantes). O nível de workflow deste ambiente é mostrado na Figura 44.

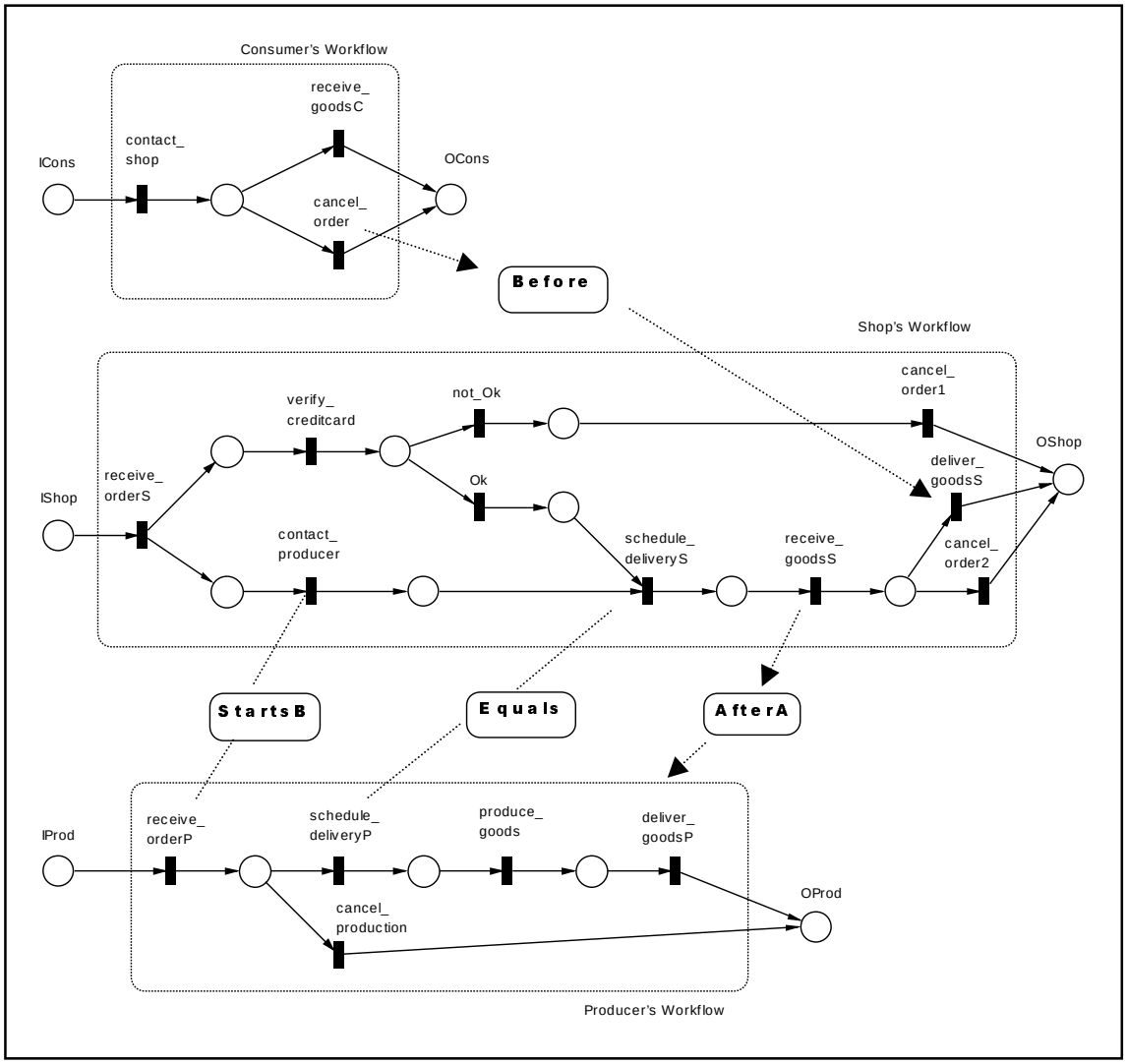

Figura 44: Nível de workflow para o exemplo da loja virtual.

O workflow do consumidor é bastante simples. Ele deve iniciar o processo contactando a loja para pedir o produto que deseja comprar, e então esperar pela chegada do mesmo ou cancelar o pedido. No entanto, o pedido só pode ser cancelado se o produto ainda não tiver sido enviado pela loja. Isto define a relação *cancel_order antes de deliver_goodsS*. A loja, após receber o pedido, inicia duas atividades paralelas: verificação do cartão de crédito do consumidor e contato com o fabricante do produto. Se houver qualquer problema com o cartão de crédito, a loja cancela o

pedido. Se não houver problema, a loja pode agendar a entrega do produto, atividade que deve ocorrer simultaneamente ao agendamento feito pelo fabricante (*schedule_deliveryS igual a schedule_deliveryP*). Se houver problema com o cartão de crédito, *schedule_deliveryS* não será executada e, consequentemente, *schedule_deliveryP* também não. Usando um time-out na tarefa *schedule_deliveryP*, o produtor pode ter a opção de cancelar a produção. Finalmente, há também a relação de que a loja receberá o produto depois do fabricante enviá-lo.

Para construir o nível de coordenação deste exemplo, é usado o seguinte arquivo de definição das interdependências (as tarefas *cancel_order* e *schedule_deliveryP* têm time-outs do tipo B).

```
before “cancel_order” “deliver_goodsS” time_outB
startsB “contact_producer” “receive_orderP”
equals “schedule_deliveryS” “schedule_deliveryP” no_time_out
                                                  time_outB
afterA “receive_goodsS” “deliver_goodsP”
```

A PN para o nível de coordenação deste exemplo é apresentada na Figura 45. Ela foi simulada e analisada com o Visual Simnet, validando o modelo. A análise da *coverability tree* apresentou doze estados finais. Alguns destes estados, apesar de corretos, indicam que o modelo pode ser melhorado. Por exemplo, se houver problema com o cartão de crédito do comprador, os workflows terminarão corretamente do ponto de vista funcional, mas haverá um *token* “morto” no lugar localizado entre as tarefas *contact_producer* e *schedule_deliveryS* no workflow da loja, indicando que este é um workflow mal estruturado, segundo van der Aalst [van der Aalst 98].

Este exemplo usou apenas dependências temporais entre as tarefas, mas dependências de gerenciamento de recursos também poderiam ser usadas. Por exemplo, a loja poderia possuir um estoque que definiria um caminho alternativo ao de contactar o fabricante. A tarefa de retirar os produtos do estoque teria uma dependência de volatilidade, indicando que o estoque é limitado.

Finalmente, é necessário comentar que este exemplo representa apenas uma situação hipotética. Não foi objetivo tratar detalhes do cenário modelado, tais como as conseqüências do cancelamento do pedido por parte do consumidor (devolução do dinheiro, devolução dos produtos para o fabricante, etc). O objetivo deste exemplo foi apenas mostrar como os mecanismos de coordenação podem ser utilizados em uma situação prática.

7.3. Exemplo para uma situação de CSCW (*Computer Supported Cooperative Work*)

Nesta seção é mostrado um exemplo modelando uma situação de autoria colaborativa, que é uma atividade muito estudada em CSCW [Prakash 99]. O ambiente é composto por três usuários. Um deles é o dono do documento, que pode editá-lo, abri-lo e fechá-lo para a escrita dos demais autores. O usuário A é um usuário que pode ler o documento ou editá-lo, desde que o dono tenha liberado a escrita. O usuário B é um usuário especial, que sempre pode editar o documento, independentemente da autorização do dono.

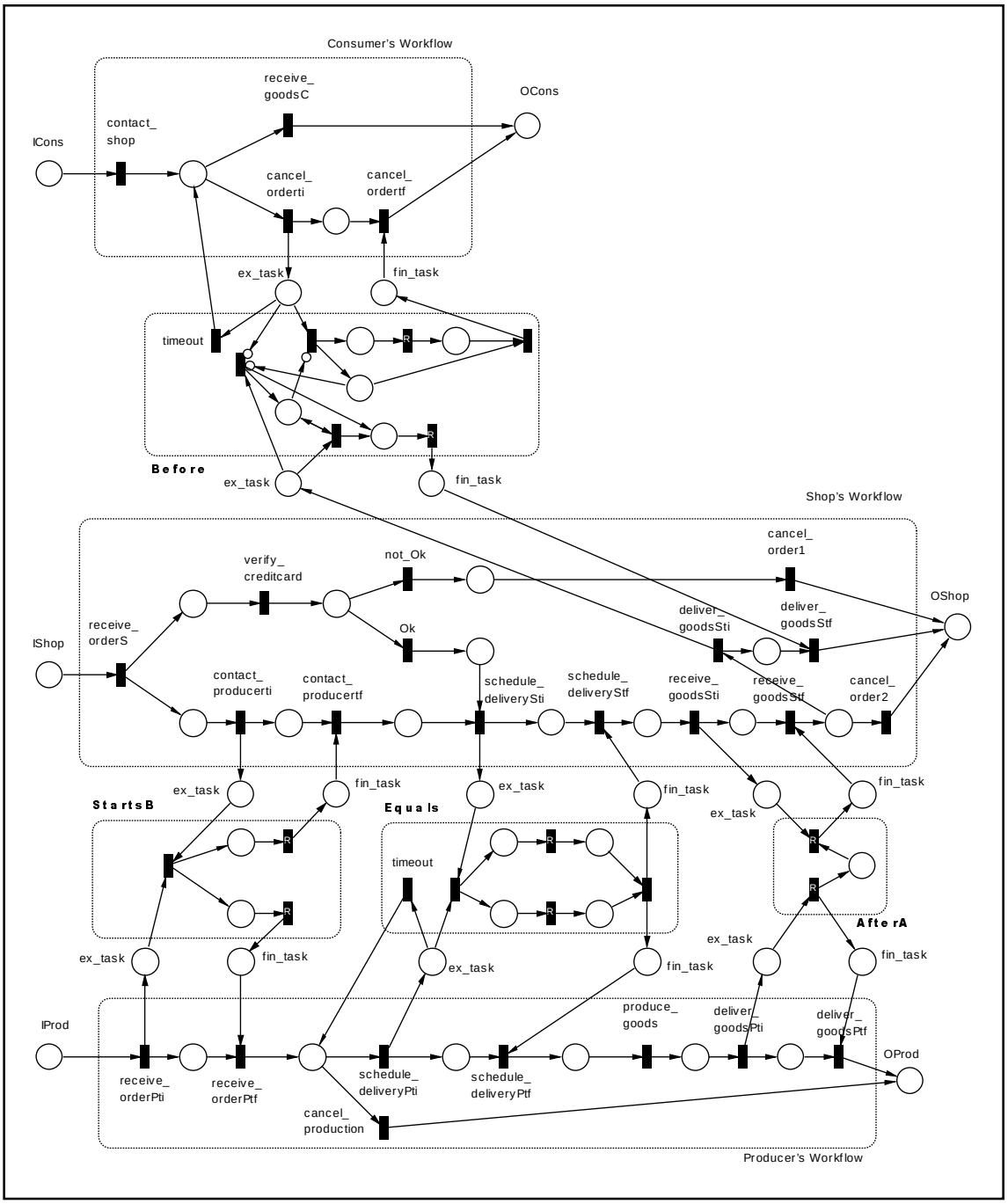

Figura 45: Nível de coordenação para o exemplo da loja virtual.

Para este cenário, foram definidas três interdependências entre as tarefas. A primeira delas diz respeito à exclusão mútua na edição do documento (apenas um usuário pode editá-lo de cada vez). As outras duas dependências estão relacionadas à autorização para o usuário A editar o documento (ele só pode editar depois da liberação por parte do dono e antes que ele o feche para escrita). A representação do cenário descrito no nível de *workflow* é mostrada na Figura 46.

A construção do nível de coordenação deste exemplo é feita com o seguinte arquivo de dependências:

```
div 1 “edita” “editaA” “editaB”
afterB “editaA” “libera_escrita”
before “editaA” “proíbe_escrita”
```

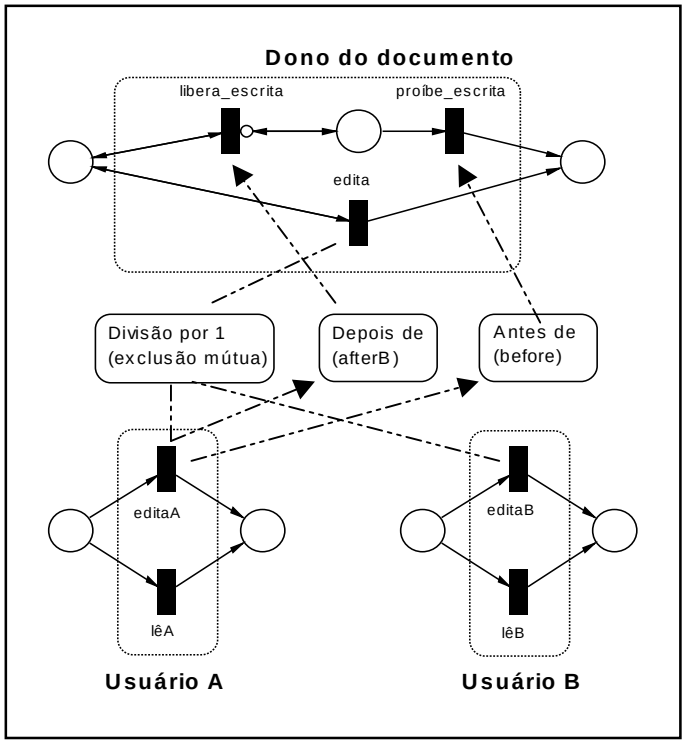

Figura 46: Nível de *workflow* do exemplo de autoria colaborativa.

O modelo no nível de coordenação é mostrado na Figura 47. A análise de verificação indicou que não houve nenhuma situação de *deadlock*, exceto por dois estados finais corretos. A análise de validação comprovou que o sistema funciona como esperado: não ocorrem edições simultâneas e o usuário A não viola as autorizações do dono do arquivo. A análise de desempenho pode ser realizada para medir, por exemplo, o tempo médio que um usuário espera para conseguir editar o documento, dadas as taxas com que cada usuário requisita o recurso de edição.

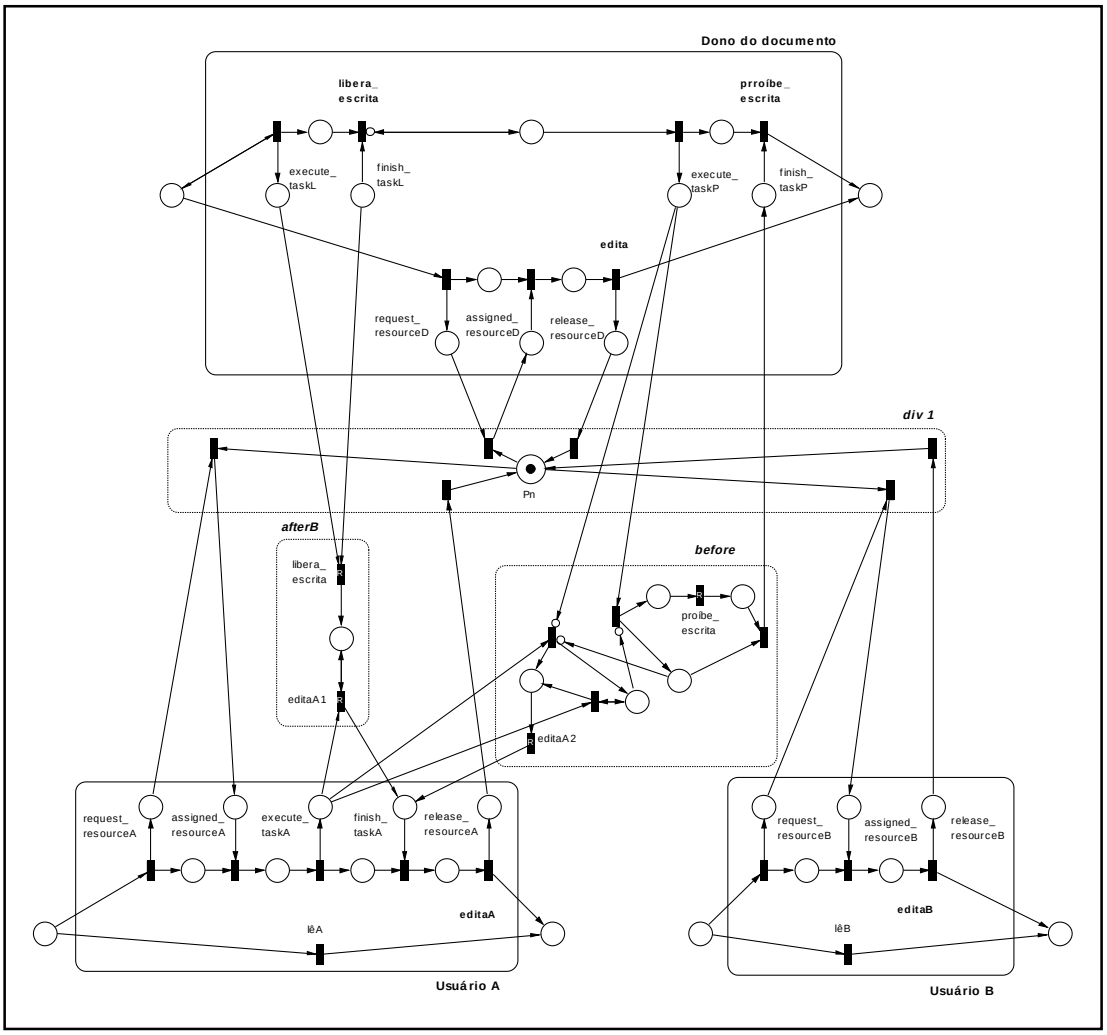

Figura 47: Nível de coordenação para o exemplo de autoria colaborativa.

8. Contribuições

Esta seção destaca, de maneira resumida, as contribuições deste trabalho:

- Visão estendida de animação por computador (Seção 3).
- Criação de mecanismos de coordenação entre as tarefas (componentes da CAV) e separação entre dependências temporais e de gerenciamento de recursos.
  - Modelagem destes mecanismos em PNs "convencionais".
  - Modelagem em PNs e alto nível (Apêndice A).
- Dependências temporais:
  - Expansão e implementação do modelo de [Allen 84] e criação de novas relações temporais.
- Gerenciamento de recursos:
  - Noção mais ampla de recurso: não apenas o agente, mas também qualquer artefato necessário à realização da tarefa.
  - Criação e modelagem de dependências básicas.
- Linguagem para a especificação de dependências.
- Esquema para automatização da passagem do nível de *workflow* para o de coordenação.
  - Implementação para o Visual SimNet (PNs convencionais).

O estágio atual de desenvolvimento permite a criação de um modelo em PNs para o nível de coordenação de um sistema colaborativo, que serve para avaliá-lo antes de sua implementação, detectando possíveis falhas ou situações imprevistas no seu comportamento.

A próxima contribuição esperada será a utilização dos mecanismos da CAV como elementos controladores da interação em uma implementação de CVE. Isso dará uma outra perspectiva à biblioteca, que passará a ser vista não só como ferramenta de teste, mas também como ferramenta para a coordenação de tarefas em CVEs.

9. Conclusão

A coordenação de atividades interdependentes em ambientes colaborativos é um problema que deve ser tratado para garantir a eficiência da colaboração. A separação entre atividades e dependências, e a utilização de mecanismos de coordenação é uma maneira de lidar com este problema, que traz a vantagem da reutilização dos componentes em outras situações de colaboração.

O uso de PNs para a modelagem dos mecanismos de coordenação se mostrou adequado tanto pela capacidade de análise e simulação das mesmas quanto pela sua descrição hierárquica, que permitiu definir a estrutura da coordenação em diferentes níveis de abstração (*workflow* , coordenação e especificação de tarefas). No entanto, um problema típico de PNs é a explosão de estados, que pode ocorrer quando o número de tarefas interdependentes for muito grande. Atualmente, está sendo investigado como PNs de alto nível podem simplificar os mecanismos de coordenação da biblioteca para minimizar este problema (ver Apêndice A).

Os CVEs foram a motivação inicial para o desenvolvimento dos mecanismos da CAV, mecanismos que acabaram se mostrando bastante genéricos e adequados para avaliação do comportamento de uma série de outros sistemas colaborativos, tais como *workflows* interorganizacionais e sistemas multiusuário. Atualmente os mecanismos de coordenação estão sendo implementados para a utilização em ambientes virtuais colaborativos. A coordenação entre as atividades poderá permitir que este tipo de ambiente seja utilizado para a realização de tarefas colaborativas mais complexas que as atualmente realizadas, basicamente controladas pelo protocolo social.

10. Referências

[Allen 84] J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23: 123-154. 1984.

[Anderson 99] M. Anderson. Workflow Interoperability – Enabling E-Commerce. *WfMC White Paper*. <http://www.aiim.org/wfmc/standards/docs/if4-a.pdf>. April 1999.

[Bause 96] F. Bause and P. S. Kritzinger. *Stochastic Petri Nets: An Introduction to the Theory*. Advanced Studies in Computer Science, Verlag Vieweg.1996.

[Bull 92] S. A. Bull. *FlowPath Functional Specification*. September 1992.

[De Cindio 88] F. De Cindio. G. De Michelis and C. Simone. *The Communication Disciplines of CHAOS*. In *Concurrency and Nets*, pp. 115-139. Springer-Verlag, 1988.

[De Martino 92] J. M. De Martino and R. Köhling. Production Rendering on a Local Area Network. *Computer & Graphics*, 16(3): 317-324. Pergamon Press, 1992.

[Del Bimbo 96] A. Del Bimbo and E. Vicario. Visual Programming of Virtual Worlds Animation. *IEEE Multimedia*, 3(1):40-49. Spring 1996.

[Dellarocas 96] C. N. Dellarocas. *A Coordination Perspective on Software Architecture: Towards a Design Handbook for Integrating Software Components*; PhD Thesis, Dept. of Electrical Engineering and Computer Science, MIT. 1996.

[Edwards 96] W. K. Edwards. Policies and Roles in Collaborative Applications. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW´96)*, pp. 11-20. 1996.

[Ellis 93] C. A. Ellis and G. J. Nutt. *Modeling and Enactment of Workflow Systems*. In M. A. Marsan (Ed.). *Application and Theory of Petri Nets 1993*, pp. 1-16. Lecture Notes in Computer Science, 691. Springe-Verlag, 1993.

[Faure 99] F. Faure, C. Faisstnauer et al. Collaborative animation over the network. *Proc. of Computer Animation´99*. <http://www.cg.tuwien.ac.at/~francois/Public/Work/papers/collabAnim.html>.

[Fekete 95] J. D. Fekete, E. Bizouarn et al. TicTacToon: A Paperless System for Professional 2D Animation. *SIGGRAPH 95, Conf. Proc.*, pp. 79-89. 1995.

[Frécon 98] E. Frécon and A. A. Nöu. Building Distributed Virtual Environments to Support Collaborative Work. *Proc. of the Symposium on Virtual Reality Software and Technology (VRST´98)*, pp. 105-119. 1998.

[Furuta 94] R. Furuta and P. D. Stotts. Interpreted Collaboration Protocols and their use in Groupware Prototyping. *Proc. of the Conf. on Computer-Supported Cooperative Work (CSCW´94)*, pp. 121-131. 1994.

[Garbe 97] W. Garbe. *Visual Simnet V.1.37 – Stochastic Petri-Net Simulator*. <http://home.arcor-online.de/wolf.garbe/simnet.html>. 1997.

[Genrich 86] H. J. Genrich. *Predicate/Transition Nets*. In W. Brauer, W. Reisig and G. Rozenberg (Eds.). *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986*, pp. 207-247. Lecture Notes in Computer Science, 254. Springer-Verlag, 1986.

[Green 96] M. Green. Animation in the Virtual World. *Proc. of Computer Animation´96*, pp. 5-12. 1996.

[Hagsand 96] O. Hagsand. Interactive Multiuser VEs in the DIVE System. *IEEE Multimedia*, 3(1): 30-39. Spring 1996.

[Holt 85] A. W. Holt. *Coordination Technology and Petri Nets*. In G. Rozenberg (Ed.). *Advances in Petri Nets*, pp. 278-296. Lecture Notes in Computer Science, 222. Springer-Verlag, 1985.

[Holt 88] A. W. Holt. Diplans: A New Language for the Study and Implementation of Coordination. *ACM Transactions on Office Information Systems*, 6(2):109-125. April 1988.

[Jensen 86] K. Jensen. *Coloured Petri Nets*. In W. Brauer, W. Reisig and G. Rozenberg (Eds.). *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986*, pp. 248-299. Lecture Notes in Computer Science, 254. Springer-Verlag, 1986.

[Jern 98] M. Jern. *Information Drill-Down Using Web Tools*. In J. Vince and R. Earnshaw (Eds.). *Virtual Worlds on the Internet*, pp. 71-84. IEEE Computer Society Press, 1998.

[Li 98] D. Li and R. Muntz. COCA: Collaborative Objects Coordination Architecture. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW´98)*, pp. 179-188. 1998.

[MacIntyre 98] B. MacIntyre and S. Feiner. A Distributed 3D Graphics Library. *SIGGRAPH 98, Conf. Proc.*, pp. 361-370. 1998.

[Magalhães 98a] L. P. Magalhães. Modeling and Analysing Computer Animations. *Proc. SIBGRAPI´98 - International Symposium on Computer Graphics, Image Processing and Vision*, pp. 18-19. 1998.

[Magalhães 98b] L. P. Magalhães, A. B. Raposo and I. L. M. Ricarte. Animation Modeling with Petri Nets. *Computer & Graphics*, 22(6): 735-743. Pergamon Press, 1998.

[Malone 90] T. W. Malone and K. Crowston. What Is Coordination Theory and How Can It Help Design Cooperative Work Systems? *Proc. of the Conf. on Computer-Supported Cooperative Work (CSCW´90)*, pp. 371-380. 1990.

[Marsan 84] M. A. Marsan, G. Conte and G. Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2): 93-122. May 1984.

[Molloy 82] M. K. Molloy. Performance analysis using stochastic Petri Nets. *IEEE Transactions on Computers*, 31(9): 913-917. 1982.

[MPEG-4 99] ISO/IEC JTC1/SC29/WG11 *Overview of the MPEG-4 Standard*. March 1999. <http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm>.

[Murata 89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4): 541-580. April 1989.

[Perlin 96] K. Perlin and A. Goldberg. Improv: A System for Scripting Interactive Actors in Virtual Worlds. *SIGGRAPH 96, Conf. Proc.*, pp. 205-216. 1996.

[Prakash 99] A. Prakash. *Group Editors*. In M. Beaudouin-Lafon (Ed.). *Computer Supported Co-operative Work*. Trends in Software 7. John Wiley & Sons, 1999.

[Ramamoorthy 80] C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri Nets. *IEEE Transactions in Software Engineering*, 6(5): 440-449. September 1980.

[Raposo 96] A. B. Raposo. *Um Sistema Interativo de Animação no Contexto ProSIm*. Tese de Mestrado, DCA - FEEC - Unicamp. 1996.

[Rohlf 94] J. Rohlf and J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. *SIGGRAPH 94, Conf. Proc.*, pp. 381-394. 1994.

[Schmidt 92] K. Schmidt and L. J. Bannon. Takig CSCW Seriously - Supporting Articulation Work. *Computer Supported Cooperative Work*, 1(1-2): 7-40. 1992.

[Starke 99] P. H. Starke. *INA – Integrated Net Analyzer*. Institute für Informatik – Humboldt-Universität zu Berlin. <http://www.informatik.hu-berlin.de/~starke/ina.html>. 1999.

[Stotts 89] P. D. Stotts and R. Furuta. Petri-Net-Based Hypertext: Document Structure with Browsing Semantics. *ACM Trans. on Information Systems*, 7(1):3-29. January 1989.

[Thalmann 85] N. Magnenat-Thalmann and D. Thalmann. *Computer Animation: Theory and Practice*. Springer-Verlag, 1985.

[van der Aalst 94] W. M. P. van der Aalst, K. M. van Hee and G. J. Houben. Modelling and analysing workflow using a Petri-net based approach. *Proc. of the 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pp. 31-50. 1994. <http://wwwis.win.tue.nl/~wsinwa/wfm_adv.ps>

[van der Aalst 97] W. M. P. van der Aalst. *Verification of Workflow Nets*. In P. Azéma and G. Balbo (Eds.). *Application and Theory of Petri Nets 1997*, pp. 407-426. Lecture Notes in Computer Science, 1248. Springer-Verlag, 1997.

[van der Aalst 98] W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1): 21-66. 1998. <http://wwwis.win.tue.nl/~wsinwa/jcsc.ps>

[VRML 97] VRML Consortium. *The Virtual Reality Modeling Language Specification*. International Standard ISO/IEC DIS 14772-1. 1997. <http://ww.vrml.org/Specifications/VRML97>.

[Waters 97] R. C. Waters and J. Barrus. The Rise of Shared Virtual Environments. *IEEE Spectrum*, 34(3): 20-25. March 1997.

Apêndice A – Redes de Petri de Alto Nível

Redes de Petri de alto nível (*high-level PNs*) englobam, dentre outros tipos, as chamadas redes de predicado/transição [Genrich 86] e as redes coloridas [Jensen 86]. A característica mais importante das PNs de alto nível é a capacidade de diferenciação entre os *tokens*, definindo tipos para eles (chamados *tokens* coloridos). Os arcos possuem expressões com variáveis e constantes que definem como será feita a transmissão dos *tokens*. A mesma variável em um arco de entrada e um arco de saída de uma transição denotam o mesmo tipo de *token*. Uma transição está habilitada se houver pelo menos uma possibilidade de substituição das variáveis em *tokens* coloridos.

No exemplo da Figura A1, a transição *t1* está habilitada porque há dois *tokens* da cor *a* no lugar *P1*, de modo que é possível substituir a variável *x* pela cor *a*. Após o disparo de *t1*, o lugar *P1* ficará com um *token* da cor *b*, e o lugar *P2* receberá um *token* da cor *a*. A transição *t2*, ainda na mesma figura, também está habilitada e, neste caso, há três possibilidades diferentes de disparo, pois as variáveis *x* e *y* podem representar qualquer combinação dos três *tokens* coloridos existentes *P3*. Por exemplo, se *x* substituir *a* e *y* substituir *b*, após o disparo de *t2*, o lugar *P3* ficará com o *token* de cor *c* e *P4* ficará com os *tokens a* e *b*. Seria possível também retirar os *tokens a* e *c*, ou os *tokens b* e *c*. No terceiro caso da mesma figura, a transição *t3* não está habilitada, pois o arco de entrada exige dois *tokens* de uma mesma cor (*x*) em *P5*.

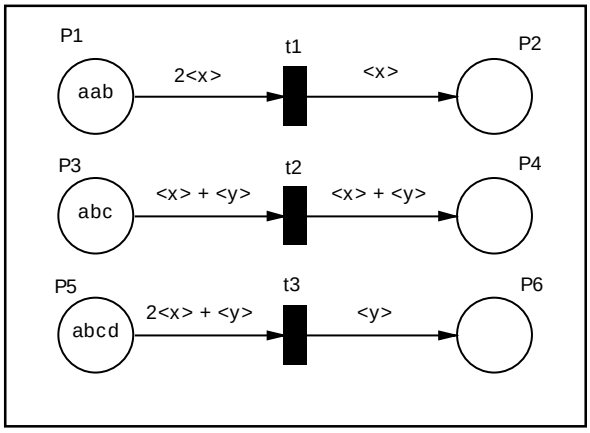

Figura A1: Exemplo de PN de alto nível.

As redes de alto nível poderão ser importantes para futuras implementações dos modelos apresentados, pois elas evitam a explosão de estados quando o número de usuários (WF-Nets no nível de *workflow*) for elevado. Cada *job token* pode ser identificado e diferenciado dos *tokens* que representam recursos e dos *tokens* de controle nos mecanismos de coordenação, simplificando o modelo final.

O que há de mais significativo na implementação por PNs de alto nível, é que não há a necessidade de cada tarefa possuir os lugares *request_resource*, *assigned_resource*, *execute_task*, *finish_task* e *release_resource*. É possível utilizar apenas um lugar de cada tipo que serve a todas as tarefas, cujos *job tokens* estão identificados por suas cores.

A seguir são mostrados os modelos de todos os mecanismos de coordenação utilizando PNs de alto nível.

A.1. Dependências temporais

Os exemplos desta seção e da próxima utilizam a expansão de tarefas em suas formas simplificadas, mostradas na Figura 41.

- *tarefa A equals tarefa B*: este modelo é mostrado na Figura A2. As duas tarefas agora compartilham lugares *execute_tasks e finish_tasks* comuns. Comparando este modelo com o da Figura 13 (mesmo mecanismo modelado com PNs "convencionais"), nota-se que houve uma diminuição no número de lugares. Observe também que os arcos, diferentemente do caso da Figura A1, definem constantes (cores *a* e *b*), e não variáveis.

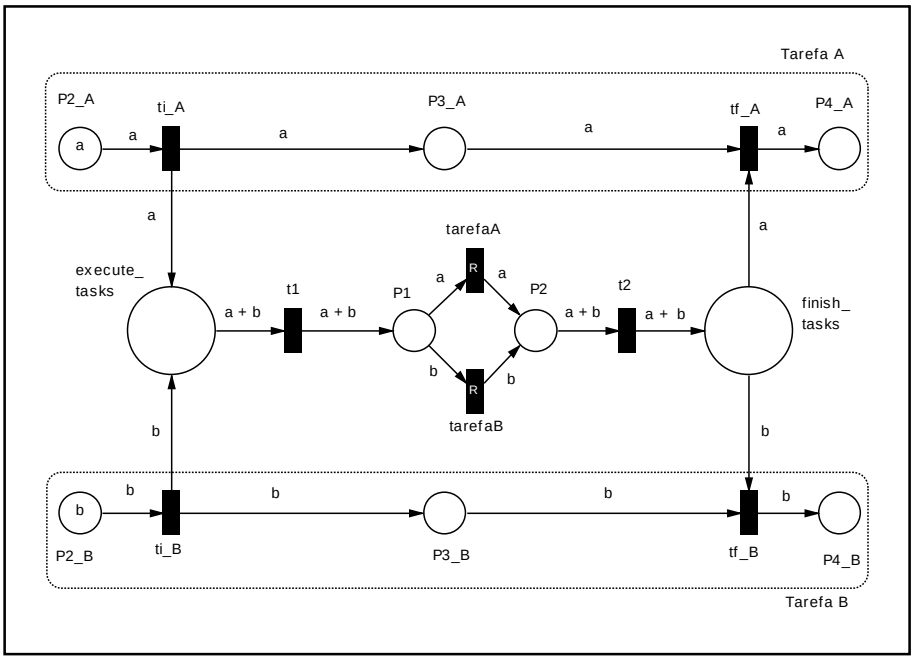

Figura A2: Mecanismo de coordenação para a relação *tarefa A equals tarefa B* usando PN de alto nível.

- *tarefa A starts tarefa B*: assim como no caso anterior, o modelo utilizando PNs de alto nível é mais simples que os que usam PNs convencionais. Compare os modelos das Figuras A3 e A4 com os das Figuras 16 e 17, respectivamente.
- *tarefa A finishes tarefa B*: os modelos para as duas variações de mecanismos de coordenação para esta dependência são mostrados nas Figuras A5 e A6. Compare com as Figuras 18 e 19, respectivamente.
- *tarefa B afte tarefa A*: neste caso (Figura A7), o modelo não se diferencia muito do que usa PNs convencionais (Figura 20), exceto pelo fato dos lugares *execute_tasks* e *finish_tasks* serem comuns para as duas tarefas. A situação em que a tarefa B pode ser executada mais de uma vez a cada execução da tarefa A (afterB) é modelada de maneira similar às PNs convencionais, colocando um arco de retorno da transição *tarefaB* para o lugar *P1*.
- *tarefa A before tarefa B*: no modelo da Figura A8, além dos arcos inibidores, é também usado um arco com peso *b + not(a)*, habilitando a transição *t3* apenas se houver um *token* de cor *b* e nenhum de cor *a* em *execute_tasks*. Compare com o modelo da Figura 21.
- *tarefa A meets tarefa B*: compare a Figura A9 com a Figura 22.

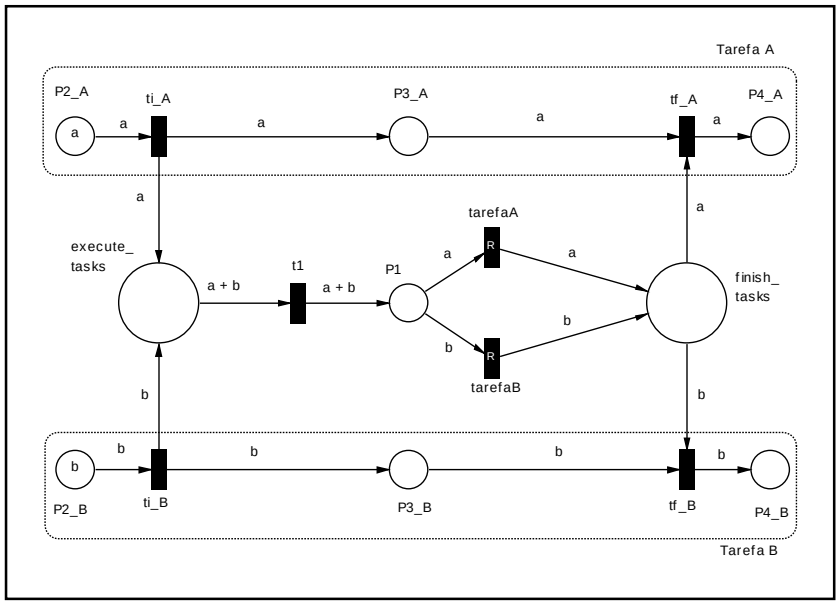

Figura A3: Mecanismo de coordenação para *tarefa A startsB tarefa B* usando PN de alto nível.

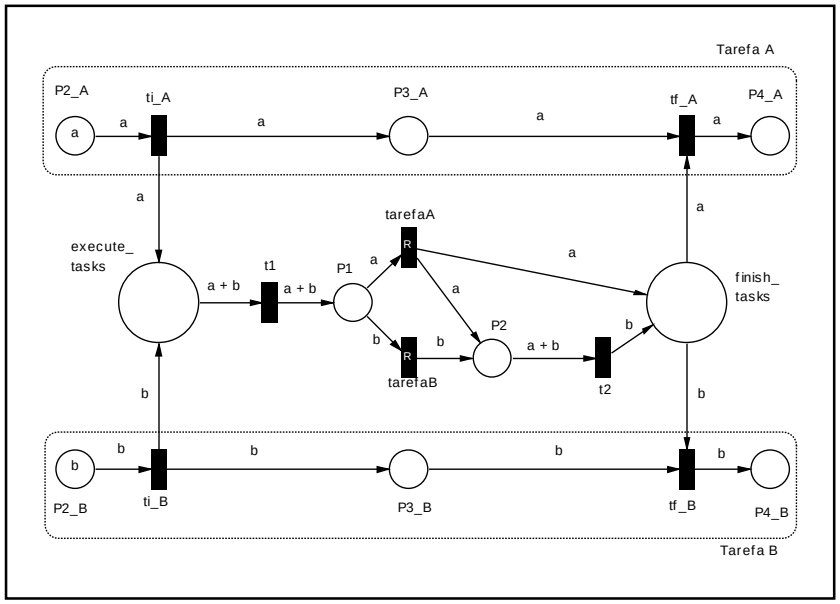

Figura A4: Mecanismo de coordenação para *tarefa A startsA tarefa B*.

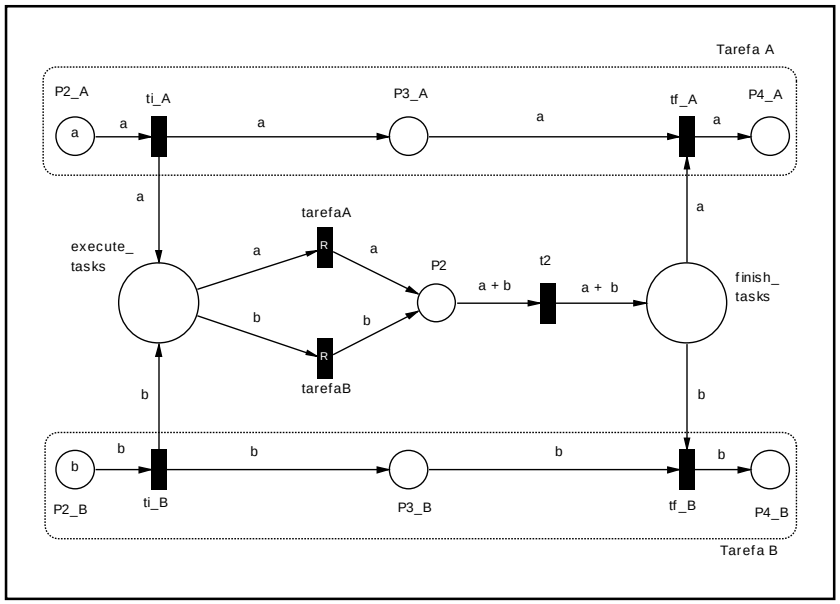

Figura A5: Mecanismo de coordenação para *tarefa A finishesB tarefa B.*

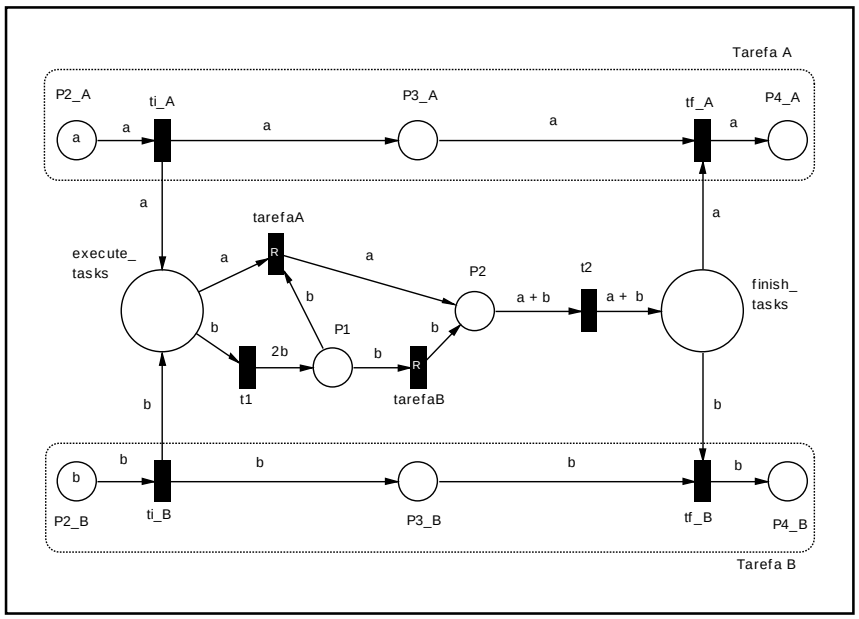

Figura A6: Mecanismo de coordenação para *tarefa A finishesA tarefa B.*

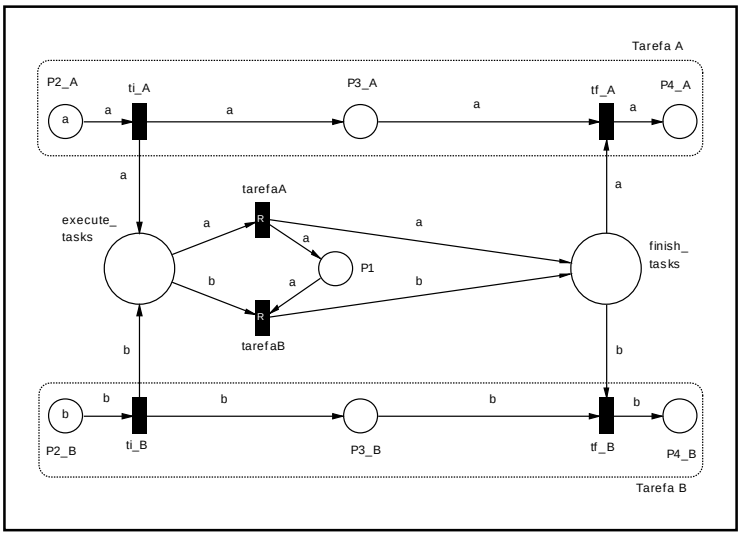

Figura A7: Mecanismo de coordenação para *tarefa B afterA tarefa A*.

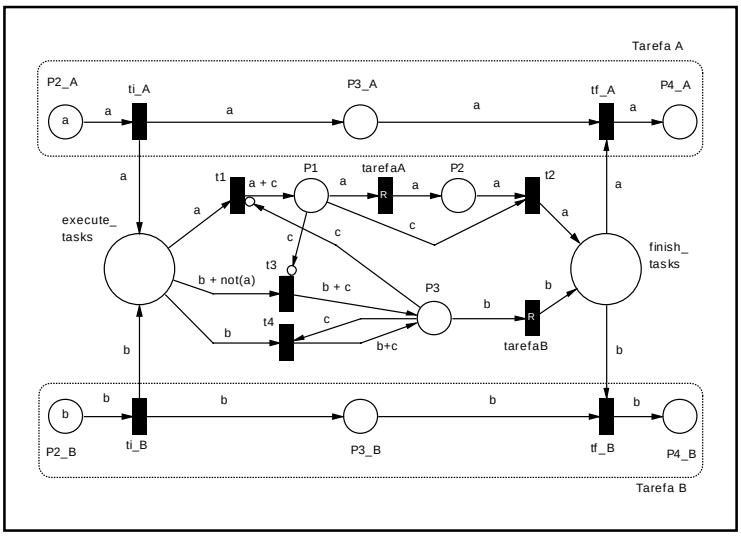

Figura A8: Mecanismo de coordenação para *tarefa A before tarefa B*.

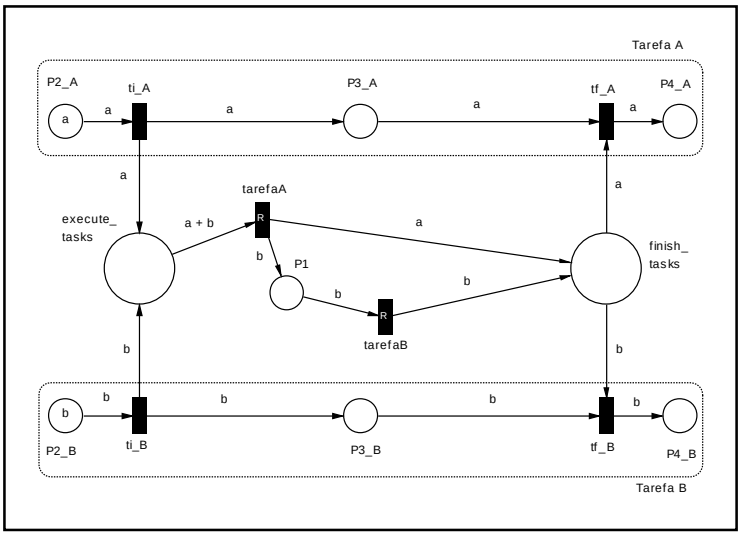

Figura A9: Mecanismo de coordenação para *tarefa A meets tarefa B*.

– *tarefa A overlaps tarefa B*: compare os modelos das Figuras A10 e A11 com os das Figuras 23 e 24, respectivamente.

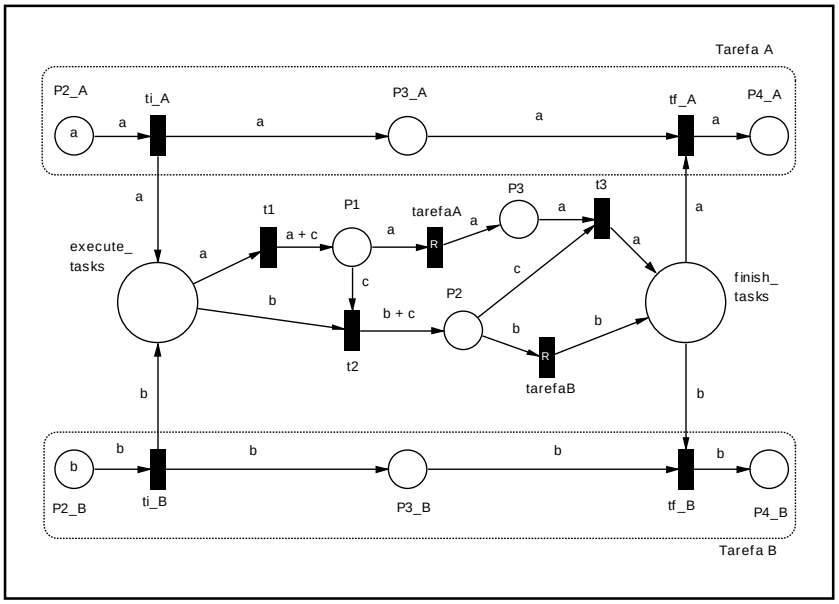

Figura A10: Mecanismo de coordenação para *tarefa A overlapsB tarefa B.*

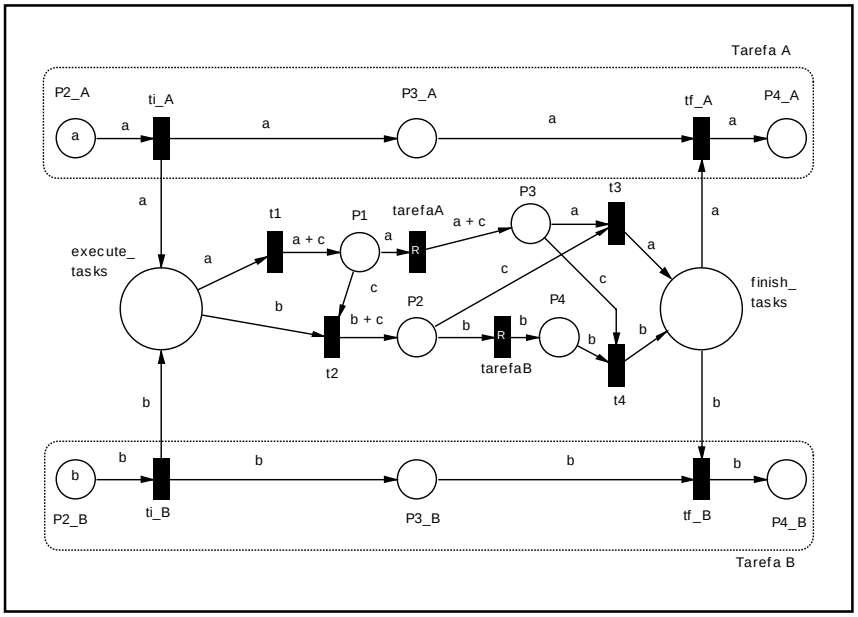

Figura A11: Mecanismo de coordenação para *tarefa A overlapsA tarefa B.*

– *tarefa A during tarefa B*: mais uma vez, o uso de PNs de alto nível gerou modelos mais simples que os anteriores. Compare as Figuras A12 e A13 com as Figuras 25 e 26, respectivamente.

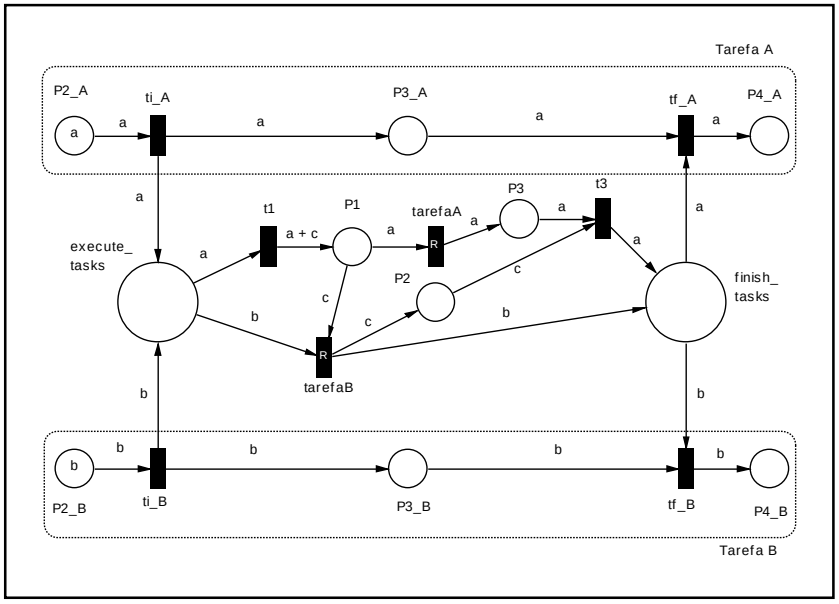

Figura A12: Mecanismo de coordenação para *tarefa A duringA tarefa B*.

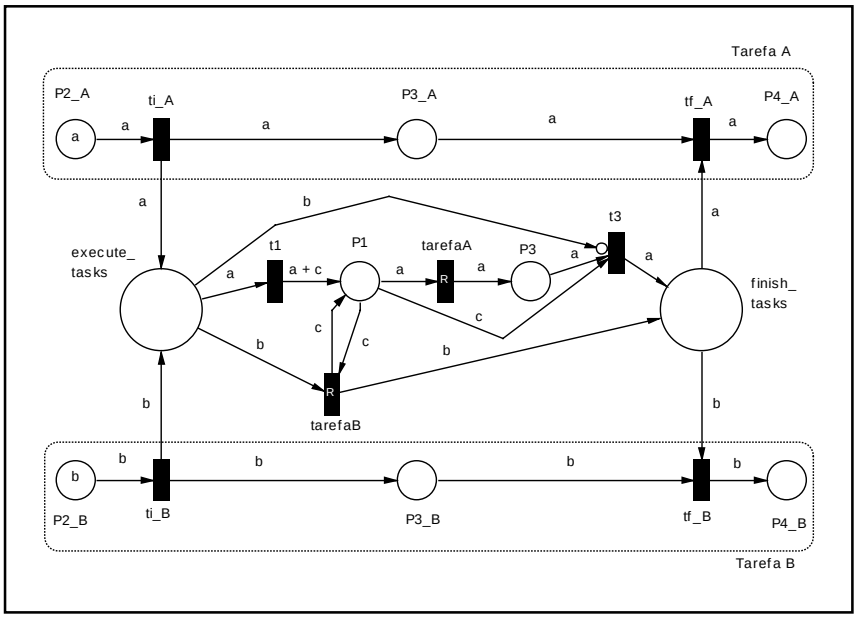

Figura A13: Mecanismo de coordenação para *tarefa A duringB tarefa B*.

A.2. Dependências de gerenciamento de recursos

Este tipo de dependência mostra de maneira ainda mais contundente como as PNs de alto nível podem simplificar os modelos dos mecanismos de coordenação. Isso porque neste tipo de dependência as tarefas não precisam ser identificadas por cores (*tokens a* e *b* nos exemplos anteriores), podendo ser usadas variáveis que são substituídas por *tokens* de qualquer tarefa. O resultado são modelos bem mais simples que os de PNs convencionais, com a vantagem de não aumentarem em complexidade quando o número de tarefas que compartilham um recurso aumenta.

– *divisão por N*: a Figura A14 mostra o gerenciador da divisão por N (compare com a Figura 27). Os arcos com variável <*x*> garantem que a tarefa que requisitou o recurso o receberá. A adição de uma nova tarefa que também possa solicitar o recurso se dá simplesmente conectando a nova tarefa aos lugares

request_resources, *assigned_resources* e *release_resources*, sem necessidade de novos componentes no gerenciador.

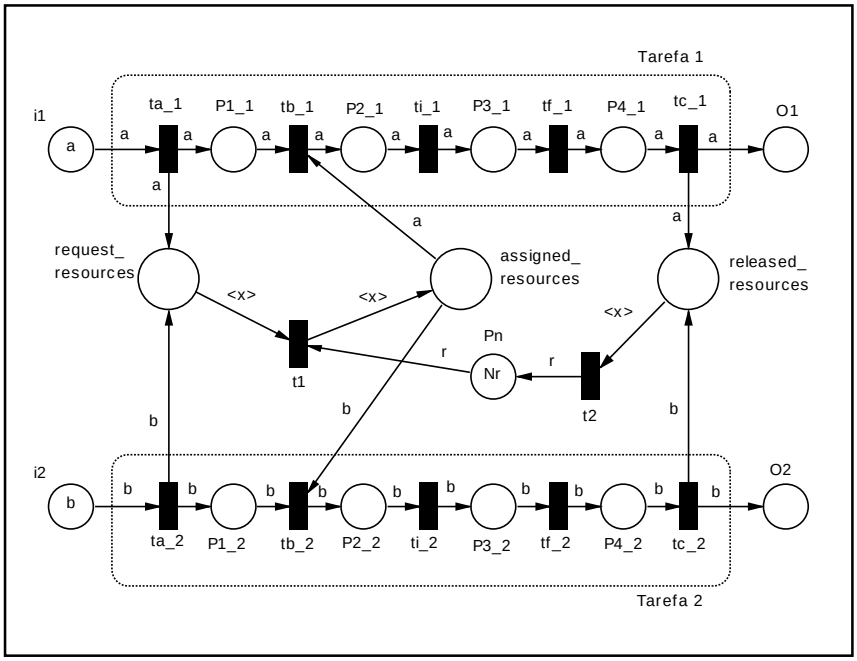

Figura A14: Mecanismo de coordenação para a *divisão por N*, usando PNs de alto nível.

- *simultaneidade N*: a Figura A15 mostra o gerenciador de *simultaneidade 2* com *tokens* coloridos (compare com a Figura 29). Os arcos com expressões *<x>* + *<y>* junto à transição *t1* garantem que duas tarefas diferentes que requisitarem o recurso poderão recebê-lo, se eles estiverem disponíveis no lugar *Pn*. A adição de uma nova tarefa para requisitar o recurso se dá simplesmente conectando a nova tarefa aos lugares *request_resources*, *assigned_resources* e *release_resources*, sem necessidade de novos componentes no gerenciador como na Figura 30.
- *volatilidade N*: o caso em que a volatilidade se refere a todas as tarefas (volB) é mostrado na Figura A16 (compare com a Figura 32). Neste modelo, o lugar *Pn* possui *N tokens* do tipo *r*, indicando as *N* possíveis utilizações do recurso. Os *tokens a* e *b* em *P1* servem para garantir que a tarefa não utilizará o recurso enquanto ela não o tiver liberado (evitar recursividade). A cada nova tarefa adicionada para compartilhar o recurso, um *token* com a cor da nova tarefa deve ser adicionado a *P1*. O modelo da volatilidade para cada tarefa (volA) é construído substituindo os *N tokens* do tipo *r* em *Pn* por *N tokens* da cor de cada tarefa que compartilha o recurso (no exemplo, haveria *N tokens* da cor *a* e *N* da cor *b*). Além disso, o arco saindo de *Pn* deve ter a expressão *<x>* para garantir que sairá um *token* da cor da tarefa que recebeu o recurso.
- *divisão por N + simultaneidade M*: a combinação da *simultaneidade M* com a *divisão por N* é feita simplesmente colocando $N \times M$ *tokens* no lugar *Pn* do modelo da simultaneidade M (Figura A15). Este modelo também resolve o problema desta configuração, descrito na Seção 6.2.4 (uma tarefa liberando recursos muito mais rápido que a outra), pois o arco *<x>* + *<y>* saindo de *release_resources* garante que os *tokens* representando os recursos (cor *r*) só retornam ao lugar *Pn* quando duas tarefas diferentes os liberam.

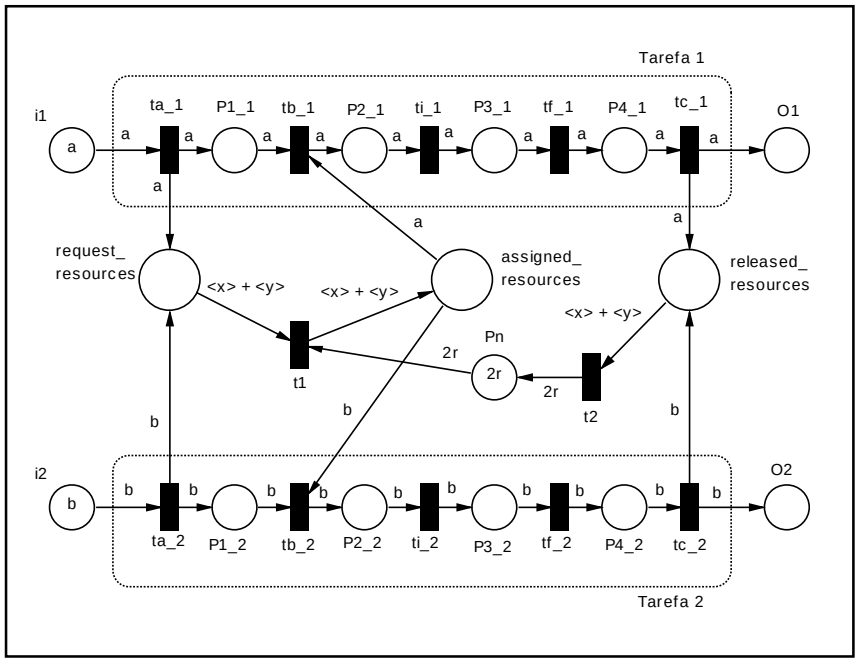

Figura A15: Mecanismo de coordenação para a *simultaneidade 2*.

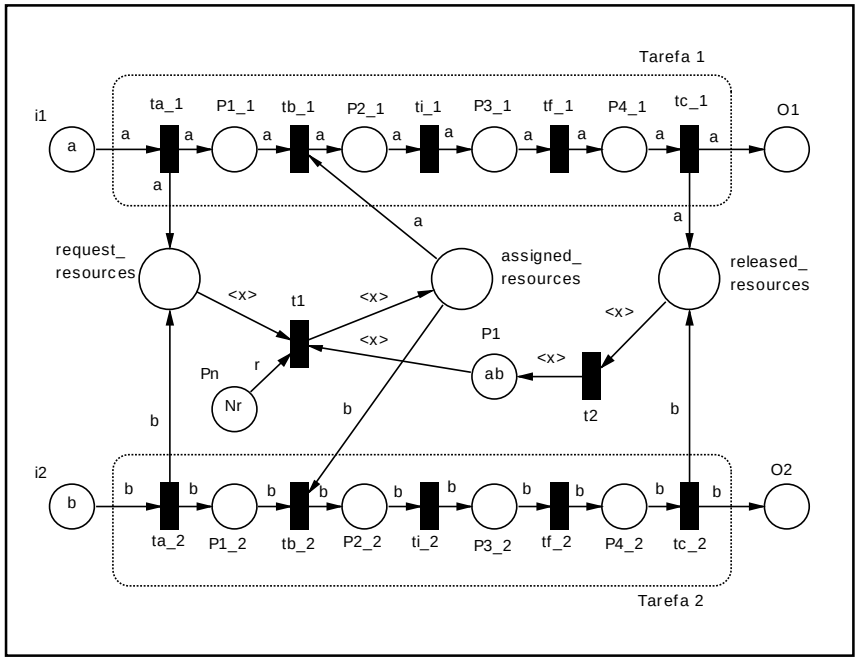

Figura A16: Mecanismo de coordenação para a *volatilidade N*.

- *divisão por N + volatilidade M*: a combinação da divisão por N com a volatilidade M é construída simplesmente adicionando o lugar *Pm* (indicando a volatilidade) junto ao modelo da divisão por N. Este modelo é mostrado na Figura A17 (compare com a Figura 34).
- *simultaneidade N + volatilidade M*: assim como no caso anterior, este mecanismo de coordenação é construído adicionando-se o lugar *Pm* ao modelo da simultaneidade N.

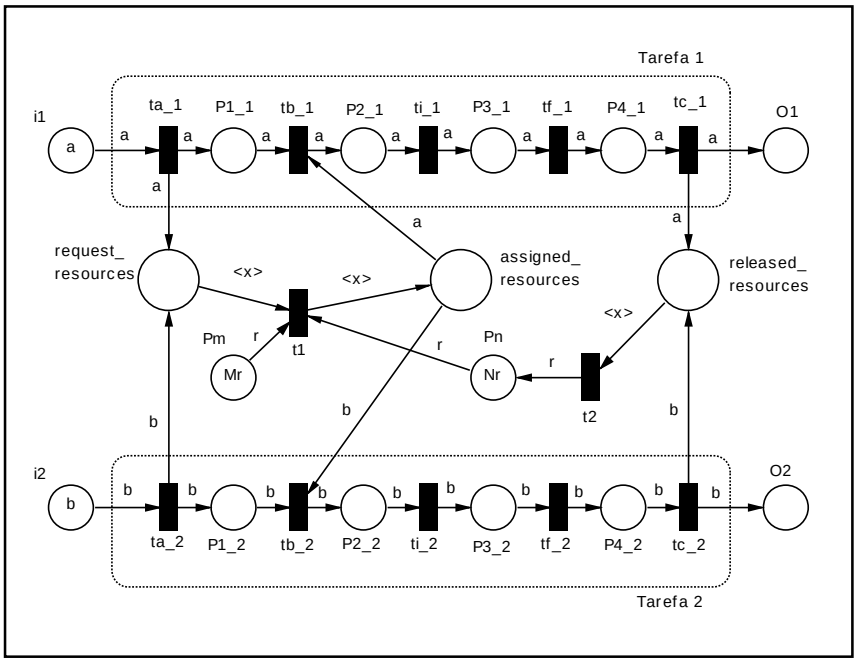

Figura A17: Mecanismo de coordenação para a *divisão por N + volatilidade M*.

– *divisão por N + simultaneidade M + volatilidade Q*: este gerenciador é obtido adicionando o lugar *Pq* (volatilidade) ao gerenciador da combinação divisão por N + simultaneidade M. O modelo para N = 3, M = 2 e Q = 4 é mostrado na Figura A18 (compare com a Figura 36).

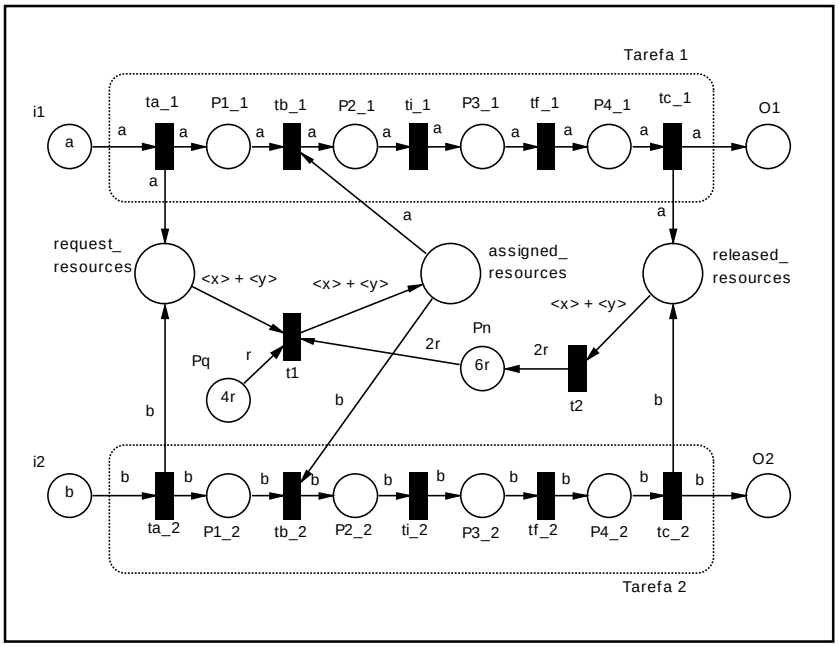

Figura A18: Mecanismo de coordenação para a *divisão por 3 + simultaneidade 2 + volatilidade 4*.

Em resumo, um estudo preliminar sobre o uso de PNs de alto nível mostrou que elas são de grande utilidade para a simplificação dos modelos propostos, principalmente quando o número de tarefas dependentes for grande. Essa simplificação é muito importante para evitar a explosão de estados. No entanto, o uso

de PNs de alto nível sacrifica algumas características importantes do modelo básico de PN, tais como a facilidade de entendimento dos modelos e a capacidade de realização de análises (a maior parte de ferramentas disponíveis só analisam PNs convencionais ou PNs coloridas sem a capacidade de substituição de variáveis). Portanto, ainda é necessário um estudo mais aprofundado sobre a viabilidade da implementação do modelos utilizando PNs de alto nível na CAV.