UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Tese de Doutorado

Coordenação em Ambientes Colaborativos Usando Redes de Petri

Autor: ALBERTO BARBOSA RAPOSO
Orientador: PROF. DR. LÉO PINI MAGALHÃES
Co-orientador: PROF. DR. IVAN LUIZ MARQUES RICARTE

TESE DE DOUTORADO

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Coordenação em Ambientes Colaborativos Usando Redes de Petri

Autor: **Alberto Barbosa Raposo**

Orientador: **Prof. Dr. Léo Pini Magalhães**
Co-orientador: **Prof. Dr. Ivan Luiz Marques Ricarte**

Banca Examinadora:

Prof. Dr. Léo Pini Magalhães – DCA / FEEC / UNICAMP
Prof. Dr. Clésio Luis Tozzi – DCA / FEEC / UNICAMP
Prof. Dr. Rafael Santos Mendes – DCA / FEEC / UNICAMP
Prof. Dr. **Walter da Cunha Borelli** – DT / FEEC / UNICAMP
Prof. Dr. Paulo César Masiero – ICMC / USP-São Carlos
Prof. Dr. Hugo Fuks – DI / PUC-Rio

Tese submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de **Doutor em Engenharia Elétrica**. Área de concentração: **Engenharia de Computação.**

26 de outubro de 2000.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Raposo, Alberto Barbosa
R182c Coordenação em ambientes colaborativos usando redes de Petri / Alberto Barbosa Raposo.--Campinas, SP: [s.n.], 2000.

Orientadores: Léo Pini Magalhães, Ivan Luiz Marques Ricarte.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Grupos de trabalho – Processamento de dados. 2. Redes de Petri. I. Magalhães, Léo Pini. II. Ricarte, Ivan Luiz Marques. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Resumo

A coordenação das interdependências entre atividades colaborativas é uma tarefa complexa, de difícil modelagem computacional. Este trabalho define uma série de interdependências que ocorrem freqüentemente entre tarefas colaborativas e apresenta um conjunto de mecanismos de coordenação para a especificação e controle da interação entre essas tarefas. Estes mecanismos são genéricos e podem ser reutilizados em uma série de ambientes colaborativos, tais como *workflows* interorganizacionais e ambientes virtuais colaborativos. A separação entre atividades (tarefas) e dependências (controladas pelos mecanismos de coordenação) permite o uso de diferentes políticas de coordenação em um mesmo ambiente colaborativo, sendo necessário apenas trocar os mecanismos de coordenação. Os mecanismos são modelados por redes de Petri, que oferecem um suporte matemático para análise e simulação do comportamento do ambiente colaborativo. Com o modelo baseado em redes de Petri, é possível prever e testar o comportamento de um ambiente de suporte ao trabalho colaborativo antes mesmo de sua implementação.

Abstract

The coordination of interdependencies among activities in collaborative environments is a very important and difficult task. This work defines a group of interdependencies that frequently occurs among collaborative tasks and presents a set of coordination mechanisms for the specification and control of interaction among these tasks. These mechanisms are generic and can be reused in several collaborative environments, such as interorganizational workflows and collaborative virtual environments. The separation between activities (tasks) and dependencies (controlled by the coordination mechanisms) allows the use of different coordination policies in the same collaborative environment by changing the coordination mechanisms. These mechanisms are modeled using Petri nets, which have a strong theoretical support for analysis and simulation of the collaborative environment's behavior. Using the Petri nets based model, it is possible to anticipate and test the behavior of a computer supported collaborative system even before its implementation.

Agradecimentos

À FAPESP, pelo apoio financeiro através da bolsa de doutorado.

Aos meus orientadores, Prof. Dr. Léo Pini Magalhães e Prof. Dr. Ivan Luiz Marques Ricarte, pela ajuda nos momentos mais críticos do trabalho (e nos momentos menos críticos também).

Ao DCA – FEEC – UNICAMP, pela infra-estrutura.

À minha esposa Ana Beatriz, à minha filha Ana Letícia pela paciência e carinho que sempre demonstraram e, para não ser injusto, a todos os outros filhos que eu ainda possa vir a ter.

A meus pais, meus irmãos e meus sogros pelo apoio.

A todos os colegas e professores que de alguma forma contribuíram neste trabalho.

A todos aqueles que se sintam decepcionados por não estarem nestes agradecimentos.

Índice

1. Introdução

O incremento do uso dos computadores pessoais, o aumento de velocidade e a expansão das redes de computadores, juntamente com o desenvolvimento de tecnologias multimídia, ocorridos na década de 80, permitiram que os computadores passassem a ser utilizados não só para a realização de tarefas individuais, mas também para a comunicação e realização de tarefas entre pessoas [Scrivener 94]. De fato, tarefas colaborativas, tais como reuniões, troca de correspondências (eletrônicas ou não), conversas por telefone, encontros informais no corredor e coordenação com secretárias e outros membros de uma equipe ocupam grande parte do tempo de trabalho das pessoas [Beaudouin-Lafon 99]. Entretanto, ainda é relativamente precário o suporte dado por computadores a este tipo de atividade. Esta é uma das motivações para as pesquisas em CSCW (*Computer Supported Cooperative Work*).

CSCW é a área de estudo interessada no trabalho em conjunto de grupos de pessoas com a ajuda de computadores [Palmer 94]. É um tema multidisciplinar que envolve profissionais das áreas de computação, automação, antropologia, sociologia, psicologia social, economia, teoria organizacional, educação e de outras áreas interessadas no estudo do trabalho colaborativo. Uma das definições mais conhecidas diz que "CSCW deve ser entendido como um esforço no sentido de compreender a natureza e as características do trabalho cooperativo com o objetivo de projetar tecnologias computacionais adequadas [para dar suporte a este tipo de trabalho]" [Bannon 91]. Ainda segundo os mesmos autores, "o foco é entender para melhor auxiliar o trabalho cooperativo".

Uma definição para trabalho colaborativo freqüentemente citada na literatura [Bannon 91], [Schmidt 92], [Wilson 94] é a de Karl Marx, escrita em 1867. Segundo Marx, trabalho colaborativo é definido como "múltiplos indivíduos trabalhando juntos de maneira planejada no mesmo processo de produção, ou em processos de produção diferentes, mas conectados". No âmago desta definição está a noção de planejamento, responsável por garantir que o trabalho coletivo seja resultante do conjunto de tarefas individuais.

A noção de planejamento presente na definição de Marx é materializada em CSCW pelo que foi chamado de "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 do trabalho em grupo, o mapeamento destes objetivos em tarefas, a seleção dos participantes, a distribuição das tarefas entre eles e a coordenação da realização das atividades (Figura 1.1).

Um aspecto importante do trabalho colaborativo é a noção de interdependência entre as tarefas. Essa noção serve para diferenciar trabalho colaborativo de outros tipos de trabalho em grupo. Um exemplo é a diferença entre dirigir em um comboio e dirigir no trânsito de uma cidade [Grosz 96]. No primeiro caso, os motoristas têm algum sistema de comunicação pré-estabelecido e um objetivo comum que depende do sucesso dos outros motoristas nas suas tarefas (todos os carros devem chegar ao destino; se algum precisar de ajuda, os outros certamente o socorrerão). Isso caracteriza um trabalho colaborativo. Por outro lado, ao dirigir no trânsito de uma cidade não há colaboração. Apesar de haver interação entre os motoristas, ser possível haver comunicação entre eles e até mesmo haver

um objetivo comum (chegar a um mesmo lugar), não há nenhum planejamento prévio das tarefas e o sucesso de cada motorista em atingir seu objetivo não depende dos demais motoristas.

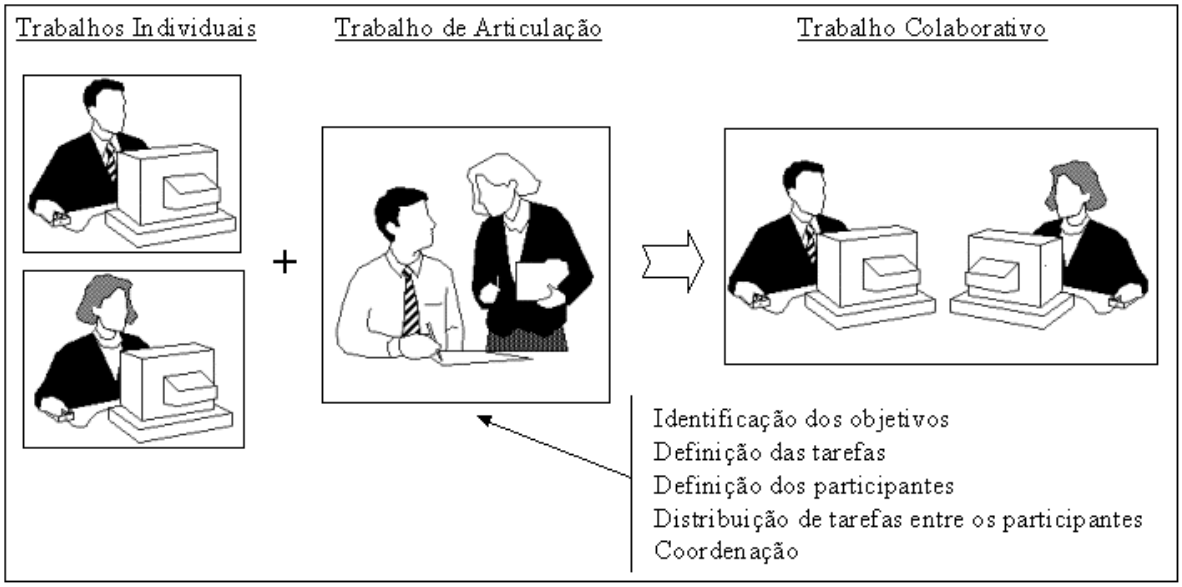

Figura 1.1: Trabalho de articulação.

Apesar da interdependência positiva entre as tarefas na colaboração (i.e., um participante precisa que o trabalho do outro seja bem sucedido), ela nem sempre é harmoniosa. É preciso haver coordenação entre as atividades para garantir a eficiência da colaboração. Sem coordenação, há o risco dos participantes se envolverem em tarefas conflitantes e/ou repetitivas. A coordenação é definida como "o ato de gerenciar interdependências entre as atividades realizadas para se atingir um objetivo" [Malone 90].

A coordenação é a mais importante dentre as atividades do trabalho de articulação, pois representa o aspecto dinâmico do mesmo, precisando ser "renegociada" de maneira quase contínua ao longo de todo o tempo que durar a colaboração. As outras atividades do trabalho de articulação (Figura 1.1) são concluídas antes do trabalho colaborativo se iniciar e normalmente não precisam ser alteradas.

Muitas vezes, o termo "coordenação" é usado como sinônimo de "trabalho de articulação". Um exemplo é a definição de *groupware* de C. Ellis [Ellis 91]. Segundo ele, sistemas para apoio ao trabalho em grupo (*groupware*) são compostos de três tecnologias básicas: comunicação, colaboração e coordenação. A primeira está relacionada à interação entre os participantes do grupo e é realizada por ferramentas de email, *chat*, *newsgroup*, videoconferência e outras, que normalmente são partes integrantes deste tipo de sistema. A colaboração está relacionada ao compartilhamento de informações, documentos e objetos entre os participantes do grupo e também à maneira como estes participantes percebem a noção de contexto do grupo (i.e., percepção da presença e das ações dos outros usuários). A coordenação, segundo este autor, é uma atividade em si, responsável por garantir a eficiência do trabalho colaborativo.

A coordenação, no entanto, não é necessária em alguns tipos de atividades colaborativas, tais como aquelas desenvolvidas por meio de *chats* e áudio- ou videoconferências. Estas atividades, associadas às relações sociais, são normalmente bem

controladas pelo chamado "protocolo social", caracterizado pela ausência de qualquer mecanismo de coordenação explícito entre as atividades, confiando nas habilidades dos participantes de mediar as interações (a coordenação é culturalmente estabelecida).

Por outro lado, atividades mais diretamente voltadas para o "trabalho" colaborativo (e não para as relações sociais) exigem sofisticados mecanismos de coordenação das atividades. Exemplos deste tipo de atividade são encontrados em ferramentas de autoria colaborativa, *workflows*, videogames, dentre outras. Este tipo de atividade é o principal foco deste trabalho, que apresenta um conjunto de mecanismos de coordenação para tratar as interdependências que freqüentemente ocorrem entre tarefas colaborativas.

O grande desafio ao se propor mecanismos de coordenação para 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 os tipos de colaboração 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]. Uma maneira de oferecer essa flexibilidade é por meio de uma clara separação entre "trabalho de articulação, i.e., o trabalho relacionado à coordenação das atividades e o trabalho coordenado, i.e., o trabalho cuja execução é articulada no domínio-alvo" [Simone 99].

No contexto de CSCW, um "mecanismo de coordenação" pode ser entendido como um "dispositivo de software que interage com uma aplicação específica para dar suporte ao trabalho de articulação relacionado ao campo de trabalho representado pelas estruturas de dados e funcionalidades daquela aplicação" [Schmidt 96]. No presente trabalho, entretanto, o termo "mecanismo de coordenação" é usado no contexto de modelagem do sistema, indicando uma representação do dispositivo de software que poderá ser construído para efetuar a coordenação das atividades.

Dentro desse contexto, este trabalho apresenta um conjunto de modelos de mecanismos de coordenação que podem ser reutilizados em uma série de ambientes colaborativos. Separando as atividades (tarefas) das dependências (controladas pelos mecanismos de coordenação), permite-se o uso de diferentes políticas de coordenação em um mesmo ambiente colaborativo, sendo necessário apenas trocar os mecanismos de coordenação. Além disso, por serem genéricos, os mecanismos de coordenação podem ser utilizados em ambientes colaborativos diversos, tais como *workflows* interorganizacionais e ambientes virtuais colaborativos, como será apresentado no Capítulo 4.

Devido à complexidade dos inter-relacionamentos entre as tarefas colaborativas, é importante que se tenha uma modelagem formal dos ambientes colaborativos projetados, de modo que suas características possam ser analisadas e avaliadas *a priori*. Para modelar os mecanismos de coordenação apresentados neste trabalho, são usadas redes de Petri (PNs – *Petri Nets*). A representação gráfica das PNs, além de ser de fácil compreensão, permite o encapsulamento de detalhes, oferecendo um método adequado para a modelagem dos diferentes níveis de coordenação. Além disso, as PNs oferecem um forte suporte matemático para a análise do comportamento do ambiente colaborativo e técnicas de simulação complementares. Com o modelo baseado em PNs, é possível prever e testar o comportamento de um ambiente colaborativo antes mesmo de sua implementação.

No próximo capítulo são discutidos alguns trabalhos relacionados à coordenação de atividades colaborativas. No Capítulo 3 os modelos dos mecanismos de coordenação são apresentados. Casos de estudo envolvendo diversos tipos de ambientes colaborativos são mostrados no Capítulo 4. No Capítulo 5 são apresentadas as conclusões e sugestões de trabalhos futuros. Uma apresentação formal de redes de Petri é feita no Apêndice A. O Apêndice B mostra como alguns conceitos de *workflow* são mapeados em PNs. O Apêndice C é um glossário com a definição dos conceitos freqüentemente utilizados no texto. No Apêndice D estão cinco artigos publicados diretamente relacionados com o conteúdo deste trabalho.

2. Trabalhos Relacionados

Apesar de sua reconhecida importância, a coordenação não foi incluída nos primeiros sistemas colaborativos no início da década de 80. De acordo com um artigo da época, "qualquer fator relacionado à interdependência de tarefas – coordenação – é deixado para os usuários gerenciarem da melhor maneira possível, seja por meio de bases de dados compartilhadas, ligações telefônicas, correio eletrônico, (...) ou qualquer outro meio disponível" [Holt 85].

As tentativas de incluir mecanismos de coordenação em sistemas colaborativos datam do final da década de 80, mas eles eram inicialmente restritos a cenários bastante específicos, pois os protocolos de coordenação eram rigidamente definidos, não permitindo modificações. Estudos antropológicos e sociológicos [Kling 91], [Suchman 94], [Winograd 94] mostraram que sistemas colaborativos não devem impor padrões rígidos de trabalho ou comunicação. Na verdade, eles devem prover facilidades que permitam aos usuários interpretar e explorar estes padrões, mas devendo sempre caber aos usuários a decisão de usá-los, modificá-los ou rejeitá-los [Schmidt 91], [Bardram 97]. Na década de 90, então, começaram a surgir sistemas com mecanismos de coordenação suficientemente flexíveis para permitir redefinições dinâmicas e modificações temporárias. Esta necessidade provém do fato de que um protocolo de coordenação não tem como abranger todas as situações possíveis e, portanto, inevitavelmente o usuário em algum momento se encontrará em uma situação que exige um desvio do protocolo. Desse modo, um esquema de coordenação flexível e facilmente modificável é uma característica desejável em sistemas colaborativos.

Embora a contribuição do presente trabalho esteja centrada na *modelagem* e verificação de sistemas colaborativos, procurando definir modelos de mecanismos de coordenação que podem ser usados em uma série de situações, este capítulo apresenta também alguns trabalhos relacionados à *utilização* de mecanismos de coordenação (neste caso, dispositivos de software) em sistemas colaborativos, dando especial ênfase àqueles que usam alguma variação de redes de Petri para a modelagem e/ou implementação dos mecanismos.

As Seções 2.1, 2.2 e 2.3 apresentam a coordenação do ponto de vista das aplicações colaborativas. Os sistemas discutidos utilizam os mecanismos de coordenação para interagir com as atividades colaborativas, controlando sua execução. Estes mecanismos, por sua vez, são definidos, de uma maneira geral, através de uma "linguagem" (seja uma linguagem de programação, um protocolo informal, uma representação em PN, etc.) que pode ser acessível ou não ao usuário final. A Figura 2.1 ilustra, de uma maneira genérica, a arquitetura de coordenação dos sistemas apresentados.

2.1. Sistemas de Primeira Geração

A primeira geração de sistemas colaborativos com algum tipo de mecanismo de coordenação apareceu na década de 80 e apresentava protocolos de coordenação pouco flexíveis.

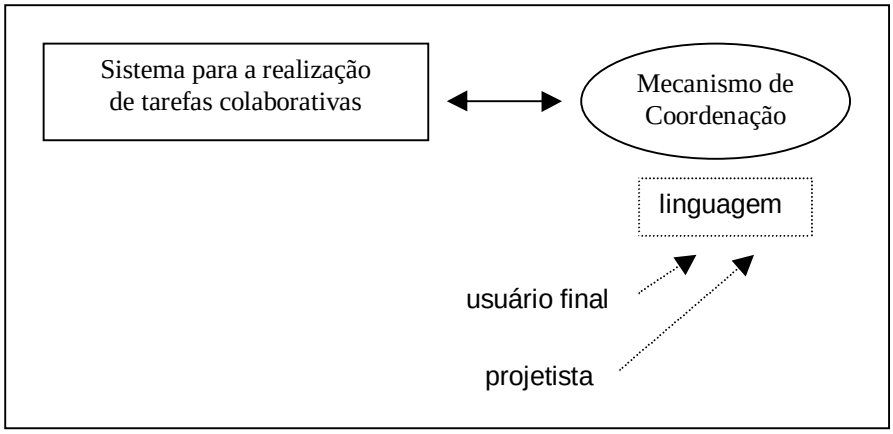

Figura 2.1: Mecanismos de coordenação em sistemas colaborativos.

2.1.1. *Coordinator*

Um dos mais significativos representantes dessa primeira geração foi *Coordinator* [Winograd 86], [Flores 88]. Ele foi desenvolvido com base em teorias lingüísticas com o objetivo de ajudar a estruturação de conversas por email. Segundo os autores, a teoria de linguagem/ação, na qual se baseia o sistema, provê um fundamento unificado para o suporte ao trabalho interativo em organizações. Esta teoria é utilizada para controlar a troca de mensagens que direciona o trabalho colaborativo.

Coordinator segue padrões pré-estabelecidos que controlam a realização das tarefas. Por exemplo, uma conversação pode começar com uma requisição de A para B. B então deve responder com uma promessa (de cumprir a requisição), uma declinação (não fará o que lhe foi requisitado) ou uma contraproposta (alterando algum aspecto na requisição). Em função da resposta de B, a conversação segue um caminho específico dentro do protocolo pré-estabelecido. O protocolo para este tipo de conversação é ilustrado na Figura 2.2 (a figura usa uma notação informal dos autores do *Coordinator* representando uma rede de conversação).

2.1.2. Diplan

Em 1985, Anatol Holt, constatando a falta de mecanismos de coordenação em ambientes de trabalho por computador, usou redes de Petri (PNs) para a coordenação de atividades neste tipo de ambiente [Holt 85]. Nesse 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, também foram identificados pontos essenciais da tecnologia de coordenação, que na época era bastante promissora para a criação de ambientes eletrônicos de trabalho. Dentre esses 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 este é um antagonismo inevitável".

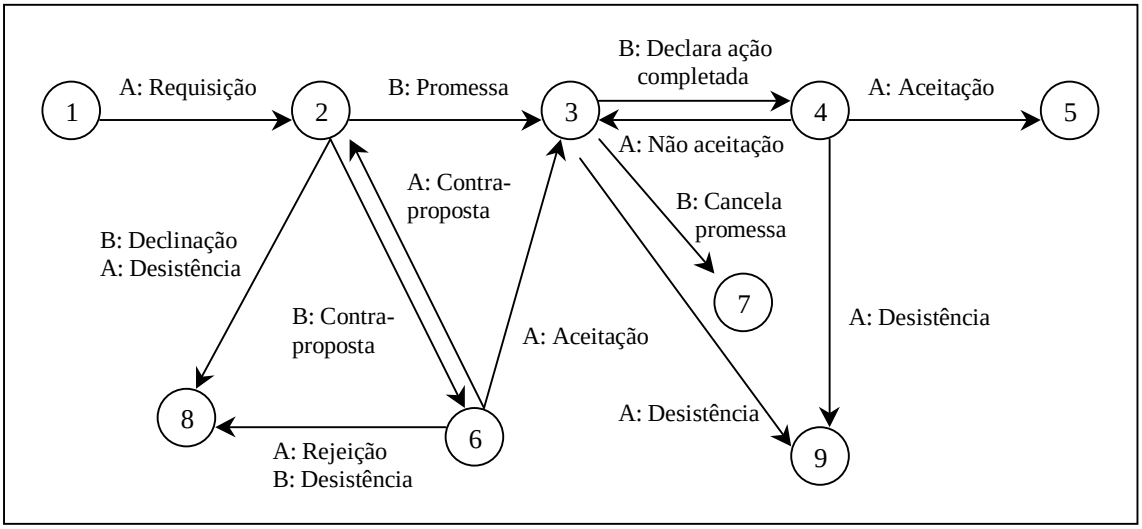

Figura 2.2: Protocolo para um tipo de conversação [Winograd 86].

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 [Genrich 86], com algumas 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.

2.1.3. CHAOS (*Commitment Handling Active Office System*)

CHAOS é outra proposta para a coordenação de atividades em automação de escritórios baseada em PNs [De Cindio 88]. CHAOS se baseia em uma teoria chamada *Communication Discipline* [Petri 77] para controlar a comunicação em um software que coordena o trabalho de membros de um escritório.

São definidos dois tipos de conversação: conversação para ação (definição de um compromisso para a realização de uma tarefa) e conversação para possibilidades (interlocutores discutem novas possibilidades para o escritório). Para cada um destes tipos de conversação foram criados vários sub-tipos, cada um modelado por uma rede de Petri específica.

Da mesma forma que *Coordinator*, CHAOS define rigidamente as redes de conversação entre os participantes, com pouca flexibilidade para situações que fujam dos padrões pré-estabelecidos. Por tratar apenas das redes de conversação em escritórios, CHAOS também é de uso mais restrito que Diplan.

2.1.4. Trellis

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.

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 atrasos e *timeouts* 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.

Trellis se diferencia de *Diplan* porque sua funcionalidade vai além dos aspectos de coordenação das atividades, criando "hiperprogramas", que misturam a navegação hipermídia ao suporte à colaboração. No entanto, uma arquitetura centralizada como a do Trellis pode não ter um desempenho adequado quando o número de usuários é muito grande [Li 98].

2.2. Sistemas de Segunda Geração

A segunda geração de sistemas colaborativos com mecanismos de coordenação surgiu na década de 90. A diferença com relação aos da geração anterior é que os de segunda geração apresentam mecanismos de coordenação mais flexíveis e acessíveis aos projetistas e usuários dos sistemas colaborativos.

2.2.1. Oval (*Objects, Views, Agents and Links*)

Oval [Malone 95] é um exemplo da segunda geração de sistemas colaborativos com mecanismos de coordenação, no qual existe uma grande preocupação com a flexibilidade e acessibilidade destes mecanismos ao usuário final.

Oval possui quatro elementos básicos: objetos, visões, agentes e ligações. Os objetos representam as coisas do mundo, tais como pessoas, tarefas, mensagens, etc. Cada objeto possui um conjunto de campos e um conjunto de ações que podem ser realizadas sobre ele. As visões mostram coleções de objetos e permitem que usuários editem objetos individualmente. Os agentes servem para realizar tarefas para os usuários, sem a necessidade da atenção direta deles, enquanto as ligações representam as relações entre objetos.

O conceito de programação para o usuário final (EUP – *end user programming*) é usado para permitir que o usuário final (não necessariamente um programador experiente) modifique o sistema sem precisar sair do domínio da aplicação e ir para o domínio de uma linguagem de programação. Dentre as modificações que o usuário pode fazer estão incluídas: criação de novos tipos de objetos e grupos de objetos, criação de novos agentes e inserção de novas ligações.

Experimentos mostraram que Oval pode ser usado para implementar as funcionalidades de várias aplicações colaborativas existentes (inclusive *Coordinator*), com a vantagem de fornecer ao usuário todo o poder da EUP, que lhe permite fazer grandes modificações em sistemas funcionais sem exagerado esforço de programação [Malone 95].

2.2.2. ABACO / Ariadne

ABACO é um protótipo que implementa uma notação (Ariadne) para a definição e manipulação de mecanismos de coordenação. Ariadne é uma notação genérica e visa facilitar a construção de mecanismos de coordenação em qualquer aplicação envolvendo trabalho colaborativo e, ao mesmo tempo, permitir que os próprios projetistas e usuários das aplicações construam e alterem estes mecanismos [Schmidt 96].

A definição da Ariadne foi baseada em estudos empíricos sobre como os participantes de uma atividade colaborativa articulam seus trabalhos. Ela define um repertório de categorias e predicados elementares no nível semântico do trabalho de articulação. Estas categorias permitem expressar os protocolos de coordenação. Além disso, Ariadne tem sua estrutura interna dividida em três níveis. Os níveis α e β são acessados pelos usuários através da interface e não requerem conhecimentos muito especializados por parte dos mesmos. O nível γ, por sua vez, é restrito aos projetistas de aplicações, que criam novas gramáticas necessárias à comunidade de usuários para definir seus protocolos de coordenação. No nível β os usuários definem ou alteram estes protocolos utilizando a gramática, enquanto no nível α, de maior nível de abstração, eles instanciam ou ativam um protocolo para uma situação particular.

Tanto Ariadne quanto Oval são ferramentas que permitem implementar sistemas colaborativos com mecanismos de coordenação flexíveis e acessíveis aos usuários. A existência do nível γ, no entanto, torna Ariadne mais completa que Oval, pois este nível torna possível a definição de uma linguagem apropriada para a construção de famílias de mecanismos de coordenação.

2.2.3. Intermezzo

Intermezzo usa os conceitos de políticas (*policies*) e papéis (*roles*) para a coordenação de colaboração [Edwards 96]. As políticas de coordenação são definidas por meio de controle de acesso aos objetos de dados que é estabelecido para grupos de usuários com um determinado papel. Os papéis são atribuídos não só de forma estática (antes da colaboração), mas também de forma dinâmica, ao longo da colaboração (usuários podem mudar de papel, por exemplo). A atribuição dinâmica de papéis permite políticas de coordenação mais flexíveis, como por exemplo, permitir acesso aos dados para as pessoas que estiverem no mesmo laboratório em determinado instante.

A partir da definição dos papéis, as políticas de coordenação são especificadas atribuindo direitos de acesso aos conjuntos de recursos regidos por estas políticas. O Intermezzo possui uma linguagem para a especificação de papéis e políticas, bem como uma série de políticas normalmente utilizadas em situações reais, tais como anonimato (usuário presente, mas sem nenhuma informação que possa identificá-lo) e pseudo-anonimato (o usuário também não é identificado, mas os outros podem tomar conhecimento de suas ações e outras informações necessárias à coordenação do trabalho em grupo).

2.2.4. COCA (*Collaborative Objects Coordination Architecture*)

COCA [Li 98] usa os conceitos de políticas e papéis de forma similar ao Intermezzo. No entanto, apresenta uma linguagem bem mais poderosa que a do Intermezzo para a

definição das políticas e papéis. Em COCA, as políticas de coordenação não se limitam a controle de acesso, mas incluem também controle de sessão, controle de concorrência, entre outros.

COCA é um *framework* genérico para o desenvolvimento de sistemas colaborativos, provendo um barramento de colaboração e uma linguagem para especificação das políticas de coordenação. As políticas são interpretadas pela máquina virtual do COCA, que é executada localmente em cada participante da colaboração.

A arquitetura do COCA é composta de três camadas. Na base está a camada de comunicação, baseada em IP *multicast* e que utiliza as noções de barramento de colaboração e de conferência. O barramento de colaboração conecta os participantes e o de conferência conecta a instância local da máquina virtual com as várias ferramentas de colaboração (por exemplo, ferramentas de vídeo, desenho, controle de sessão, etc.). No topo está a camada de aplicação, provendo serviços como controle de sessão, apresentação de dados, manipulação de interfaces, etc. Entre estas duas camadas fica a camada de colaboração, conectando-as e interpretando as políticas de coordenação. A Figura 2.3 ilustra as visões macroscópica e microscópica da colaboração com COCA.

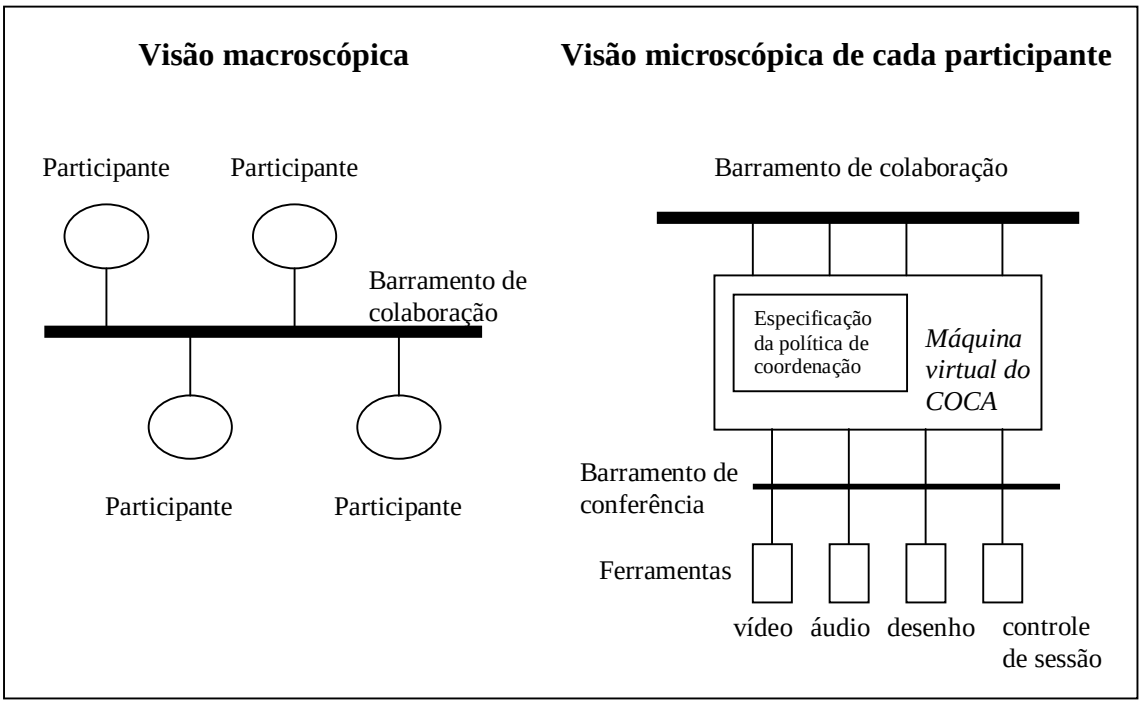

Figura 2.3: Visões macroscópica e microscópica da colaboração com o COCA [Li 98].

2.3. Sistemas de *Workflow*

Workflows [Jablonski 96] são exemplos de sistemas colaborativos com fortes componentes de coordenação. Recentemente, *workflows* interorganizacionais têm atraído a atenção dos pesquisadores, pois é um tema de importância na era da economia globalizada [Hayes 00], [Yang 00]. Neste tipo de *workflow* a cooperação ultrapassa os limites de uma única organização, sendo composto por várias organizações trabalhando de maneira colaborativa.

PRODNET II [Camarinha-Matos 98] é um exemplo de projeto visando o desenvolvimento de uma infra-estrutura de suporte aberta e flexível, adequada às necessidades de pequenas e médias organizações. A arquitetura do PRODNET II implementa uma clara separação entre serviços de suporte e mecanismos de controle.

VEC (*Virtual Enterprise Co-ordinator*) [Ludwig 99] é outro sistema de *workflow* interorganizational. O foco principal deste sistema é gerenciar a colaboração através de *gateways* (configurados por meio de acordo entre as organizações) que fornecem aos parceiros externos uma maneira controlada de acessar os processos internos de uma organização, mantendo a liberdade de alterações de detalhes internos destes processos.

METEOR (*Managing End-To-End OpeRations*) [Infocosm 00] é um exemplo de sistema acadêmico que se tornou um produto comercial. Ele permite especificar tarefas humanas e automáticas e as dependências entre elas, gerando código automaticamente a partir da especificação gráfica. Canais de comunicação entre dispositivos de *workflow* distribuídos e heterogêneos são implementados por meio de CORBA e Java.

Um exemplo de sistema de *workflow* interorganizacional que utiliza redes de Petri é o COSM [Merz 97]. COSM usa agentes móveis para a coordenação de *workflows* interorganizacionais, uma abordagem que se mostrou especialmente adequada para situações de comércio eletrônico, em que os parceiros não querem “amarrar” a colaboração (por exemplo, para fazer uma pesquisa de preços com vários fornecedores e optar pela melhor oferta). A máquina virtual que controla o comportamento de agentes móveis no COSM é modelada por meio de uma rede de Petri.

2.4. Coordenação em Outros Contextos

Esta seção sai do contexto das aplicações colaborativas, apresentando alguns exemplos de coordenação em outros tipos de aplicações, tais como programas *multi-threaded* (Seção 2.4.1), atividades humanas (Seção 2.4.2) e sistemas de agentes móveis (Seção 2.4.3).

2.4.1. Linguagens de Coordenação

Em paralelo aos mecanismos de coordenação de sistemas colaborativos, também foram desenvolvidas as chamadas linguagens de coordenação, que propõem a separação computação/coordenação para aplicações *multi-threaded* [Gelernter 92].

A idéia em que se baseiam as linguagens de coordenação é que se pode construir um modelo de programação completo a partir de duas partes separadas: o modelo de computação e o modelo de coordenação. O modelo de computação é realizado pelas linguagens de programação convencionais e permite a construção de atividades isoladas (em uma única *thread* de execução). O modelo de coordenação é o que conecta estas atividades isoladas, estabelecendo o controle de execução das *threads* e a comunicação entre elas. As linguagens de coordenação englobam este modelo de coordenação, provendo recursos para a criação de atividades e comunicação entre elas.

A mais conhecida linguagem de coordenação é Linda [Gelernter 92], que não é uma linguagem de programação completa, mas uma extensão que pode ser adicionada a outra linguagem de programação para prover suporte à criação, comunicação e sincronização de processos. Existem várias implementações englobando o modelo Linda, tais como C-Linda e Fortran-Linda.

2.4.2. Percepção (*Awareness*)

Embora não possa ser considerado um mecanismo de coordenação de acordo com a definição apresentada no Capítulo 1 (não é um dispositivo de software), o conceito de percepção (*awareness*) é citado por muitos autores como uma forma de garantir a coordenação de tarefas colaborativas [Dourish 92], [Gutwin 98], [Araújo 99].

A percepção é definida como o "entendimento das atividades dos outros, o que provê um contexto para sua própria atividade" [Dourish 92]. Ela envolve saber quem está usando o sistema, o que os outros usuários estão fazendo (alterações ocorrendo no objeto de trabalho) e como estas alterações aconteceram (o que, junto com a comunicação e o conhecimento do contexto, permitem entender porque elas aconteceram) [Dix 97].

A importância da percepção como coordenadora de atividades colaborativas pode ser observada em vários níveis. Em um nível de abstração mais alto, a percepção das ações dos outros permite aos participantes a estruturação de suas próprias atividades para evitar duplicações ou conflitos. Em um nível de abstração mais baixo, a percepção do conteúdo das ações permite um controle mais detalhado da atividade compartilhada e do comportamento sinergético do grupo como um todo [Dourish 92].

Vários estudos têm sido realizados com o objetivo de criar um *framework* com os elementos necessários à completa percepção dos usuários [Benford 95], [Gutwin 96].

Além da importância no processo de colaboração em si, a capacidade de percepção dos usuários também serve como "indicadora de atividade social" [Ackerman 96], pois é sabido que as pessoas são atraídas por sistemas com grande atividade e, por outro lado, tendem a não utilizar sistemas que não são usados por um número elevado de pessoas (problema da massa crítica de usuários [Markus 90], [Grudin 94]).

2.4.3. Agentes

A tecnologia de agentes tem sido vista como detentora de um grande potencial para a coordenação de atividades colaborativas [Jennings 94], [Kosoresow 98].

Um exemplo de sua aplicação é o uso de agentes para a coordenação de atividades de grupos que se comunicam via email [Kreifelts 91]. Neste caso, os agentes são chamados de mediadores e podem iniciar ou participar de uma conversação. Assim como nos mecanismos de coordenação da primeira geração (*Coordinator* e CHAOS, por exemplo), o comportamento e as tarefas dos mediadores são definidos de acordo com um protocolo de comunicação pré-estabelecido. A diferença é que os agentes são capazes de realizar certas tarefas no contexto da conversação, tais como distribuir mensagens para um grupo de acordo com os interesses e competências de seus membros.

Outras aplicações de agentes para atividades de coordenação incluem a sincronização de objetos multimídia distribuídos [Al-Salqan 96] e o gerenciamento de *workflows* interorganizacionais [Merz 97].

A relação entre agentes e coordenação não está restrita ao uso de agentes para gerenciar atividades entre usuários humanos. Em ambientes multi-agentes, os agentes também precisam coordenar suas atividades com as dos outros agentes e de outras entidades que encontram durante sua execução no ambiente hospedeiro, criando uma nova dimensão para o conceito de coordenação [Cabri 00].

2.5. O Contexto do Trabalho

Este trabalho se insere no contexto dos sistemas colaborativos. No entanto, o objetivo não é implementar mecanismos de coordenação, mas sim modelá-los, para que se possa fazer uma avaliação prévia do sistema colaborativo e corrigir possíveis problemas antes da implementação (Figura 2.4).

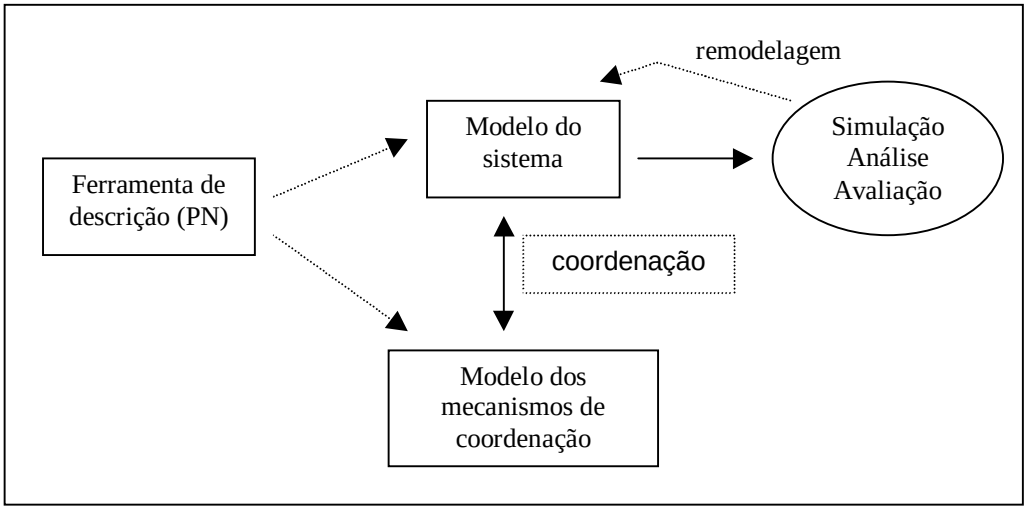

Figura 2.4: Esquema para modelagem e avaliação dos sistemas colaborativos.

Como ferramenta de descrição são utilizadas redes de Petri. No entanto, seria também possível utilizar outras ferramentas para a descrição dos sistemas, tais como redes hierárquicas [Huber 90] e SDL (*Specification and Description Language*) [Saracco 89].

Um trabalho que está bastante relacionado ao aqui apresentado é o de van der Aalst, que percebeu a necessidade de tratar o conteúdo da estrutura de coordenação de *workflows* interorganizacionais. Segundo ele, "a semântica dos mecanismos necessários para modelar *workflows* interorganizacionais deve ser definida antes de resolver questões técnicas (principalmente sintáticas)" [van der Aalst 99]. Ele expandiu seu trabalho anterior sobre redes de workflows (WF-Nets – ver Apêndice B) [van der Aalst 98] para modelar e analisar *workflows* interorganizacionais.

Apesar de motivados por questões semelhantes e fazendo uso da mesma ferramenta de modelagem (PNs), existem significativas diferenças entre o presente trabalho e o de van der Aalst. O principal objetivo do trabalho de van der Aalst é verificar formalmente a correção de *workflows* interorganizacionais e sua consistência com os diagramas de seqüências de mensagens usados para especificar a interação entre os processos. O trabalho aqui apresentado, por outro lado, tem como objetivo definir um conjunto de interdependências entre tarefas colaborativas e prover modelos de mecanismos de coordenação genéricos (usando PNs) que podem ser usados para especificar a interação em vários tipos de ambientes colaborativos (*workflow* interorganizacional é um destes tipos, como será mostrado em um exemplo no Capítulo 4). A correção e consistência dos ambientes modelados podem ser verificadas por meio de ferramentas de simulação e análise de PNs. Para facilitar esta tarefa, também foi desenvolvida uma ferramenta capaz de inserir estes mecanismos de coordenação entre as tarefas interdependentes de uma PN modelada.

O próximo capítulo apresenta o conjunto de modelos de mecanismos de coordenação definido e o protótipo desenvolvido para simulação e análise de ambientes colaborativos que usam estes mecanismos.

3. Modelos de Coordenação

O modelo de coordenação utilizado neste trabalho, e aqui apresentado nas Seções 3.1 e 3.2, é inspirado na filosofia das linguagens de coordenação, que separam a computação da coordenação. No caso de atividades colaborativas, a separação é feita entre tarefas e interdependências (controladas por mecanismos de coordenação).

Uma das vantagens da separação entre tarefas e interdependências é a possibilidade de alterar as políticas de coordenação simplesmente trocando os mecanismos de coordenação para as dependências, sem a necessidade de alterar o "núcleo" do sistema colaborativo. Além disso, as interdependências e seus mecanismos de coordenação podem ser reutilizados. É possível "caracterizar diferentes tipos de dependências e identificar os processos de coordenação que podem ser usados para gerenciá-los" [Malone 94], criando um conjunto de dependências e respectivos mecanismos de coordenação capaz de abranger um variado espectro de aplicações.

A separação entre atividades e dependências também tem sido usada em trabalhos relacionados à teoria da coordenação [Malone 90], [Malone 94] e à área de arquitetura de software (por exemplo, SYNOPSIS [Dellarocas 96]). Em particular, SYNOPSIS é uma linguagem para a descrição da arquitetura de sistemas que suporta duas abstrações ortogonais: atividades, que representam as peças funcionais de uma aplicação, e dependências, que descrevem as relações de interconexão entre as atividades. Sistemas executáveis podem ser gerados a partir de descrições SYNOPSIS por meio de sucessivos refinamentos, substituindo atividades abstratas por versões mais especializadas e gerenciando as dependências com processos de coordenação, até que todos os elementos da descrição sejam suficientemente específicos para a geração do código.

Na área de sistemas colaborativos, a separação entre atividades e dependências é utilizada em sistemas como ABACO/Ariadne (ver capítulo anterior) e o Reconciler [Simone 99]. O Reconciler pode ser visto como um componente autônomo, construído independentemente das aplicações, usado para mediar os mecanismos de coordenação de grupos de trabalho diferentes. Seu principal objetivo é gerenciar a interoperabilidade entre os grupos no nível semântico, conciliando as visões dos grupos por meio do tratamento de conflitos de terminologia, de unidades, etc.

Outro trabalho que merece destaque usa as interdependências entre as atividades para o gerenciamento de *workflows* [Attie 96]. Nesse caso, as interdependências são definidas como "restrições à ocorrência e à ordem temporal dos eventos", e são controladas por mecanismos de coordenação definidos como autômatos finitos que garantem que a respectiva dependência não seja violada. O objetivo é criar um *scheduler* global que satisfaça todas as dependências definidas para o *workflow*.

Tomando como base a separação acima discutida, a contribuição do trabalho apresentada neste capítulo possui duas partes distintas. Na primeira delas (Seção 3.1), um conjunto genérico de interdependências entre atividades colaborativas é definido. Na segunda parte, mecanismos de coordenação para essas interdependências são modelados utilizando PNs (Seção 3.2 em diante).

3.1. Definição das Dependências

As interdependências entre tarefas colaborativas serão aqui divididas em duas grandes classes: dependências temporais e de gerenciamento de recursos. 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. As dependências de gerenciamento de recursos são complementares às temporais e lidam com a distribuição dos recursos entre as tarefas.

Esta separação entre dependências temporais e de gerenciamento de recursos está de acordo com os níveis de coordenação do modelo proposto por Ellis e Wainer [Ellis 94]. Segundo esse modelo, a coordenação de sistemas colaborativos pode ocorrer em dois níveis, o de atividades e o de objetos. "No nível de atividades, o modelo de coordenação descreve o seqüenciamento das atividades que compõem um procedimento. No nível de objetos, o modelo de coordenação descreve como o sistema lida com o acesso seqüencial ou simultâneo de múltiplos participantes ao mesmo conjunto de objetos" [Ellis 94].

O conjunto de interdependências apresentado a seguir não tem a pretensão de abranger todas as possíveis interdependências entre tarefas. A idéia principal é prover um número limitado (mas suficientemente abrangente) de dependências e respectivos mecanismos de coordenação que serão testados em uma série de aplicações colaborativas. A partir do uso, poderá surgir a necessidade de novas dependências, que seriam então agregadas a este conjunto. O trabalho optou pela extensibilidade, e não pela completude do conjunto de dependências.

3.1.1. Dependências Temporais

As dependências temporais estabelecem a ordem para a execução das tarefas. Um exemplo de dependência temporal ocorre em sistemas de comércio eletrônico, onde o cancelamento de um pedido por parte do usuário só pode ocorrer antes do produto lhe ser enviado [Raposo 00a] (ver exemplo no Capítulo 4 e o artigo no Apêndice D). Outro exemplo ocorre em sistemas de revisão, que só podem iniciar a atividade de revisão após o final da atividade de escrita do documento [Ellis 94].

O conjunto de dependências temporais aqui apresentadas é baseado nas relações temporais definidas por J. F. Allen [Allen 83], [Allen 84]. Segundo Allen, há um conjunto de relações primitivas e mutuamente exclusivas que podem ser aplicadas sobre intervalos de tempo (e não sobre instantes de tempo). Esta álgebra de intervalos é baseada em sete relações básicas ilustradas na Figura 3.1.

Definindo formalmente as relações da Figura 3.1, sejam os intervalos de tempo $X = [x_i, x_f)$ e $Y = [y_i, y_f)$:

- **X equals Y** (X *igual a Y*) se e somente se $x_i = y_i$ e $x_f = y_f$. (X e Y são o mesmo intervalo de tempo.)
- **X starts Y** (*X inicia Y*) se e somente se $x_i = y_i$ e $x_f < y_f$. (X e Y começam juntos, mas X termina antes de Y.)
- **X finishes Y** (X *finaliza Y*) se e somente se $x_i > y_i$ e $x_f = y_f$. (X e Y terminam juntos, mas X começa depois de Y.)

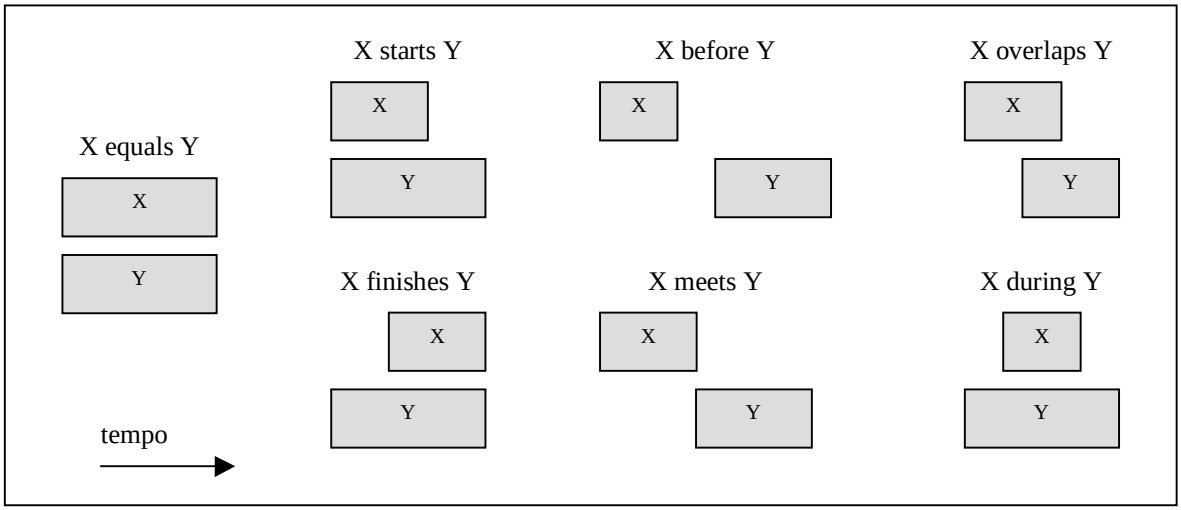

Figura 3.1: As sete relações temporais básicas entre intervalos de tempo.

X before Y (X *antes de Y*) se e somente se $x_f < y_i$. (X ocorre antes de Y, e eles não se sobrepõem.)

X meets Y (X *encontra Y*) se e somente se $x_f = y_i$. (Y começa imediatamente após o término de X – não há intervalo entre X e Y.)

X overlaps Y (X *sobrepõe Y*) se e somente se $x_i < y_i < x_f < y_f$. (X começa antes de Y, que começa antes de X terminar; X deve terminar antes de Y.)

X during Y (X *durante Y*) se e somente se $x_i > y_i$ e $x_f < y_f$. (X está totalmente contido em Y.)

A partir das relações acima, são estabelecidos axiomas que definem a lógica temporal. Por exemplo, existem axiomas para provar que as relações acima são mutuamente exclusivas e exaustivas [Zaidi 99]:

1) Se (X Ri Y), sendo Ri uma das relações acima, não existe um outro Rj ≠ Ri dentre as relações acima para o qual (X Rj Y) seja verdadeiro (exclusão mútua).
2) Para dois intervalos quaisquer X e Y, sempre existe uma relação Ri para a qual (X Ri Y) ou (Y Ri X) é verdadeira.

Outros axiomas são definidos para definir relações de transitividade (por exemplo, se *X during Y* e *Y before Z*, é inferido que *X before Z*) [Allen 83].

A lógica de Allen não faz nenhuma menção explícita à duração dos intervalos, o que é uma característica importante para a sua grande aceitação. Ela é usada em uma série aplicações nas mais variadas áreas, por exemplo, para a sincronização de apresentações multimídia [Wahl 94] e para a criação de *scripts* que comandam sensores e acionadores no gerenciamento da interação com o usuário em ambientes imersivos [Pinhanez 97].

Assim, a lógica temporal de Allen foi definida em um contexto em que são importantes propriedades como a definição de um conjunto mínimo de relações básicas, a exclusão mútua entre estas relações e a possibilidade de realizar combinações e inferências sobre elas.

As dependências temporais entre atividades colaborativas se inserem em outro contexto. Por esta razão foram necessárias algumas adaptações às relações básicas do modelo de Allen, adicionando algumas variações das relações originalmente propostas e

criando outras novas. A principal diferença do contexto de tarefas colaborativas é que em muitos casos é possível relaxar algumas restrições impostas pelas relações originais, como será explicado a seguir. O resultado da adaptação das relações de Allen para o contexto de dependências temporais entre atividades colaborativas é o conjunto de treze dependências temporais apresentado a seguir.

Considere-se duas tarefas T1 e T2, que ocorrem nos intervalos $[t1_i, t1_f)$ e $[t2_i, t2_f)$, respectivamente:

- **T1 equals T2**: esta dependência estabelece que as duas tarefas devem ocorrer simultaneamente. É a mesma relação originalmente definida por Allen ($t1_i = t2_i$ e $t1_f = t2_f$).
- **T1 startsA T2**: as duas tarefas começam juntas e a primeira deve terminar antes. É a mesma relação original ($t1_i = t2_i$ e $t1_f < t2_f$).
- **T1 startsB T2**: variação da relação original, relaxando a obrigação de qual tarefa deve terminar antes. Neste caso, a única obrigação é que as duas tarefas iniciem juntas ($t1_i = t2_i$). Esta variação é importante porque em algumas situações o objetivo pode ser apenas disparar a execução simultânea de duas tarefas, não importando quando elas terminarão. Um exemplo é quando um usuário realiza alguma mudança na tela de uma aplicação WYSIWIS[1], que deve ser imediatamente propagada para todos os outros usuários que compartilham a aplicação. Nesse caso, só importa definir que a alteração original dispara (startsB) as alterações nos outros usuários.
- **T1 finishesA T2**: as duas tarefas terminam juntas e a primeira deve começar depois. É a relação original ($t1_i > t2_i$ e $t1_f = t2_f$).
- **T1 finishesB T2**: da mesma forma que startsB, esta dependência é obtida a partir da relação original, relaxando a restrição de que T1 deve começar depois de T2. Neste caso, a única obrigação é que ambas as tarefas terminem juntas ($t1_f = t2_f$). Assim como ocorre com startsB, esta dependência é importante em situações em que não importa quando as tarefas começam, mas apenas que elas devem terminar simultaneamente.
- **T2 afterA T1**: T2 só pode ser executada se T1 já tiver terminado antes (a restrição ocorre na execução de T2; T1 pode ser executada livremente). Esta dependência é a relação de pré-requisito, freqüentemente usada em aplicações colaborativas. Neste caso, T2 pode ser executada apenas uma vez após cada execução de T1 ($t1_{f,n} < t2_{i,n}$, $\forall n > 0$, onde o sub-índice *n* indica a n-ésima execução da tarefa).
- **T2 afterB T1**: variação da dependência anterior, na qual T2 pode ser executada várias vezes após uma única execução de T1 ($t1_{f,1} < t2_{i,n}$, $\forall n > 0$).
- **T1 before T2**: do ponto de vista da lógica temporal, esta relação pode ser vista como oposta à anterior, mas ela gera um mecanismo de coordenação totalmente diferente, como será visto mais adiante (formalmente, a definição é a mesma: $t1_f < t2_i$). No entanto, diferentemente da dependência anterior, a restrição aqui ocorre na execução de T1, que não poderá mais acontecer caso T2 já tenha iniciado sua execução. Não há restrições na execução de T2 (T2 não fica esperando a execução de T1, como ocorreria em T2 afterA T1).

[1] WYSIWIS (*What You See Is What I See*) é um tipo de interface para aplicações colaborativas na qual todos os usuários compartilham a mesma visão da interface [Stefik 87]. Alterações locais são propagadas para todos os usuários.

T1 meets T2: T2 deve começar imediatamente após o término de T1. É a mesma relação original ($t1_f = t2_i$).

T1 overlapsA T2: T1 começa antes de T2 e T2 deve começar antes do término de T1, que por sua vez deve terminar antes de T2. É a relação original ($t1_i < t2_i < t1_f < t2_f$).

T1 overlapsB T2: variação da relação original, na qual não importa qual tarefa deve terminar antes. A única obrigação é que T1 comece antes de T2 e T2 comece antes do término de T1 ($t1_i < t2_i < t1_f$).

T1 duringA T2: T1 deve ser totalmente realizada durante a execução de T2. Neste caso, uma única execução de T1 é permitida durante uma execução de T2 ($t1_{i,n} > t2_{i,n}$ e $t1_{f,n} < t2_{f,n}$, $\forall n > 0$).

T1 duringB T2: variação da dependência anterior, na qual T1 pode ser realizada várias vezes durante uma única execução de T2 ($t1_{i,n} > t2_{i,m}$ e $t1_{f,n} < t2_{f,m}$, $\forall m > 0$ e $\forall (n \geq m)$).

Estas treze dependências temporais são ilustradas na Figura 3.2.

O conjunto de dependências apresentado nesta seção engloba apenas as dependências associadas à sincronização das tarefas. Dependências relacionadas ao uso de recursos podem aparecer em paralelo às temporais e são o assunto da próxima seção.

3.1.2. Dependências de Gerenciamento de Recursos

As dependências de gerenciamento de recursos são complementares às temporais e podem aparecer independentemente destas. As dependências apresentadas nesta seção lidam com o gerenciamento do acesso seqüencial ou simultâneo a um mesmo recurso. Um exemplo é quando dois ou mais participantes desejam alterar simultaneamente o mesmo trecho de um documento em um sistema de autoria colaborativa [Raposo 00b] (ver exemplo no Capítulo 4 e o artigo no Apêndice D).

Neste contexto, o termo "recurso" está sendo usado de maneira mais ampla que o normalmente usado em *workflows* (ver Apêndice B), referindo-se não apenas ao agente que realiza a tarefa, mas também a qualquer artefato necessário à realização da tarefa.

São definidas três dependências básicas de gerenciamento de recursos:

Compartilhamento ou divisão de recursos: um número limitado de recursos precisa ser compartilhado entre várias tarefas. É um caso muito comum que ocorre, por exemplo, quando vários usuários querem editar simultaneamente um documento ou quando vários usuários querem usar uma impressora.

Simultaneidade no uso de recursos: o recurso só fica disponível se um determinado número de tarefas requisitá-lo simultaneamente. Ocorre, por exemplo, na abertura de canais de comunicação (precisam de dois usuários para serem abertos) ou no caso de uma máquina que precisa de mais de um operador para funcionar.

Volatilidade de recursos: indica se após o uso o recurso volta a estar disponível. Documentos e impressoras são exemplos de recursos não voláteis (em situações normais, estão novamente disponíveis após o uso). Folhas de papel e mercadorias em estoque são exemplos de recursos voláteis (não estão mais disponíveis após o uso ou a venda, respectivamente).

As dependências de gerenciamento de recurso, diferentemente das temporais, não são relações binárias. É possível, por exemplo, que duas ou mais tarefas compartilhem um

recurso. Além disso, essas dependências requerem um parâmetro para indicar o número de instâncias do recurso compartilhado, o número de tarefas que devem solicitar um recurso simultaneamente, ou o número de vezes que um recurso pode ser utilizado (volatilidade).

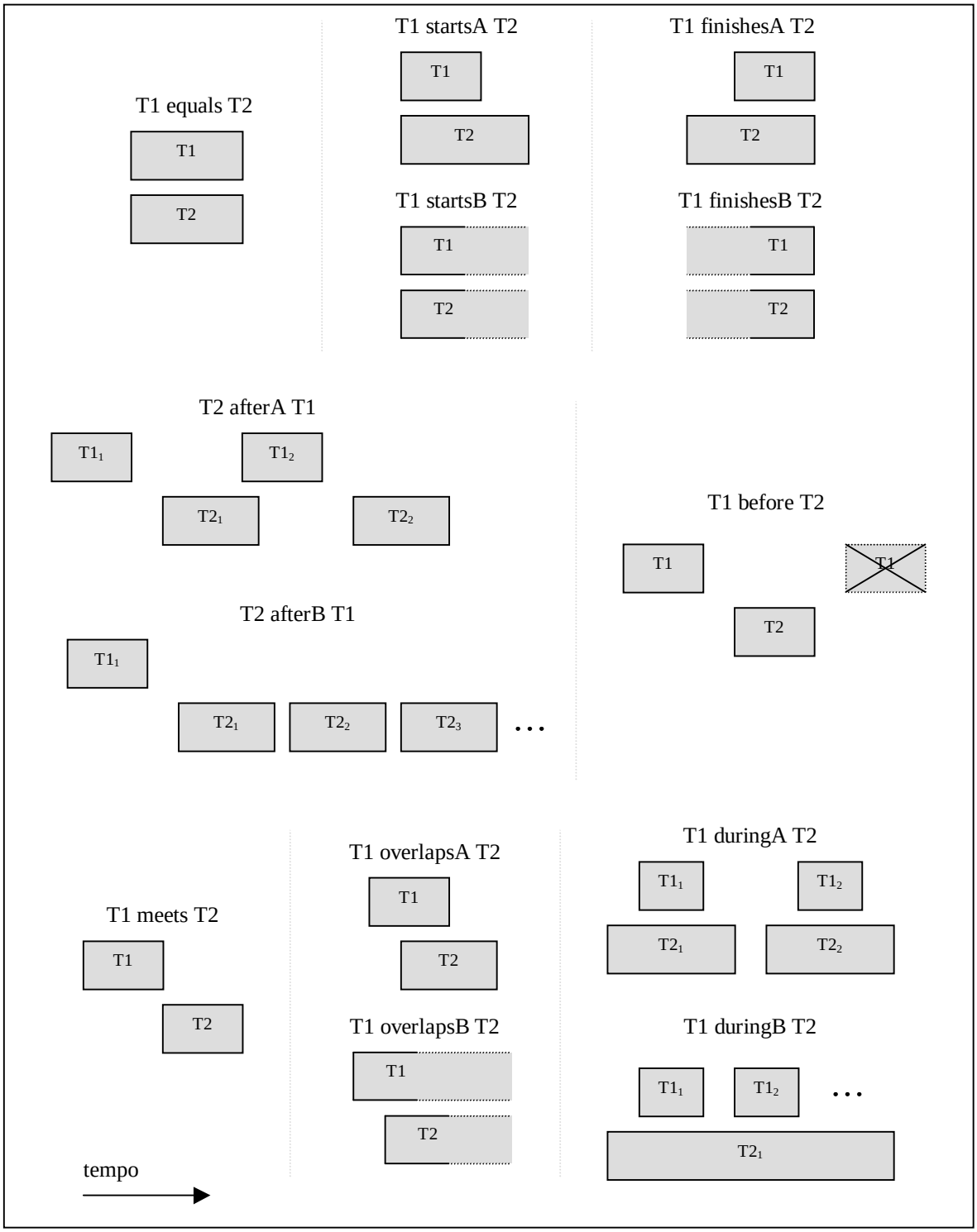

Figura 3.2: As treze dependências temporais entre tarefas.

A Figura 3.3 ilustra as três dependências básicas de gerenciamento de recursos, considerando que as tarefas *Ti* requisitam um determinado recurso.

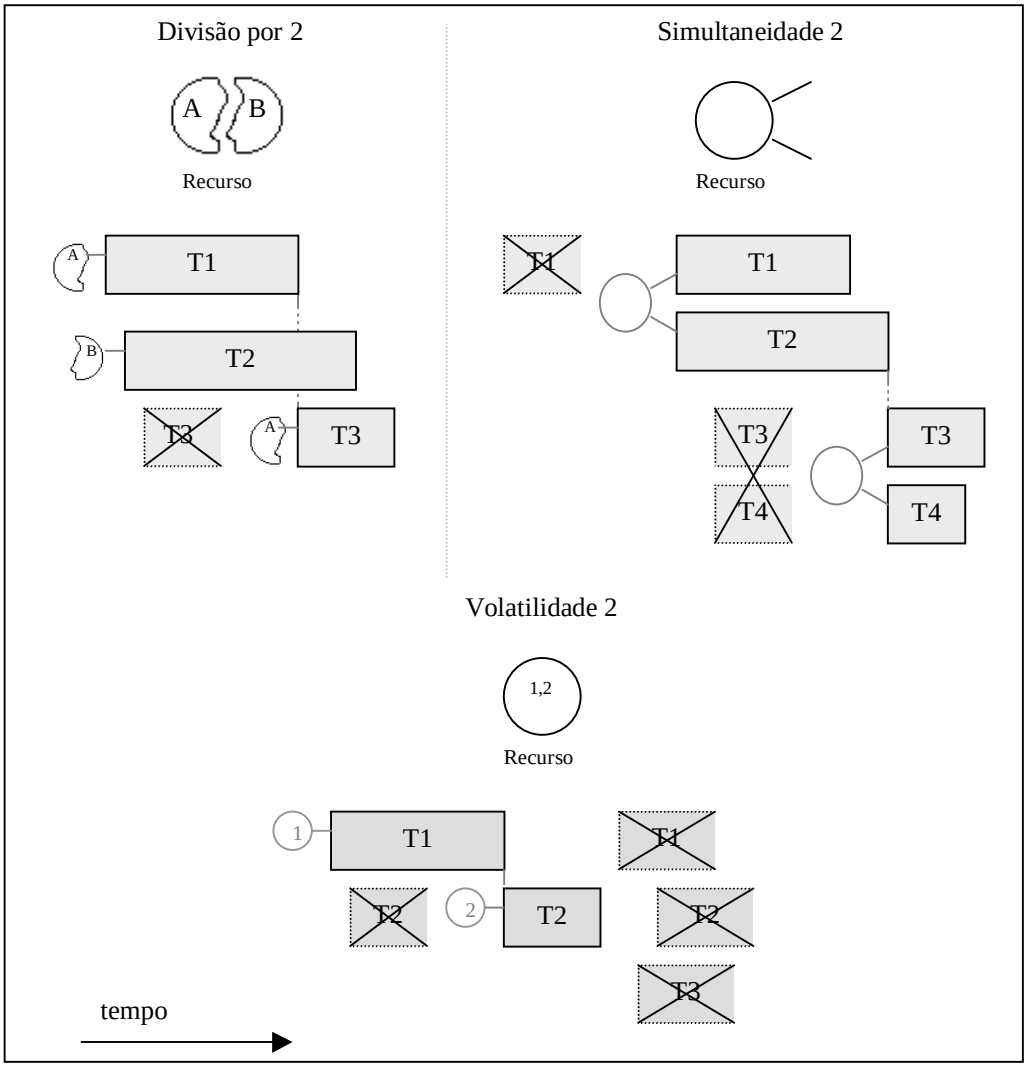

Figura 3.3: As três dependências básicas de gerenciamento de recursos.

A partir das três dependências básicas acima apresentadas é possível criar dependências derivadas para as combinações entre elas:

Divisão por M com Simultaneidade N: representa a situação em que até M grupos de N tarefas podem compartilhar um recurso simultaneamente.

Divisão por M com Volatilidade N: situação em que até M tarefas podem compartilhar o recurso simultaneamente e o recurso só pode ser usado N vezes.

Simultaneidade M com Volatilidade N: grupos de M tarefas podem compartilhar o recurso, que pode ser usado até N vezes.

Divisão por M com Simultaneidade N com Volatilidade Q: até M grupos de N tarefas podem compartilhar o recurso simultaneamente. O recurso pode ser utilizado até Q vezes.

A Figura 3.4 ilustra as quatro dependências compostas, considerando novamente que as tarefas *Ti* requisitam um determinado recurso.

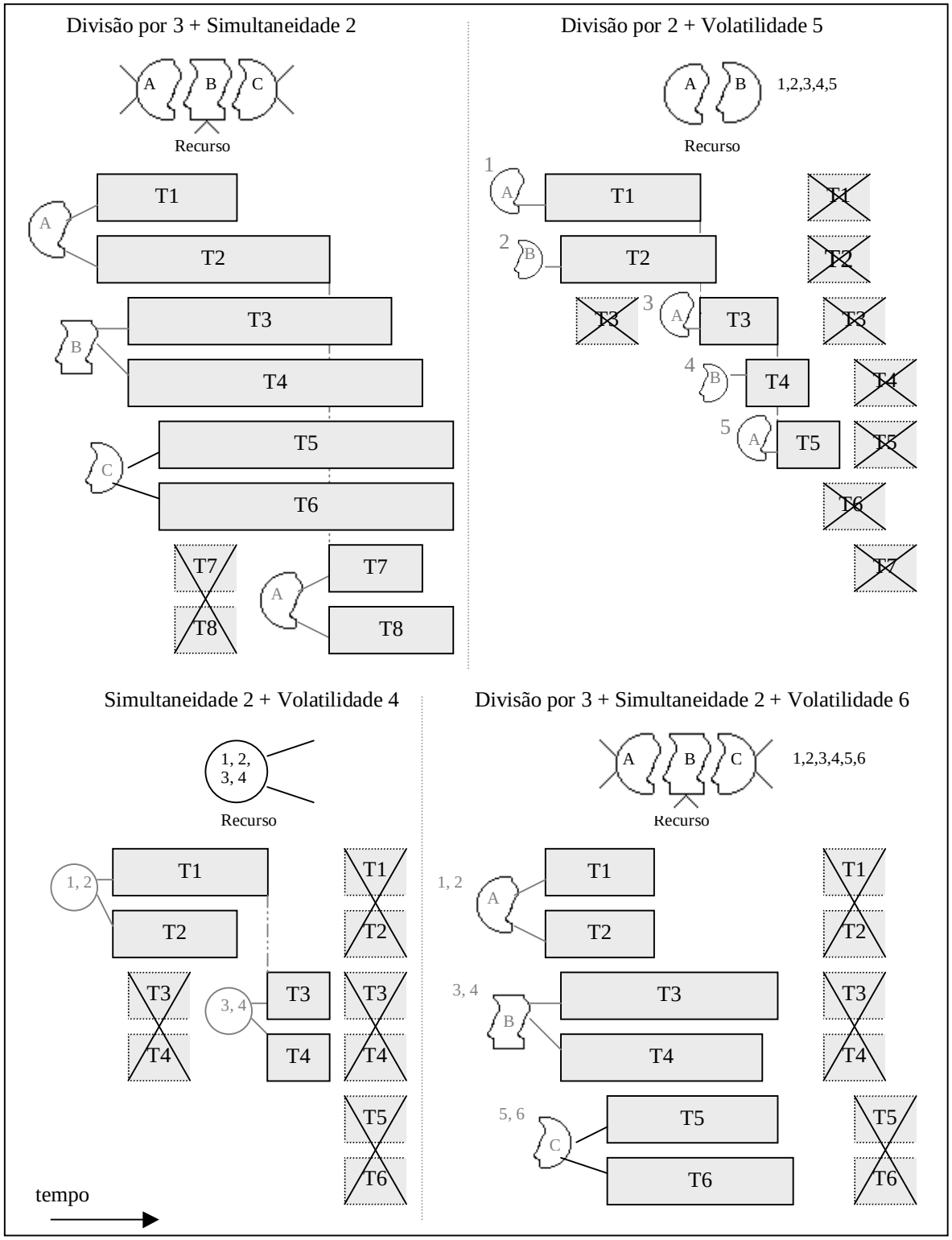

Figura 3.4: As quatro dependências compostas de gerenciamento de recursos.

3.2. Níveis Hierárquicos do Modelo de Coordenação

Após a definição do conjunto de possíveis interdependências entre as tarefas, na seqüência serão tratados os modelos dos mecanismos de coordenação para controlar essas interdependências. Para isso, será modelado, usando PNs, o conjunto de mecanismos de coordenação associados às interdependências apresentadas na seção anterior.

São construídas várias PNs padrões que modelam gerenciadores de recursos e a sincronização entre as tarefas para os vários tipos de interdependências descritas na seção anterior. A idéia é fazer com que o projetista do sistema colaborativo se preocupe apenas com a definição da rede que modela o seqüenciamento das tarefas e com a definição das interdependências entre elas e não mais com os mecanismos para gerenciar estas dependências, pois eles já estão pré-definidos.

No esquema proposto, o ambiente colaborativo é modelado em três níveis hierárquicos: nível de *workflow*[2], nível de coordenação e nível de especificação das tarefas. No nível de *workflow* são definidos o seqüenciamento das tarefas e as dependências entre elas. No nível de coordenação, as tarefas interdependentes são expandidas (de acordo com o modelo que será mostrado mais adiante) e os mecanismos de coordenação são inseridos entre elas. O nível de especificação das tarefas expande a tarefa propriamente dita em uma PN que modela sua execução. Este último nível não será tratado neste trabalho pois, para efeito de simulação e análise, é possível considerar as tarefas como "caixas pretas", não importando os detalhes de sua execução. A Figura 3.5 ilustra a relação entre estes três níveis.

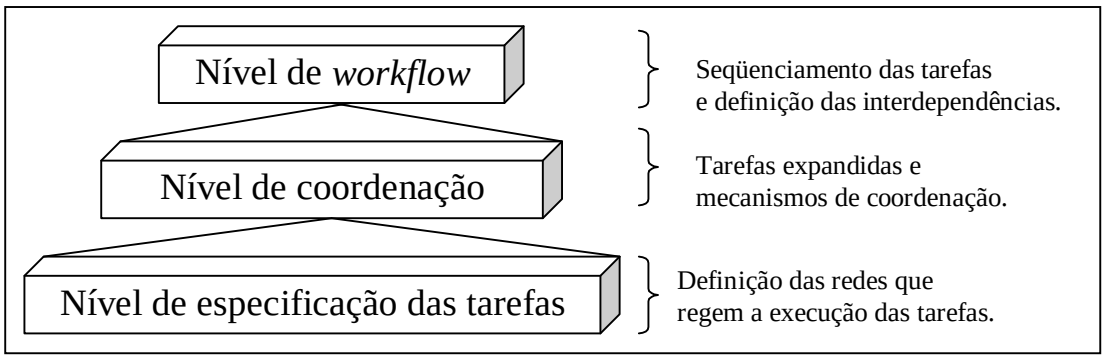

Figura 3.5: Os níveis hierárquicos do modelo de coordenação.

No nível de *workflow*, cada participante da colaboração tem seu comportamento modelado por uma PN. Suas tarefas são representadas por transições da rede. A relação entre as tarefas é definida por meio de interdependências. A Figura 3.6 mostra o nível de *workflow* para uma situação hipotética de colaboração. Nessa figura, aparecem duas PNs representando a seqüência de tarefas de dois participantes. As dependências existentes entre as tarefas também estão representadas na figura. Note-se que as dependências podem ocorrer entre tarefas de participantes diferentes (*Tarefa1a overlapsB Tarefa1b*, por exemplo) ou entre tarefas de um mesmo participante (*Tarefa3b duringB Tarefa2b*).

[2] O nível de *workflow* tem esse nome porque nesse nível o ambiente colaborativo é visto basicamente como uma seqüência de atividades executadas pelos vários participantes da colaboração, o que é, em essência, a definição de *workflow* [Workflow 98] – ver Apêndice B.

Durante a passagem do nível de *workflow* para o nível de coordenação, as tarefas (transições) que possuírem interdependências com outras tarefas (representadas por retângulos não preenchidos no nível de *workflow* – ver Figura 3.6) são expandidas em subredes como a da Figura 3.7 [van der Aalst 94]. Os cinco lugares associados às transições (*request_resource*, *assigned_resource*, *start_task*, *finish_task* e *release_resource*) representam a interação com o gerenciador de recursos e com o agente que 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 *start_task* e *finish_task* marcam, respectivamente, o início e o final da tarefa (interação com o agente que realiza a tarefa). 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 independentes, 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 *start_task* como lugar de entrada e *finish_task* como lugar de saída). Estas duas subredes representam exatamente os dois tipos de dependências discutidos na seção anterior (temporais e de gerenciamento de recursos) – ver Figura 3.8.

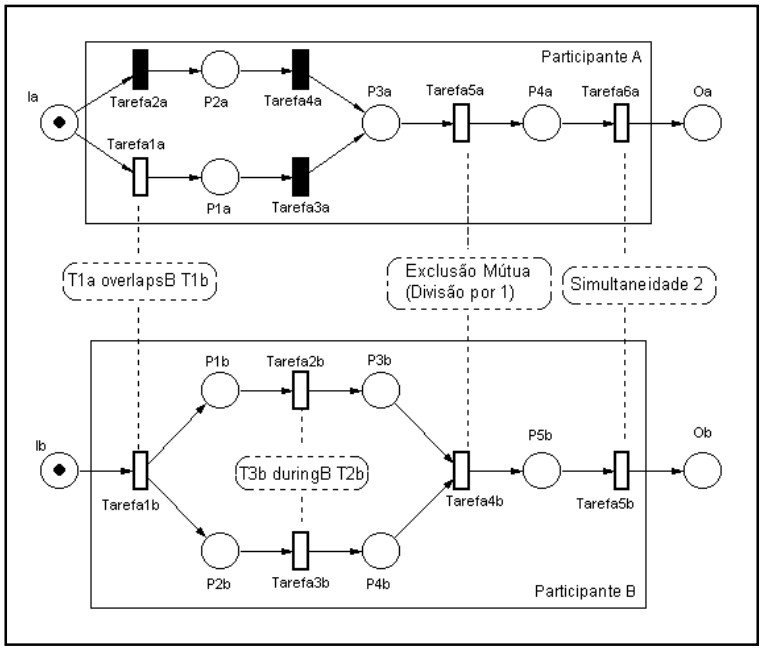

Figura 3.6: Exemplo de modelo no nível de *workflow*.

Como os mecanismos de coordenação para dependências temporais e de gerenciamento de recursos estão em subredes independentes, é possível simplificar a expansão das PNs. Se a tarefa possui somente dependências temporais, ela pode ser expandida apenas com os lugares *start_task* e *finish_task*. Da mesma forma, se ela possuir somente dependências de gerenciamento de recursos, ela pode ser expandida apenas com os lugares *request_resource*, *assigned_resource* e *release_resource*. A tarefa só precisa ser expandida por completo, como na Figura 3.7, se possuir os dois tipos de dependências. A Figura 3.9 ilustra os modelos simplificados de expansão de tarefas.

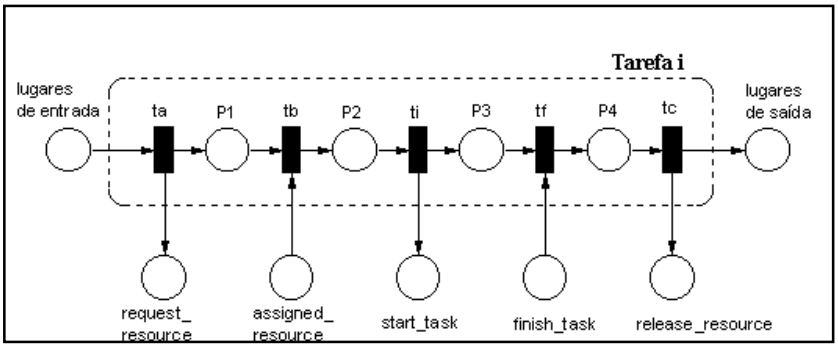

Figura 3.7: Estrutura de uma tarefa expandida no nível de coordenação.

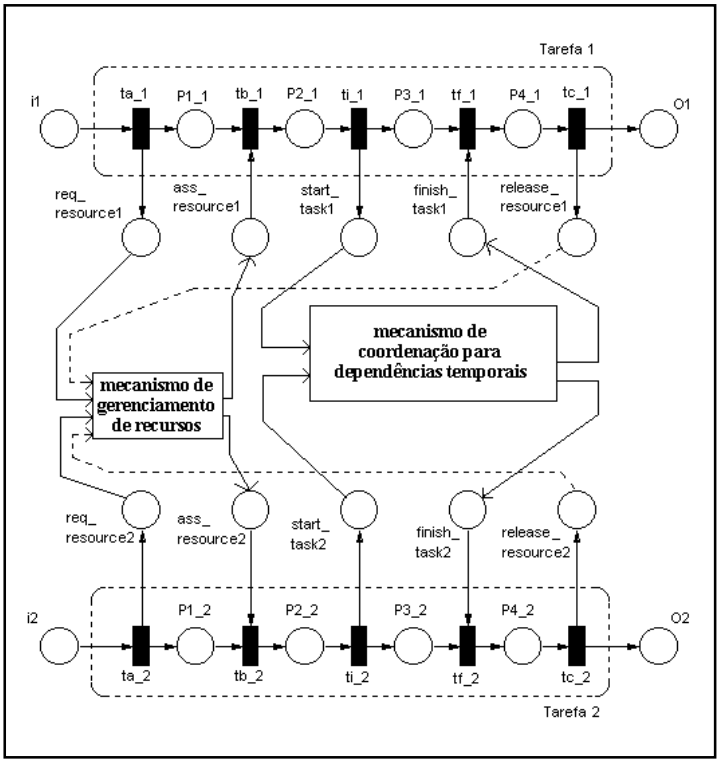

Figura 3.8: Duas tarefas expandidas e os mecanismos de coordenação que podem ser inseridos entre elas.

Após a expansão das tarefas, os mecanismos de coordenação correspondentes às dependências definidas no nível de *workflow* são inseridos entre as tarefas expandidas. Com o modelo do sistema colaborativo no nível de coordenação é possível prever o comportamento do mesmo por meio das análises e simulações pertinentes.

O objetivo dos mecanismos de coordenação apresentados nas próximas seções 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 3.7 (ou as versões simplificadas da Figura 3.9) e os mecanismos de coordenação reutilizáveis, o que pode ser feito automaticamente (ver Seção 3.5). Dessa forma, o projetista do sistema colaborativo só precisa modelá-lo no nível de *workflow* para poder analisar seu comportamento.

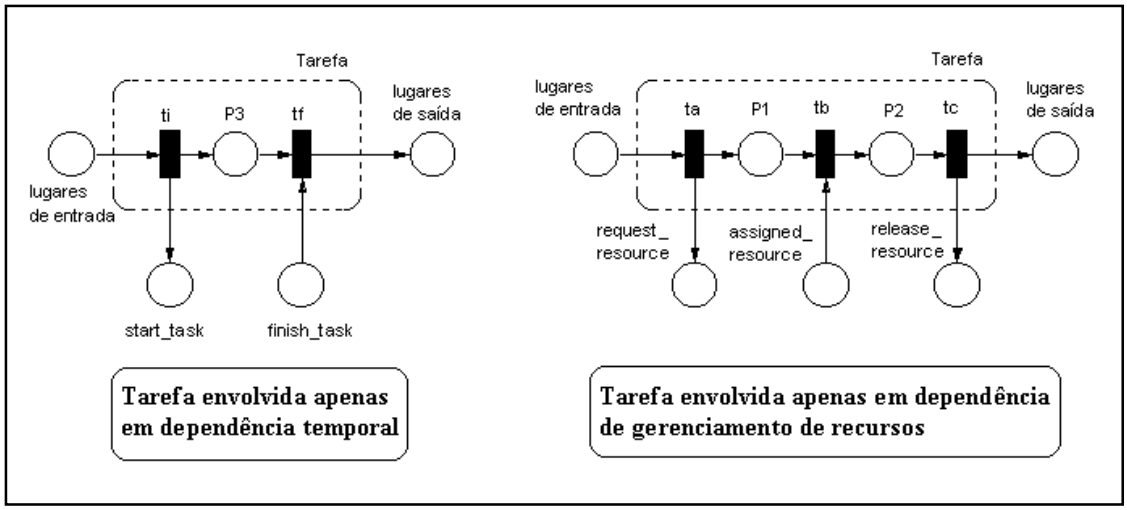

Figura 3.9: Simplificações do modelo de tarefa expandida

3.3. Mecanismos de Coordenação: Modelos Usando PNs Simples

Os modelos de mecanismos de coordenação apresentados nas próximas seções servem para garantir que as dependências estabelecidas entre as tarefas não serão violadas nos modelos. Estes mecanismos são PNs inseridas nas "caixas" da Figura 3.8. Como já comentado, mecanismos de coordenação para dependências temporais são inseridos entre os lugares *start_task* e *finish_task* do modelo expandido da tarefa. Mecanismos de coordenação para dependências de gerenciamento de recursos, por sua vez, são colocados entre *request_resource*, *assigned_resource* e *release_resource*. Dessa forma, os dois tipos de dependências podem coexistir de maneira independente.

3.3.1. Dependências Temporais

Esta seção apresenta os mecanismos de coordenação para as treze dependências temporais ilustradas na Figura 3.2.

3.3.1.1. Tarefa 1 equals 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 *start_task* de ambas) e que elas terminarão juntas (*tokens* enviados simultaneamente aos lugares *finish_task*). A subrede mostrada na Figura 3.10 representa o modelo proposto para a coordenação deste tipo de dependência utilizando PNs simples (um modelo similar utilizando PNs de alto nível será mostrado na Seção 3.4).

Na Figura 3.10, a transição *t1* é uma atividade de controle que constitui ao mesmo tempo um *AND-join* e um *AND-split* (ver Apêndice B), 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 a real execução das tarefas. 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 [Ramamoorthy 80]). 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 (ver Apêndice A).

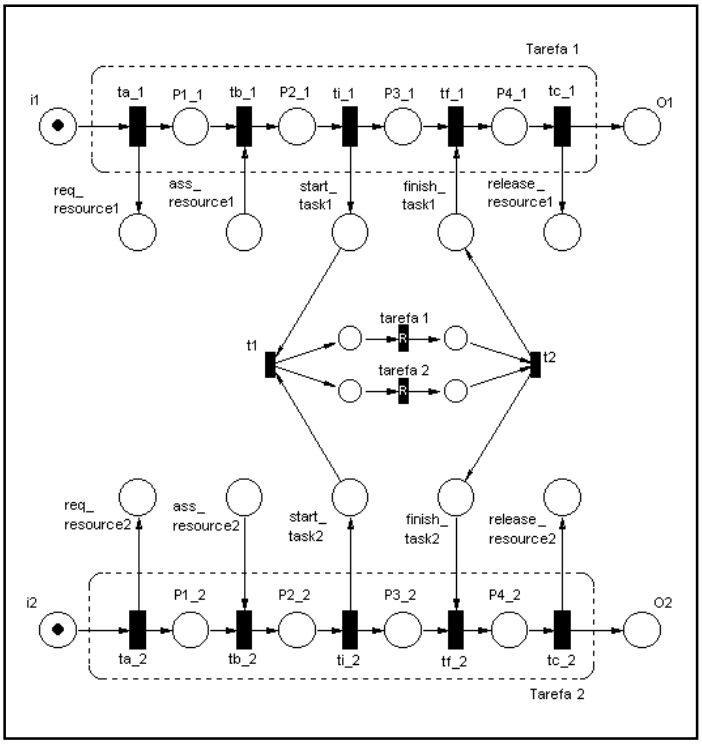

Figura 3.10: Mecanismo de coordenação para a dependência *tarefa 1 equals tarefa 2*.

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

Do ponto de vista da lógica temporal, o modelo da Figura 3.10 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 reduzir a possibilidade *deadlocks*. Isso porque as tarefas 1 e 2 podem, por exemplo, pertencer a caminhos alternativos (condicionais) de PNs diferentes. Desse modo, caso o participante 1 opte pelo caminho que passe pela tarefa 1, e o participante 2 opte 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 possibilidade seria acrescentar um mecanismo de *timeout*[3], de modo que uma tarefa

[3] É importante ressaltar que os *timeouts* aparecem como opção nos modelos dos mecanismos de coordenação com o objetivo de dar continuidade às simulações (evitar *deadlocks*). O disparo dos *timeouts*, no entanto, pode indicar a existência de problemas que precisarão ser tratados na implementação do ambiente colaborativo. Em resumo, os *timeouts* não aparecem aqui como solução de controle, mas apenas de modelagem.

alternativa 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 *timeout* (denominado *timeoutA*) é mostrado na Figura 3.11, onde estão instanciados como *time_out1* e *time_out2*.

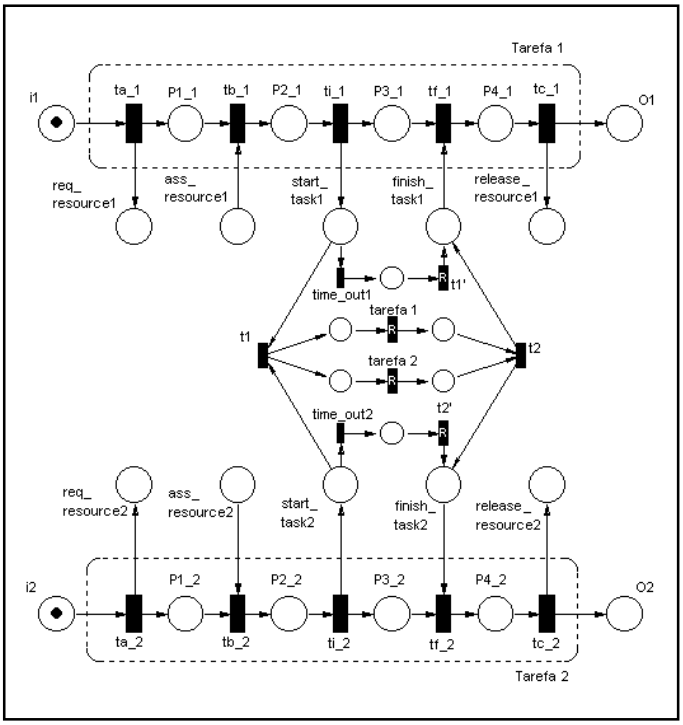

Figura 3.11: Mecanismo de coordenação para *tarefa 1 equals tarefa 2* com *timeoutA*.

Na Figura 3.11 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 *timeouts*.

O modelo da Figura 3.11 permite a execução de tarefas alternativas após um certo tempo, mesmo contrariando a relação de dependência. Uma outra possibilidade propõe um *timeout* ligando o *start_task* ao lugar de entrada *i* da tarefa, de modo que ela volte ao seu estado inicial decorrido um certo tempo de espera e a colaboração possa seguir um caminho alternativo que não passe pela tarefa bloqueada. Caso haja dependência de gerenciamento de recursos, é necessário também colocar um *token* em *release_resource*, para que o gerenciador de recursos continue funcionando corretamente. Essa alternativa é mostrada na Figura 3.12 (este tipo de *timeout* é chamado *timeoutB*).

3.3.1.2. Tarefa 1 starts Tarefa 2

Essa relação estabelece que as duas tarefas começam juntas. Em uma das variações (*startsB*) não importa qual tarefa termina primeiro. Na outra variação (*startsA*), a tarefa 1 deve ser finalizada antes da outra.

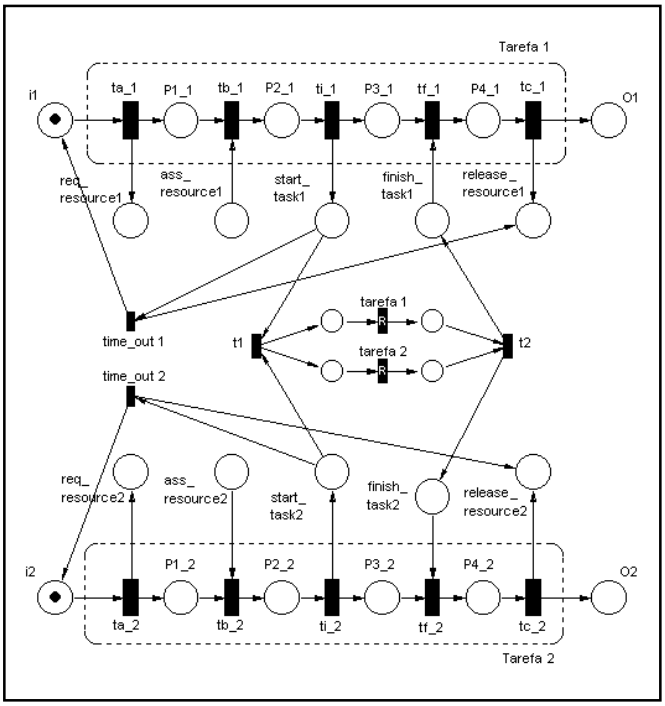

Figura 3.12: Mecanismo de coordenação para *tarefa 1 equals tarefa 2* com *timeoutB*.

A forma sem a restrição de uma tarefa terminar primeiro é na verdade um subconjunto da relação anterior (*tarefa 1 equals tarefa 2*), onde só a primeira metade da subrede é utilizada (a que garante que as duas tarefas começam juntas). Essa relação é modelada na Figura 3.13 (note-se que a tarefa 1 tem um *timeoutB*).

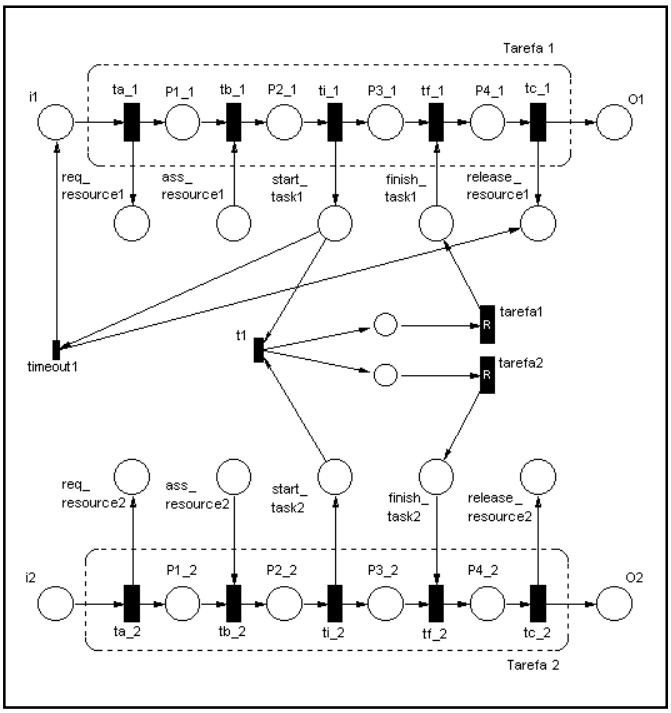

Figura 3.13: Mecanismo de coordenação para a dependência *tarefa 1 startsB tarefa 2*.

Para garantir que a tarefa 1 termine antes da tarefa 2 (*startsA*), 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 3.14 mostra o mecanismo de coordenação para esta dependência. Note-se que a única diferença com relação ao mecanismo de coordenação da relação *tarefa 1 equals tarefa 2* (Figura 3.10) é que a tarefa 1 não espera a tarefa 2 terminar (o *finish_task1* recebe o *token* logo após a realização da tarefa 1).

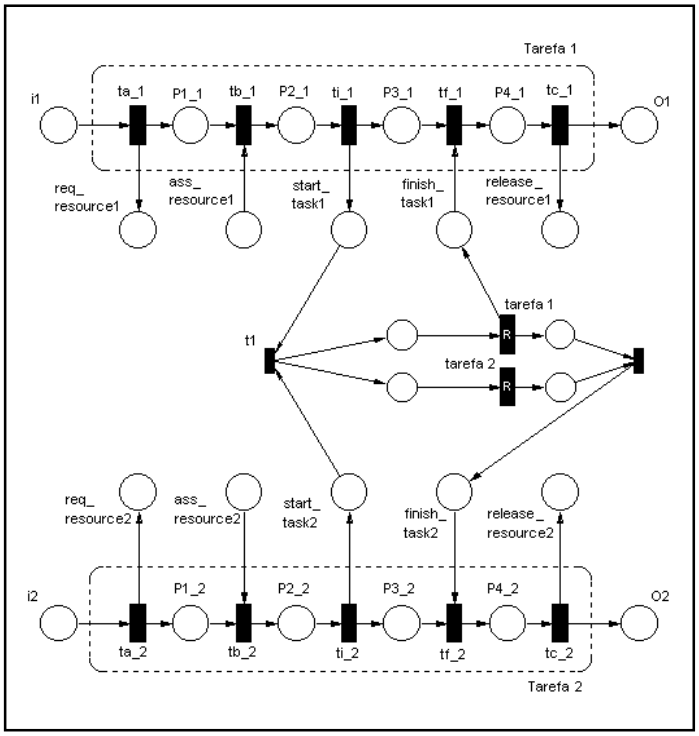

Figura 3.14: Mecanismo de coordenação para a dependência *tarefa 1 startsA tarefa 2*.

Nas dependências *tarefa 1 starts tarefa 2* é possível utilizar os mesmos *timeouts* da dependência anterior (a Figura 3.13 mostrou um exemplo de uso do *timeoutB*).

3.3.1.3. Tarefa 1 finishes Tarefa 2

Assim como no caso anterior, é possível modelar esta relação de duas maneiras. A primeira (*finishesB*) 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 equals 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 3.15 (desta figura em diante os mecanismos de coordenação para dependências temporais serão mostrados apenas com os lugares *start_task* e *finish_task* do modelo completo de tarefas da Figura 3.7, já que o restante do modelo não é utilizado neste tipo de mecanismo – o que trata das dependências temporais, como se pode ver nas Figuras 3.10 a 3.14).

No modelo da Figura 3.15 existem *timeouts* para evitar que uma tarefa fique esperando indefinidamente o término da outra.

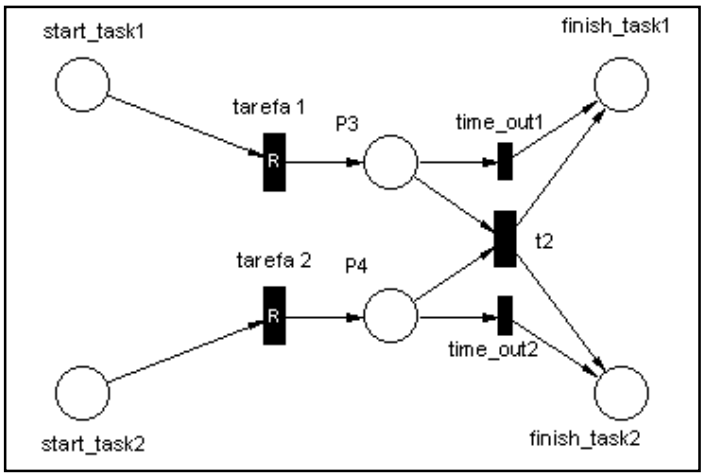

Figura 3.15: Mecanismo de coordenação para a dependência *tarefa 1 finishesB tarefa 2*.

A definição original desta relação (*finishesA*) exige que a tarefa 1 comece após a tarefa 2. Para modelá-la (Figura 3.16), 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 *timeout*, pois a tarefa 1 só começa se a outra estiver sendo executada, não existindo a possibilidade de sua execução ficar "bloqueada".

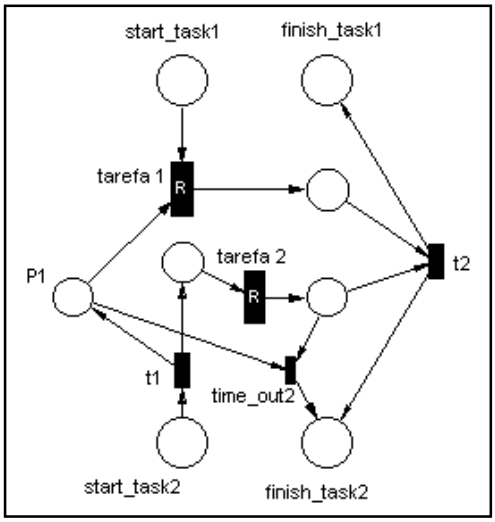

Figura 3.16: Mecanismo de coordenação para a dependência *tarefa 1 finishesA tarefa 2*.

Na Figura 3.16 observa-se 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 *timeout* para evitar que a tarefa 1 fique esperando indefinidamente pelo início da tarefa 2, caso ela termine sem a tarefa 2 ter sido iniciada.

3.3.1.4. Tarefa 2 after Tarefa 1

Esta é uma dependência temporal que derivou da relação *before* da álgebra de intervalos de Allen. A dependência *tarefa 2 after tarefa 1* impõe uma restrição sobre a execução da tarefa 2, que só pode ser executada após o término da tarefa 1. A tarefa 1 não tem nenhuma restrição. A relação *tarefa 1 before 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).

A relação *tarefa 2 after 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 3.17).

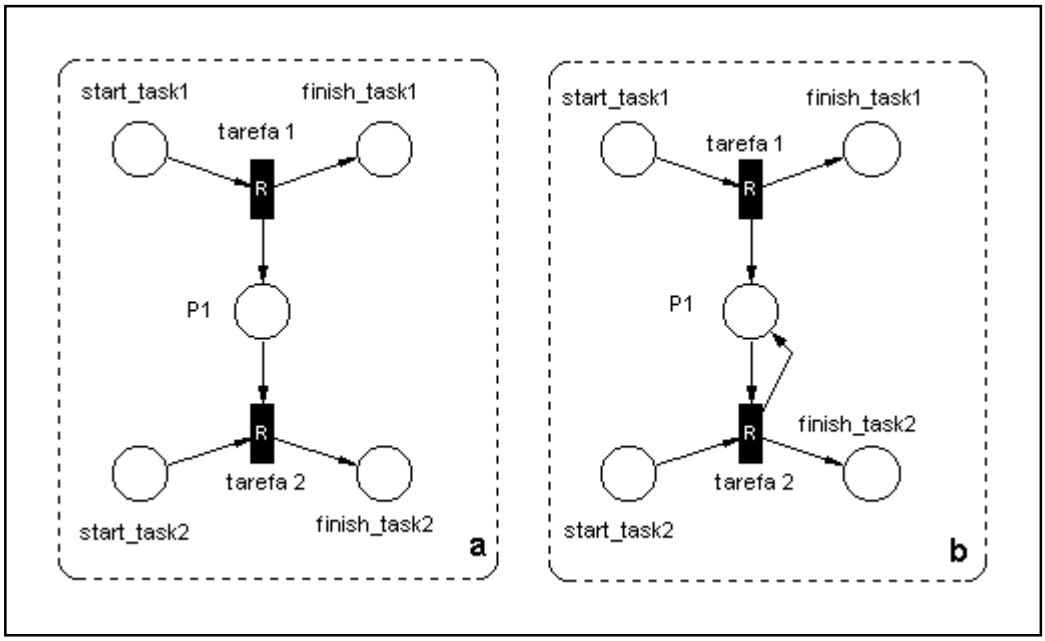

Figura 3.17: Mecanismos de coordenação para as dependências *tarefa 2 after tarefa 1*.

No modelo da Figura 3.17a (*afterA*), 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). A outra alternativa é estabelecer que uma única execução da tarefa 1 dá direito a inúmeras execuções da tarefa 2 (*afterB*). Para isso, a única alteração necessária no modelo é adicionar um arco ligando a transição *tarefa 2* ao lugar *P1* (Figura 3.17b). Observa-se que em ambos os casos a rede não é *k-bounded* (ver Apêndice A), a menos 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 até 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 3.17a). Este valor indicaria o número de vezes que a tarefa 2 poderia ocorrer (número de *tokens* em P1).

Para reduzir a possibilidade de *deadlocks*, os modelos acima podem ser acrescidos dos dois tipos de *timeout* discutidos anteriormente.

3.3.1.5. Tarefa 1 before Tarefa 2

Como já comentado, a restrição nesta relação ocorre sobre a tarefa 1, que não poderá mais ser executada após o início da 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), que só permitem o disparo da transição de destino se o lugar de origem estiver vazio. O núcleo do modelo criado para esta relação é a transição *t1* (Figura 3.18). Esta transição determina a execução da tarefa 2, e fica inibida se houver *tokens* em *start_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, conseqüentemente, a execução da tarefa 1. Se a tarefa 2 ainda não tiver iniciado sua execução (sem *token* em *P1*), *t2* pode ser disparada, enviando *tokens* para *P2* (habilita a tarefa 1) 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 *start_task1* (a tarefa 1 não poderá ocorrer mais). É sempre aconselhável colocar um *timeoutB* na tarefa 1 para que ela sempre volte ao seu estado inicial quando não puder mais ser executada (ou seja, depois que a tarefa 2 tiver iniciado sua execução).

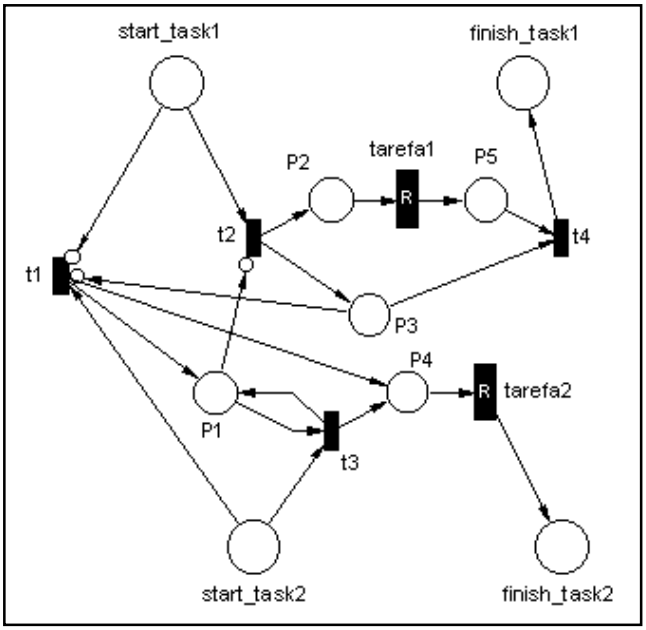

Figura 3.18: Mecanismo de coordenação para a dependência *tarefa 1 before tarefa 2*.

3.3.1.6. Tarefa 1 meets Tarefa 2

De acordo com esta dependência, a tarefa 2 deve começar imediatamente após o término da 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 3.19, esta dependência é satisfeita colocando o lugar *start_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 *start_task2* após seu término. A conclusão da tarefa 1, habilita a tarefa 2 (*token* em *P1*). O modelo apresentado possui um *timeout*, para evitar que a tarefa 1 espere indefinidamente a habilitação da tarefa 2. A tarefa 2 também pode possuir um *timeoutA* (executando uma tarefa alternativa após um tempo de espera) ou um *timeoutB* (retornando ao estado inicial).

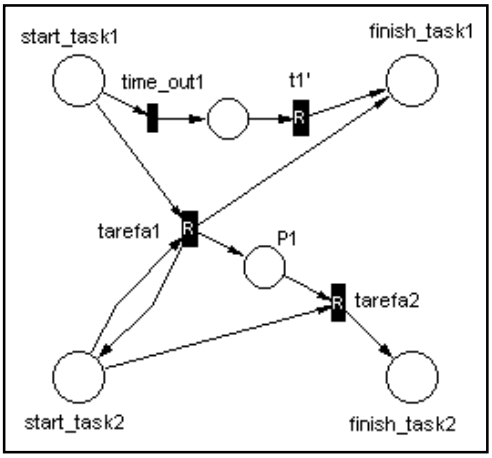

Figura 3.19: Mecanismo de coordenação para a dependência *tarefa 1 meets tarefa 2*.

3.3.1.7. Tarefa 1 overlaps Tarefa 2

Esta relação também permite a criação de dois modelos diferentes. De acordo com a definição original de Allen (*overlapsA*), a tarefa 2 deve começar antes do término da tarefa 1, e esta deve terminar antes da tarefa 2. O primeiro modelo apresentado (*overlapsB*), no entanto, relaxa a segunda parte da definição, não se preocupando com qual das duas tarefas termina primeiro. Este modelo é apresentado na Figura 3.20.

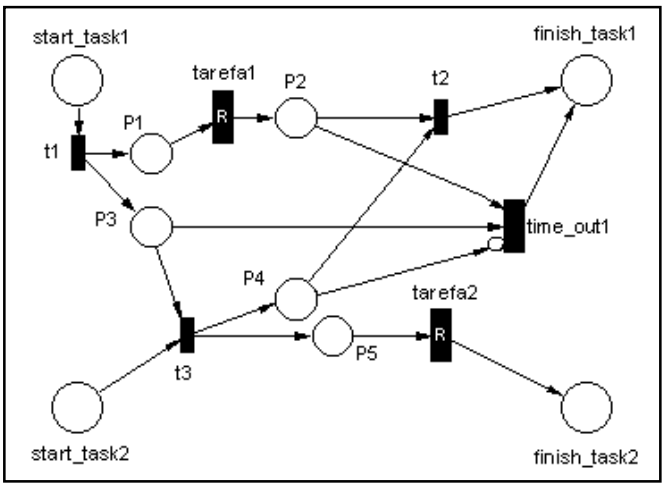

Figura 3.20: Mecanismo de coordenação para a dependência *tarefa 1 overlapsB tarefa 2*.

Pelo mecanismo da Figura 3.20, 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 *timeout* para não ficar esperando indefinidamente o início da tarefa 2. O *timeout* 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 mecanismo de coordenação de *tarefa 1 overlapsA tarefa 2*, é necessário acrescentar um *AND-join* (constituído por *P6*, *P7* e *t4* – Figura 3.21) para garantir que a tarefa 2 não termine antes da tarefa 1.

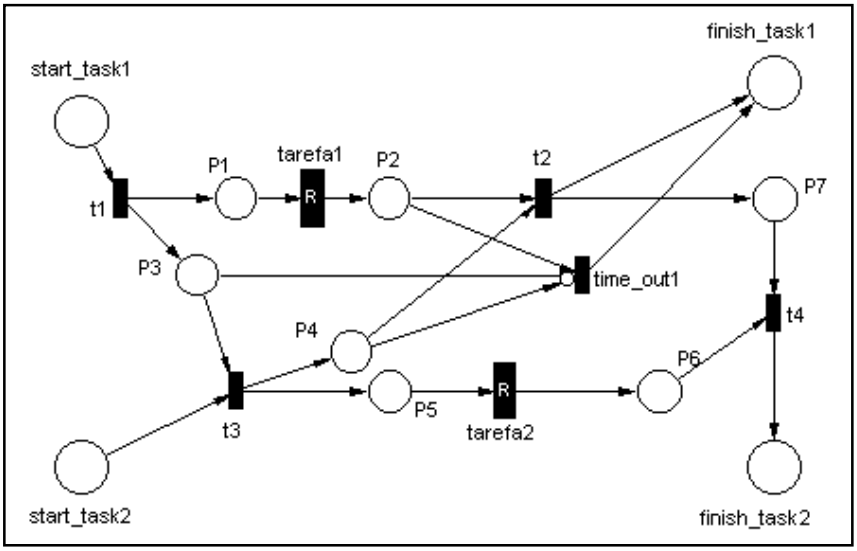

Figura 3.21: Mecanismo de coordenação para a dependência *tarefa 1 overlapsA tarefa 2*.

Em ambos os casos (*overlapsA* e *overlapsB*), a tarefa 2 pode possuir um *timeoutA* ou um *timeoutB*.

3.3.1.8. Tarefa 2 during 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 (*duringA*), a tarefa 2 pode ser executada apenas uma vez a cada execução da tarefa 1. No segundo caso (*duringB*), a tarefa 2 pode ser executada quantas vezes forem necessárias durante a execução da tarefa 1.

No modelo da Figura 3.22 (*duringA*), 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 término da *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 *timeout*, inibido pela presença de um *token* em *start_task2* (i.e., a tarefa 1 não encerra se a tarefa 2 estiver pronta para começar).

Para permitir que a tarefa 2 seja executada mais de uma vez durante a execução da tarefa 1 (*duringB*) são necessárias algumas modificações no modelo (Figura 3.23). 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 *start_task2*, impedindo que a tarefa 1 se encerre enquanto a tarefa 2 estiver pronta para ser executada.

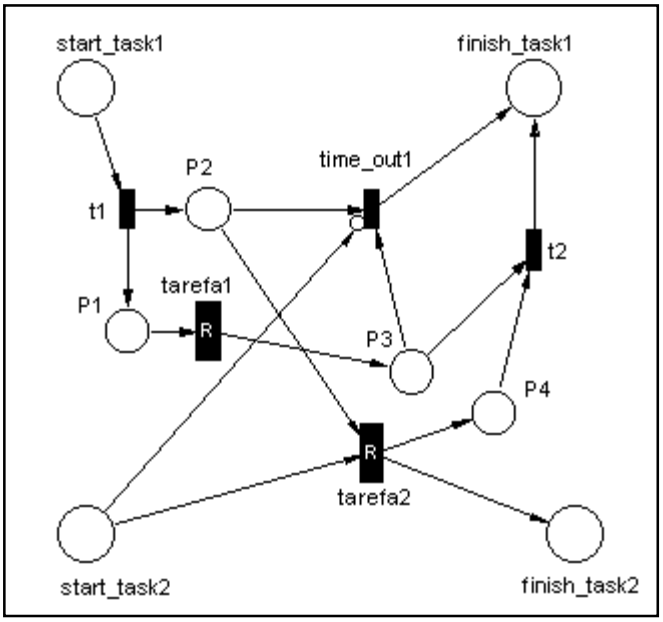

Figura 3.22: Mecanismo de coordenação para a dependência *tarefa 2 duringA tarefa 1*.

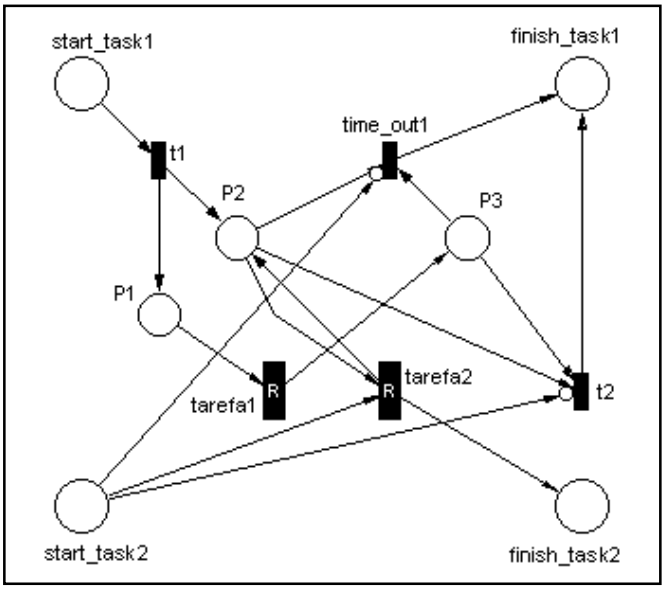

Figura 3.23: Mecanismo de coordenação para a dependência *tarefa 2 duringB tarefa 1*.

Em ambos os casos, a tarefa 2 pode possuir um *timeout* que a retorna ao seu estado inicial após certo tempo de espera (*timeoutB*) ou um *timeout* que permite a execução de uma tarefa alternativa (*timeoutA*).

3.3.2. Dependências de Gerenciamento de Recursos

Esta seção apresenta os mecanismos de coordenação para as sete dependências de gerenciamento de recursos mostradas nas Figuras 3.3 e 3.4.

3.3.2.1. Divisão (Compartilhamento) 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. 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 3.24 apresenta o modelo para o caso de duas tarefas compartilhando três instâncias do recurso (N = 3).

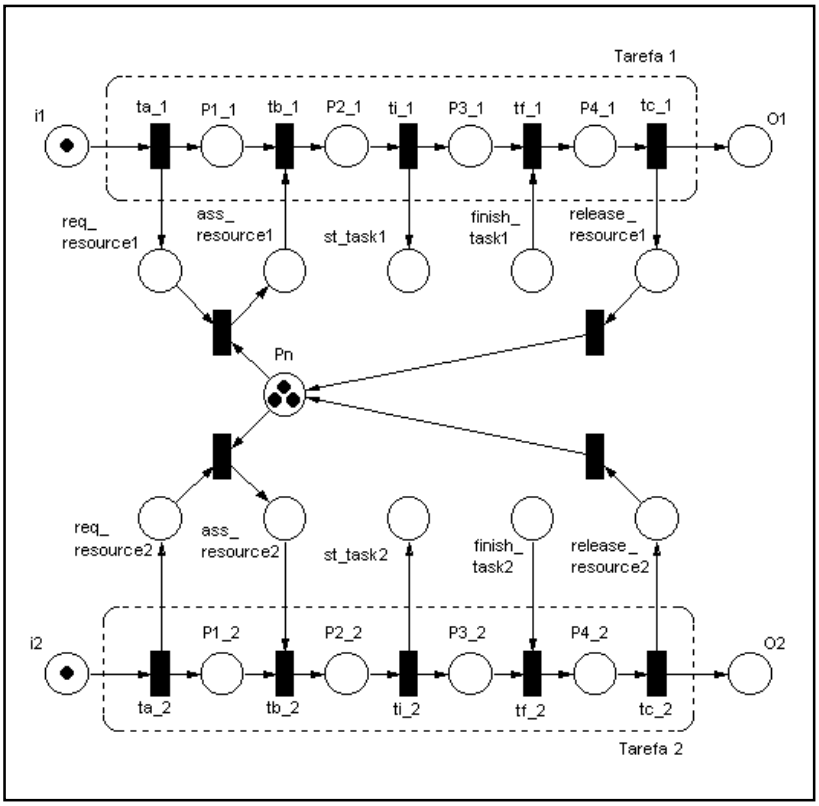

Figura 3.24: Gerenciador de recursos – *divisão por 3*.

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 3.25.

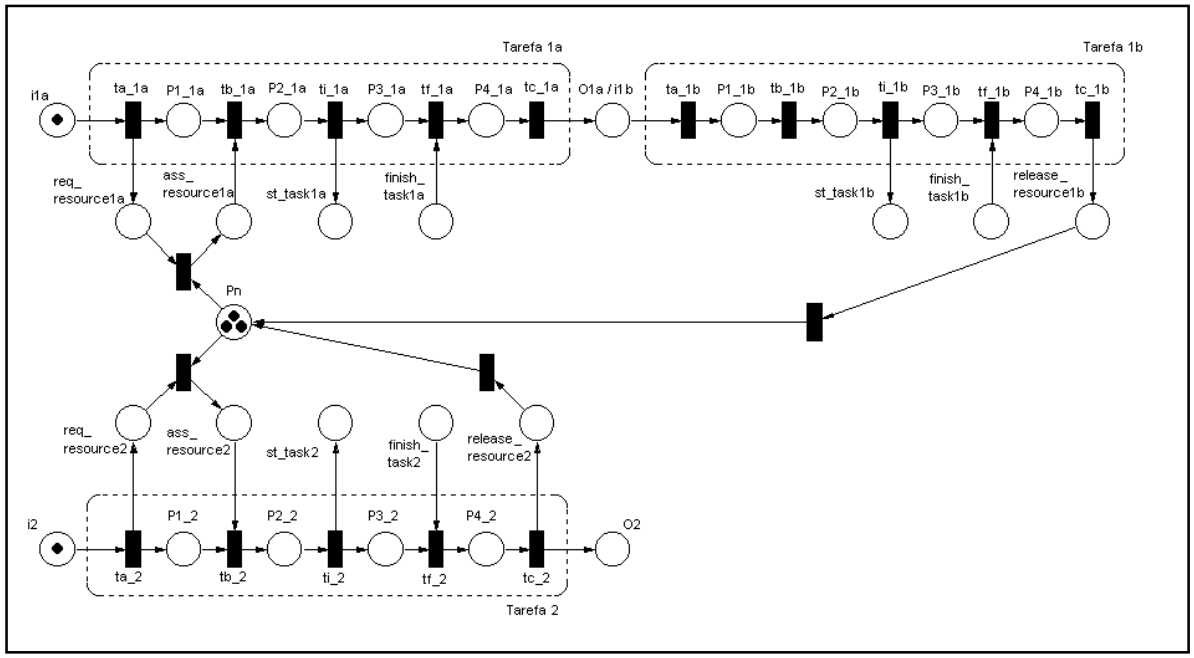

Figura 3.25: Gerenciador de recursos – *divisão por 3*, com o recurso sendo utilizado por duas tarefas consecutivas.

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

3.3.2.2. Simultaneidade N

Este tipo de gerenciador estabelece que um recurso só é alocado para N tarefas simultaneamente. O modelo (Figura 3.26) 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 3.26, 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). Na figura também é definido um *timeout* para a tarefa 1, devolvendo o *token* de *request_resource* para o lugar de entrada *i1*, evitando que ela espere indefinidamente até que as outras N-1 tarefas requisitem o recurso.

No modelo da Figura 3.26, há apenas duas tarefas exigindo simultaneidade 2, de modo que só há como utilizar o recurso se ambas o estiverem requisitando. Na Figura 3.27 é mostrado um exemplo onde três tarefas exigem simultaneidade 2. Nesse caso, quando duas das três tarefas requisitarem o recurso, ele será alocado a elas. A montagem deste modelo segue o da Figura 3.26, no qual a tarefa deve possuir um caminho entre *request_resource* e *assigned_resource* (há uma 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*.

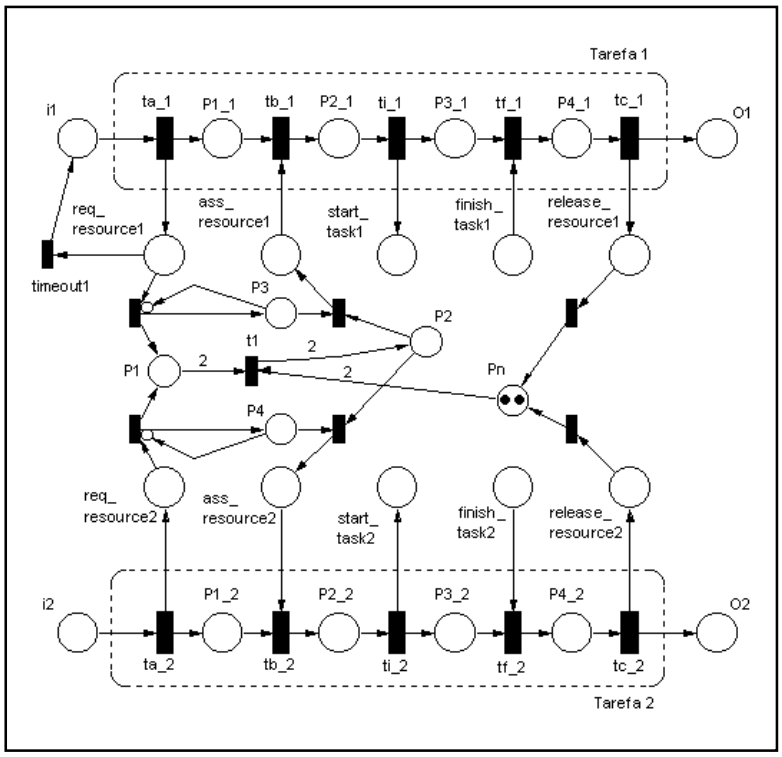

Figura 3.26: Gerenciador de recursos – *simultaneidade 2*, com duas tarefas podendo requisitar o recurso.

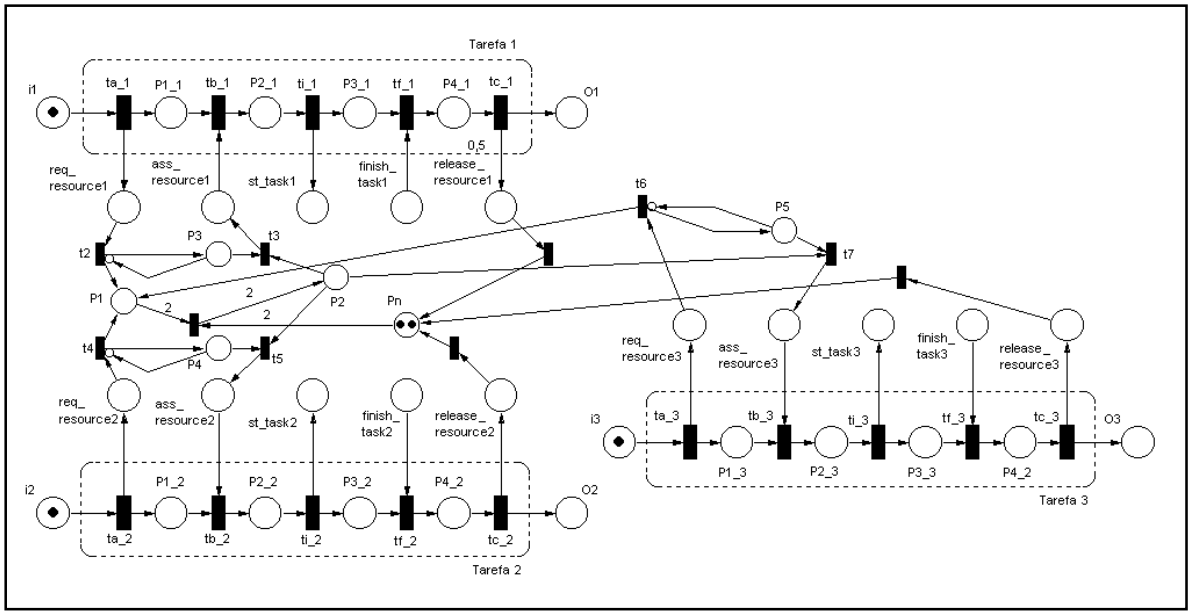

Figura 3.27: Gerenciador de recursos – *simultaneidade 2*, com três tarefas podendo requisitar o recurso.

3.3.2.3. Volatilidade N

A volatilidade N de um recurso indica que ele não pode ser reutilizado mais de N vezes por uma tarefa (*volatilidadeA*). O modelo é bastante simples, diferenciando-se do modelo da *divisão por N* apenas pelo fato do *token* não retornar ao lugar *Pn*. Na Figura 3.28 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 *timeout* (opcional) só estará habilitado após o término dos recursos (arco inibidor).

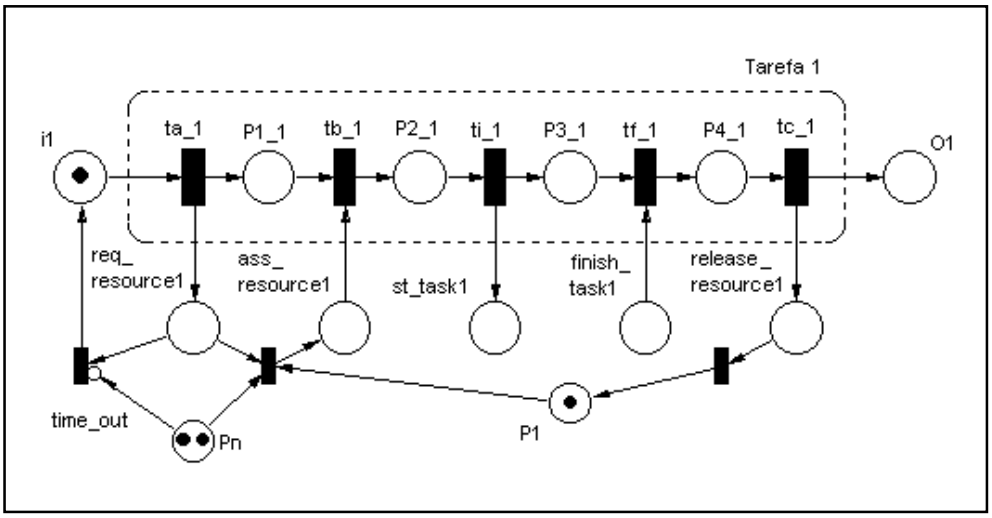

Figura 3.28: Gerenciador de recursos – *volatilidadeA 2* (em cada tarefa).

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 3.29 (*volatilidade B*).

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.

3.3.2.4. Divisão por M com Simultaneidade N

Esta situação permite que até M grupos de N tarefas compartilhem um determinado recurso. O modelo é praticamente idêntico ao da *simultaneidade N* (Figuras 3.26 e 3.27), apenas colocando $N \times M$ *tokens* no lugar *Pn*. A Figura 3.30 mostra o modelo para M = 3 e N = 2 ($2 \times 3 = 6$ *tokens* em *Pn*).

A análise deste modelo através de simulações detectou uma possível situação incorreta. 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 com a redefinição dos modelos ou por meio de PNs de alto nível (ver Seção 3.4).

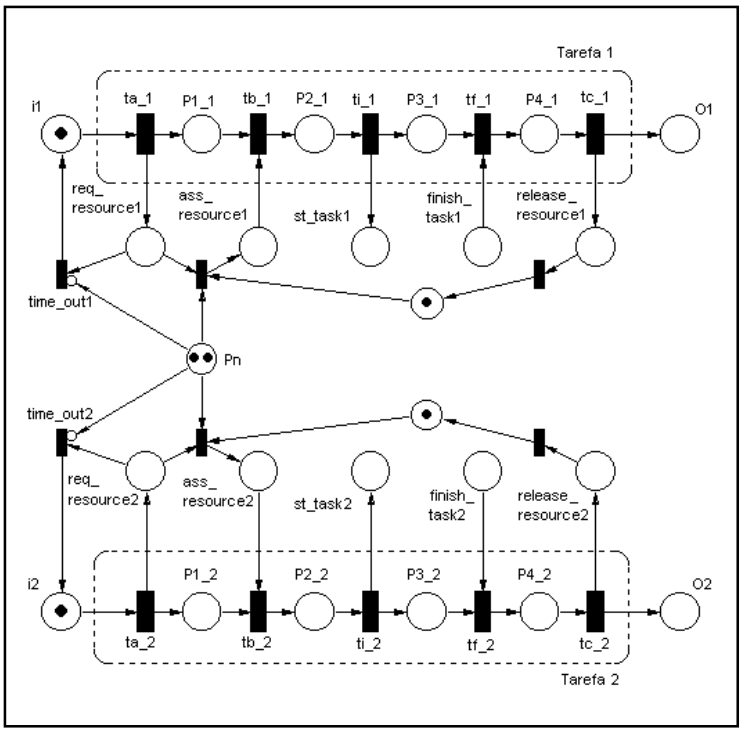

Figura 3.29: Gerenciador de recursos – *volatilidadeB 2* (para todas as tarefas).

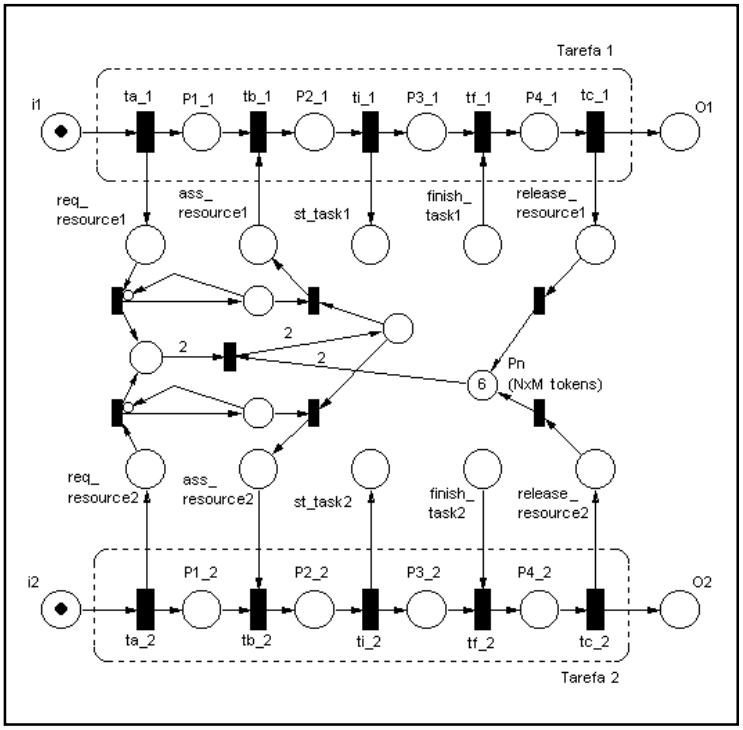

Figura 3.30: *Divisão por 3 com simultaneidade 2.*

3.3.2.5. Divisão por M com Volatilidade N

Nessa 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 3.24), apenas acrescentando o lugar *Pn*, representando a volatilidade do recurso, e os respectivos *timeouts* (opcionais). Na Figura 3.31, o modelo está representado para o caso de M = 2 e N = 4.

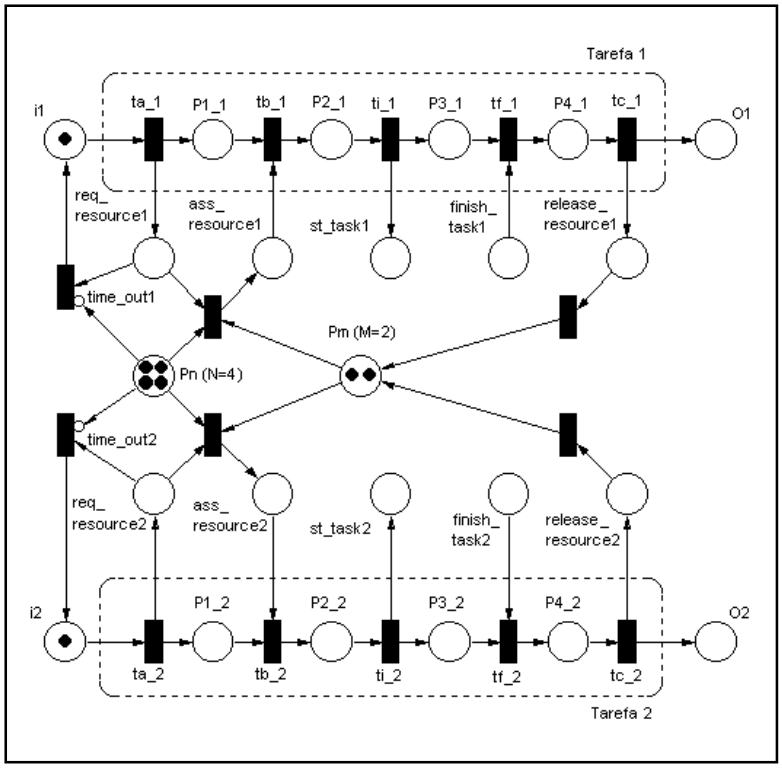

Figura 3.31: *Divisão por 2 com volatilidade 4*.

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 3.28).

3.3.2.6. Simultaneidade N com 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 3.26), apenas acrescentando o lugar *Pm*, relativo à volatilidade do recurso, e os respectivos *timeouts*. Na Figura 3.32, o modelo é apresentado para o caso de N = 2 e M = 4.

É importante ressaltar que 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 $M \times N$ *tokens* em *Pm*.

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

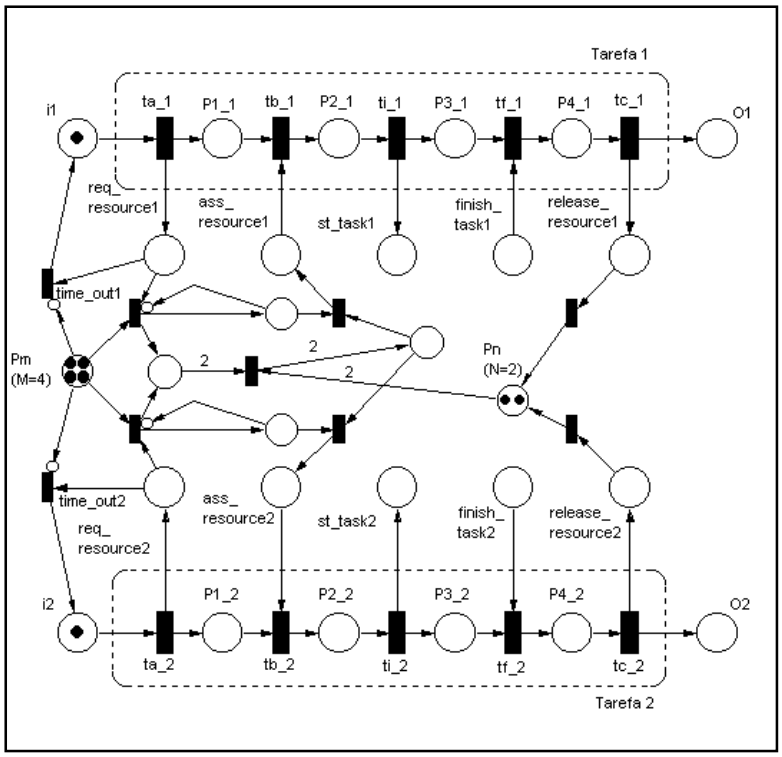

Figura 3.32: *Simultaneidade 2 com volatilidade 4.*

3.3.2.7. Divisão por Q com Simultaneidade N com Volatilidade M

Esta situação 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 com simultaneidade N* (Figura 3.30), apenas acrescentando o lugar *Pm* e (opcionalmente) os *timeouts*. O modelo para o caso de Q = 2, N = 2 e M = 6 é mostrado na Figura 3.33.

3.4. Mecanismos de Coordenação: Modelos Usando PNs de Alto Nível

Redes de Petri de alto nível (*high-level PNs*), de acordo com Murata [Murata 89], englobam, dentre outros tipos, as chamadas redes de predicado/transição [Genrich 86] e as redes coloridas [Jensen 86]. No texto, portanto, o termo "PN de alto nível" se refere à definição de Murata (ver Apêndice A). A principal característica deste tipo de rede é a possibilidade de diferenciação entre os *tokens* (chamados *tokens* coloridos). Os arcos possuem expressões com variáveis e constantes que definem como será feita a transmissão dos *tokens*.

PNs de alto nível, em geral, produzem modelos mais compactos (i.e., menos lugares e transições) que os equivalentes em PNs convencionais, pois nas PNs de alto nível os *tokens* também carregam informação. Essa característica das PNs de alto nível é importante para a implementação dos mecanismos de coordenação porque reduz a possibilidade da explosão

de estados[4] em sistemas onde o número de tarefas e interdependências for elevado [Raposo 00c] (ver artigo no Apêndice D).

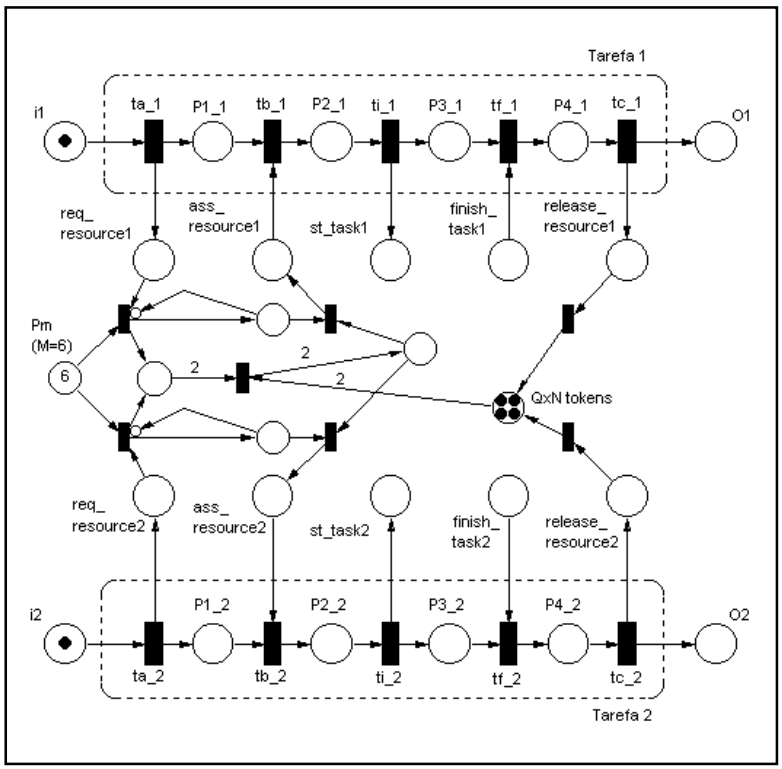

Figura 3.33: *Divisão por 2 com simultaneidade 2 com volatilidade 6*.

O que há de mais significativo na implementação utilizando PNs de alto nível é que não há a necessidade de cada tarefa possuir os lugares *request_resource*, *assigned_resource*, *start_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.

Assim, as duas seções que se seguem remodelam todos os mecanismos de coordenação utilizando PNs de alto nível.

3.4.1. Dependências Temporais

Os mecanismos de coordenação ilustrados nas figuras desta seção utilizam o modelo simplificado de expansão de tarefas da Figura 3.9.

Tarefa A equals Tarefa B: este mecanismo de coordenação é mostrado na Figura 3.34. As duas tarefas agora compartilham lugares *start_tasks* e *finish_tasks* comuns. Os arcos definem constantes (cores *a* e *b*), cada uma representando o *job token* de uma tarefa. A transição *t1* só é disparada, dando início à execução das tarefas, quando houver um *token* de cada uma dessas duas cores em *start_tasks*. O mesmo vale para a transição *t2*, que determina o final das tarefas quando houver um *token* de cada cor

[4] Modelos em PNs simples tendem a se tornar muito grandes para análise, mesmo para sistemas de tamanhos modestos [Murata 89].

em *P2*. Comparando este modelo com o da Figura 3.10 (mesmo mecanismo modelado com PNs simples), nota-se que houve uma diminuição no número de lugares.

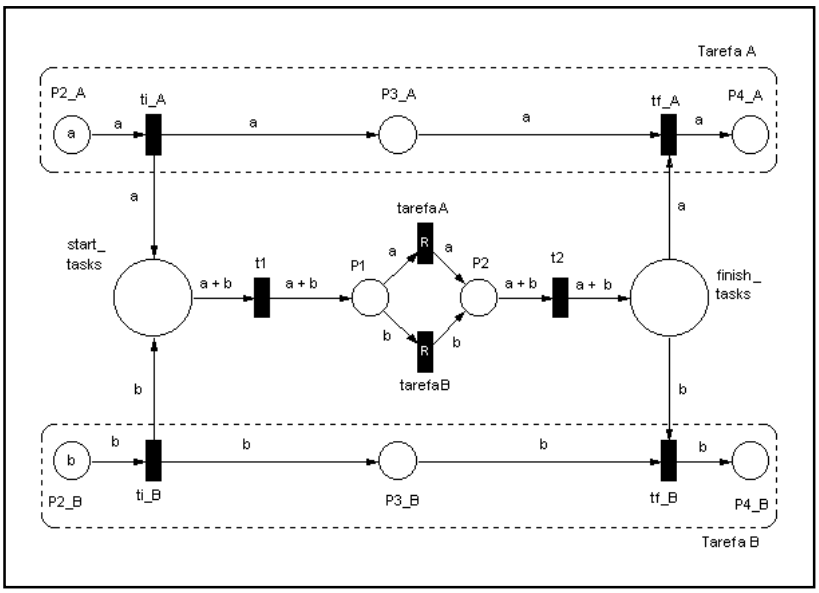

Figura 3.34: Mecanismo de coordenação para *tarefa A equals tarefa B* usando PN de alto nível.

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. As Figuras 3.35 e 3.36 mostram as duas variações desta dependência (são equivalentes às Figura 3.13 e 3.14, respectivamente).

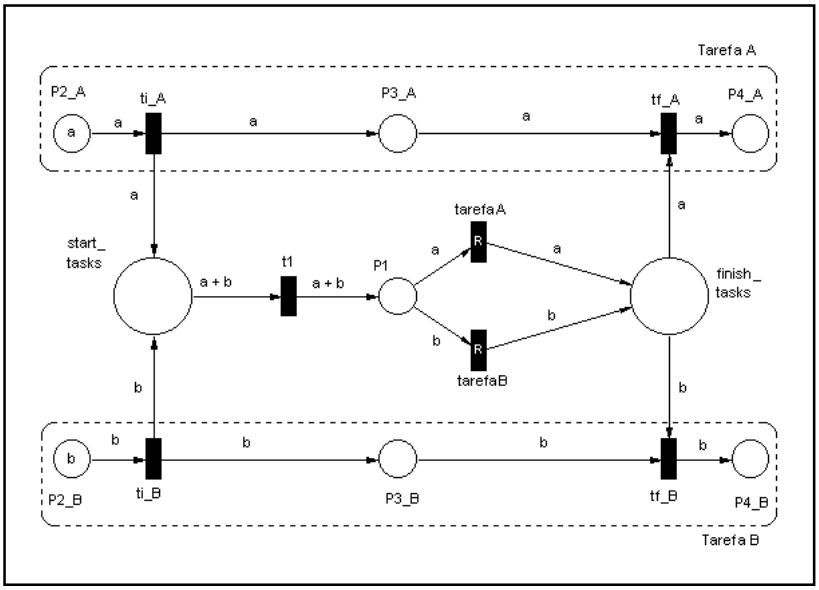

Figura 3.35: Mecanismo de coordenação para *tarefa A startsB tarefa B* usando PN de alto nível.

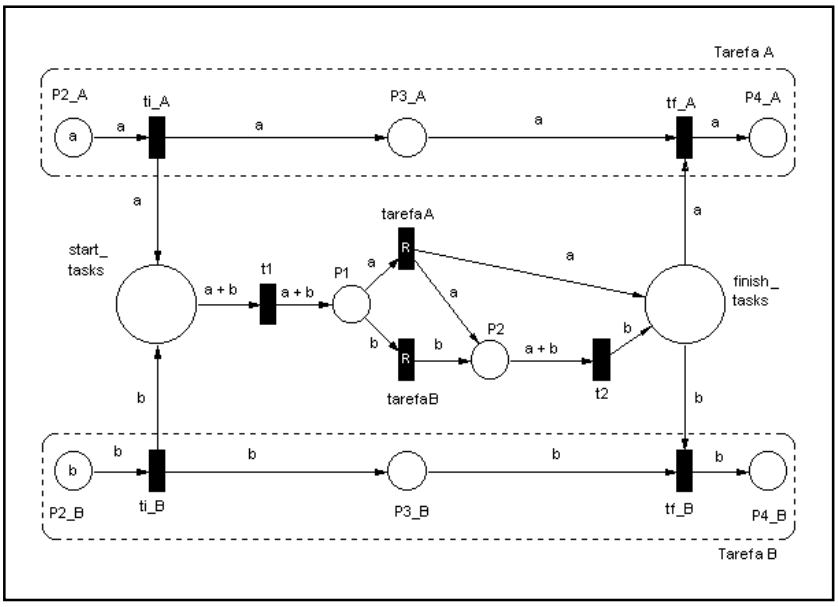

Figura 3.36: Mecanismo de coordenação para *tarefa A startsA tarefa B* usando PN de alto nível.

Tarefa A finishes Tarefa B: os modelos para as duas variações desta dependência são mostrados nas Figuras 3.37 e 3.38. No caso da Figura 3.38 (*finishesA*), a transição *t1* coloca dois *tokens* de cor *b* no lugar P1, indicando à tarefa A que a tarefa B já começou. A transição *t2* garante que ambas as tarefas terminam juntas. Estas figuras são equivalentes às Figuras 3.15 e 3.16.

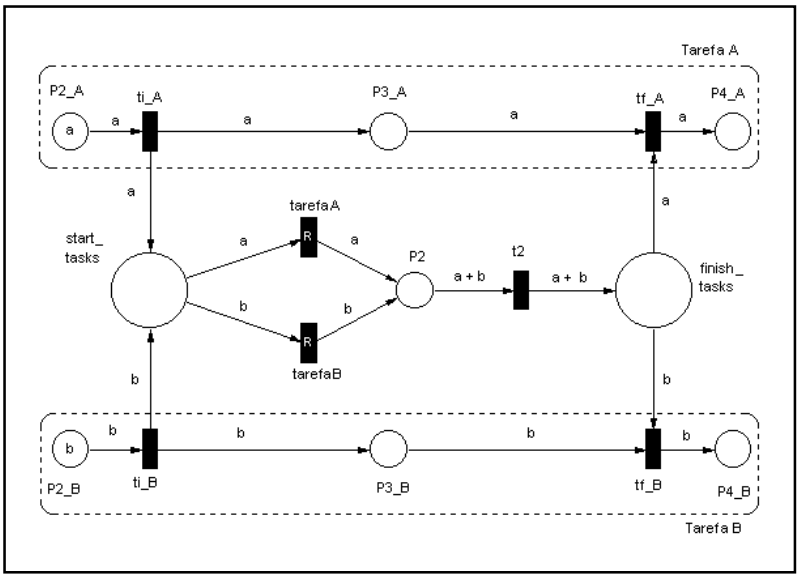

Figura 3.37: Mecanismo de coordenação para *tarefa A finishesB tarefa B* usando PN de alto nível.

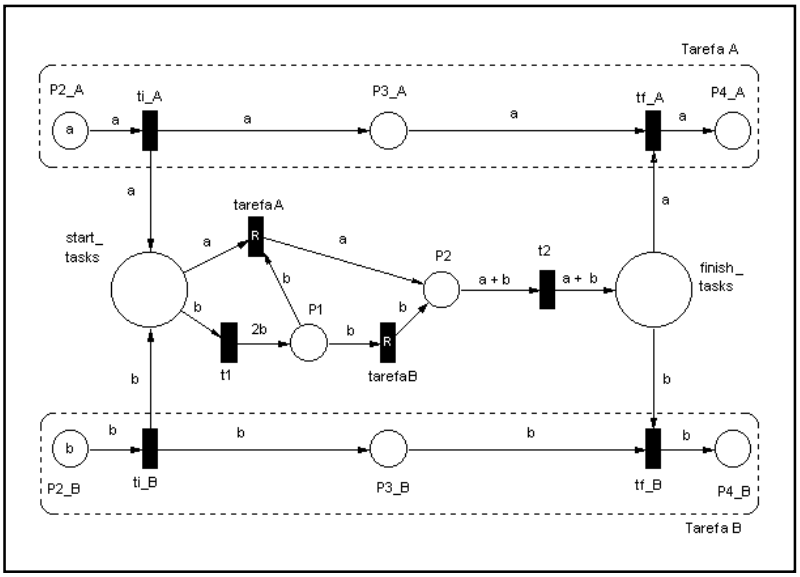

Figura 3.38: Mecanismo de coordenação para *tarefa A finishesA tarefa B* usando PN de alto nível.

Tarefa B after Tarefa A: neste caso (Figura 3.39 - *afterA*) o mecanismo de coordenação não se diferencia muito do que usa PNs convencionais (Figura 3.17), exceto pelo fato dos lugares *start_tasks* e *finish_tasks* serem comuns a ambas as tarefas. A situação em que a tarefa A pode ser executada mais de uma vez após a execução da tarefa B (*afterB*) é modelada de maneira similar às PNs convencionais, colocando um arco de retorno da transição *tarefaB* para o lugar *P1*.

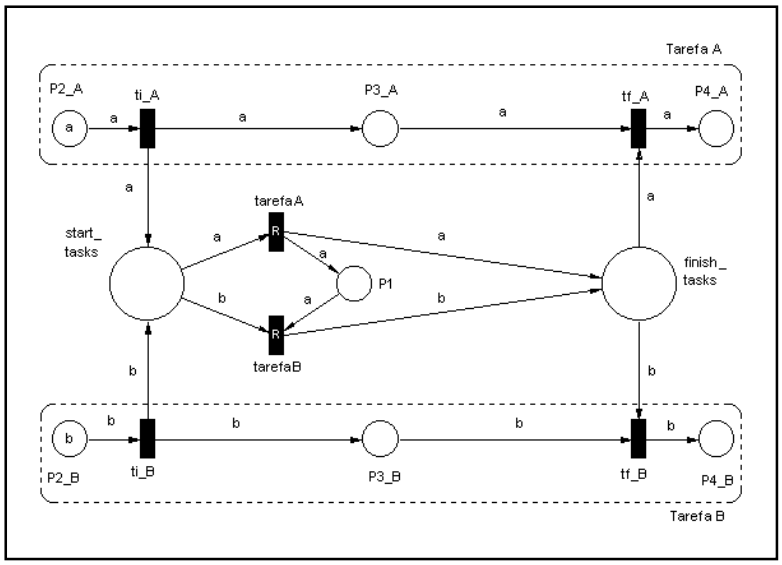

Figura 3.39: Mecanismo de coordenação para *tarefa B afterA tarefa A* usando PN de alto nível.

Tarefa A before Tarefa B: no modelo da Figura 3.40, além dos arcos inibidores, é usado um arco com peso *b + not(a)*, habilitando a transição *t3* apenas se houver um *token* da cor *b* e nenhum da cor *a* em *start_tasks*. Isso implica que a tarefa B só pode

ser executada se a tarefa A não estiver pronta. Existem também *tokens* de uma terceira cor *c* que aparecerão em *P1* e *P3* para indicar, respectivamente, que a tarefa A está sendo executada (inibindo temporariamente a tarefa B) e que a tarefa B já foi executada (inibindo "para sempre" a tarefa A).

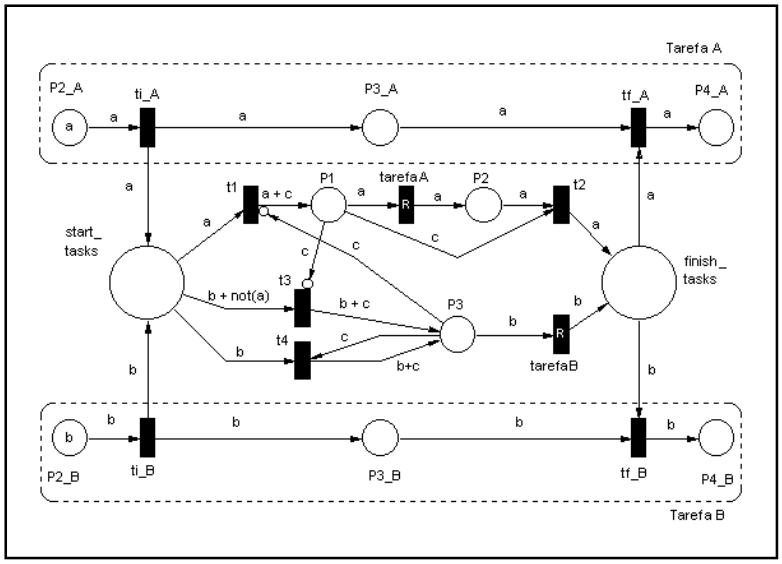

Figura 3.40: Mecanismo de coordenação para *tarefa A before tarefa B* usando PN de alto nível.

Tarefa A meets Tarefa B: a Figura 3.41 ilustra o mecanismo de coordenação para esta dependência. A tarefa A só pode ser executada quando as duas estiverem prontas (*tokens a* e *b* em *start_tasks*) e, ao seu final, ela libera o *token b* para *P1*, que permitirá a execução da tarefa B.

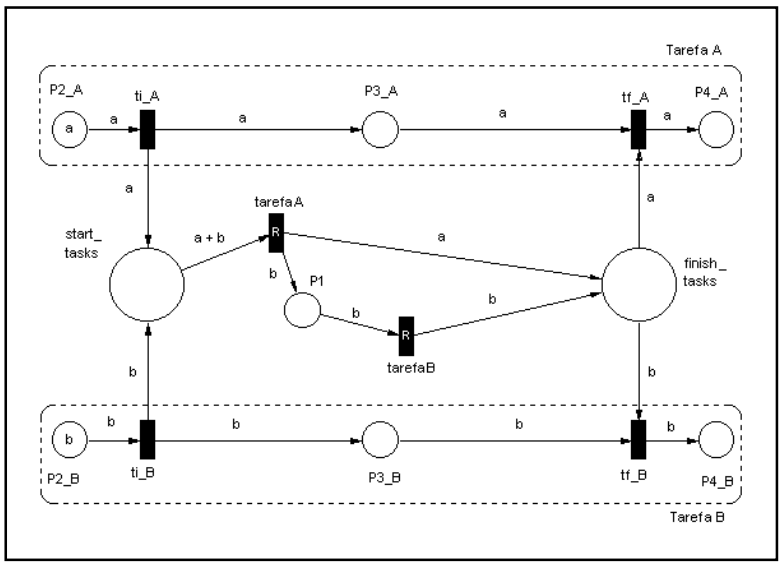

Figura 3.41: Mecanismo de coordenação para *tarefa A meets tarefa B* usando PN de alto nível.

Tarefa A overlaps Tarefa B: as Figuras 3.42 e 3.43 ilustram os mecanismos de coordenação para as duas variações desta dependência. Em ambos os casos existe uma cor auxiliar *c* que indica à tarefa B que a tarefa A já se iniciou (*token c* em *P1* habilita *t2*). No caso da Figura 3.43 (*overlapsA*), onde a tarefa A deve terminar antes, um *token c* em *P3* (habilitando *t4*) serve para indicar à tarefa B que a tarefa A já terminou.

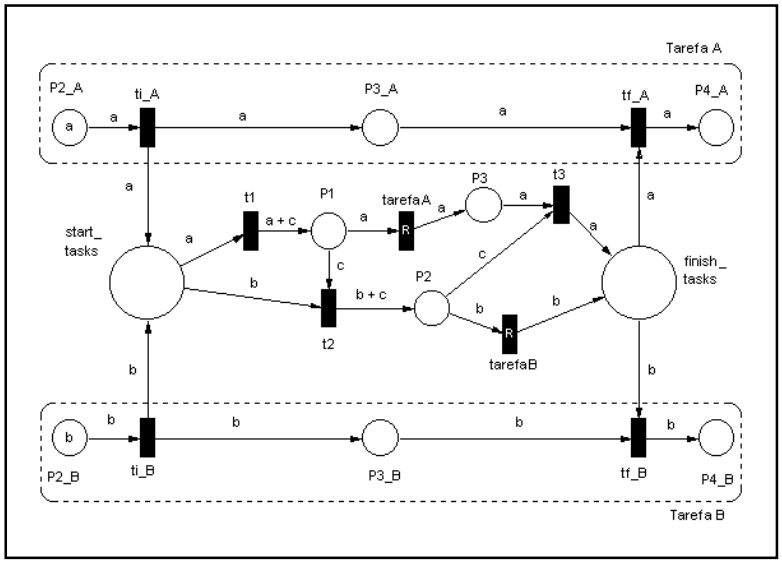

Figura 3.42: Mecanismo de coordenação para *tarefa A overlapsB tarefa B* usando PN de alto nível.

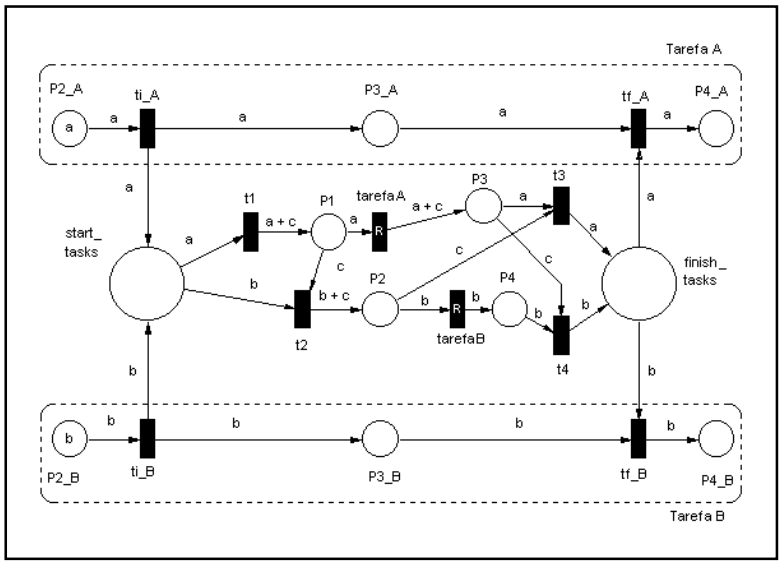

Figura 3.43: Mecanismo de coordenação para *tarefa A overlapsA tarefa B* usando PN de alto nível.

Tarefa B during Tarefa A: nas Figuras 3.44 e 3.45, um *token c* em *P1* indica que a tarefa A está em execução, portanto a tarefa B está habilitada. Esse mesmo *token c* é colocado em *P2* (Figura 3.44) ao término da tarefa B, permitindo o final da tarefa A

(habilita *t3*). Na Figura 3.45 (*duringB*) o *token c* retorna a *P1* após a execução da tarefa B, permitindo novas execuções da tarefa B enquanto a tarefa A não termina (o disparo de *t3* recolhe o *token c* de *P1*).

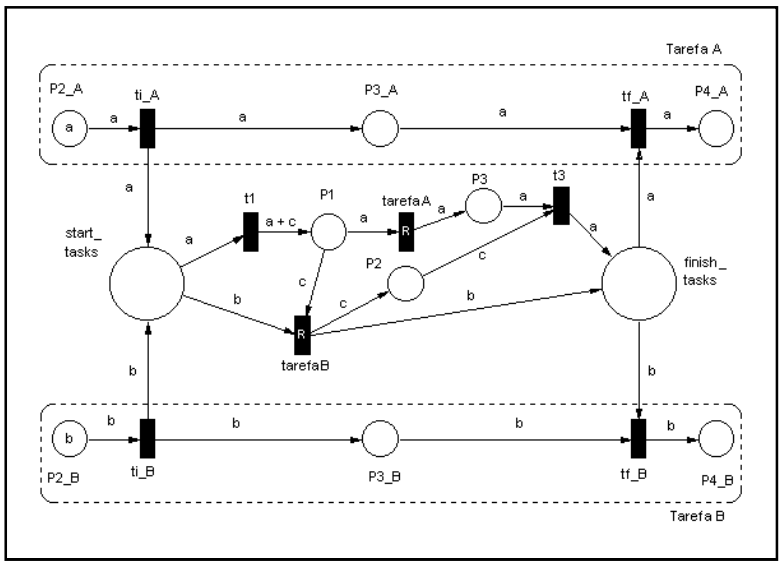

Figura 3.44: Mecanismo de coordenação para *tarefa B duringA tarefa A* usando PN de alto nível.

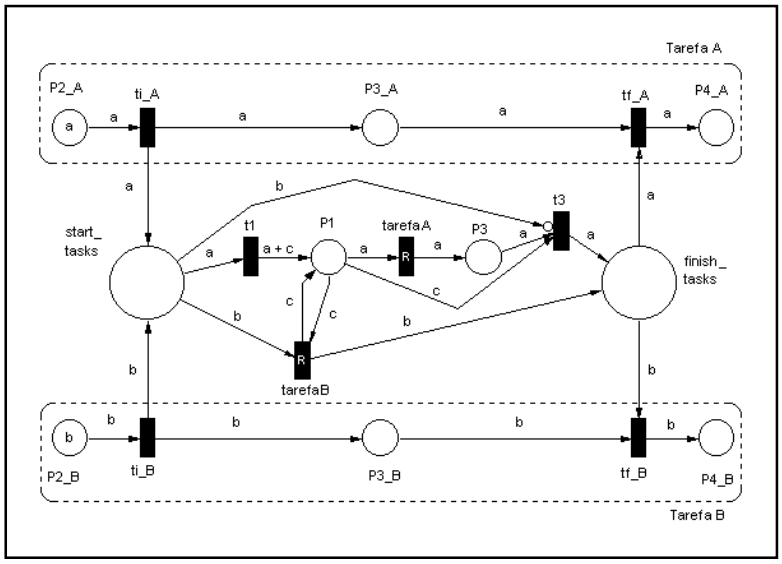

Figura 3.45: Mecanismo de coordenação para *tarefa B duringB tarefa A* usando PN de alto nível.

3.4.2. Dependências de Gerenciamento de Recursos

Os mecanismos de coordenação para este tipo de dependência mostram de maneira ainda mais contundente como as PNs de alto nível podem simplificar os modelos. Isso porque nesse tipo de dependência os arcos podem usar variáveis que são substituídas por *tokens* de qualquer tarefa. Os resultados são modelos bem mais simples que os de PNs

convencionais, com a vantagem adicional de não aumentarem em complexidade quando o número de tarefas que compartilham um recurso aumenta.

A seguir são mostrados os mecanismos de coordenação para as dependências de gerenciamento de recursos.

Divisão por N: a Figura 3.46 mostra o gerenciador da divisão por N (equivalente ao da Figura 3.24). Os arcos com variável <*x*> garantem que a tarefa que requisitou o recurso o receberá. Os N *tokens* de cor *r* em *Pn* indicam as instâncias do recurso. A adição de uma terceira 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.

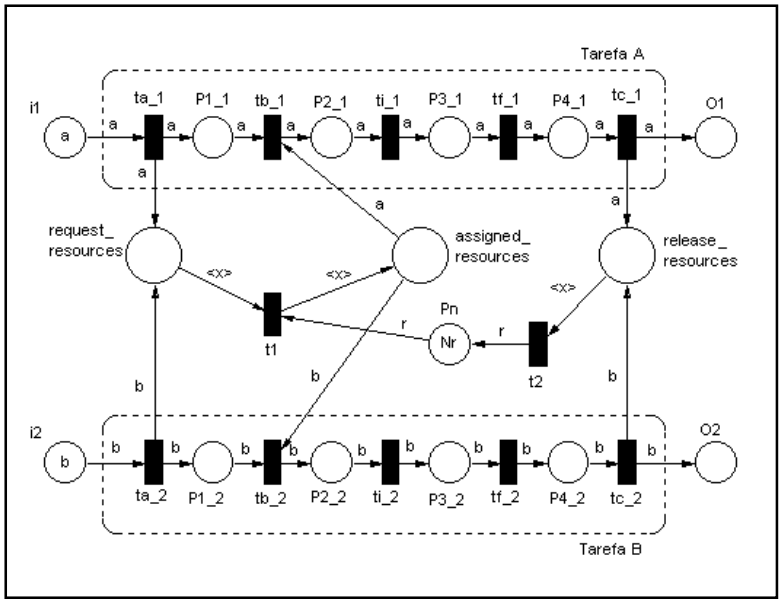

Figura 3.46: Gerenciador de recursos – *divisão por N* usando PN de alto nível.

Simultaneidade N: a Figura 3.47 mostra o gerenciador de *simultaneidade 2* com *tokens* coloridos (equivalente ao da Figura 3.26). 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. A Figura 3.48 mostra o caso em que três tarefas podem requisitar o recurso (equivalente ao da Figura 3.27).

Volatilidade N: o caso em que a volatilidade se refere a todas as tarefas (*volatilidadeB*) é mostrado na Figura 3.49 (equivalente ao da Figura 3.29). 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 mesma 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 (*volatilidadeA*) 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, os arcos juntos a *Pn* devem ter a expressão <*x*> para garantir que sairá um *token* da cor da tarefa que recebeu o recurso e entrará um *token* da cor da tarefa que liberou o recurso.

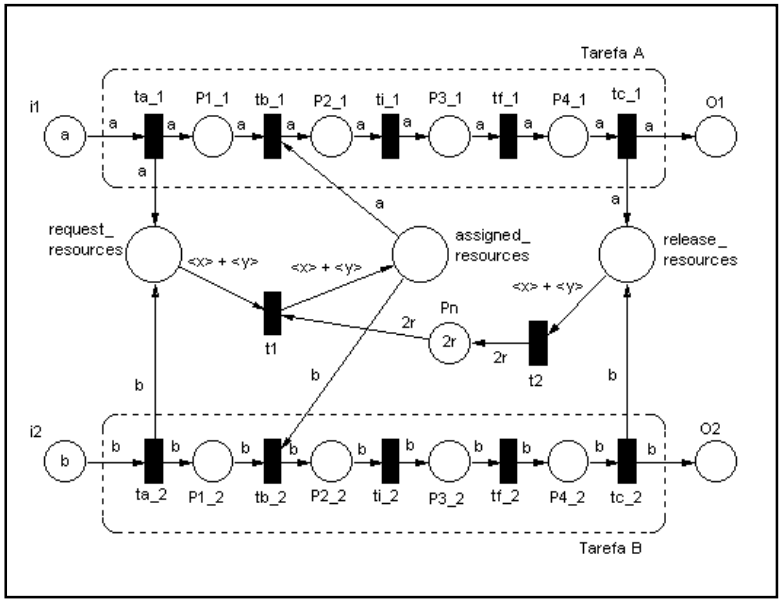

Figura 3.47: Gerenciador de recursos – *simultaneidade 2*, com 2 tarefas podendo requisitar o recurso.

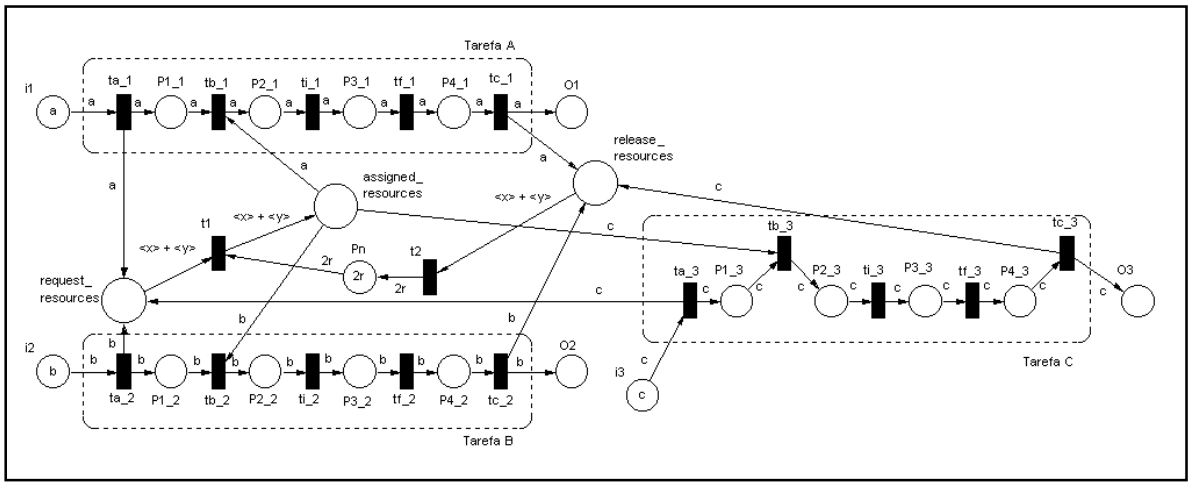

Figura 3.48: Gerenciador de recursos – *simultaneidade 2*, com 3 tarefas podendo requisitar o recurso.

Divisão por M com Simultaneidade N: a combinação da *simultaneidade N* com a *divisão por M* é feita simplesmente colocando N × M *tokens* de cor *r* no lugar *Pn* do modelo da *simultaneidade M* (Figura 3.47). Este modelo também resolve o problema dessa configuração, descrito na Seção 3.3.2.4 (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.

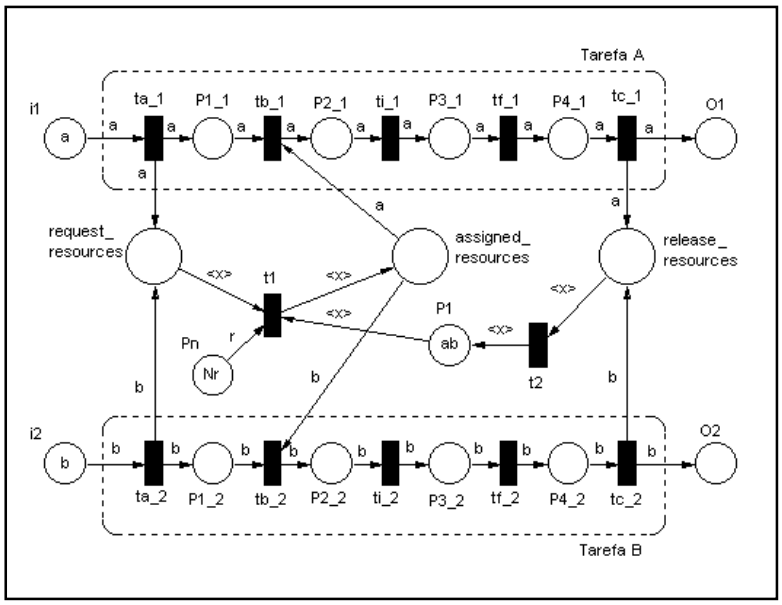

Figura 3.49: Gerenciador de recursos – *volatilidade N* usando PN de alto nível.

Divisão por N com 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 3.50 (equivalente ao da Figura 3.31).

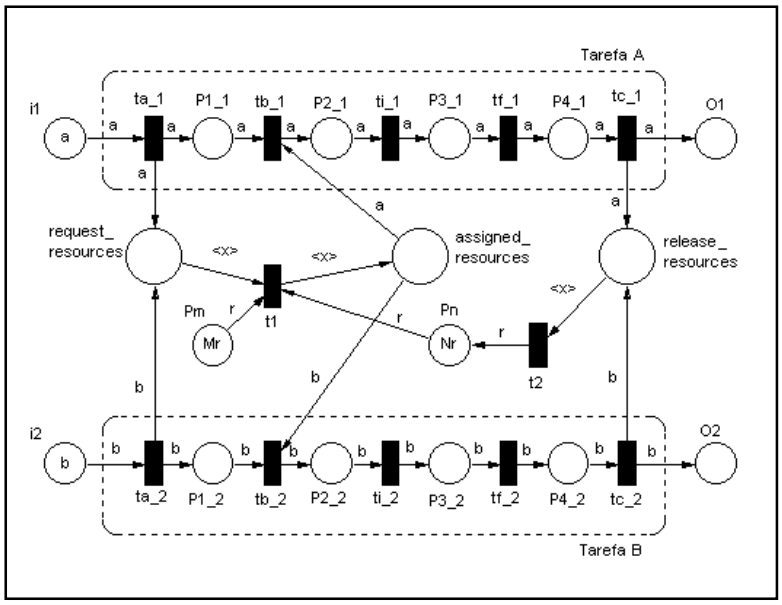

Figura 3.50: Gerenciador de recursos – *divisão por N com volatilidade M* usando PN de alto nível.

Simultaneidade N com Volatilidade M: assim como no caso anterior, este mecanismo de coordenação é construído adicionando-se o lugar *Pm* ao modelo da *simultaneidade N* (Figura 3.47).

Divisão por N com Simultaneidade M com Volatilidade Q: este gerenciador é obtido adicionando o lugar *Pq* (volatilidade) ao gerenciador da combinação *divisão por N com simultaneidade M*. O modelo para N = 3, M = 2 e Q = 4 é mostrado na Figura 3.51 (equivalente ao da Figura 3.33).

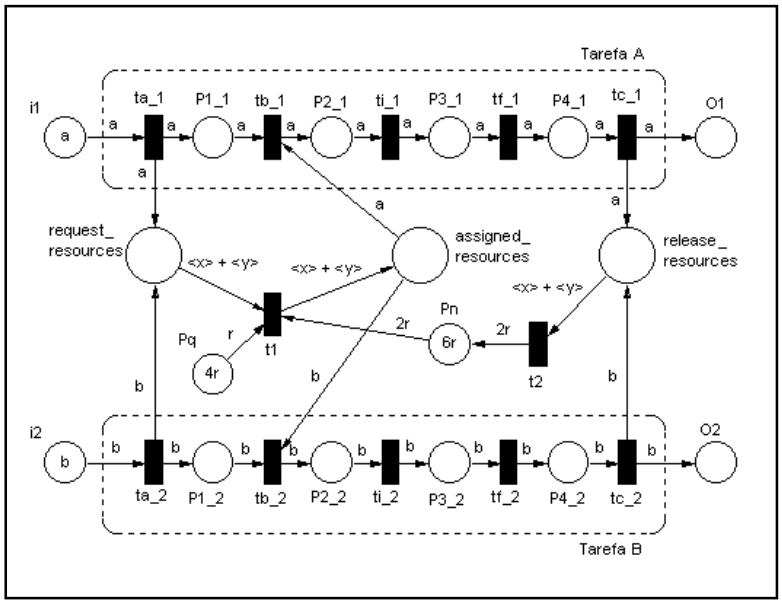

Figura 3.51: Gerenciador de recursos – *divisão por 3 com simultaneidade 2 com volatilidade 4*.

3.5. Protótipo para Simulação e Análise dos Mecanismos de Coordenação

A análise do comportamento de um ambiente colaborativo modelado por meio dos mecanismos de coordenação aqui apresentados 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 de coordenação é proposto um esquema com três componentes: uma 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. Assim, foi criado um protótipo conforme ilustrado na Figura 3.52, que mostra como os componentes mencionados estão relacionados.

A seguir será apresentada a linguagem que define as dependências entre as tarefas. Esta linguagem independe do simulador de PN usado. Na Seção 3.5.2 será mostrada a implementação para uma ferramenta de simulação específica pois, como os arquivos de entrada variam de acordo com a ferramenta utilizada, o programa deve ser adaptado de acordo com a ferramenta utilizada.

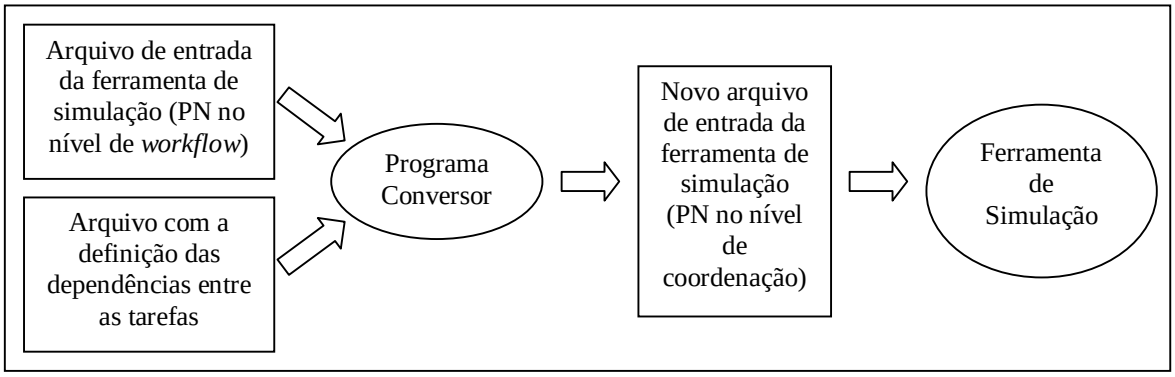

Figura 3.52: Esquema para a utilização dos mecanismos de coordenação com um simulador de PNs.

3.5.1. Linguagem para a Definição das Interdependências

Seguindo o esquema proposto na Figura 3.52, o usuário só precisa saber utilizar o simulador de PNs (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. Esse arquivo é escrito em uma linguagem bastante simples que será apresentada a seguir.

A linguagem criada define todas as interdependências 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> equals <tarefa 2>*, onde <tarefa1> e <tarefa 2>, como já comentado, devem ser os nomes das tarefas (transições) na PN criada no simulador. Os *timeouts* são opcionais e representam, respectivamente, os *timeouts* aplicados sobre a <tarefa1> e a <tarefa2>. O <time_out> pode assumir, nesse caso, três valores: time_outA, time_outB ou no_time_out. Assim, a dependência mostrada na Figura 3.10, onde nenhum *timeout* é aplicado, é definida como

```
equals “<tarefa1>” “<tarefa2>”
ou
equals “<tarefa1>” “<tarefa2>” no_time_out no_time_out
```

A dependência da Figura 3.11 é definida como

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

A da Figura 3.12, como

```
equals “<tarefa1>” “<tarefa2>” time_outB time_outB
```

Se fosse desejado colocar um *timeout* 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 *timeout* apenas na segunda tarefa, a definição seria

```
equals “<tarefa1>” “<tarefa2>” no_time_out time_outA
```

A relação *<tarefa1> starts <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>]
```

Assim como na relação anterior, os *timeouts* são opcionais e podem assumir os três valores anteriormente definidos. A dependência da Figura 3.13, portanto, é definida como

```
startsB “<tarefa1>” “<tarefa2>” time_outB
ou
startsB “<tarefa1>” “<tarefa2>” time_outB no_time_out
```

A da Figura 3.14, por sua vez, é definida como

```
startsA “<tarefa1>” “<tarefa2>”
```

A relação *<tarefa1> finishes <tarefa2>* também é definida da mesma maneira.

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

O *timeout* continua sendo opcional para o primeiro caso (*finishesA*), mas só a tarefa 1 pode tê-lo. A segunda variação da relação (*finishesB*) não aceita nenhum tipo de *timeout* (qualquer *timeout* definido após esta relação deve ser ignorado). Isso porque já existem *timeouts* embutidos no mecanismo de coordenação (Figura 3.15).

Para a relação *<tarefa2> after <tarefa1>*, o único *timeout* que pode existir se refere à <tarefa2>. Isso porque a <tarefa1> (que ocorre antes) nunca será bloqueada e, portanto, ela não tem necessidade de *timeout*. Este *timeout* pode assumir os valores time_outA, time_outB ou no_time_out (se houver um segundo *timeout* na definição da relação, ele é ignorado).

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

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

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

A relação *<tarefa1> meets <tarefa2>* pode ter apenas o *timeout* referente à <tarefa2>, pois o da <tarefa1> já está pré-definido no modelo da Figura 3.19. Este *timeout* pode assumir os mesmos três valores dos casos anteriores e, para diminuir a possibilidade de erro na definição da dependência, sempre que houver dois *timeouts* definidos, só o segundo será considerado (os casos anteriores só consideram o primeiro porque eles se referiam à primeira tarefa).

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

Assim como no caso anterior, para a relação *<tarefa1> overlaps <tarefa2>* apenas o *timeout* para a segunda tarefa deve ser considerado e pode assumir os mesmos três valores dos casos anteriores. Isso porque os modelos (Figuras 3.20 e 3.21) já definem um *timeout* embutido para a <tarefa1>.

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

Para relação *<tarefa2> during <tarefa1>*, o *timeout* para a <tarefa1> já está pré-definido no modelo (Figuras 3.22 e 3.23). Portanto, o único *timeout* que pode aparecer na definição desta relação se refere à <tarefa2>. Mais uma vez, o <time_out> pode assumir os valores time_outA, time_outB ou no_time_out.

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

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

A definição da *divisão por N* possui 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 *timeouts* associados a cada uma delas.

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

A Figura 3.24 (duas tarefas compartilhando três recursos, sem *timeouts*) é definida como

```
div 3 “<tarefa1>” “<tarefa2>”
```

Para definir, por exemplo, uma situação em que três tarefas compartilham dois recursos, onde apenas a última tarefa possui *timeout*, se escreveria

```
div 2 “<tarefa1>” “<tarefa2>” “<tarefa3>” no_time_out no_time_out time_out
```

Este gerenciador de recursos (divisão por N) pode apresentar uma variação (Figura 3.25), 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 3.25, 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 3.26, onde ocorre simultaneidade 2 entre duas tarefas é definida como

```
sim 2 “<tarefa1>” “<tarefa2>”
```

A Figura 3.27 (simultaneidade 2 entre três tarefas) é definida como

```
sim 2 “<tarefa1>” “<tarefa2>” “<tarefa3>”
```

A *volatilidade* de recursos apresenta duas variações. No primeiro caso (*volatilidadeA*), a volatilidade se refere a cada tarefa (Figura 3.28) e no segundo caso (*volatilidadeB*), a todas as tarefas (Figura 3.29).

```
volA <N> “<tarefa1>” [“<tarefaX>”]k [<time_out>] [<time_out>]k
ou
volB <N> “<tarefa1>” [“<tarefaX>”]k [<time_out>] [<time_out>]k
```

As combinações entre os três mecanismos básicos de gerenciamento de recursos 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 com simultaneidade N é:

```
div_sim <M> <N> “<tarefa1>” [“<tarefaX>”]k [<time_out>] [<time_out>]k
```

A Figura 3.30 (divisão por 3 com simultaneidade 2 entre duas tarefas, sem *timeouts*) 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
```

Seguindo a lógica, a combinação das três relações básicas (*divisão por Q com simultaneidade N com 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 3.33 (divisão por 2 com simultaneidade 2 com volatilidade 6, sem *timeouts*) é definido como

```
div_sim_volB 2 2 6 “<tarefa1>” “<tarefa2>”
```

A BNF a seguir define formalmente a linguagem apresentada.

```
<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.

3.5.2. O Programa Conversor

Esse programa é responsável por converter o arquivo de entrada do simulador de PNs no nível de *workflow* para um arquivo no nível de coordenação, estendendo as tarefas interdependentes e inserindo os mecanismos de coordenação entre elas. 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 estas 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. Ou seja, cada transição que possui uma dependência será transformada em uma estrutura como a da Figura 3.7 (ou suas versões simplificadas, mostradas na Figura 3.9).
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 para a ferramenta Visual Simnet [Garbe 97].

3.5.3. Utilizando a Ferramenta Visual Simnet

No protótipo desenvolvido, a ferramenta de simulação de PNs mostrada na Figura 3.52 é o Visual Simnet [Garbe 97], que é *freeware* e tem a capacidade de modelar e analisar PNs convencionais e estocásticas (cujos disparos das transições são 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 de domínio público 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*) e permite a exportação para outras ferramentas mais poderosas (por exemplo o INA[5] – *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, entre outras propriedades, a lista de *tokens*, definindo a marcação inicial da rede. Cada transição possui, entre 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 do programa) 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, sempre que possível serão usados os modelos da Figura 3.9 para a expansão. Para isso, a lTE também possui informação sobre o tipo de relação em que cada transição está envolvida.

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 da representação gráfica 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 sistema colaborativo.

O uso de PNs provê suporte matemático para análise e simulação do comportamento do ambiente colaborativo. Com o modelo baseado PNs e o protótipo apresentado, é possível prever e testar o comportamento de um ambiente de suporte ao trabalho colaborativo antes

[5] O INA, que é uma outra ferramenta de análise de PNs para a qual o Visual Simnet pode exportar os modelos criados, não trata transições com reserva de *tokens*. Portanto, alguns mecanismos de coordenação não podem ser exportados para esta ferramenta. No entanto, todos os exemplos deste trabalho foram simulados apenas com o Visual Simnet.

mesmo de sua implementação. Os modelos de coordenação apresentados neste capítulo são bastante genéricos e podem ser reutilizados em uma série de ambientes colaborativos, como será mostrado em casos de estudo no capítulo seguinte.

4. Casos de Estudo

Este capítulo mostra exemplos de uso dos modelos propostos na Seção 3.3 em uma série de ambientes colaborativos. Em todos os casos, o projetista do ambiente constrói as PNs que definem as possíveis seqüências de tarefas para cada participante e estabelece as dependências existentes entre essas tarefas (nível de *workflow*). A partir da definição do nível de *workflow*, o nível de coordenação é construído com as ferramentas apresentadas no capítulo anterior e submetido às diversas formas de análise (verificação, validação e desempenho – ver Apêndice A).

Os modelos utilizados usam PNs simples e foram simulados com o Visual Simnet, usando o protótipo descrito na Seção 3.5. Alguns dos exemplos aqui estudados foram apresentados em artigos mostrados no Apêndice D.

O capítulo inicia mostrando algumas combinações típicas entre relações temporais e de gerenciamento de recursos (Seção 4.1). Depois é estudado um exemplo genérico, desprovido de qualquer interpretação real, cujo objetivo é mostrar como usar os mecanismos de coordenação e as ferramentas de simulação e análise (Seção 4.2). As seções seguintes apresentam casos de estudos para diferentes classes de aplicações colaborativas: interação multiusuário via *whiteboard*, autoria colaborativa (*groupware*), *workflow* interorganizacional e ambiente virtual colaborativo.

4.1. Combinações entre Relações Temporais e de Gerenciamento de Recursos

Para ilustrar possíveis usos das relações temporais e dos gerenciadores de recursos, são mostrados nesta seção alguns exemplos de combinações típicas que envolvem ao mesmo tempo dependências temporais e de gerenciamento de recursos. As combinações são realizadas simplesmente pela justaposição das duas subredes (a da relação temporal e a de gerenciamento de recursos) no modelo estendido de tarefa (Seção 3.2).

O primeiro exemplo combina a relação temporal *tarefa 1 equals 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 4.1, tem duas subredes distintas entre as tarefas: a da relação temporal entre os lugares *start_task* e *finish_task* e a do gerenciador de recursos, entre os lugares *request_resource*, *assigned_resource* e *release_resource*. O gerenciador de recursos garante que os *tokens* chegarão juntos aos *assigned_resources* de ambas as tarefas. O mecanismo para a relação temporal garante que as duas tarefas começarão e teminarão suas execuções simultaneamente (*tokens* saindo juntos dos *start_task* e chegando juntos aos *finish_task*). Ambas as tarefas possuem *timeoutA* para a relação temporal e nenhum *timeout* para a de gerenciamento de recursos. O arquivo que define estas dependências é assim descrito na linguagem mostrada no capítulo anterior:

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

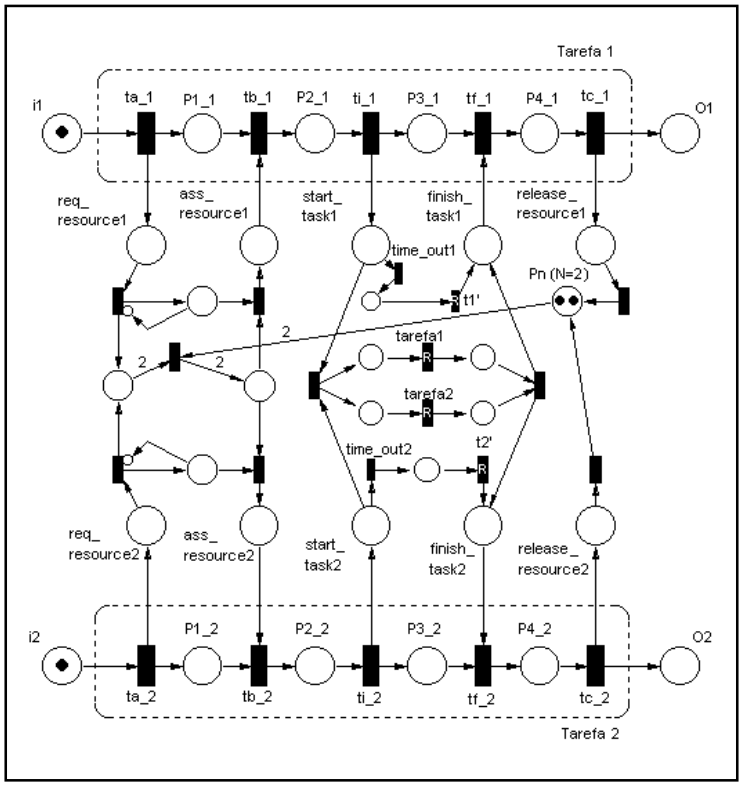

Figura 4.1: *Tarefa 1 equals Tarefa 2* e *Simultaneidade 2*.

O segundo exemplo combina a relação temporal *tarefa 2 duringB tarefa 1* 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 4.2) é uma combinação das subredes para a relação temporal e para o gerenciador de recursos.

As dependências da Figura 4.2 são definidas da seguinte forma (o único *timeout* existente é o que já está pré-definido no modelo da relação *duringB*):

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

Simulações realizadas com o modelo da Figura 4.2 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 *start_task2*. Uma possível solução seria colocar um *timeoutB* entre *start_task2* e o lugar de entrada *i2*, também liberando o *token* para o *release_resource2*. Dessa forma a rede poderia seguir um caminho alternativo que não passasse pela tarefa 2. Outra solução seria colocar um *timeoutA* para a tarefa 2 na relação temporal, permitindo a execução de uma tarefa alternativa caso a tarefa 1 não seja executada após um certo intervalo de tempo.

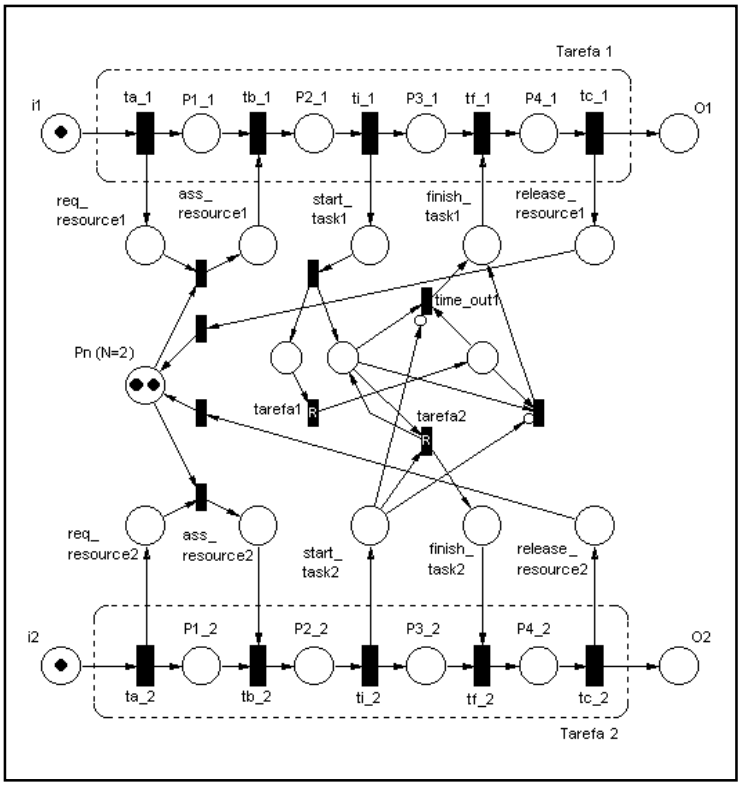

Figura 4.2: *Tarefa 2 duringB Tarefa 1* e *Divisão por 2*.

Como último exemplo de possíveis combinações, será mostrado um caso em que as dependências estabelecidas não permitem que nenhuma das tarefas seja executada. É o caso em que se combina a relação *tarefa 1 equals 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 *token* chegar a *start_task*. Como a outra tarefa não conseguirá colocar um *token* em *start_task*, a transição *t1* (Figura 4.3) nunca estará habilitada, causando um *deadlock* inevitável.

O problema desta última combinação, embora possa ser intuitivamente detectado, serve para mostrar como o modelo em PNs permite a detecção de erros na modelagem através da simulação do funcionamento da rede. Essa é uma das motivações do uso de modelos em PNs, pois em situações mais complexas, onde os erros são menos previsíveis, a simulação do comportamento da rede assume grande importância.

4.2. Exemplo Genérico

Este exemplo não se preocupa em modelar uma situação específica de colaboração. Ele parte de PNs desprovidas de qualquer interpretação real para mostrar como utilizar os mecanismos de coordenação em uma situação qualquer.

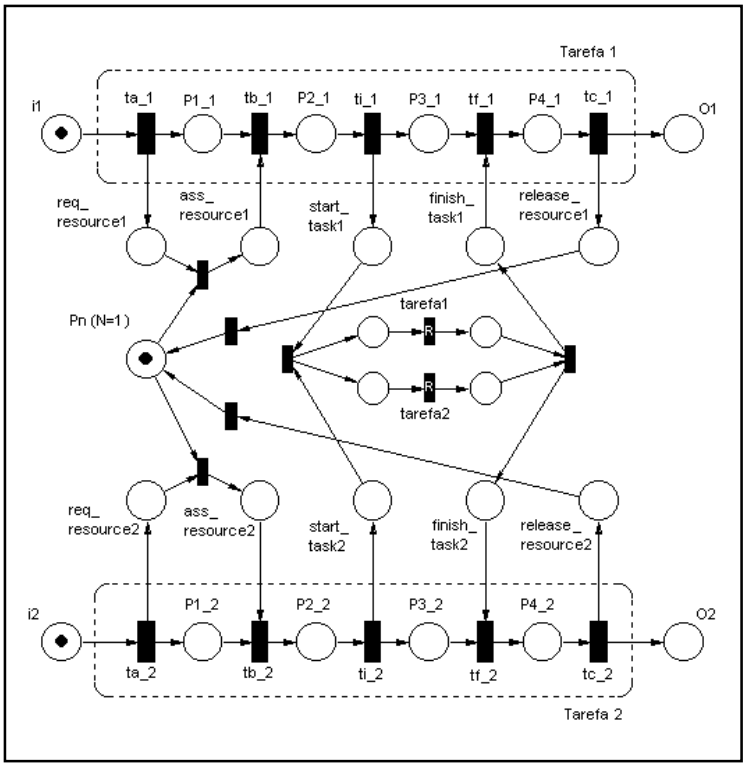

Figura 4.3: *Tarefa 1 equals Tarefa 2* e *Divisão por 1*. Situação de *deadlock*.

A Figura 4.4 mostra o nível de *workflow* do modelo (este exemplo já foi mostrado na Figura 3.6). Existem duas PNs que poderiam representar dois participantes de uma atividade colaborativa, a primeira delas (Participante A) se inicia com um roteamento condicional exclusivo (*OR-split* e depois *OR-join* – ver Apêndice B) e a segunda (Participante 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. Observa-se que as dependências podem ocorrer entre tarefas de participantes diferentes ou entre tarefas de um mesmo participante.

O arquivo que define estas dependências é mostrado a seguir:

```
overlapsB “T1a” “T1b”
duringB “T3b” “T2b”
div 1 “T5a” “T4b”
sim 2 “T6a” “T5b”
```

Com o uso das ferramentas descritas na Seção 3.5, o trabalho de modelagem do projetista termina no nível de *workflow*. A passagem para o nível de coordenação é feita inserindo os mecanismos de coordenação entre as tarefas interdependentes (retângulos não preenchidos na figura). O modelo no nível de coordenação, usando os mecanismos modelados em PNs simples, é mostrado na Figura 4.5. Apesar de parecer complexa, a rede mostrada na Figura 4.5 é bastante modular e facilmente montada a partir da rede no nível de *workflow* e dos mecanismos e coordenação pré-definidos.

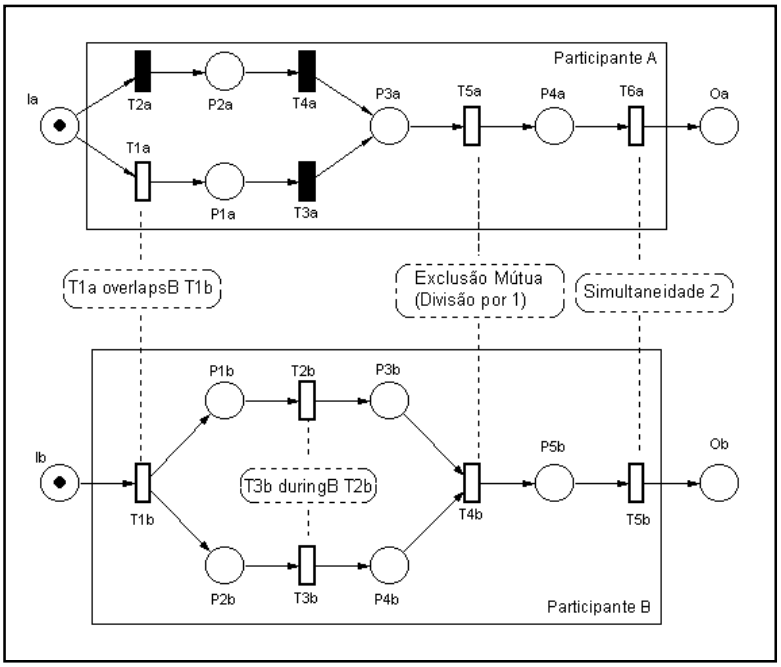

Figura 4.4: Nível de *workflow* do exemplo genérico.

A partir do modelo no nível de coordenação, foi possível realizar uma série de análises pertinentes. A análise de verificação através da *coverability tree* revelou, dentre os 518 estados possíveis para a PN, apenas duas situações de erro devido a *deadlocks*.

A primeira delas ocorre quando o participante A opta por seguir o caminho passando por *T2a*. Nesse caso, *T1a* não será executada e, conseqüentemente, *T1b* também não, pois ela precisa se sobrepor a *T1a*. O participante B, portanto, fica bloqueado antes da realização de *T1b* e o participante A será bloqueado mais à frente, por ocasião da tarefa *T6a*, que exige simultaneidade 2 com a *T5b*. O *timeout* fazendo com que *T1b* volte ao seu estado inicial não resolveria, pois a rede do participante 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*). Uma alternativa para evitar a possibilidade deste *deadlock* seria uma alteração no modelo no nível de *workflow* criando, por exemplo, uma dependência adicional *T2a overlapsB T1b*, de modo que *T1b* seja executada independente do caminho seguido pelo participante A. No entanto, quando a rede modela um sistema real, este tipo de alternativa pode não ser aceitável.

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

Outros tipos de análise, tais como *liveness*, *boundness*, persistência e distância síncrona entre transições (ver Apêndice A) podem ser realizadas com o modelo em PN, caso o projetista julgue necessário. 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 uma situação concreta.

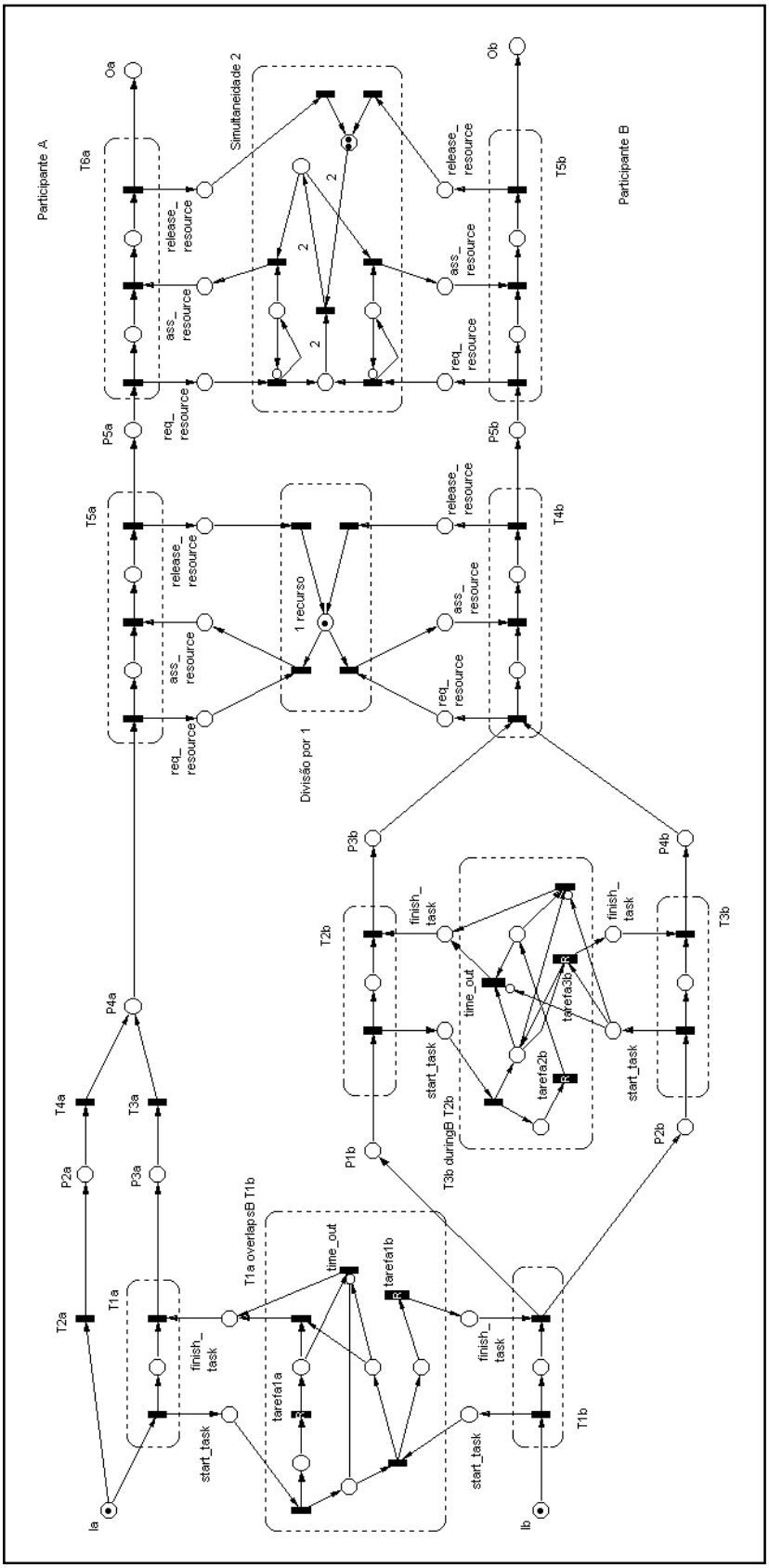

Figura 4.5: Nível de coordenação do exemplo genérico.

4.3. *Whiteboard* Compartilhado

Este exemplo ilustra uma situação típica da classe de aplicações que envolve interação multiusuário. Nesse ambiente hipotético, dois usuários interagem por meio de um *whiteboard* compartilhado. Para controlar a interação nesse ambiente, é estabelecido o modelo da Figura 4.6 (nível de *workflow*).

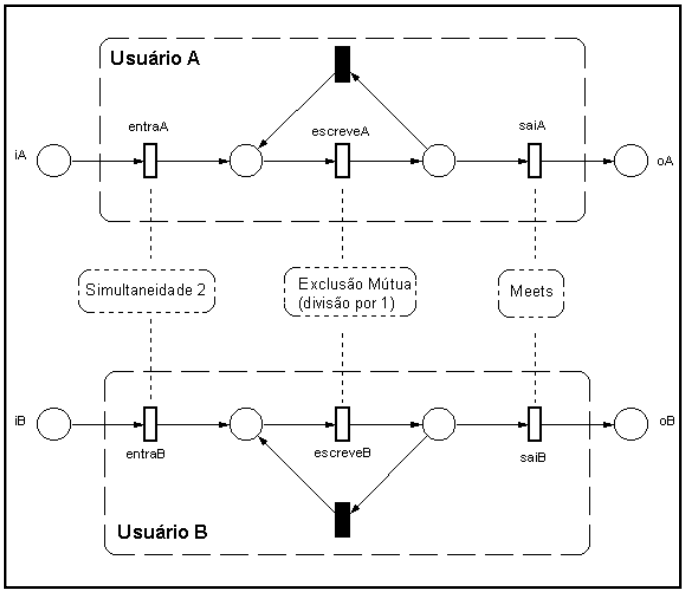

Figura 4.6: Nível de *workflow* do *whiteboard* compartilhado.

De acordo com o modelo estabelecido, cada usuário pode iniciar a interação (“entrar”) no ambiente, escrever várias vezes no *whiteboard* (estrutura de iteração – ver Apêndice B) e depois encerrar a interação (“sair”). O *whiteboard* aqui definido é um canal para comunicação síncrona entre dois usuários. Por esta razão, ele só está disponível quando os dois usuários o requisitarem, definindo a relação de *simultaneidade 2* entre as tarefas de iniciar a interação. Também é definido que apenas um usuário pode escrever no *whiteboard* de cada vez. Nesse caso, fica estabelecida uma relação de exclusão mútua (divisão por 1) entre as tarefas de escrever no *whiteboard*. Finalmente, também é definido que o canal de comunicação deve ser fechado após a saída do primeiro usuário. Por esta razão, é definida a dependência *saiA meets saiB*, garantindo que assim que o usuário A encerrar a interação, o outro também encerrará.

Usando a linguagem para a definição das interdependências, o seguinte arquivo é definido para este exemplo:

```
sim 2 “entraA” “entraB” time_out time_out
div 1 “escreveA” “escreveB” time_out time_out
meets “saiA” “saiB”
```

O modelo para o nível de coordenação deste exemplo é mostrado na Figura 4.7. Mais uma vez, vale a pena ressaltar que o modelo da Figura 4.7 é automaticamente obtido a partir do modelo da Figura 4.6, expandindo-se as tarefas interdependentes e inserindo os mecanismos de coordenação adequados entre elas.

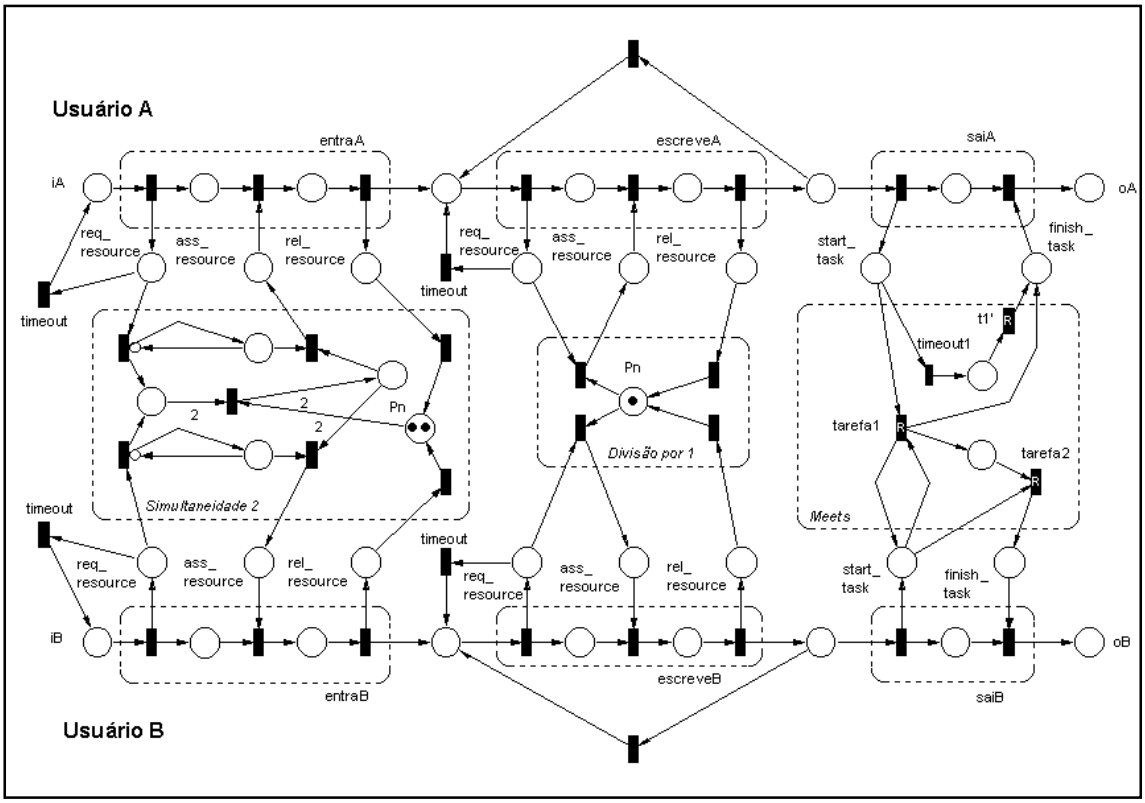

Figura 4.7: Nível de coordenação do *whiteboard* compartilhado.

A análise de verificação por meio da *coverability tree* indicou 861 estados possíveis para a PN, dentre os quais oito representam situações de *deadlock*. Na verdade, sete destes estados finais são decorrentes de disparos precipitados dos *timeouts*. A análise de validação mostrou que estes sete *deadlocks* podem ser evitados se os *timeouts* possuírem tempos adequados. O oitavo *deadlock* encontrado é o estado final considerado correto, com *tokens* em *oA* e *oB*, indicando que os dois usuários chegaram ao final de suas atividades.

Neste exemplo, análises de desempenho poderiam ser realizadas para medir, por exemplo, o tempo médio de espera para um usuário poder escrever no *whiteboard*, dada a taxa com que o outro requisita o uso do *whiteboard*.

Vale a pena ressaltar que a situação modelada é hipotética. A interação via *whiteboard* compartilhado poderia ser modelada utilizando outros mecanismos de coordenação (por exemplo, sem a restrição de ambos terem que iniciar a interação juntos).

O exemplo apresentado nesta seção foi também modelado em PNs de alto nível em [Raposo 00c] (ver Apêndice D).

4.4. Autoria Colaborativa

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 permitido a escrita a outros usuários. O usuário B é um usuário especial, que sempre pode editar o documento, independentemente da autorização do dono.

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 4.8.

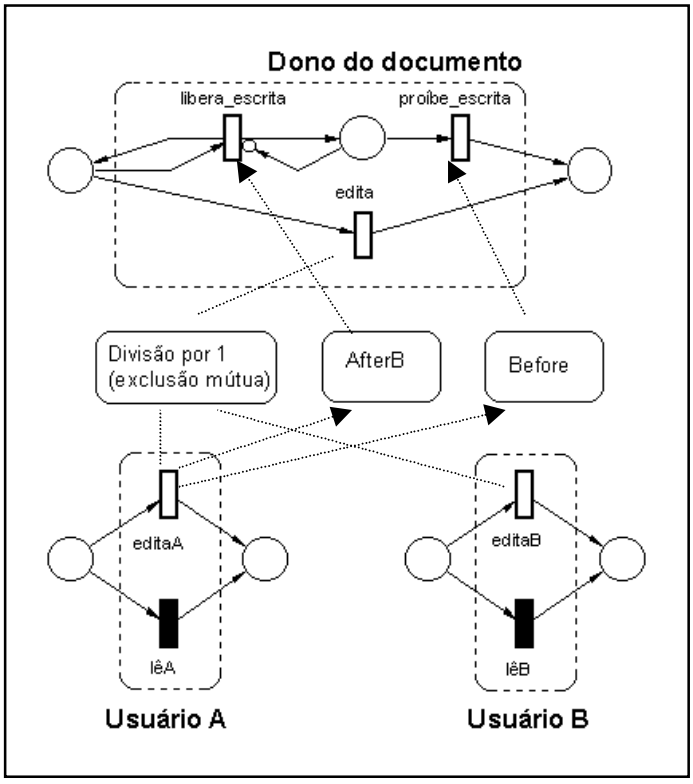

Figura 4.8: Nível de *workflow* do exemplo de autoria colaborativa.

A construção do nível de coordenação deste exemplo usando as ferramentas apresentadas no capítulo anterior é feita com o seguinte arquivo de dependências:

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

O modelo no nível de coordenação é mostrado na Figura 4.9.

A análise de verificação indicou que existem 429 estados possíveis para a PN da Figura 4.9, dentre os quais apenas dois representam situações de *deadlock*. No entanto, ambas são situações corretas, indicando que os usuários chegaram ao final de suas tarefas. A análise de validação comprovou que o sistema funciona como esperado: não ocorrem edições simultâneas do documento e o usuário A não viola as autorizações do dono o arquivo. A análise de desempenho, neste caso, poderia 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.

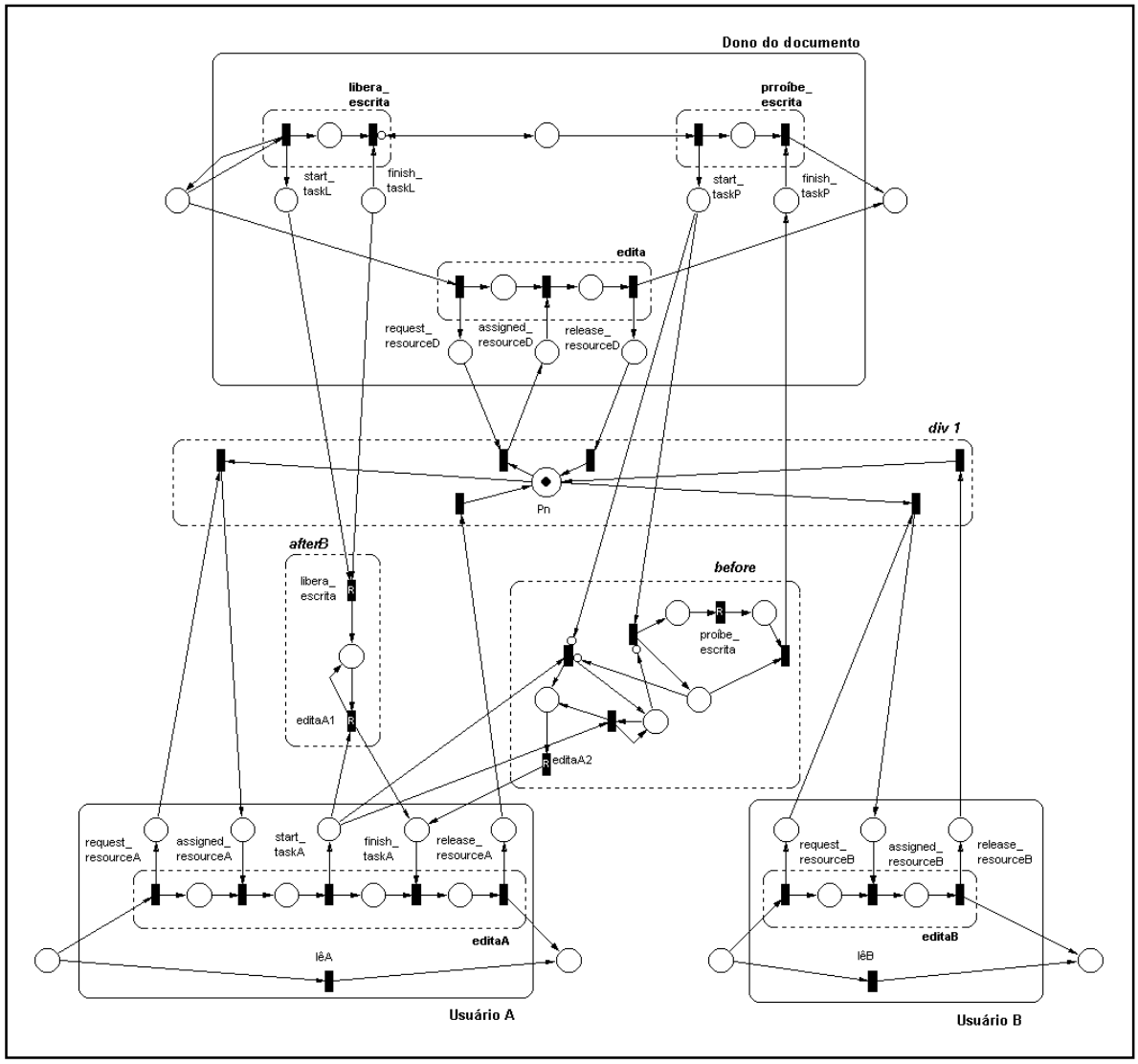

Figura 4.9: Nível de coordenação do exemplo de autoria colaborativa.

4.5. *Workflow* Interorganizacional

A crescente globalização da economia e a necessidade de cooperação entre as organizações formam um cenário promissor para os *workflows* interorganizacionais, que ultrapassam os limites de uma única organização. Os limites das organizações passam a ser mais fluidos do que antes. Múltiplas empresas que participam de um segmento de mercado compartilhado passam a planejar, implementar e gerenciar colaborativamente o fluxo de bens, serviços e informações ao longo de cadeias de valores que têm sua eficiência aumentada [Yang 00].

A necessidade de colaboração entre as organizações coloca uma série de questões que podem ser separadas em duas grandes classes: as relacionadas à definição de uma infra-estrutura de comunicação conjunta e as relacionadas à coordenação dos *workflows* colaborativos [Raposo 00a].

A primeira classe de problemas lida com a integração de ambientes heterogêneos de software em uma infra-estrutura comum de comunicação. Os problemas podem ser resumidos às dificuldades em se estabelecer um "contrato de interoperabilidade" entre as organizações participantes [Anderson 99]. Este contrato deve estabelecer, entre outros aspectos, quais dispositivos de *workflow* de uma organização devem ser capazes de interagir com dispositivos das outras organizações, a tecnologia de transporte, o protocolo de comunicação e políticas de segurança. Atualmente já estão sendo elaborados padrões para tratar este tipo de problema [Hayes 00], tais como a Interface 4 da WfMC (*Workflow Management Coalition*) [WfMC 96], o SWAP (*Simple Workflow Access Protocol*) da IETF (*Internet Engineering Task Force*) [SWAP 99], [Bolcer 99] e o Wf-XML, também da WfMC [WfMC 00].

A segunda classe de problemas – relacionada à coordenação de *workflows* cooperativos – aparece em um nível de abstração mais alto. Nesse caso, assume-se que o ambiente de comunicação entre os parceiros está claramente definido e a maior preocupação é como coordenar as atividades colaborativas. Os *workflows* interorganizacionais exigem mecanismos de coordenação mais complexos que os dos sistemas colaborativos em geral, pois eles também precisam prover suporte para a colaboração "inter-grupo" e não só para a "intra-grupo" (i.e., dentro de um grupo já estabelecido e com certas convenções definidas) [Simone 99].

É com essa segunda classe de problemas que os modelos de coordenação aqui apresentados estão relacionados. Com os mecanismos apresentados, é possível modelar a interação entre vários *workflows* e verificar a consistência e correção dos mesmos por meio de análise e simulação das PNs geradas. Isso é importante porque as organizações só gastarão tempo e dinheiro para o estabelecimento de uma infra-estrutura conjunta de comunicação com seus parceiros quando tiverem certeza da eficiência da estrutura de coordenação de seus *workflows* cooperativos.

Para ilustrar o uso dos mecanismos de coordenação em *workflows* interorganizacionais, é mostrado um exemplo de um ambiente composto por três "organizações" independentes: um consumidor, uma loja virtual e um produtor.

O exemplo representa uma situação típica de comércio eletrônico, onde o consumidor contacta uma loja virtual para comprar algo. A loja, no entanto, é apenas um intermediário entre o consumidor e o produtor, tendo que contactar este último para receber o produto a ser enviado para o consumidor. O nível de *workflow* deste exemplo é mostrado na Figura 4.10, destacando aspectos da relação entre a loja e o produtor.

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* (consumidor) *before deliver_goodsS* (loja). 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. Caso contrário, a loja pode agendar a entrega do produto, atividade que deve ocorrer simultaneamente ao agendamento feito pelo fabricante (*schedule_deliveryS equals schedule_deliveryP*). Se houver problema com o cartão de crédito, *schedule_deliveryS* não será executada e, conseqüentemente, *schedule_deliveryP* também não. Usando um *timeout* 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.

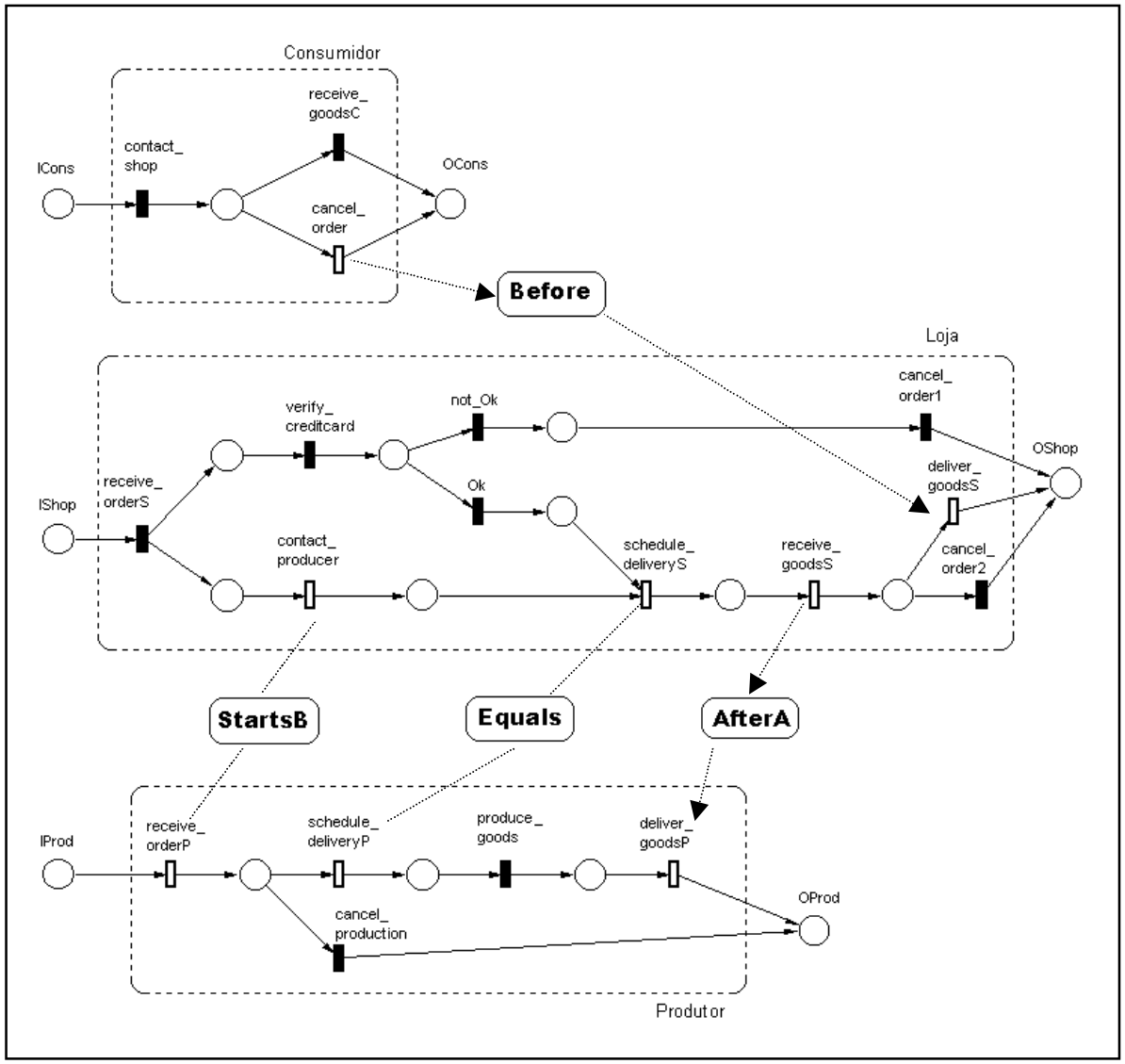

Figura 4.10: Nível de *workflow* do exemplo de *workflow* interorganizacional.

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 *timeouts* do tipo B, ou seja, *timeouts* que retornam os *tokens* aos locais de entrada das tarefas após um certo tempo de espera).

```
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 4.11. A análise da *coverability tree* mostrou que a rede pode atingir 1532 estados, sendo doze deles estados finais (*deadlocks*). Alguns destes estados finais, 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 [van der Aalst 98]. A análise de desempenho poderia ser realizada, por exemplo, para medir o tempo médio de espera dos consumidores, dadas as taxas com que os produtos são fabricados.

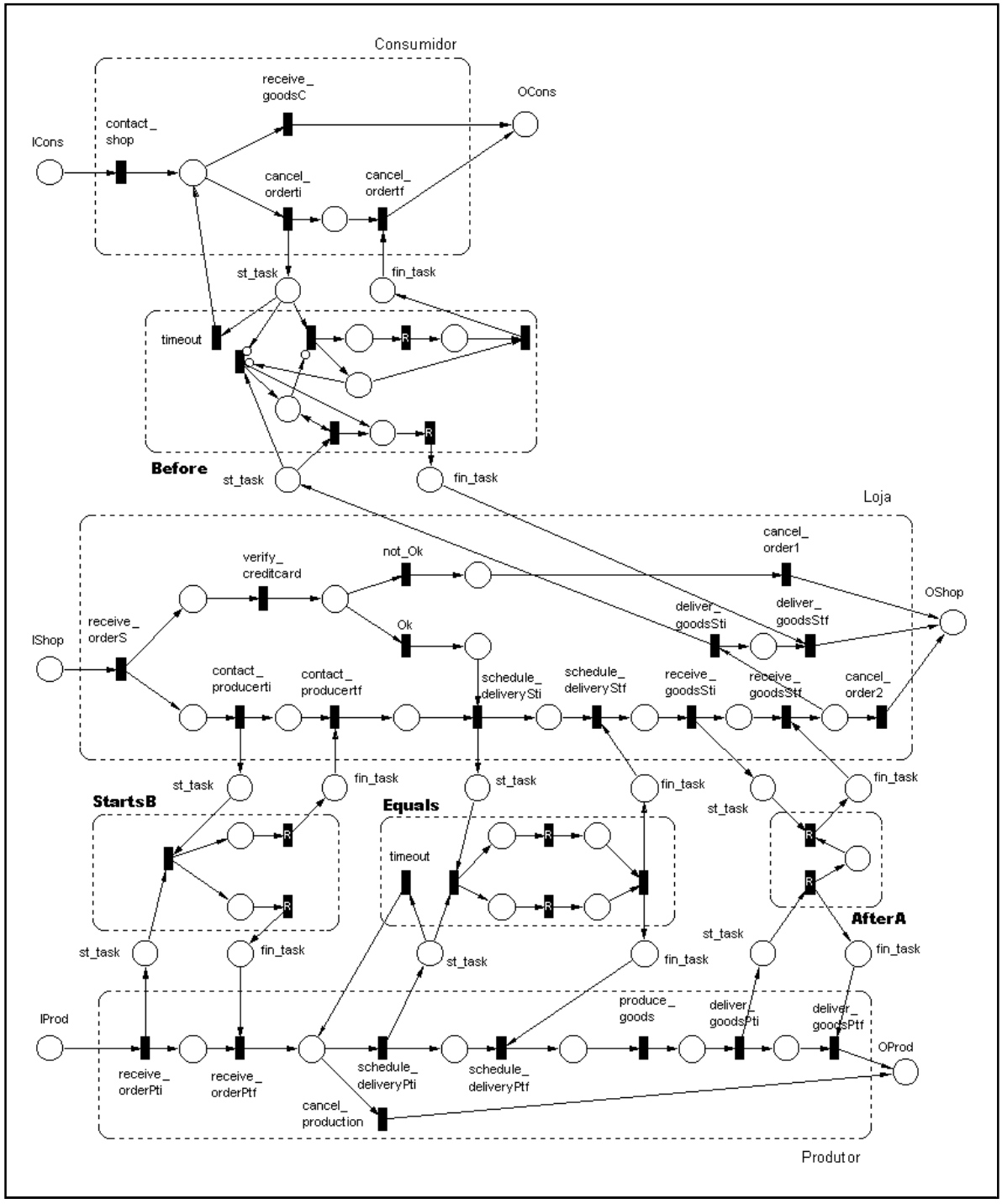

Figura 4.11: Nível de coordenação do exemplo de *workflow* interorganizacional.

Esse 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 esse exemplo não teve o objetivo de 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 foi apenas mostrar como os mecanismos de coordenação podem ser utilizados em mais uma situação prática.

4.6. Ambiente Virtual Colaborativo

Os ambientes virtuais colaborativos ou CVEs (*Collaborative Virtual Environments*) são "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].

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).

Pelas suas características, os CVEs apresentam grande potencial para o suporte ao trabalho colaborativo. Até o presente momento, no entanto, o desenvolvimento de CVEs tem sido essencialmente voltado para atividades de "lazer" (controladas pelo protocolo social), permitindo basicamente que usuários se comuniquem remotamente ou naveguem por ambientes tridimensionais [Frécon 98]. Para que os CVEs possam ser efetivamente utilizados como ferramenta de trabalho colaborativo é necessário, dentre outros aspectos, investir no planejamento e coordenação das atividades. É nesse sentido que os mecanismos de coordenação aqui apresentados poderão ser úteis [Raposo 00d].

Da mesma forma que nos exemplos anteriores, cada usuário de um CVE pode ser representado por uma PN independente. A interação colaborativa no mundo virtual é definida pelas interdependências entre as tarefas realizadas pelos usuários e coordenada pelos mecanismos de coordenação. Isso permitirá prever e testar o que pode acontecer no mundo virtual antes de sua implementação. Uma experiência semelhante foi realizada usando PNs para controlar o comportamento de animações modeladas por computador [Magalhães 98].

Para ilustrar o uso do modelo de coordenação em CVEs foi implementado um caso de estudo onde um usuário interage com um agente (representando um possível segundo usuário). O exemplo modela uma espécie de videogame baseado na segunda tarefa de Hércules, da mitologia grega. Segundo a lenda [Borges 00], [Leadbetter 00], Hércules precisava matar a Hidra de Lerna, um monstro com nove cabeças que renasciam quando cortadas. Para conseguir atingir seu objetivo, Hércules contou com a colaboração do sobrinho Iolau, que cauterizava cada uma das cabeças que o tio cortava, impedindo que elas nascessem de novo. A última cabeça, no entanto, não podia ser cortada. Para matar o

monstro, eles tiveram que enterrá-la em uma cova profunda e colocar uma grande pedra em cima.

O modelo no nível de *workflow* para este "videogame" é mostrado na Figura 4.12. Existem duas PNs idênticas, uma para o usuário e outra para o agente. Cada um pode assumir o papel de Hércules (parte de cima das redes) ou de Iolau (parte de baixo das redes). Hércules deve pegar a espada, cortar as oito cabeças e depois empurrar o monstro para a cova e colocar a pedra em cima. Iolau deve pegar a tocha, cauterizar as cabeças cortadas e cavar o buraco.

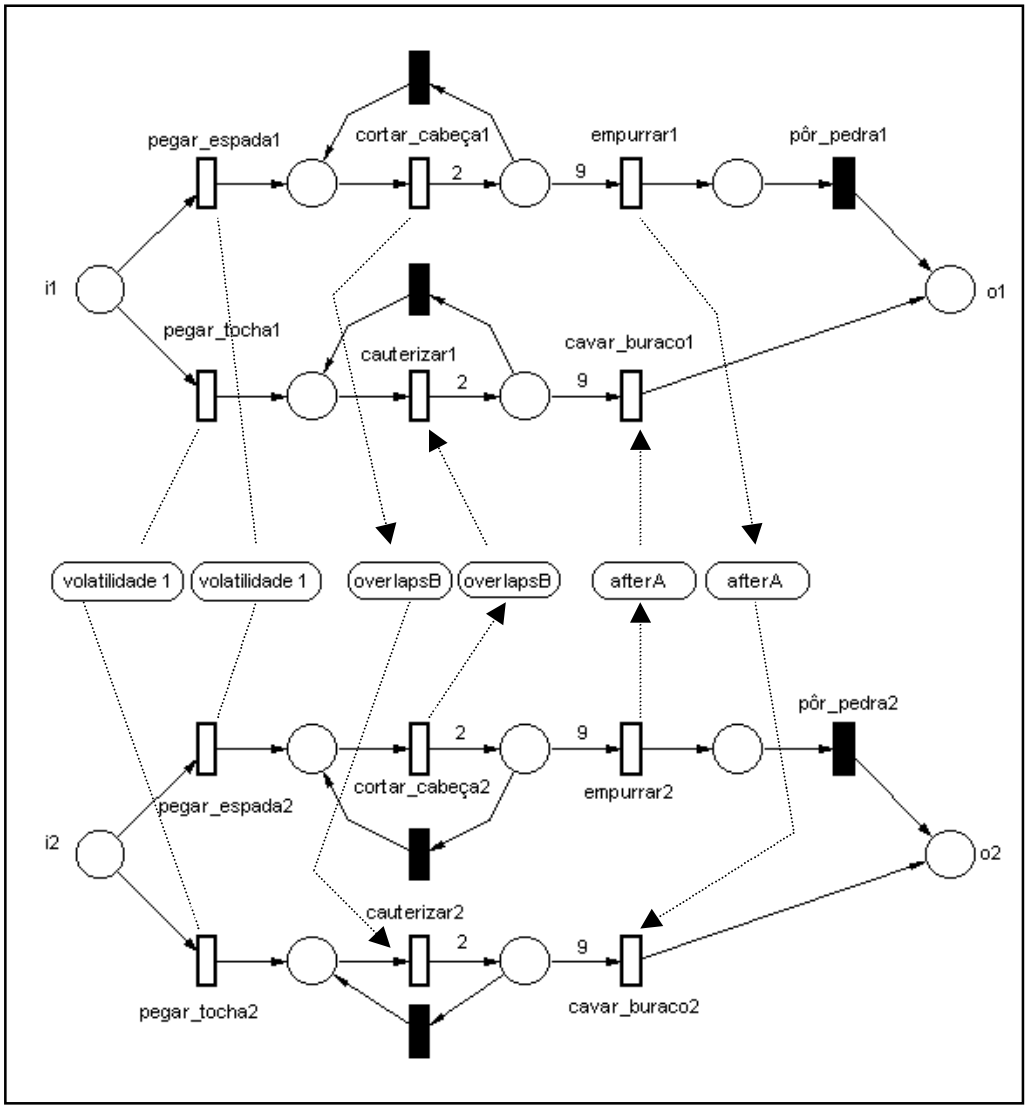

Figura 4.12: Nível de *workflow* para o ambiente virtual colaborativo.

A definição de quem (usuário ou agente) vai assumir que papel (Hércules ou Iolau) depende da primeira tarefa. A *volatilidade 1* garante que cada um assumirá um papel diferente (só há uma espada e uma tocha disponíveis). A cauterização após o corte de cada cabeça é garantida com dependências do tipo *overlapsB*. Também existe a dependência de que Hércules só pode empurrar a Hidra no buraco se ele já tiver sido cavado (*afterA*).

O arquivo para a construção do nível de coordenação para o exemplo da Figura 4.12 é o seguinte:

```
volB 1 "pegar_espada1" "pegar_espada2″
volB 1 "pegar_tocha1" "pegar_tocha2"
overlapsB "cortar_cabeça1" "cauterizar2"
overlapsB "cortar_cabeça2" "cauterizar1"
afterA "empurrar1" "cavar_buraco2"
afterA "empurrar2" "cavar_buraco1"
```

A rede no nível de coordenação é mostrada na Figura 4.13. As simulações com o Visual Simnet detectaram 804 estados possíveis, sendo sete estados finais. No entanto, os seis estados finais indesejados (o sétimo é o correto, com *tokens* em *o1* e *o2*) podem ser eliminados com o uso de *timeouts* adequados. Por exemplo, se os dois "atores" iniciarem a tarefa *pegar_espada*, apenas um deles a obterá. Com o *timeout*, o que não conseguiu pegar a espada pode voltar ao estado inicial e pegar a tocha, dando prosseguimento à colaboração.

Este exemplo foi implementado utilizando o blaxxun Contact [Blaxxun 00], um cliente para comunicação multimídia que oferece recursos como suporte a VRML (*Virtual Reality Modeling Language*), *chats*, *message boards*, avatares (representação física dos usuários no ambiente virtual), interação com agentes, etc. Os recursos do blaxxun Contact utilizados nessa implementação foram o suporte à visualização de VRML e os avatares com movimentos pré-definidos.

O "videogame" é visualizado na forma de VRML e a interação com o usuário ocorre por meio de botões definidos em um *applet* Java que interage com o mundo VRML por meio da EAI (*External Authoring Interface*). A EAI é uma interface para permitir que ambientes (programas) externos acessem os objetos de uma cena VRML. Dessa forma, ao clicar sobre os botões do *applet*, o usuário determina a execução de uma tarefa no mundo virtual. A forma como este exemplo foi implementado permite que os mecanismos de coordenação atuem sobre os botões da interface, habilitando-os ou não caso as respectivas tarefas possam ser executadas ou não. Por exemplo, o botão *put stone* na interface do usuário que assume o papel de Hércules só estará habilitado após o término da tarefa "empurrar". A Figura 4.14 mostra alguns quadros do "videogame" implementado (as nove cabeças da Hidra são repesentadas por nove "monstros" na implementação).

Para dar maior dinamismo ao "videogame", o agente possui um comportamento aleatório, podendo demorar ou não para executar as tarefas que lhe são atribuídas. Por exemplo, quando o usuário assume o papel de Hércules, o agente (no papel de Iolau) pode ou não cauterizar a cabeça cortada por Hércules antes dela renascer (o que é modelado inserindo um *timeoutB* na tarefa "cauterizar" do modelo da Figura 4.13).

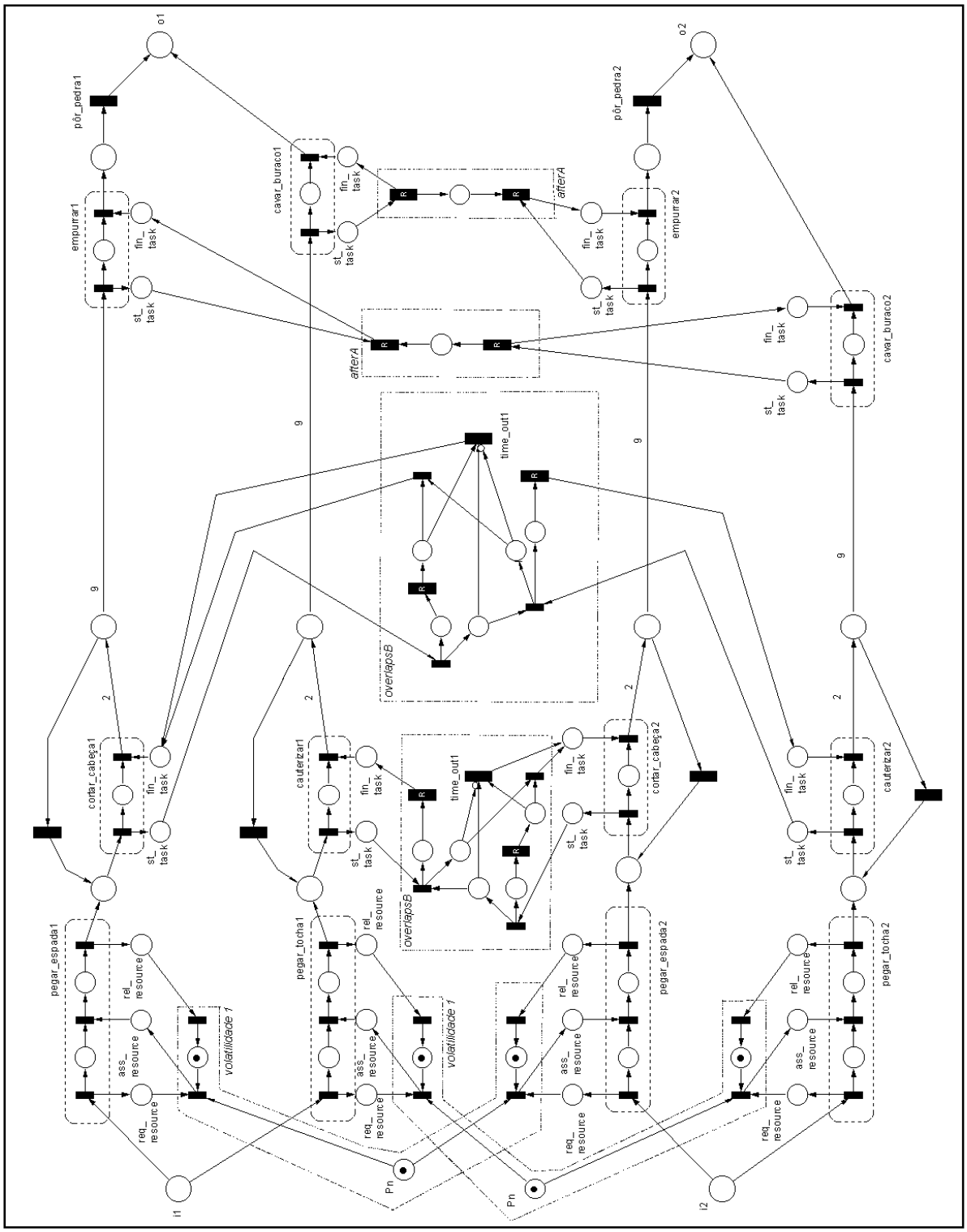

Figura 4.13: Nível de coordenação para o ambiente virtual colaborativo.

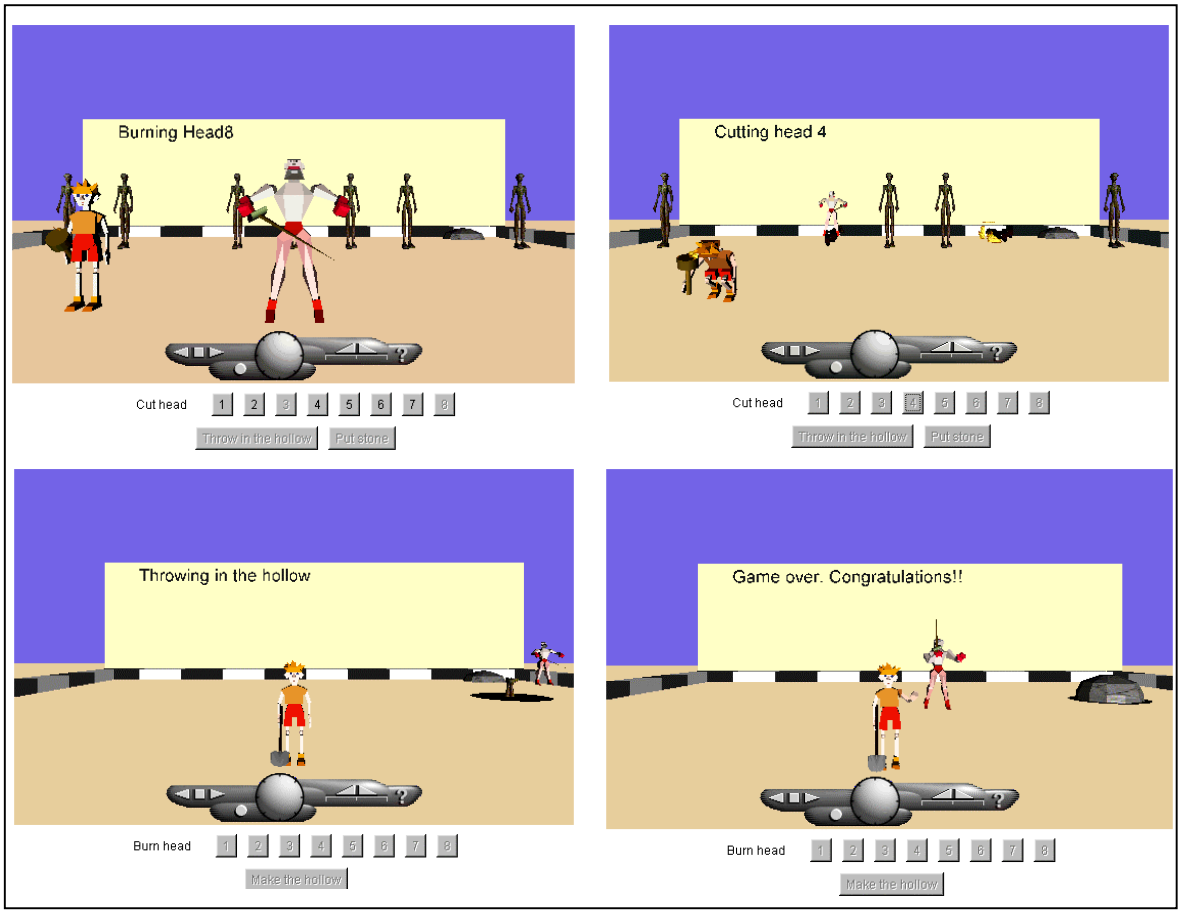

Figura 4.14: Quadros do “videogame” implementado.

Como foi mostrado nos exemplos deste capítulo, os mecanismos de coordenação modelados servem como ferramentas de simulação e análise para verificar a correção e validar a eficiência de vários tipos de ambiente colaborativo antes de sua implementação. No entanto, ainda existe uma distância entre o modelo e a implementação, como identificado no exemplo de CVE, já que não existe uma ferramenta para “traduzir” os modelos em componentes de software que pudessem ser efetivamente utilizados na implementação dos diversos ambientes colaborativos. Esse e outros aspectos serão abordados no capítulo de conclusões a seguir.

5. Conclusões

Este trabalho abordou a questão da coordenação entre tarefas em ambientes computacionais de apoio ao trabalho colaborativo sob a óptica da modelagem destes ambientes. A coordenação é a principal atividade do trabalho de articulação, que é o trabalho extra necessário para garantir a eficiência do trabalho colaborativo a partir das tarefas individuais.

O Capítulo 2 mostrou que a preocupação, por parte dos projetistas de sistemas colaborativos, com o suporte ao trabalho de articulação tem aumentado continuamente. Por esta razão, têm surgido sistemas que oferecem mecanismos de coordenação cada vez mais flexíveis e acessíveis não só aos projetistas de aplicações colaborativas mas também aos usuários finais.

Com base nessa preocupação, o Capítulo 3 apresentou um modelo de coordenação para tarefas colaborativas baseado em três níveis de abstração. O projetista do ambiente colaborativo se concentra no nível mais alto (nível de *workflow*), onde são definidas as seqüências de tarefas a serem realizadas pelos participantes da colaboração e as interdependências entre estas tarefas. O nível intermediário (nível de coordenação) é obtido a partir do anterior, expandindo as tarefas interdependentes segundo um modelo pré-definido e inserindo os mecanismos de coordenação, também pré-definidos, entre estas tarefas. O nível de coordenação pode ser obtido automaticamente a partir do nível de *workflow* utilizando o protótipo desenvolvido. No nível de coordenação é possível fazer a simulação e a análise do ambiente modelado. O nível de abstração mais baixo é o nível de especificação de tarefas, onde cada tarefa é expandida em uma subrede que modela sua execução (importante para a etapa de implementação do ambiente colaborativo).

A abordagem utilizada para a criação do modelo de coordenação foi baseada na separação entre tarefas e interdependências e na utilização de mecanismos de coordenação, o que traz a vantagem da reutilização dos mecanismos em várias situações de colaboração. Além disso, várias políticas de coordenação podem ser utilizadas em um mesmo ambiente colaborativo, bastando para isso trocar os mecanismos de coordenação. A uniformidade do modelo também facilita a padronização, desde que os elementos necessários sejam incorporados aos modelos de referência existentes [WfMC 98], [WfMC 99].

O trabalho apresentou duas contribuições distintas: a definição de um conjunto genérico de interdependências entre tarefas colaborativas e a modelagem dos mecanismos de coordenação baseados em PNs para tratar estas dependências.

Com relação ao conjunto de interdependências, ele foi dividido em duas partes, uma voltada para as dependências temporais e outra para as de gerenciamento de recursos. As dependências temporais foram baseadas nas relações temporais de Allen [Allen 84], que foram expandidas e adaptadas ao domínio de interesse (tarefas colaborativas). Quanto às dependências de gerenciamento de recursos não é de conhecimento do autor nenhum trabalho que classifique tais dependências no contexto.

O conjunto de interdependências apresentado, no entanto, não pretende ser completo. Acredita-se que seja muito difícil estabelecer um *framework* com todas as possíveis interdependências entre tarefas, principalmente quando não se deseja trabalhar com um domínio de aplicação específico. Por esta razão, o trabalho optou pela possibilidade de

extensão dos modelos, ao invés de tentar provar a completude do conjunto de dependências proposto.

Com relação aos mecanismos de coordenação, a escolha de PNs como ferramenta de modelagem se deu pela simplicidade e o grande número de ferramentas computacionais para suporte à modelagem, simulação e análise das mesmas. Além disso, as PNs se mostraram adequadas para definir a estrutura de coordenação nos diferentes níveis de abstração (níveis de *workflow*, coordenação e especificação de tarefas).

Um problema recorrente no uso de PNs é a explosão de estados. A análise de modelos baseados em PNs tende a se tornar muito complexa, mesmo em sistemas de porte modesto [Murata 89]. Por essa razão, o uso de PNs exige um balanço entre generalidade e capacidade de análise. O uso de PNs para aplicações específicas geralmente vem acompanhado de modificações no modelo básico para aumentar as possibilidades de análise. Neste trabalho optou-se pela generalidade, utilizando o modelo básico de PNs para a modelagem dos mecanismos. Como forma de amenizar o problema da explosão de estados, também foram apresentados modelos para os mecanismos de coordenação utilizando PNs de alto nível. Os modelos em PNs de alto nível de fato são mais compactos que os que usam PNs convencionais (i.e., têm menos estados e transições). No entanto, o uso de PNs de alto nível sacrifica algumas características importantes do modelo básico, tais como a facilidade de entendimento dos modelos e a disponibilidade de ferramentas para análise e simulação (muitas ferramentas não simulam este tipo de rede ou usam noções diferentes do conceito de PN de alto nível – não aceitam variáveis, por exemplo).

Devido à clara separação entre a definição das dependências e os mecanismos de coordenação, aplicações que não usam PNs podem seguir as idéias aqui apresentadas, utilizando o conjunto de dependências definido e criando mecanismos adequados para coordená-las.

Outra contribuição deste trabalho foi a criação da linguagem para a definição das dependências e a implementação do protótipo que automatiza a passagem do nível de *workflow* para o nível de coordenação dos modelos, simplificando o trabalho do projetista do ambiente colaborativo. A linguagem para a definição das dependências está diretamente relacionada ao conjunto de dependências definido e, quando uma nova dependência for criada, ela pode ser estendida. O protótipo implementado é dependente da ferramenta de simulação usada (Visual Simnet), mas pode ser usado como guia para futuras implementações que usem outras ferramentas.

O Capítulo 4 mostrou como o modelo de coordenação desenvolvido é bastante genérico e pode ser utilizado (sempre pensando em modelagem, simulação e análise) em uma série de ambientes colaborativos. Os dois últimos exemplos, em particular, mostraram como os resultados deste trabalho podem ser aplicados em sistemas colaborativos relativamente recentes e com grandes potenciais para se tornarem importantes aplicações no futuro próximo: *workflows* interorganizacionais e CVEs.

Os *workflows* interorganizacionais se enquadram no contexto da economia globalizada e das organizações virtuais, onde a cooperação entre as organizações é fundamental para melhorar a utilização de recursos, aumentar a qualidade dos produtos e baixar os custos de produção. Os *workflows* interorganizacionais e os mecanismos de coordenação aqui apresentados estão no nível de abstração mais alto da integração entre os sistemas de informação das organizações virtuais [Hasselbring 00]. Esta integração passa por várias camadas, até chegar no nível de abstração mais baixo, que é a integração de *middleware* por meio de tecnologias tais como CORBA. A coordenação das atividades

interorganizacionais é uma questão bastante complexa, pois ela envolve a cooperação "inter-grupo", que exige um "contrato de interoperabilidade" entre as organizações envolvidas, definindo uma ontologia comum, questões de segurança, etc. Os mecanismos de coordenação aqui apresentados são úteis nesse contexto como forma de testar e validar a cooperação entre as organizações antes da implementação dos mecanismos de integração definidos no "contrato de interoperabilidade".

Os CVEs já existem há algum tempo, mas até hoje têm sido basicamente utilizados para atividades que não necessitam de nenhum tipo de coordenação, tais como a navegação por ambientes tridimensionais. A possibilidade de colocar vários usuários "imersos" em um ambiente 3D, no entanto, representa um grande potencial para a realização de tarefas colaborativas mais complexas, tais como elaboração de projetos de arquitetura, *design* de produtos, simulações de treinamento, etc. Estas tarefas exigem mecanismos de coordenação para garantir a eficiência da colaboração. Como em todos os outros exemplos, os mecanismos de coordenação em PNs servem para testar o comportamento do ambiente virtual antes de sua implementação. Além disso, os mecanismos de coordenação aqui apresentados podem servir como formas de guiar a implementação de CVEs que sejam realmente capazes de realizar tarefas colaborativas. Para ilustrar essa possibilidade, foi implementado um pequeno "videogame" envolvendo dois participantes.

Os mecanismos de coordenação modelados, juntamente com o protótipo implementado, servem por enquanto como ferramentas de análise e simulação para verificar a correção e validar a eficiência de ambientes colaborativos antes de sua implementação. No entanto, como identificado na implementação do CVE (Seção 4.6), ainda não há como utilizar diretamente estes mecanismos na implementação dos sistemas colaborativos. Como sugestão de trabalho futuro, um próximo passo poderá ser a criação de componentes de software que implementem estes mecanismos de coordenação como "caixas pretas", colocadas entre as tarefas colaborativas, de forma a garantir que suas interdependências não sejam violadas. Os componentes de coordenação serão capazes de receber e gerar eventos de/para as tarefas colaborativas, controlando sua execução. Dessa forma, eventos relacionados às tarefas colaborativas (início de uma tarefa, por exemplo) poderão acarretar eventos de saída que controlarão a execução das tarefas interdependentes (bloqueando a execução de outra tarefa, por exemplo). Os modelos em PNs dos mecanismos de coordenação controlarão internamente, de forma transparente para o projetista do ambiente colaborativo, o comportamento dos componentes de software.

Outra possibilidade de trabalho futuro é a reavaliação dos mecanismos propostos para sua adequação a domínios específicos, tais como *workflows* interorganizacionais ou CVEs. Neste caso, se for detectada a existência de novos tipos de interdependências, estas devem ser acrescentadas ao conjunto definido e os respectivos mecanismos de coordenação devem ser modelados. Este trabalho poderia englobar também a remodelagem dos mecanismos de coordenação usando outros tipos de PNs ou mesmo outros tipos de ferramentas de modelagem.

Com relação ao problema da explosão de estados, uma possibilidade além da remodelagem dos mecanismos com outros tipos de PNs seria a análise modular das redes, em que partes do modelo podem ser tratadas independentemente das outras [Christensen 92]. Outra possibilidade é a utilização de metodologias como CPM (*Critical Path Method*) e PERT (*Program Evaluation and Review Technique*) [Nilsson 86] para concentrar o esforço de coordenação nas atividades do caminho crítico em um ambiente complexo.

Finalmente, é importante reforçar que a coordenação de tarefas interdependentes em ambientes colaborativos é um problema que precisa ser tratado para garantir a eficiência da colaboração. A separação entre atividades e interdependências, juntamente com o uso de mecanismos de coordenação reutilizáveis, são passos em direção a este objetivo.

Referências

[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.

[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.

[van der Aalst 99] W. M. P. van der Aalst. Interorganizational Workflows: An approach based on Message Sequence Charts and Petri Nets. *Systems Analysis-Modelling-Simulation*, 34(3): 335-367. 1999.

[Ackerman 96] M. S. Ackerman and B. Starr. Social Activity Indicators for Groupware. *IEEE Computer*, 29(6): 37-42. June 1996.

[Al-Salqan 96] Y. Al-Salqan and C. K. Chang. Temporal Relations and Synchronization Agents. *IEEE Multimedia*, 3(2): 30-39. Summer 1996.

[Allen 83] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11): 832-843. November 1983.

[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*. April 1999.

[Araújo 99] R. M. Araújo and M. R. S. Borges. The Role of Awareness in Collaborative Improvement of Software Processes. *International Workshop on Groupware (CRIWG'99)*. 1999.

[Attie 96] P. C. Attie et al. Scheduling workflows by enforcing intertask dependencies. *Distributed Systems Engineering Journal*, 3(4): 222-238. December 1996.

[Bannon 91] L. J. Bannon and K. Schmidt. *CSCW: Four Characters in Search of a Context*. In J. M. Bowers and S. D. Benford (Eds.). *Studies in Computer Supported Cooperative Work*, pp. 3-16. North-Holland, 1991.

[Bardram 97] J. E. Bardram. Plans As Situated Action: An Activity Theory Approach to Workflow Systems. *Proc. of the 5th European Conf. on Computer-Supported Cooperative Work (ECSCW'97)*, pp. 17-32. 1997.

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

[Beaudouin-Lafon 99] M. Beaudouin-Lafon. *Preface*. In M. Beaudouin-Lafon (Ed.). *Computer Supported Co-operative Work*, pp. xi-xiii. Trends in Software 7, John Wiley & Sons, 1999.

[Benford 95] S. Benford, J. Bowers et al. User Embodiment in Collaborative Virtual Environments. *Proc. of the Conf. on Human Factors in Computing Systems (CHI'95)*. 1995.

[Blaxxun 00] blaxxun interactive. *blaxxun Contact 4.4*. <http://www.blaxxun.com/products/contact>. Agosto 2000.

[Bolcer 99] G. A. Bolcer and G. Kaiser. SWAP: Leveraging the Web To Manage Workflow. *IEEE Internet Computing*, 3(1): 85-88. January-February 1999.

[Borges 00] J. L. Borges e M. Guerrero. *O Livro dos Seres Imaginários*. Ed. Globo, 2000.

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

[Cabri 00] G. Cabri, L. Leonardi and F. Zambonelli. Mobile-Agent Coordination Models for Internet Applications. *IEEE Computer*, 33(2): 82-89. February 2000.

[CACI 00] CACI. *SIMPROCESS*. <http://www.simprocess.com>. 2000.

[Camarinha-Matos 98] L. M. Camarinha-Matos and H. Afsarmanesh. Flexible Coordination in Virtual Enterprises. *Proc. of the 5th Int. Workshop on Intelligent Manufacturing Systems (IMS'98)*, pp. 43-48. 1998.

[Christensen 92] S. Christensen and L. Petrucci. *Towards a Modular Analysis of Coloured Petri Nets*. In K. Jensen (Ed.). *Application and Theory of Petri Nets 1992*, pp. 113-133. Lecture Notes in Computer Science, 616. Springer-Verlag, 1992.

[Cohen 89] G. Cohen et al. Algebraic Tools for the Performance Evaluation of Discrete Event Systems. *Proc. of the IEEE*, 77(1): 39-57. January 1989.

[Coolahan 83] J. E. Coolahan and N. Roussopoulos. Timing Requirements for Time Driven Systems Using Augmented Petri Nets. *IEEE Transactions on Software Engineering*, SE-9(9). September 1983.

[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.

[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.

[Dix 97] A. Dix. Challenges for Cooperative Work on the Web: An Analytical Approach. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 6(2-3): 135-156. 1997.

[Dourish 92] P. Dourish and V. Bellotti. Awareness and Coordination in Shared Workspaces. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW´92)*, pp. 107-114. 1992.

[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 91] C. A. Ellis, S. J. Gibbs and G. L. Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, 34(1): 38-58. January 1991.

[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. Springer-Verlag, 1993.

[Ellis 94] C. A. Ellis and J. Wainer. A Conceptual Model of Groupware. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW´94)*, pp. 79-88. 1994.

[Flores 88] F. Flores et al. Computer Systems and the Design of Organizational Interaction. *ACM Transactions on Office Information Systems*, 6(2): 153-172. April 1988.

[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.

[Gelernter 92] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2): 97-107. February 1992.

[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.

[Ghezzi 89] C. Ghezzi et al. A General Way to Put Time in Petri Nets. *Proc. of the SIGSOFT 5th Int. Workshop on Software Specification and Design*, pp. 60-67. 1989.

[Grosz 96] B. J. Grosz. Collaborative Systems. *AI Magazine*, 17(2): 67-85. Summer 1996.

[Grudin 94] J. Grudin. Groupware and Social Dynamics: Eight Challenges for Developers. *Communications of the ACM*, 37(1): 92-105. January 1994.

[Gutwin 96] C. Gutwin and S. Greenberg. Workspace Awareness for Groupware. *Proc. of the Conf. on Human Factors in Computing Systems (CHI'96)*, pp. 208-209. 1996.

[Gutwin 98] C. Gutwin and S. Greenberg. Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW´98)*, pp. 207-216. 1998.

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

[Hasselbring 00] W. Hasselbring. Information System Integration. *Communications of the ACM*, 43(6): 33-38. June 2000.

[Hayes 00] J. G. Hayes et al. Workflow Interoperability Standards for the Internet. *IEEE Internet Computing*, 4(3): 37-45. May-June 2000.

[Holliday 87] M. A. Holliday and M. K. Vernon. A Generalized Timed Petri Net Model for Performance Analysis. *IEEE Transactions on Software Engineering*, SE-13(12): 1297-1310. December 1987.

[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.

[Huber 90] P. Huber, K. Jensen and R. M. Shapiro. *Hierarchies in Coloured Petri Nets*. In G. Rozenberg (Ed.). *Advances in Petri Nets 1990*, pp. 313-341. Lecture Notes in Computer Science, 483. Springer-Verlag, 1990.

[Infocosm 00] Infocosm. *METEOR (Managing End-To-End OpeRations)*. Site comercial <http://infocosm.com/html/products.html>. Site acadêmico (LSDIS, Dept. of Computer Science, U. of Georgia) <http://lsdis.cs.uga.edu/proj/meteor/meteor.html>. 2000.

[Jablonski 96] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Publishing, 1996.

[Jennings 94] D. Jennings. *On the Definition and Desirability of Autonomous User Agents in CSCW*. In J. H. Connolly and E. A. Edmonds (Eds.). *CSCW and Artificial Intelligence*, pp. 161-173. Springer-Verlag, 1994.

[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.

[Kling 91] R. Kling. Cooperation, Coordination and Control in Computer-Supported Work. *Communications of the ACM*, 34(12): 83-88. December 1991.

[Kosoresow 98] A. P. Kosoresow and G. E. Kaiser. Using Agents to Enable Collaborative Work. *IEEE Internet Computing*, 2(4): 85-87. July-August 1998.

[Kreifelts 91] T. Kreifelts, F. Victor et al. *A design tool for autonomous group agents*. In J. M. Bowers and S. D. Benford (Eds.). *Studies in Computer Supported Cooperative Work*, pp. 131-144. North-Holland, 1991.

[Leadbetter 00] R. Leadbetter. *Hydra*. <http://www.pantheon.org/mythica/articles/h/hydra.html>. Consulta: Maio 2000.

[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.

[Ludwig 99] H. Ludwig and K. Whittingham. Virtual Enterprise Co-ordinator – Agreement-Driven Gateways for Cross-Oganisational Workflow Management. *Proc. of the Int. Conf. on Work Activities Coordination and Collaboration (WACC'99)*, pp. 29-38. 1999.

[Magalhães 98] 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. 357-370. 1990.

[Malone 94] T. W. Malone and K. Crowston. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1): 87-119. March 1994.

[Malone 95] T. W. Malone, K.-W. Lai and C. Fry. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *ACM Transactions on Information Systems*, 13(2): 177-205. April 1995.

[Markus 90] M. L. Markus and T. Connoly. Why CSCW Applications Fail: Problems in the Adoption of Interdependent Work Tools. *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.

[Merlin 76] P. M. Merlin and D. J. Farber. Recoverability of Communication Protocols – Implications of a Theoretical Study". *IEEE Transactions on Communications*, COM-24(9): 1036-1043. September 1976.

[Merz 97] M. Merz, B. Liberman and W. Lamersdorf. Using Mobile Agents to support Interorganizational Workflow Management. *International Journal on Applied Artificial Intelligence*, 11(6). September 1997.

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

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

[Nilsson 86] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Pubs, 1986.

[Palmer 94] J. D. Palmer and N. A. Fields. Guest Editors' Introduction – Computer-Supported Cooperative Work. *IEEE Computer*, 27(5): 15-17. May 1994.

[Peterson 77] J. L. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3): 223-252. September 1977.

[Petri 62] C. A. Petri. *Kommunikation mit Automaten*. Schriften des IIM Nr. 3, Bonn: Institute für Instrumentelle Mathematik. 1962.

[Petri 77] C. A. Petri. *Modelling as a Communication Discipline*. In H. Beilner and E. Gelenbe (Eds.). *Measuring, modelling and evaluating computer systems*. North Holland, 1977.

[Pinhanez 97] C. S. Pinhanez, K. Mase and A. Bobick. Interval Scripts: a Design Paradigm for Story-Based Interactive Systems. *Proc. of the Conf. on Human Factors in Computing Systems (CHI'97)*, pp. 287-294. 1997.

[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 00a] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments. Aceito para a publicação no *International Journal of Computer Systems Science & Engineering*. Special Issue on Flexible Workflow Technology Driving the Networked Economy. CRL Publishing. September 2000.

[Raposo 00b] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Mecanismos de Coordenação para Ambientes Colaborativos. *Anais do VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia (SBMIDIA'00)*, pp. 247-258. Natal, RN. Junho 2000.

[Raposo 00c] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach. *4th World Muticonference on Systemics, Cybernetics and Informatics (SCI'2000) Proceedings, Vol. I – Information Systems*, pp. 195-200. Orlando, EUA. Julho 2000.

[Raposo 00d] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Coordination Mechanisms for Collaborative Virtual Environments. *WRV'2000 Proc. – III Workshop on Virtual Reality*, pp. 277-278. Gramado, RS. Outubro 2000.

[Saracco 89] R. Saracco, J. R. W. Smith and R. Reed. *Telecommunications systems engineering using SDL*. North-Holland, 1989.

[Schmidt 91] K. Schmidt. Riding a Tiger, or Computer Supported Cooperative Work. *Proc. of the 2nd European Conf. on Computer-Supported Cooperative Work (ECSCW'91)*, pp. 1-16. 1991.

[Schmidt 92] K. Schmidt and L. J. Bannon. Taking CSCW Seriously – Supporting Articulation Work. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 1(1-2): 7-40. 1992.

[Schmidt 96] K. Schmidt and C. Simone. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 5(2-3): 155-200. 1996.

[Scrivener 94] S. A. R. Scrivener and S. Clark. *Introducing Computer-Supported Cooperative Work*. In S. A. R. Scrivener (Ed.). *Computer-Supported Cooperative Work – The multimedia and networking paradigm*, pp. 19-38. Unicom Seminars Ltd., 1994.

[Simone 99] C. Simone, G. Mark and D. Giubbilei. Interoperability as a Means of Articulation Work. *Proc. of the Int. Conf. on Work Activities Coordination and Collaboration (WACC'99)*, pp. 39-48. 1999.

[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.

[Stefik 87] M. Stefik et al. WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems*, 5(2): 147-168. April 1987.

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

[Suchman 94] L. Suchman. Do Categories Have Politics? *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 2: 177-190. 1994.

[SWAP 99] SWAP Working Group. <http://www.ics.uci.edu/~ietfswap/>. October 1999.

[Wahl 94] T. Wahl and K. Rothermel. Representing Time in Multimedia Systems. *Proc. IEEE 1st Int. Conf. on Multimedia Computing and Systems (ICMC'94)*, pp. 538-543. 1994.

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

[WfMC 96] Workflow Management Coalition. *Interface 4 – Interoperability Abstract Specification*. WFMC-TC-1012. <http://www.aiim.org/wfmc/standards/docs/if4-a.pdf> October 1996.

[WfMC 98] Workflow Management Coalition**.** *Interface 5 – Audit Data Specification – Version 1.1*, WFMC-TC-1015. <http://www.aiim.org/wfmc/standards/docs/if4v11b.pdf> September 1998.

[WfMC 99] Workflow Management Coalition**.** *Interface 1 – Process Definition Interchange Process Model – Version 1.1*, WFMC-TC-1016-P. <http://www.aiim.org/wfmc/standards/docs/if19910v11.pdf> October 1999.

[WfMC 00] Workflow Management Coalition. *Workflow Standard – Interoperability Wf-XML Binding*. WFMC-TC-1023, Version 1.0. <http://www.aiim.org/wfmc/standards/docs/tc1023v10.pdf> May 2000.

[Wilson 94] P. Wilson. *Introducing CSCW – What It Is and Why We Need It*. In S. A. R. Scrivener (Ed.). *Computer-Supported Cooperative Work – The multimedia and networking paradigm*, pp. 1-18. Unicom Seminars Ltd., 1994.

[Winograd 86] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, 1986.

[Winograd 94] T. Winograd. Categories, Disciplines, and Social Coordination. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 2: 191-197. 1994.

[Workflow 98] The Workflow Automation Corporation. *Workflow Automation – New Opportunities for Dramatic Results*. White paper. 1998.

[Yang 00] J. Yang and M. P. Papazoglou. Interoperation Support for Electronic Business. *Communications of the ACM*, 43(6): 39-47. June 2000.

[Zaidi 99] A. K. Zaidi. On Temporal Logic Programming Using Petri Nets. *IEEE Trans. on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 29(3): 245-254. May 1999.

Apêndice A – Redes de Petri

A.1. Modelagem

Rede de Petri (PN - *Petri Net*) [Petri 62] é uma ferramenta de modelagem aplicável a uma série de sistemas, especialmente àqueles 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 (*places*).
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. 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.

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 $P_o \in t\bullet$. $\bullet t$ 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.

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.

Na PN dada como exemplo na Figura A.1, 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*. Note-se que 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_i é dado por $M_{k+1} = M_k + A^T \times e_i$, onde e_i é um vetor coluna com 1 na posição *i* e 0 nas demais posições e $M_k = [q_1\ q_2\ ...\ q_m]^T$, onde q_j indica a quantidade de *tokens* no lugar P_j (estado atual a rede). A matriz $A = [a_{ij}]$ representa a topografia da rede e é chamada matriz de incidência da PN. Essa matriz tem dimensões $n \times m$ (*n* é o número de transições e *m* o número de lugares), e o elemento $a_{ij} = a_{ij}^+ - a_{ij}^-$, onde a_{ij}^+ indica quantos *tokens* o lugar P_j vai receber quando a transição t_i disparar (peso do arco partindo de t_i para P_j – ou $w(t_i, P_j)$) e a_{ij}^- indica quantos *tokens* ele vai perder ($w(P_j, t_i)$). Para o exemplo da Figura A.1, a matriz de incidência é:

$$A = \begin{bmatrix} -2 & 0 & 0 & 1 \\ 0 & -1 & -1 & 1 \end{bmatrix}$$

Na matriz acima, o elemento a_{11} = -2, por exemplo, indica que 2 *tokens* serão retirados de P_1 quando t_1 for disparada. O elemento a_{24} = 1 indica que 1 *token* será adicionado a P_4 quando t_2 for disparada. A determinação do próximo estado, disparando t_2 a partir do estado inicial (M_0) é feita de acordo com a equação:

$$M_1 = M_0 + A^T \times e_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 & 0 \\ 0 & -1 \\ 0 & -1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

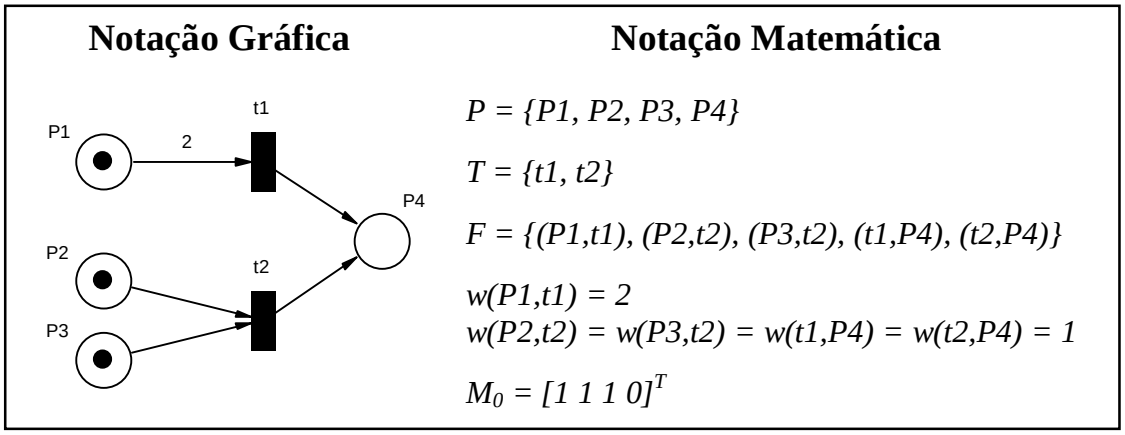

Figura A.1: Notação gráfica e notação matemática de PNs.

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

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 (na Figura A.2 o arco ligando P_1 a t_1 é um arco inibidor; nesse caso a transição está habilitada porque não há *tokens* em P_1).

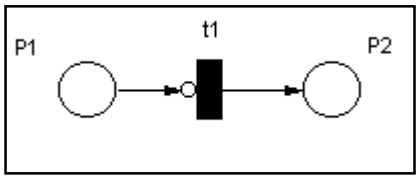

Figura A.2: Arco inibidor.

A noção de tempo em PNs é importante em algumas situações como na avaliação de desempenho dos sistemas modelados. O modelo básico de PN, no entanto, 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. Por esta razão, foram desenvolvidos vários métodos para a inclusão de tempo em PNs, tanto de forma estocástica como determinística. Nas PNs estocásticas [Bause 96] são estabelecidas funções de probabilidade para o tempo de disparo de uma transição. Neste tipo de PN, a transição dispara algum tempo depois de habilitada, tempo este determinado pela função de probabilidade associada a ela. PNs estocásticas são adequadas para a análise de desempenho dos sistemas modelados [Molloy 82], [Marsan 84], [Holliday 87], enquanto as determinísticas são adequadas para sistemas dependentes de limites de tempo (sistemas de tempo real, por exemplo) [Ghezzi 89].

As técnicas para a representação do tempo em PNs também podem ser separadas em duas grandes classes: as que incorporam tempo aos lugares e as que o incorporam às transições. No primeiro caso (por exemplo, [Coolahan 83], [Cohen 89]) é estabelecido um tempo de espera para o *token* em cada lugar. Uma vez que um *token* chega a um lugar, ele só pode contribuir para a habilitação de alguma transição após decorrido esse tempo de espera. A outra classe de técnicas associa tempo ao disparo das transições. É possível associar a uma transição um intervalo de tempo $[t_{min}, t_{max}]$, onde t_{min} e t_{max} representam, respectivamente, o tempo mínimo e o máximo que deve decorrer entre a habilitação da transição e seu disparo [Merlin 76]. Os *timeouts* dos modelos apresentados se enquadram nesta categoria, no caso particular em que $t_{min} = t_{max}$, definindo um tempo exato para o disparo após sua habilitação. Uma outra técnica para associar tempo às transições consiste em associar uma duração Δ ao disparo da transição [Ramamoorthy 80], [Holliday 87]. Neste caso, os *tokens* são retirados dos lugares de entrada no início do disparo da transição e, somente após transcorrido o intervalo Δ, são entregues aos lugares de saída. Este tipo de disparo não-instantâneo de transições é também chamado de disparo com reserva de *tokens*, e foi usado nos modelos para encapsular a execução real das tarefas (que não é instantânea) no nível de coordenação (são as transições marcadas com a letra "R" nos modelos).

PNs de alto nível (*high-level PNs*), de acordo com Murata [Murata 89], 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. Os modelos apresentados em PNs de alto nível seguem a definição de Murata para este tipo de rede, explicada a seguir.

No exemplo da Figura A.3, 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*.

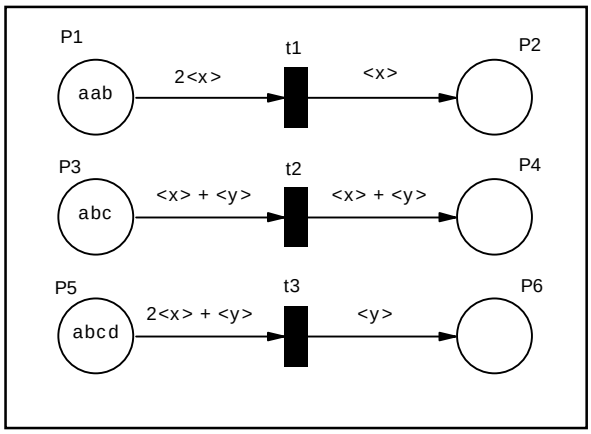

Figura A.3: Exemplo de PN de alto nível.

As PNs de alto nível podem ser decompostas em PNs simples se o número de cores for finito. Para isso, basta decompor cada lugar *P* da rede em um conjunto de lugares, cada um relativo a uma cor de *token* que *P* pode receber e decompor cada transição *t* em um conjunto de transições, cada uma representando uma das possíveis maneiras de *t* disparar [Murata 89]. Justamente por serem "composições estruturadas" de PNs simples, as redes de alto nível são importantes para a redução do problema da explosão de estados, típico de PNs convencionais. A identificação dos *tokens* e a possibilidade de utilizar variáveis e expressões nos arcos podem simplificar significativamente modelos de sistemas mais complexos.

A.2. Análise

Além das importantes características de modelagem, que permitem a simulação dos sistemas modelados, as PNs também oferecem importantes ferramentas de análise do sistema modelado.

A eficiência de uma ferramenta de modelagem de sistemas depende de dois fatores antagônicos: poder de modelagem e poder de decisão [Peterson 77]. O primeiro se refere à capacidade da ferramenta em modelar corretamente os sistemas, e o segundo se refere à capacidade de analisar diferentes versões do sistema e de determinar suas propriedades. Em geral, à medida que se aumenta o poder de modelagem, se diminui o poder de decisão. PNs, genericamente falando, apresentam um compromisso entre esses dois fatores. As PNs simples tendem mais para o poder de modelagem, enquanto modelos mais específicos (as extensões de PNs) são desenvolvidos para aumentar o poder de decisão, em detrimento do poder de modelagem.

PNs oferecem a possibilidade de analisar várias propriedades e problemas associados aos sistemas modelados. Dois tipos de propriedades podem ser estudadas com o modelo em PN: as estruturais e as comportamentais [Murata 89]. As propriedades estruturais dependem apenas da topologia da rede, sendo independentes da marcação inicial M_0. Estas propriedades podem ser obtidas a partir da matriz de incidência da rede. As propriedades

comportamentais, por sua vez, são dependentes do estado inicial da PN e servem para analisar o comportamento dinâmico da mesma.

Uma outra classificação das possíveis análises em PNs, as separam em três categorias: análise de verificação, de validação e de desempenho [van der Aalst 98]. Esse tipo de classificação foi o adotado no texto.

A análise de verificação é realizada para descobrir se a rede apresenta *deadlocks*, se atinge algum estado não permitido, se há transições mortas, etc. A análise de verificação é baseada nas propriedades comportamentais 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 A.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 $M_2 = [\,1\ 0\ 0\ 1\,]^T$.

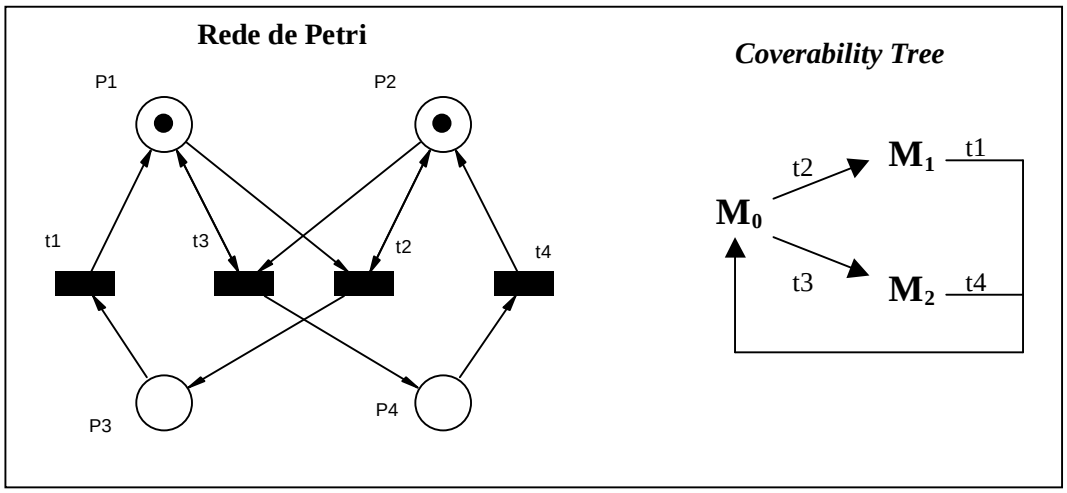

Figura A.4: Exemplo de PN e *coverability tree*.

Formalmente [Murata 89], um estado M_d é alcançável a partir de M_0 se

$$B_f \Delta M = 0$$

onde $\Delta M = M_d - M_0$ e B_f é uma matriz de dimensões $(m\text{-}r) \times m$, onde m é o número de lugares da rede e r é o posto (*rank*) da matriz de incidência A. B_f é definida como

$$B_f = [I_\mu : -A_{11}^T (A_{12}^T)^{-1}]$$

onde I_μ é a matriz identidade de ordem $\mu = m - r$ e A_{11} e A_{12} são partições da matriz A na seguinte forma

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} \vdots\ r \\ \vdots\ n\text{-}r \end{matrix}$$

(A_{11}: $m\text{-}r$ colunas; A_{12}: r colunas)

Liveness: há algum estado ou seqüência de estados que não será mais alcançado, indicando um possível *deadlock*? A partir da equação da *reachability*, é possível deduzir que um estado M_d não é alcançável se $B_f \Delta M \neq 0$.

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? Em outros termos, 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 o conjunto de possíveis seqüências de disparos a partir de qualquer marcação M e $\sigma(t_i)$ o número de vezes que a transição t_i dispara em uma seqüência $\sigma \in \Sigma$, a distância síncrona entre as transições t_i e t_j é dada por $d_{i,j} = max_\sigma \mid \sigma(t_i) - \sigma(t_j) \mid$, $\sigma \in \Sigma$. No exemplo da Figura A.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 esses pares de transições estão associados a eventos independentes.

O segundo tipo de análise, a de validação, é feita para garantir que a rede esteja corretamente definida e corresponda com exatidão ao sistema modelado. Os testes são feitos por meio de simulação iterativa de situações para verificar se a rede funciona como esperado.

A análise de desempenho, por sua vez, está associada às PNs com tempo. Ela 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 [Magalhães 98] – ver Apêndice D, cadeias de Markov [Molloy 82] e outras técnicas [Ramamoorthy 80], [Marsan 84], [Holliday 87].

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 e coordenação de ambientes colaborativos. Com o uso de PNs é possível prever e testar o comportamento desses ambientes antes de sua implementação.

Apêndice B – *Workflows* e Redes de Petri

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 coordenar o fluxo de trabalho em um ambiente de trabalho distribuído" [Bull 92].

O sistema de gerenciamento de *workflow* (WFMS – *workflow management system*) é o responsável pela coordenação do *workflow*. Em última instância, seu objetivo é fazer com que as atividades sejam executadas pelas pessoas certas nos momentos certos [van der Aalst 98]. Uma definição mais formal diz que um WFMS é um sistema para "o suporte às tarefas distribuídas em uma companhia. O WFMS é baseado em estruturas de controle (i.e., *workflows*), que são usadas para disparar, coordenar e supervisionar as atividades" [Jablonski 96].

Redes de Petri e suas variações podem ser usadas para a modelagem dos WFMS [Ellis 93], [van der Aalst 98]. A vantagem das PNs é que, graças ao suporte à modelagem, simulação e análise, elas permitem verificar a correção e validar a eficiência do *workflow* antes de sua implementação. A possibilidade de prever os resultados de uma seqüência de atividades antes de sua execução tem sido considerada cada vez mais importante nas organizações. Algumas ferramentas atuais de *workflow* têm oferecido recursos para isso, embora usem abordagens menos formais que as PNs. O SIMPROCESS [CACI 00] é um exemplo de ferramenta comercial que usa técnicas de simulação e análise para avaliar alternativas antes de implementá-las.

B.1. Estruturas Básicas

Redes de Petri e suas variações são bastante usadas para a modelagem de sistemas de *workflow*. Um exemplo são as ICNs (*Information Control Nets*), uma variação de PNs para modelagem de *workflows* [Ellis 93]. Outro exemplo de PNs usadas para a modelagem de *workflows* são as *Workflow Nets* (WF-Nets), uma classe de PNs adequada para a representação, validação e verificação de conjuntos de tarefas interdependentes (ver Seção B.2) [van der Aalst 94].

Independentemente do modelo específico utilizado (ICN, WF-Net, etc.), os *workflows* apresentam alguns tipos de conexões básicas e estruturas lógicas que podem ser 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 esses 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 B.1 mostra o modelo de PNs para as conexões básicas de *workflow*.

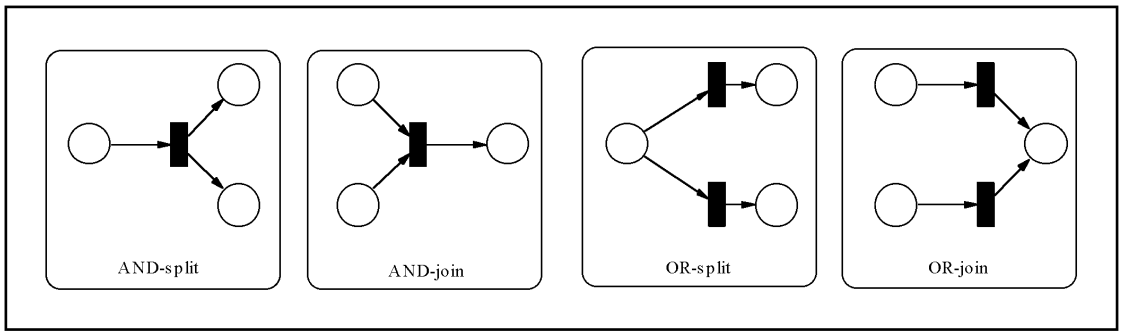

Figura B.1: Conexões básicas de *workflow*.

Com relação ao *OR-split*, o modelo apresentado na Figura B.1 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 B.2.

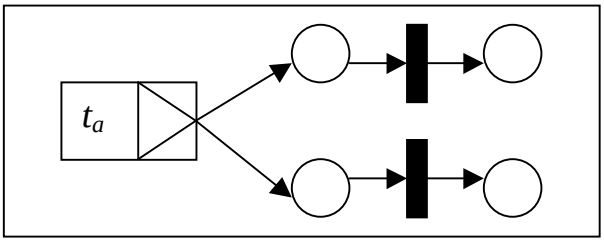

Figura B.2: *OR-split* explícito

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*. Ele representa trilhas de tarefas executadas em paralelo. O condicional exclusivo é um trecho iniciado por um *OR-split* e finalizado por um *OR-join*. O condicional exclusivo 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. Ele é modelado por uma combinação das quatro conexões básicas. A iteração é a possibilidade de repetição de uma seqüência de tarefas. Ela é modelada por um *OR-split* com uma saída retornando a algum lugar da PN. Estas quatro estruturas lógicas são mostradas na Figura B.3.

B.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]:

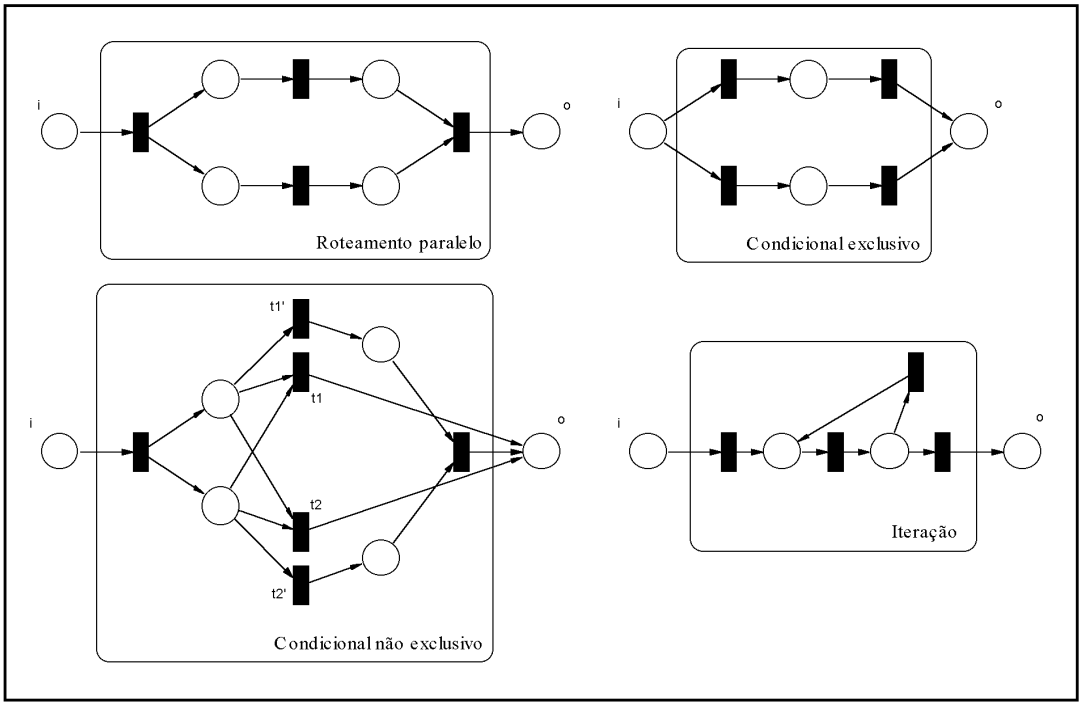

Figura B.3: Estruturas lógicas típicas de *workflows*.

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 B.1.

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 B.4 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, o uso de PNs permite o encapsulamento de detalhes, pois qualquer uma das tarefas poderia estar representando uma outra WF-Net (sub-procedimento).

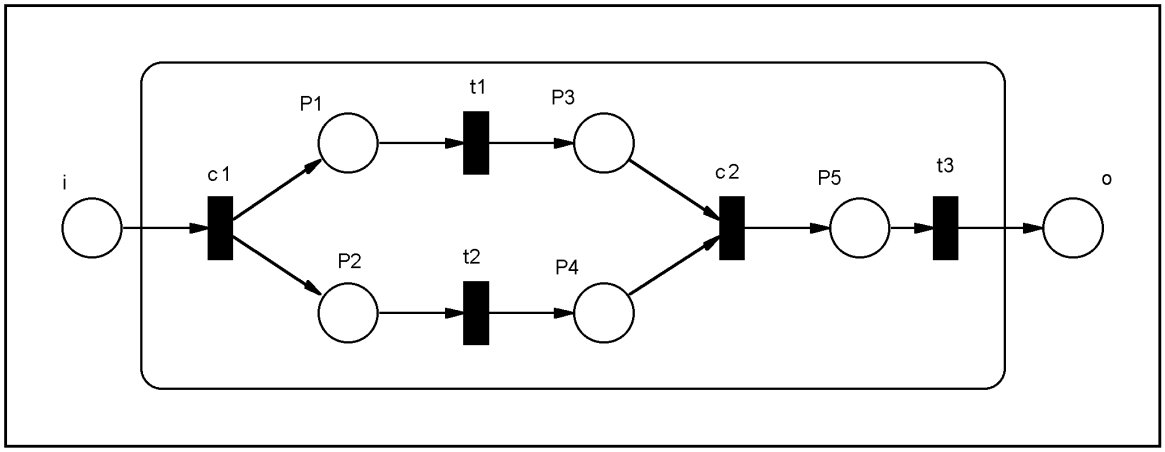

Figura B.4: Exemplo de WF-Net

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.

O modelo utilizado neste trabalho é semelhante às WF-Nets, com as seguintes diferenças básicas:

1. No modelo aqui utilizado não há rigor quanto à necessidade de cada participante do ambiente colaborativo ter seu comportamento modelado por uma WF-Net formal. Em outras palavras, as PNs no nível de *workflow* não precisam forçosamente atender ao requisito *(ii)* acima, pois o comportamento projetado para um participante pode aceitar tarefas "pendentes", o que seria inaceitável em *workflows*.
2. A tarefa em *workflows* é necessariamente atômica. No modelo utilizado, uma tarefa pode encapsular uma série de sub-tarefas que só serão expandidas no nível de especificação de tarefas.

Apêndice C – Glossário

Análise de desempenho – Realizada em PNs para avaliar a capacidade do sistema modelado atingir certos requisitos, tais como tempo médio de espera, uso de recursos, *throughput times*, etc.

Análise de validação – Realizada em PNs para garantir que as mesmas estejam corretamente definidas e correspondam com exatidão aos sistemas modelados. Os testes são feitos com simulações iterativas.

Análise de verificação – Realizada em PNs para descobrir se a rede apresenta *deadlocks*, se atinge algum estado não permitido, se há transições mortas, etc.

AND-join – Conexão de PNs que finaliza um roteamento paralelo, composta por uma transição com mais de um lugar de entrada.

AND-split – Conexão de PNs que inicia um roteamento paralelo, composta por uma transição com mais de um lugar de saída.

Awareness – Percepção. Em trabalho colaborativo, é como o entendimento das atividades dos outros provê um contexto para sua própria atividade.

BNF – *Backus Normal Form*. Notação para escrever gramáticas.

Coordenação – Ato de gerenciar interdependências entre atividades.

Coverability tree – Árvore de estados que uma PN pode atingir.

CSCW – *Computer Supported Cooperative Work*. Área de estudo interessada no trabalho em grupo com o auxílio de computadores.

CVE – Ambiente Virtual Colaborativo (*Collaborative Virtual Environment*). Simulação em tempo real de um mundo real ou imaginário onde os usuários estão simultaneamente presentes e podem navegar e interagir com outros usuários e com objetos do mundo.

Deadlock – Em PNs significa atingir um estado final da rede.

Dependências de gerenciamento de recurso – Interdependências relativas ao acesso seqüencial ou simultâneo a um mesmo recurso.

Dependências temporais – Interdependências que estabelecem a ordem de execução das tarefas.

EAI – *External Authoring Interface*. Interface para permitir que uma cena VRML interaja com um programa Java na forma de *applet*.

EUP – *End User Programming*. Conceito de programação levado ao usuário final da aplicação.

Explosão de estados – Problema característico das PNs, que tendem a se tornar muito grandes para análise mesmo em sistemas de tamanhos modestos.

Groupware – Sistemas para apoio ao trabalho em grupo.

ICN – *Information Control Net*. Variação de PN para modelagem de *workflows*.

Job token – *Token* que flui pela PN, representando o estado de um caso (*job*).

Mecanismo de coordenação – Dispositivo de software que interage com a aplicação para dar suporte ao trabalho de articulação.

Nível de workflow – Nível de abstração mais alto do modelo de coordenação apresentado.

Nível de coordenação – Nível de abstração intermediário do modelo de coordenação apresentado. É nesse nível que é realizada a simulação das PNs.

Nível de especificação das tarefas – Nível de abstração mais baixo do modelo de coordenação apresentado.

OR-join – Conexão de PNs que finaliza um roteamento condicional, composta de por um lugar com mais de uma transição de entrada.

OR-split – Conexão de PNs que inicia um roteamento condicional, composta de por um lugar com mais de uma transição de saída (situação de conflito).

PN – Rede de Petri (*Petri Net*).

PNs simples – PNs que seguem o modelo original.

PNs de alto nível – PNs que englobam, de uma maneira geral, as PNs de pedicado/transição, as coloridas e as de tokens individuais [Murata 89].

Protocolo social – Protocolo de coordenação culturalmente estabelecido, onde não são necessários mecanismos de coordenação explícitos.

Recurso – Agente que realiza a tarefa ou qualquer artefato necessário à realização da mesma.

Timeout – Nesse contexto, significa uma transição de uma PN com tempo, que será disparada após um certo tempo de habilitada.

TimeoutA – Nos mecanismos de coordenação apresentados, é um tipo de *timeout* que leva à execução de uma tarefa alternativa.

TimeoutB – Nos mecanismos de coordenação apresentados, é um tipo de *timeout* que retorna a tarefa à sua condição inicial (*token* nos lugares de entrada).

Trabalho colaborativo – Múltiplos indivíduos trabalhando de forma planejada em uma mesma atividade ou em atividades conectadas.

Trabalho de articulação – Esforço extra para garantir que a colaboração seja obtida a partir da soma dos trabalhos individuais.

VRML – *Virtual Reality Modeling Language*. Linguagem para a descrição e transmissão de mundos virtuais 3D pela Web.

WF-Net – *Workflow Net*. Classe de PN adequada para a representação, validação e verificação de conjuntos de tarefas interdependentes.

WFMS – *Workflow Management System*. Sistema responsável pela coordenação de um *workflow*.

Workflow – Tipo particular de *groupware* para auxiliar grupos de pessoas na execução de procedimentos de trabalho e que possui o conhecimento de como o trabalho normalmente flui em uma organização.

Workflow interorganizacional – *Workflow* envolvendo várias organizações.

Apêndice D – Artigos Publicados

Neste apêndice são mostrados cinco artigos publicados pelo autor e que estão diretamente relacionados ao conteúdo apresentado neste trabalho.

- **[Raposo 00a]** A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments. Aceito para a publicação no *International Journal of Computer Systems Science & Engineering*. Special Issue on Flexible Workflow Technology Driving the Networked Economy. CRL Publishing. September 2000.

- **[Raposo 00b]** A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Mecanismos de Coordenação para Ambientes Colaborativos. *Anais do VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia (SBMIDIA'00)*, pp. 247-258. Natal, RN. Junho 2000.

- **[Raposo 00c]** A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach. *4th World Muticonference on Systemics, Cybernetics and Informatics (SCI'2000) Proceedings, Vol. I – Information Systems*, pp. 195-200. Orlando, EUA. Julho 2000.

- **[Raposo 00d]** A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Coordination Mechanisms for Collaborative Virtual Environments. *WRV'2000 Proceedings – III Workshop on Virtual Reality*, pp. 277-278. Gramado, RS. Outubro 2000.

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

Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments

Alberto B. Raposo, Léo P. Magalhães and **Ivan L. M. Ricarte**
State University of Campinas (UNICAMP)
School of Electrical and Computer Engineering (FEEC)
Department of Computer Engineering and Industrial Automation (DCA)
CP 6101 – 13083-970 – Campinas, SP, Brazil
{alberto, leopini, ricarte}@dca.fee.unicamp.br

Abstract. The coordination of cooperative workflows occurs in parallel to the definition of a common communications infrastructure among organizations. In this paper, we present a library of coordination mechanisms modeled with Petri Nets. These mechanisms specify and control the interaction between workflow processes. The separation between activities and dependencies, managed by the coordination mechanisms, allows the reuse of these mechanisms in other environments and also the use of different coordination policies in the same environment.

1. Introduction

The increasing globalization of the world economy offers promising opportunities for the field of interorganizational workflows, which cross a single organization boundary, being composed of several organizations working cooperatively. However, the necessity of cooperation raises many problems which we can separate into two main classes, those related to the definition of a joint communications infrastructure and those related to the coordination of cooperative workflows.

The first class of problems deals with the integration of heterogeneous software environments in a common communications infrastructure. These problems can be roughly summarized by the difficulties of establishing an "interoperability contract" [1] among the cooperative organizations. This "contract" must establish, among other things, which workflow engines within an organization are capable of interoperating with which engines within other organizations, the transport technology, the communication protocol, security policies and exception handling. The Workflow Management Coalition (WfMC) is currently working on standards addressing these issues [2].

The other class of problems – those related to the coordination of cooperative workflows – appears in a higher abstraction level and, to our knowledge, it has not been addressed by the WfMC yet. Here, it is assumed that the communications environment is well-defined by the partners and the major concerns are related to the coordination of interdependent activities. Since this kind of problem is the main focus of the paper, we purposely use the term "multi-workflow environment" instead of the standard "interorganizational workflow".

Coordination can be defined as "the act of managing interdependencies between activities performed to achieve a goal" [3]. The coordination is highly dynamic, because it

needs to be "renegotiated" almost continuously during a collaborative effort. The great challenge in proposing coordination mechanisms to control a cooperative activity is to achieve the flexibility demanded by the dynamism of the interaction among the partners [4]. In other words, coordination policies vary according to the cooperation instance, or even during the evolution of the same instance. Therefore, it is essential that coordination mechanisms be flexible enough to handle these variations. It is not reasonable to think that organizations will spend time and money to establish a common communications infrastructure with their partners if they are not sure about the effectiveness of the coordination structure of their cooperative workflows.

This paper presents a library of coordination mechanisms that can be reused in several workflow environments. The main idea is to separate the coordination mechanisms from the tasks that constitute the workflow. The separation between activities (tasks) and dependencies (managed by the coordination mechanisms) allows the use of different coordination policies in the same environment by changing only the coordination mechanisms. Moreover, the same mechanisms can be used in different single- or multi-workflow environments.

To model the proposed coordination mechanisms, we use an approach based on Petri Nets (PNs). The graphical representation of PNs, besides being easy to understand, enables detail encapsulation, offering a very clear hierarchical description model, which is adequate to model the different coordination levels. Furthermore, PNs offer a strong theoretical support for the analysis of an environment's behavior and supplementary simulation techniques. With the PN based model, it is possible to anticipate and test the behavior of multi-workflow environments before their implementation.

This paper is structured as follows. The next section briefly introduces PNs. Section 3 makes an overview of previous works regarding the use of PNs in coordination, workflow applications and interorganizational workflows. The library of coordination mechanisms is presented in Section 4, and an example of use is shown in Section 5. The last section presents the conclusions and next issues.

2. Petri Nets Fundamentals

PNs [5], [6] are a modeling tool applicable to a variety of fields and systems, specially suitable for systems with concurrent events. Formally, a PN can be defined as a 5-tuple (P, T, F, w, M_0), where: $P = \{P_1, ..., P_m\}$ is a finite set of places; $T = \{t_1, ..., t_n\}$ is a finite set of transitions; $F \subseteq (P \times T) \cup (P \times T)$ is a set of arcs; $w: F \rightarrow \{1, 2, ...\}$ is a weight function; $M_0: P \rightarrow \{0, 1, 2, ...\}$ is the initial marking; with $(P \cap T) = \varnothing$ and $(P \cup T) \neq \varnothing$.

In a PN model, states are associated to places and marks (also called tokens), and events to transitions. A transition t is said to be enabled if each input place $P_i \in \bullet t$ is marked with at least $w(P_i, t)$ tokens, where $w(P_i, t)$ is the weight of the arc between P_i and t. Once enabled, a transition will fire when its associated event occurs. Firing transition t, $w(P_i, t)$ tokens are removed from each input place P_i and $w(t, P_o)$ tokens are added to each output place $P_o \in t\bullet$. Here, $\bullet t$ and $t\bullet$ means, respectively, the set of input and output places of transition t.

A very useful notation for PNs is the graphical notation (Figure 1) which will be used in the examples throughout this paper. In this notation, circles represent places, rectangles

represent transitions, dots represent tokens and arrows represent the arcs, with weights above. By definition, an unlabeled arc has weight 1.

In the PN of Figure 1, only transition *t2* is enabled; *t1* is not enabled because it would require two tokens in *P1* to fire, since *w(P1,t1)=2*. When *t2* is fired, the tokens in *P2* and *P3* are removed and *P4* receives one token. Note that the number of tokens in a PN is not necessarily conserved.

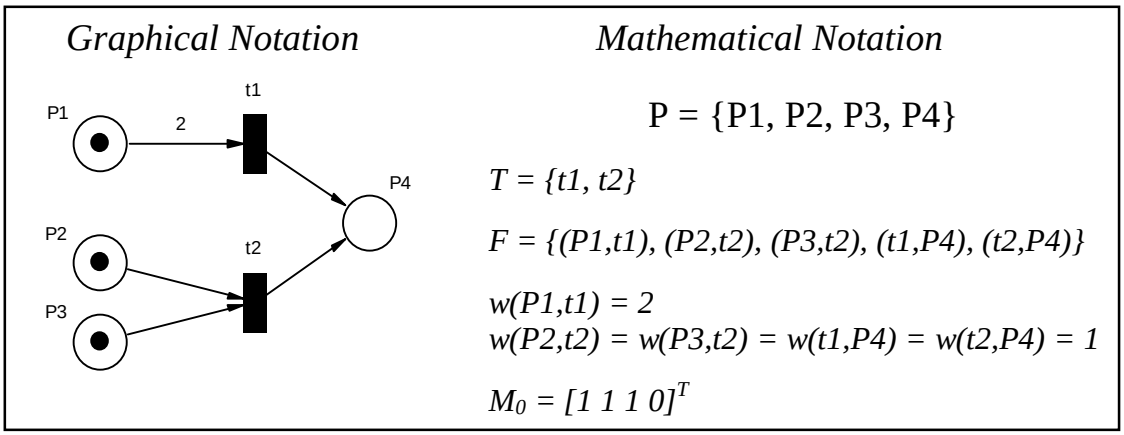

Figure 1: PNs graphical and mathematical notations.

In addition to the basic PN model, several extensions appears in the literature [6]. In this paper we use two extensions: inhibitor arcs and nets with time. An inhibitor arc connects a place *P* with a transition *t* and enables *t* only if *P* has no tokens. In the graphical notation, inhibitor arcs are represented with a circle on the edge. The basic PN model does not consider the notion of time. One way to include this notion in the model is to establish a waiting time for the tokens in a place before they enable the output transitions [7]. Time can also be associated with transitions firing. In this case, tokens do not stay in input places waiting for the firing, but they are removed from that places and some time later (firing time) are added to the output places. This kind of non-instantaneous firing is called firing with token reservation.

Besides the modeling capabilities of PNs, their support for analysis is very important and useful on the verification of workflows. "The abundance of available analysis techniques shows that Petri nets can be seen as a solver independent medium between the design of the workflow process definition and the analysis of the workflow" [8]. These analysis techniques are based on the properties of the mathematical model. Some of these properties are:

Reachability: is there a sequence of firing that reaches a given state? The coverability tree, that offers a vision of the complete sequence of transitions and states in a PN, can be used to verify this property.

Liveness: is there any state or sequence of states which will not be reached anymore, indicating a possible deadlock?

Reversibility: is it possible to return to a defined initial state M_0?

Boundness: will a place be overloaded? A PN is defined as *k*-bounded if the number of tokens in each place does not exceed *k*.

Persistence: is the firing of any pair of enabled transitions interdependent, i.e., the firing of one will disable the other? A pair of transitions with interdependent firings indicates a conflict, because only one of them will be fired (logical OR).

Three kinds of analysis can be applied to a PN model; verification, validation and performance analysis [8]. The verification analysis is used to guarantee that the net is correctly defined. It is verified whether the net has deadlocks, whether there are dead transitions, whether it reaches any undesired state, among others. In the validation analysis, it is checked whether the model works as expected. Tests are made via iterative simulations of fictitious cases to ensure that the model treats them correctly. Finally, the performance analysis evaluates the capacity of the system to achieve requisites such as average waiting time, throughput times, resource use, and so on.

Summarizing, PNs provide interesting modeling characteristics, a strong theoretical support for analysis and also a number of simulation techniques, which make them a powerful tool for workflow applications. They enable designers to preview and test the workflow behavior before implementing it.

3. Related Work

In 1985 Anatol Holt used PNs to coordinate activities in computer supported work environments [9]. He proposed a new interpretation for PNs, connecting their formal structure with the "natural structure" of human (or computational) work. Moreover, he also identified some essential aspects of coordination technology, which at that time was very promising for the creation of electronic work environments. Among these aspects, the most important was related to the flexibility of coordination mechanisms. As stated by Holt, to be useful, the desired patterns of task coordination "must be done in a flexible yet well-integrated manner, with plenty of leeway for the unpredictability of real life" [9].

Holt's work resulted in the creation of Diplan, a formal graphical language to the planning of activities involving multiple collaborative agents [10]. This language is based on Predicate/Transition Nets [11], with some alterations, such as the specification of a time for state changes (non-instantaneous transitions) and the explicit reference to the human role in the tasks' execution. The CHAOS (Commitment Handling Active Office System) is another system for the coordination of activities in office automation based on PNs [12].

PNs also constitute the basis of Trellis, a model for prototyping interaction protocols in collaborative systems [13]. The generated protocol defines how a central server must process clients' requests. Trellis uses colored PNs [14], in which tokens have a type (color) and carry information. The notion of time appears in the form of delays and timeouts to the firing of transitions. The functionality of Trellis is different from that of Diplan because the former is not restricted to the coordination of activities, creating "hyperprograms" that join hypermedia navigation to collaboration support.

The ideas related to the automation and coordination of tasks led up to the development of workflow systems [15], [16], defined as "a particular kind of groupware intended to assist groups of people in executing work procedures; they contain knowledge of how work normally flows through the organization" [17]. PNs and their extensions are also used for the modeling of this kind of system. There are many papers in the literature presenting the mapping of workflow concepts into PNs (e.g., [17], [18]). Basically, tasks are represented by transitions and jobs are represented by tokens flowing through the net. The state of a job

is given by the marking of the PN at a given instant. Control activities are also represented by transitions, which are used in basic workflow connections such as joins and splits. Examples of variations of PNs used in workflow applications are the ICNs (Information Control Nets) [17] and the Workflow Nets (WF-Nets), a class of PNs appropriate to the representation, validation and verification of sets of interdependent tasks [8].

PNs are not the only way to evaluate workflow systems before implementing them. Current workflow tools offer approaches less formal than PNs for analysis, which have a larger penetration in the business world [19]. SIMPROCESS [20] is an example of a commercial tool that uses workflow based simulation and analysis to evaluate alternatives prior to implementing them.

Recently, interorganizational workflows have attracted a lot of attention from developers. PRODNET II [19] is an example of a project aiming to develop an open and flexible support infrastructure suited to the needs of small and medium enterprises. The PRODNET II architecture implements a clear separation between support services and control mechanisms, and adopts a workflow based coordination approach.

METEOR (Managing End-To-End OpeRations) [21] is a well-succeeded academic workflow management system that generated a commercial product. METEOR specifies human and automated tasks and intertask dependencies, generating automatic code from graphical specification. Communication channels between heterogeneous and distributed workflow engines are provided by means of a CORBA and Java Web-based implementation.

PNs have also been used in interorganizational workflows. An example is the virtual machine (a PN) that controls the behavior of mobile agents in COSM (Common Open Service Market) [22]. COSM uses mobile agents to support the management of interorganizational workflows, an approach specially suitable for situations where partners do not intend to tighten their cooperation (e.g., in order to make a price survey with several suppliers and get the best offer). This is not the case covered by this paper; here we assume that partners have a well-established set of tasks in the cooperation.

To our knowledge, the work more closely related to ours is that of van der Aalst [23], who realized that it is necessary to address the contents of the coordination structure of interorganizational workflows. As stated by him “the semantics of the constructs needed to model interorganizational workflows should be defined before solving the technical issues (mainly syntactical)” [23]. He expanded his previous work on WF-Nets [8] in order to model and analyse interorganizational workflows.

Although being motivated by similar concerns and making use of the same modeling tool (PNs), there are differences between his work and ours. The main goal of van der Aalst’s work is to formally verify the correctness of interorganizational workflows and their consistency with the message sequence charts used to specify the interaction between workflow processes. Our main goal, on the other hand, is to provide a set of PN-modeled coordination mechanisms that can be used to specify such interaction in several workflow environments. The correctness and consistency of the generated workflow can be verified by means of available PN analysis and simulation tools. In order to facilitate this task, we also have developed a prototype tool capable of “inserting” those mechanisms between interdependent tasks of a PN modeled with a specific software tool.

4. A Library of Coordination Mechanisms

As already stated, this work intends to provide mechanisms to manage interdependencies between tasks in a multi-workflow environment. In order to achieve this goal, we propose a library of PN-based primitives to model coordination mechanisms for several possible interdependencies. Each coordination mechanism ensures that the associated dependency will not be violated. The idea is that workflow designers be concerned only with the definition of tasks and interdependencies among them, and not with the management of those dependencies, since mechanisms to manage them are provided by the library.

In the proposed schema, an environment is modeled in two distinct levels, *workflow* and *coordination*. In the *workflow* level, each workflow is modeled separately by a PN, in which tasks are represented by transitions, and that may include "traditional" workflow connections (splits and joins) and routings (parallel routing, conditional routing, etc). Also in this level, it is necessary to establish the interdependencies between tasks in different workflows, or even in the same workflow. The *coordination* level is built under the workflow level by the expansion of interdependent tasks according to a model defined in [18] and the insertion of corresponding coordination mechanisms between them.

In the passage from the workflow to the coordination level, each task (a transition in the workflow level) which has a dependency with another is hierarchically expanded in a system with five transitions (*ta*, *tb*, *ti*, *tf* and *tc*) and four places (*P1*, *P2*, *P3* and *P4*), as proposed by van der Aalst et al [18]. As shown in Figure 2, attached to each expanded task there are also five places that represent the interaction with the resource manager and the agent that executes the task. The places *request_resource*, *assigned_resource* and *release_resource* connect the task with the resource manager. The places *start_task* and *finish_task* connect the task with the agent that performs it, respectively indicating the beginning and the end of the task execution.

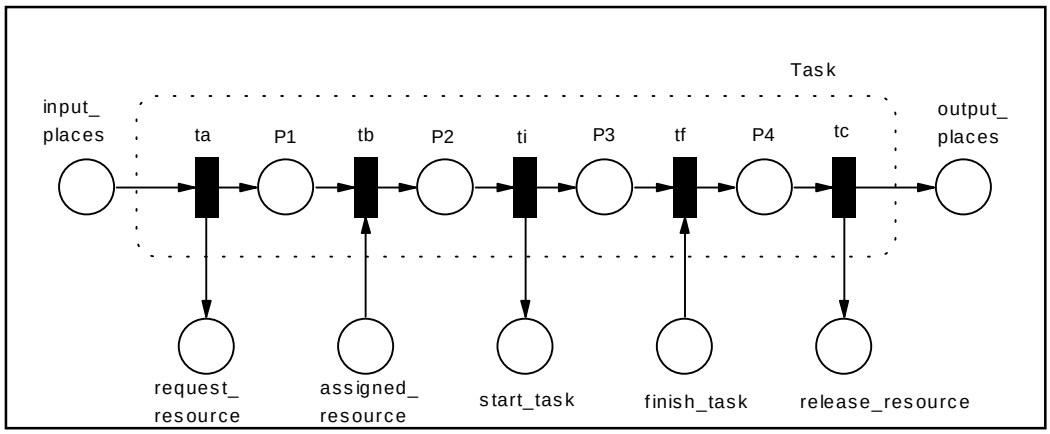

Figure 2: An expanded task in the coordination level.

Based on the model of Figure 2, it is possible to conclude that a task can be connected with two sub-nets: a resource manager (which has *request_resource* and *release_resource* as input places and *assigned_resource* as output place), and another representing the logistics of the task (which has *start_task* as input place and *finish_task* as output place).

The library recognizes two general classes of interdependencies: *temporal* and *resource management* dependencies. The former is related to the logistics sub-net cited above, while the latter is related to the resource manager sub-net.

The library construction enables to build the coordination level from the workflow level, because once the interdependencies are defined, the expansion of tasks according to the model of Figure 2 and the insertion of coordination mechanisms can be automated.

In the following sections we present the set of mechanisms of the library.

4.1. Temporal Dependencies

Temporal dependencies establish the execution order of "disconnected tasks" (e.g., tasks belonging to different workflows). By the use of coordination mechanisms proposed for temporal dependencies, it is possible to define if a task is executed before, after, or during another task.

In order to define the possible temporal dependencies between tasks, we refer to a classic temporal logic paper by J. F. Allen [24]. In this paper, Allen states that there is a set of primitive and mutually exclusive relations that can be applied to time intervals:

[x1, x2] *equals* [y1, y2] iff x1 = y1 and x2 = y2. (X and Y are the same time interval.)
[x1, x2] *starts* [y1, y2] iff x1 = y1 and x2 < y2. (X and Y start together, but X finishes before Y.)
[x1, x2] *finishes* [y1, y2] iff x2 = y2 and x1 > y1. (X and Y finish together, but X starts after Y.)
[x1, x2] *before* [y1, y2] iff x2 < y1. (X happens before Y, and they do not overlap.)
[x1, x2] *meets* (y1, y2] iff x2 = y1. (X happens before Y, which starts immediately with the end of X.)
[x1, x2] *overlaps* [y1, y2] iff x1 < y1 < x2 < y2. (X starts before Y, which starts before the end of X.)
[x1, x2] *during* [y1, y2] iff x1 > y1 and x2 < y2. (X is totally contained in Y.)

Allen defined possible relations between time intervals. We adapted these relations for the definition of temporal dependencies between tasks in multi-workflow environments, adding a couple of new relations and a few variations of those originally proposed. The following temporal dependencies are defined in the library:

task1 equals task2: this dependency establishes that both tasks must be executed simultaneously. In the coordination level, it is necessary to ensure that tasks only start when both are ready (tokens in places *start_task* – Figure 2) and that they finish together (tokens simultaneously sent to place *finish_task*).

task1 starts task2: according to Allen's definition, both tasks must start together and task1 must finish before task2 (we called this relation *startsA*). However, we also relaxed the second part of the definition, creating a variation of the relation in which it does not matter which task finishes before ([x1, x2] *startsB* [y1, y2] iff x1 = y1).

task1 finishes task2: the original definition establishes that both tasks must finish together and task1 must start after task2 (*finishesA*). Similarly to the previous dependency, it is possible to relax the definition, creating a variation in which it does not matter which task starts before ([x1, x2] *finishesB* [y1, y2] iff x2 = y2).

task2 after task1: this relation derived from the original relation *before*. In this case, there is a restriction on the execution of task2, which can only occur after the execution of task1. There is no restriction on the performance of task1. This relation models the notion of pre-requisite, frequently used in workflows. It is possible to define two variations of this relation. The first one (*afterA*) establishes that each execution of task1 enables a single execution of task2. The second variation (*afterB*) enables many executions of task2 after a single execution of task1.

task1 before task2: differently from the previous relation, the restriction here occurs on the execution of task1, which cannot be performed if task2 has started to execute. There is no restriction on the execution of task2 (i.e., task2 does not have to wait for the execution of task1, as was the case for *task2 after task1*). From the temporal logic point of view, the difference between these two relations may not be significant, but it generates totally different coordination mechanisms.

task1 meets task2: according to this dependency, task2 must begin immediately with the end of task1. In the coordination level, we achieve this by blocking the end of task1 (token in *finish_task* – Figure 2) while task2 is not ready (token in *start_task*).

task1 overlaps task2: this dependency generates two different models. The first one (*overlapsA*) follows the original definition, which establishes that task2 must start before the end of task1, which must finish before task2. The second model relaxes the last part of the original definition and does not require that task1 finishes before task2 ([x1, x2] *overlapsB* [y1, y2] iff $x1 < y1 < x2$).

task2 during task1: this dependency establishes that task2 must be executed during the execution of task1. Once more, the relation enables two different interpretations, which generates two different coordination mechanisms. In the first case (*duringA*), task2 can be executed only once during the performance of task1. In the second case (*duringB*), task2 can be executed several times.

In the following, we show the models for the coordination mechanisms related to *starts* and *during* dependencies. Other mechanisms are shown in the Appendix and the full set of coordination mechanisms is described in [25].

Figure 3 shows the coordination mechanism for the relation *Task1 startsB Task2*, without the restriction on which task should finish before. In the figure, transition *t1* is a control activity that constitutes, at the same time, an AND-join and an AND-split, ensuring the simultaneous beginning of both tasks. The transitions called *task1* and *task2* represent the logistic of the tasks. They are represented as non-instantaneous transitions with token reservation (indicated by the letter "R" in the transitions). In order to model the relation with the restriction that Task1 must finish before Task2 (*startsA*), it is necessary to add an AND-join after the tasks, ensuring that a token will be sent to place *finish_task2* only after the end of both tasks (Figure 4).

Since the goal of the coordination mechanisms is to deal with relations between tasks that sometimes belongs to complex procedures, it is interesting to add mechanisms to reduce deadlocks. For example, in relation *Task1 startsB Task2*, the first workflow could be blocked in Task1 if the second workflow has an alternative path that does not execute Task2. This kind of problem can be minimized by the use of timeouts added to the original schema.

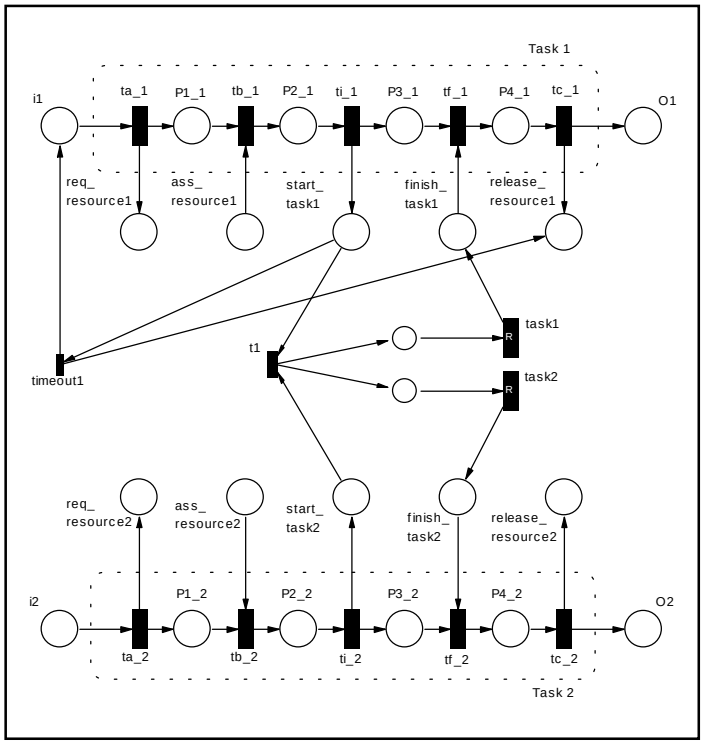

Figure 3: Coordination mechanism for *Task1 startsB Task2*.

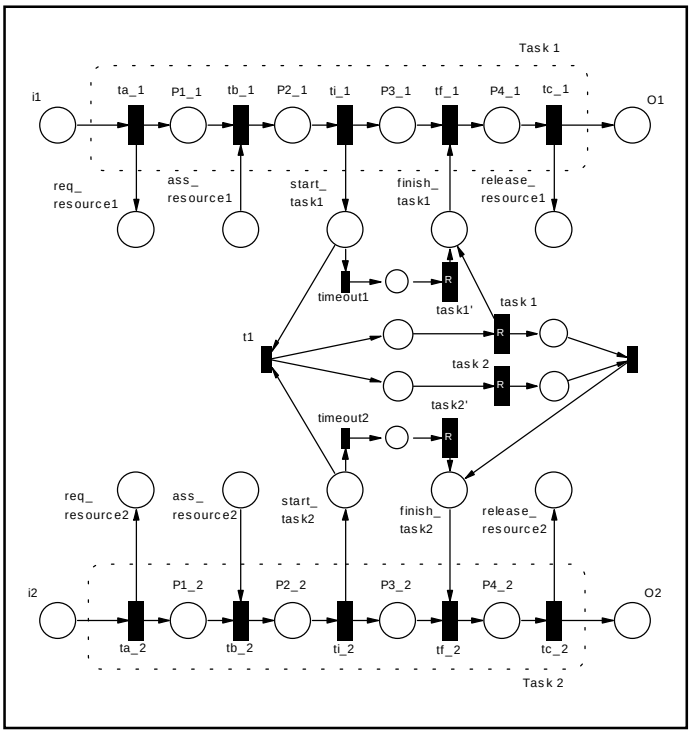

Figure 4: Coordination mechanism for *Task1 startsA Task2*.

Two kinds of timeouts are proposed. In the first kind (called *timeoutA*), an alternative task (*task'*) is executed after a certain waiting time if the original task has not been

executed yet. The other kind of timeout (*timeoutB*) is modeled by a timed transition that removes the token from *start_task* and returns it to the input places of the task after a certain waiting time. This enables to follow alternative paths that do not execute the blocked task. In this kind of timeout, it is also necessary to put a token in *release_resource* if the task have requested a specific resource. An example of *timeoutB* is shown in Figure 3 for Task1 (transition *timeout1*) and examples of *timeoutA* are shown in Figure 4 for both tasks.

The coordination mechanism for the relation *Task2 duringA Task1* is shown in Figure 5. In the model, the firing of *t1* sends tokens to places *P1* and *P2*, the latter indicating to Task2 that Task1 has already started (enabling the firing of *task2* only once). After the firing of *task1* (token in *P3*), a token will only be sent to *finish_task1* at the end of Task2 (token in *P4*). In order to avoid that Task1 waits indefinitely, there is a timeout inhibited by the presence of a token in *start_task2* (i.e., Task1 will not be finished if Task2 is ready to begin).

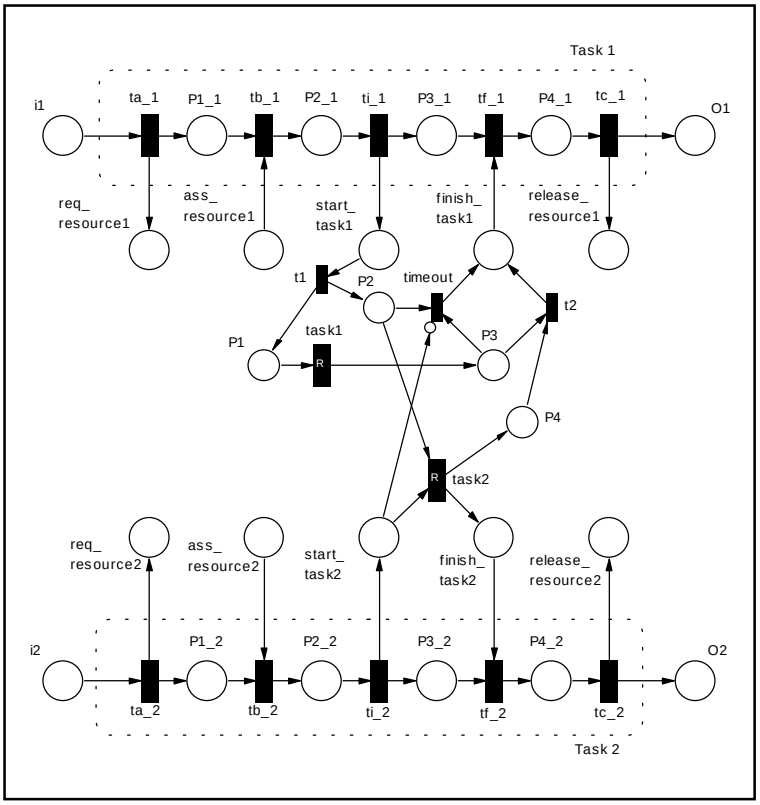

Figure 5: Coordination mechanism for *Task2 duringA Task1*.

In order to enable multiple executions of Task2 during the execution of Task1 (*duringB*), it is necessary to make a few modifications in the model of Figure 5. The first one is to add a return arc from *task2* to *P2*, enabling future firings of *task2*. In addition, transition *t2* must have *P2* as input place instead of *P4*, which is removed from the model. The reason for this is that, at the end of Task1 (firing of *t2*) the token must be removed from *P2* to avoid future occurrences of Task2. It is also necessary to include an inhibitor arc from *start_task2* to *t2*, avoiding that Task1 finishes while Task2 is ready to be executed. The resulting model is presented in Figure 6.

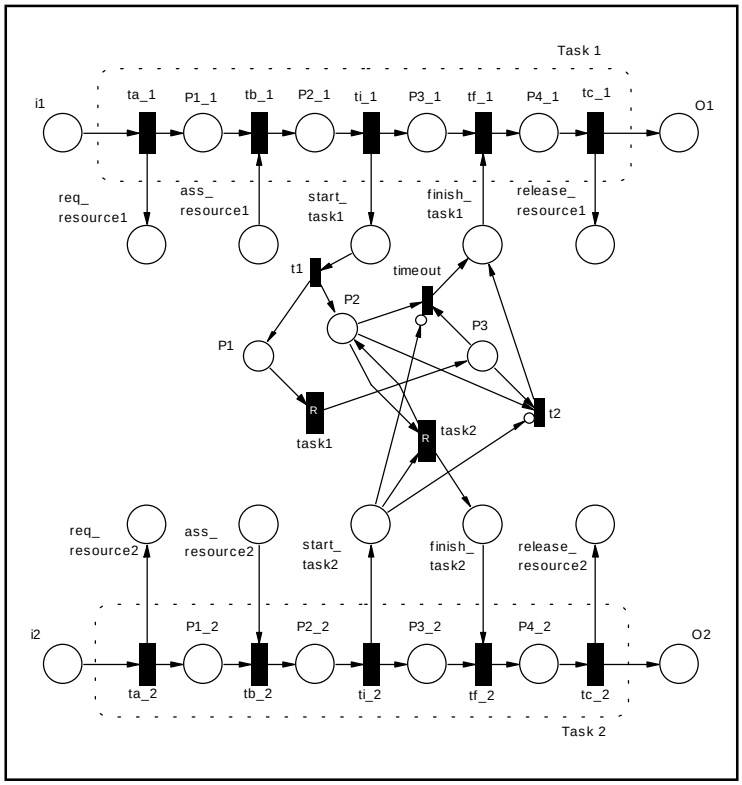

Figure 6: Coordination mechanism for *Task2 duringB Task1*.

In both cases (*duringA* and *duringB*) there is an internal timeout to avoid that Task1 waits indefinitely for Task2. In other words, Task1 may be skipped if it takes too long to starts its execution.

The coordination mechanisms presented in this section cover only those dependencies related to the synchronization of tasks. Dependencies related to the use of resources are modeled by another class of mechanisms presented in the following section.

4.2. Resource Management Dependencies

The coordination mechanisms for resource management are complementary to those presented in the previous section and can be used in parallel to them. This kind of coordination mechanism deals with the distribution of resources among the tasks. It is necessary to clarify that the term “resource” in this context refers not only to the agent that performs the task (similar to the concept of actor [17]), but also to any artifact needed to the execution of the task (e.g., a pen in a shared whiteboard interaction).

We define three basic coordination mechanisms for resource management:

Sharing: a limited number of resources needs to be shared among several tasks. It represents a very common situation that occurs, for example, when various computers share a printer.

Simultaneity: a resource is available only if a certain number of tasks requests it simultaneously. It represents, for instance, a machine that can only be used with more than one operator.

Volatility: indicates whether, after the use, the resource is available again. For example, a printer is a non-volatile resource, while a sheet of paper is volatile.

Differently from temporal dependencies, resource management dependencies are not binary relations. It is possible, for example, that more than two tasks share a resource. Moreover, each of the above mechanisms requires a parameter indicating the number of resources to be shared, the number of tasks that must request a resource simultaneously, or the number of times a resource can be used (volatility).

Figure 7 shows the coordination mechanism for the simultaneity, which assigns a resource only if *N* tasks request it simultaneously. The model has a place *Pn* with *N* tokens (in the figure, *N=2*, as indicated by the two tokens in *Pn*) that return to this place at the end of the tasks (via *release_resource*). Place *Pn* is connected to transition *t1* by an arc with weight *N*. Transition *t1* fires only when there are *N* tokens in *Pn* (indicating *N* available instances of the resource) and also *N* tokens in *P1* (indicating that N tasks have already requested the resource). After the firing, *t1* sends *N* tokens to *P2*, which distributes them among the *N assigned_resource* places. Places *P3* and *P4* are used to indicate which tasks have requested the resource and to avoid that the same task requests the resource more than once (i.e., the resource must be assigned to *N* distinct tasks).

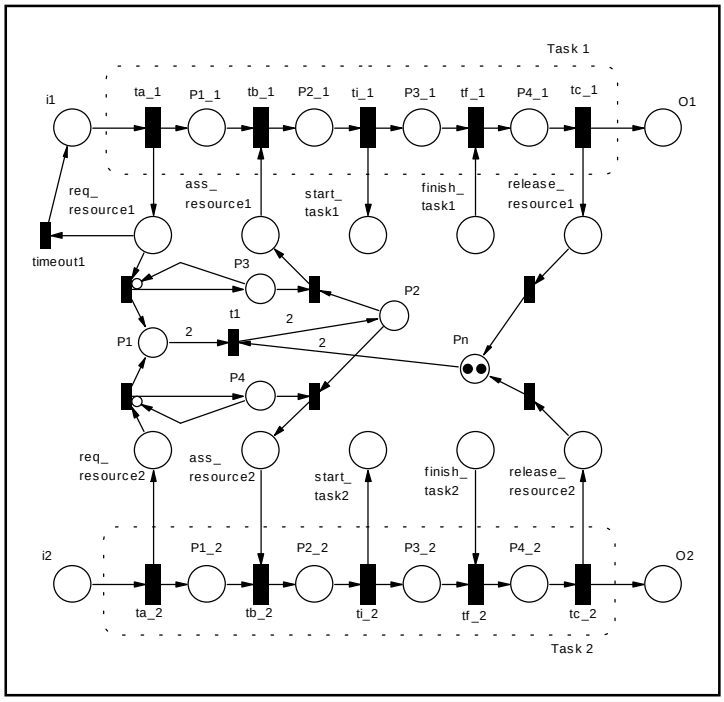

Figure 7: Resource manager for *simultaneity* (*N=2*), with two tasks able to request the resource.

The library also defines composite mechanisms from the basic ones discussed above.

S*haring M + simultaneity N*: represents the situation in which up to *M* groups of *N* tasks can share a resource.

Sharing M + volatility N: situation in which up to *M* tasks can share the resource, which can be used *N* times.

Simultaneity M + volatility N: the resource is assigned to groups of *M* tasks simultaneously. This can be done *N* times.

Sharing M + simultaneity N + volatility Q: up to *M* groups of *N* tasks can share a resource. This can be done *Q* times.

Similarly to temporal dependencies, it is possible to define timeouts for the tasks in order to reduce deadlocks. In this case, there is a single type of timeout, which is modeled by a timed transition from *request_resource* back to the input places of the task (see Figure 7).

4.3. A Prototype Implementation

In order to automate the passage from the workflow level to the coordination level of the models, by the expansion of tasks and the insertion of coordination mechanisms, we implemented an application based on three components: a PN simulation tool, a language for the definition of interdependencies among tasks, and a program capable of generating the PN in the coordination level (Figure 8).

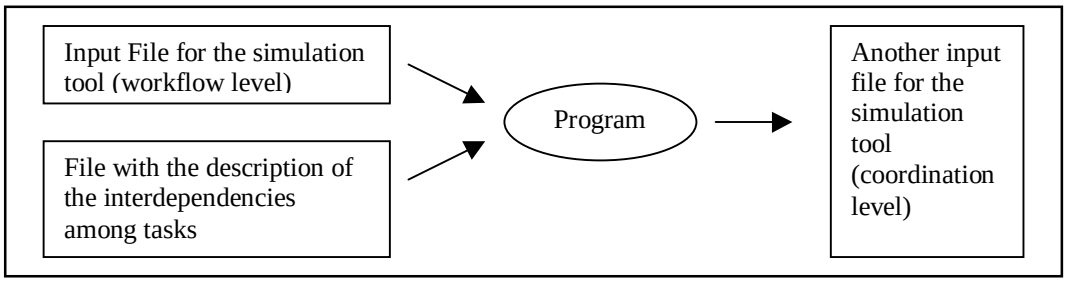

Figure 8: Schema for the use of the library with a PN simulation tool.

The specific language for the description of PNs (input files) normally varies according to the simulation tool used. However, the language for the definition of interdependencies is independent of the simulation tool. The file with the interdependencies define, one at each line, all the dependencies among tasks in the workflow level. It is necessary that tasks in this file have the same name as the transitions representing them in the workflow level. Otherwise, the program will not be able to know where to insert the coordination mechanisms. Basically, a dependency is defined as `<dependency name> [parameters] "<task1 name>" "<task2 name>" ["<taskn name>"] [timeouts]`, where parameters are needed for resource management dependencies and the list of timeouts is optional. As an example, the simultaneity dependency shown in Figure 7 is described as `sim 2 "task1" "task2" time_out`. In this example, parameter N=2 and only the first task has a timeout.

The implemented prototype uses the Visual Simnet [26] as simulation tool. This is a freeware tool, capable of modeling and analysing conventional and stochastic PNs (whose firing times are given by probability functions). It has a graphical editor and several resources for the analysis of PNs, including animated simulation, coverability tree, structural and performance analysis, among others. Besides its good analysis capacity, Visual Simnet was chosen because it has a very simple textual format for the description of

PNs (MoDeL – Model Description Language) and it allows the exportation for other tools (e.g., INA – Integrated Net Analyzer [27], a tool with further analysis capabilities[6]).

The result of the application is a file in MoDeL describing the coordination level of the defined model, which can be simulated, analysed and validated using the Visual Simnet or a tool to which it can be exported. In the next section we present some analysis results derived from the use of this tool.

5. Example

To illustrate the use of the library, we present an example of a hypothetical multi-workflow environment composed of three independent "organizations", a consumer, a shop and a producer. This environment tries to represent a typical situation in e-commerce, in which a consumer contacts a virtual shop to buy some goods. The shop, however, is just an intermediary between consumer and producer, having to contact the latter to receive the goods that will be delivered to the consumer. The workflow level of this environment is shown in Figure 9, stressing aspects of the relation between the shop and the producer.

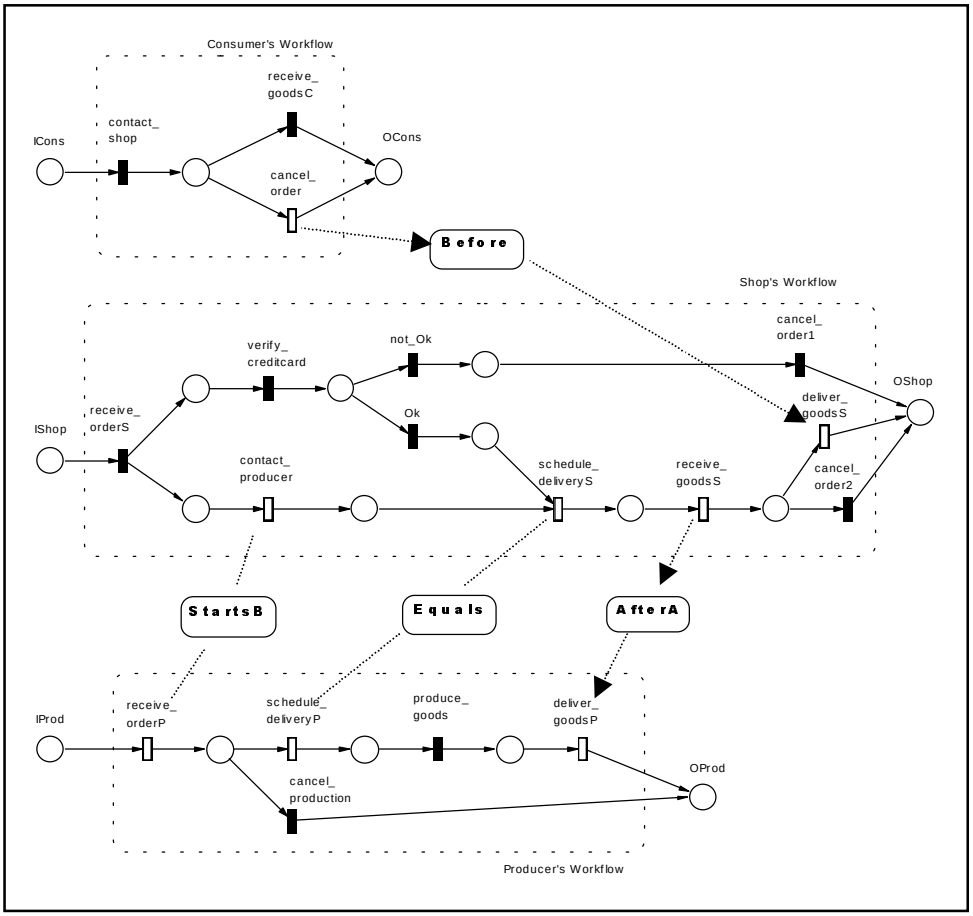

Figure 9: Workflow level of the example.

[6] INA does not handle transitions with token reservation. Therefore, some coordination mechanisms cannot be treated by this tool.

The consumer's workflow is very simple. It must start the process by contacting the shop to make the order, and then wait for the goods or cancel the order. However, the order can only be canceled if the product has not been delivered by the shop. This defines the relation *cancel_order (consumer) before deliver_goodsS (shop)*. The shop, after receiving an order, starts two parallel activities: verification of the consumer's credit card and contact with the producer. This contact *starts* the process in the producer. If there is any problem with the consumer's credit card, the shop cancels the order. Otherwise, the shop is able to schedule the delivery of the product, a task that must occur simultaneously to the scheduling made by the producer (*schedule_deliveryS equals schedule_deliveryP*). Note that, if there is a problem with the credit card, *schedule_deliveryS* will not be executed, and consequently neither will *schedule_deliveryP*. Using a timeout in task *schedule_deliveryP*, the producer has the option to follow an alternative path that cancels the production. Finally, there is also the relation that the shop receives the goods *after* the producer delivers them.

To construct the coordination level for this example using the application presented in the previous section, the following file is written to define the interdependencies among tasks (note that *cancel_order* and *schedule_deliveryP* have timeouts of type 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"
```

The model of the coordination level for this example is shown in Figure 10. At first glance, this model may seem complicated, but it is highly modular and easily built from the model of Figure 9, by expanding interdependent tasks (open rectangles) and inserting the pre-defined coordination mechanisms.

An observation should be made regarding the expansion of tasks. As can be seen in Figure 10, interdependent tasks are not expanded exactly according to the model of Figure 2. The reason is that in the case of temporal dependencies, only places *start_task* and *finish_task* are needed, therefore, we use a simplified version of the model. A similar simplification is made for the case of resource management dependencies. The complete model of a task, as in Figure 2, is only used if it has both a temporal and a resource management dependency.

The PN of the coordination level was simulated and analysed with the Visual Simnet, validating the model. The analysis of the coverability tree presented twelve final states. Some of these final states, although correct, indicates that the model could be improved. For example, if there is a problem with the credit card, the workflows will finish correctly from the functional point of view, but there will be a "dead token" in the place between *contact_producer* and *schedule_deliveryS* in the shop's workflow, which indicates a badly-structured workflow [8]. Performance could be analysed, for instance, to measure average consumers' waiting time, given the rates with which goods can be produced.

The previous example used only temporal dependencies, but resource management dependencies could also be used in a straightforward way. For example, the shop could have a stock, which defines an alternative route to that of contacting the producer. The task of getting the goods from the stock would have a volatility dependency, indicating that the stock would eventually finish.

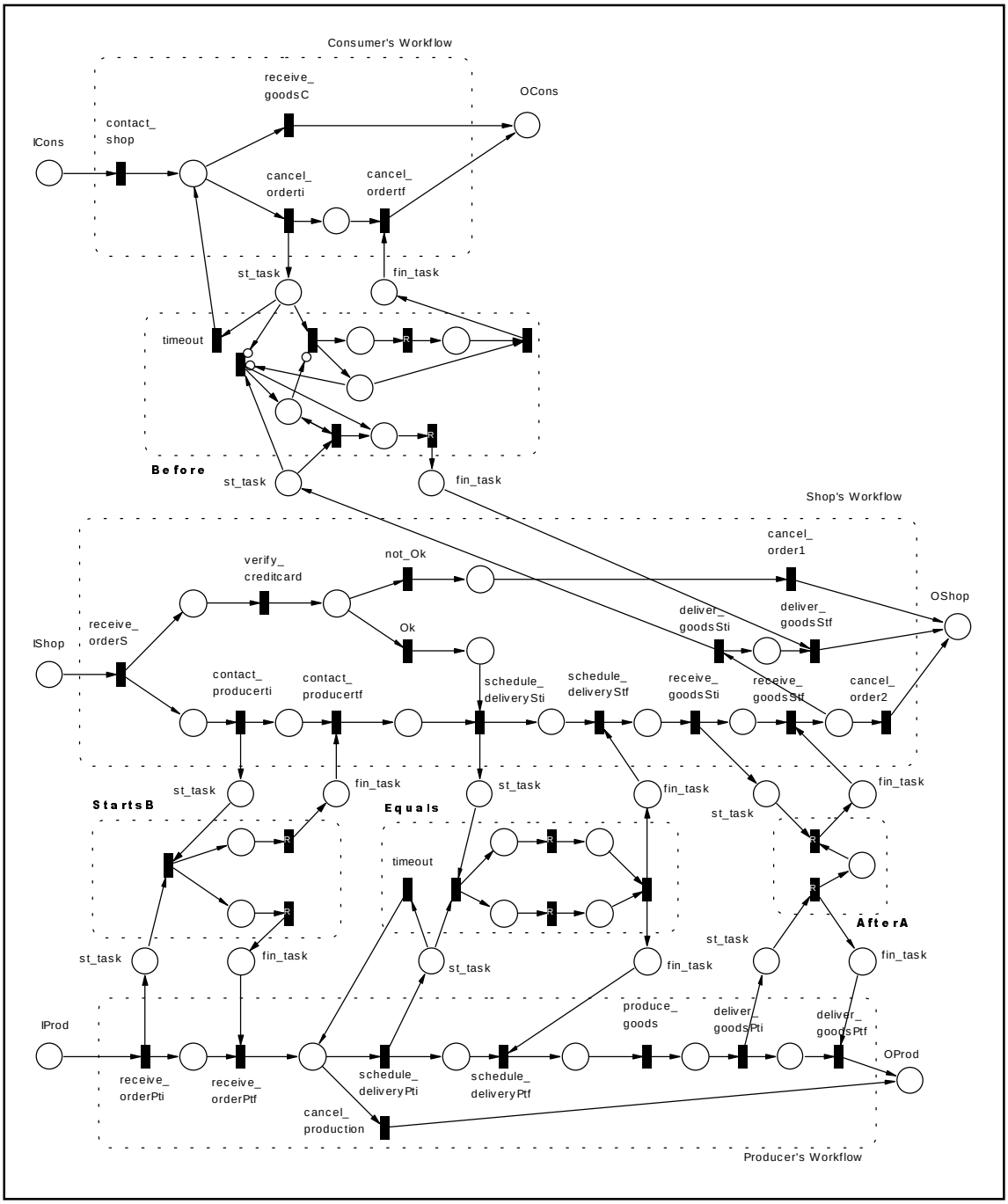

Figure 10: Coordination level of the example.

It is necessary to reinforce that the presented example represents just a hypothetical situation. We did not stress all details for the modeled scenario (e.g., the consequences of the consumer's cancellation, such as refund, devolution of the goods to the producer, and so on). Its main goal was to show how the coordination mechanisms can be used in a practical situation.

6. Conclusions

This paper intends to contribute in two different ways. The first one is the definition of a generic set of interdependencies among tasks, and the second are the PN based coordination mechanisms. Due to this clear separation, non-PN based applications can also follow the ideas presented here using the interdependencies among tasks and constructing appropriate coordination mechanisms.

Petri Nets, due to their support for modeling, simulation, and analysis, has proven to be a powerful tool for verifying the correctness and validating the effectiveness of multi-workflow environments before their actual implementation [8], [17]. Furthermore, the hierarchical description of PNs showed an appropriate way to define the coordination structure in different abstraction levels (workflow and coordination levels).

Nevertheless, a typical problem in the use of PNs is the state explosion, which can occur in our context when the number of workflows and interdependencies increases. This problem can be minimized by the use of high level PNs, such as predicate/transition and colored nets. We are currently investigating how the use of these kinds of PNs can simplify the coordination mechanisms of the library [28]. We are also studying the possibility of using methodologies such as CPM (Critical Path Method) and PERT (Program Evaluation and Review Technique) [29] to concentrate the coordination effort in the critical path of complex nets.

Another use of the coordination mechanisms are in collaborative virtual environments, where remote users can be simultaneously present and interact with each other and with the objects of the simulated 3D world. In this scenarios, users can be compared to an organization in a multi-workflow environment, because their behavior is defined by an independent workflow. The collaborative interaction in the virtual world is defined by interdependencies among tasks executed by the users and coordinated by the mechanisms here presented. Therefore, a collaborative virtual environment can be viewed as a multi-workflow environment. A similar experience has been made using PNs to control computer animation [30].

The library presented in this paper does not claim to be complete. We believe it would be very difficult to establish a framework of all possible interdependencies between tasks. For that reason, we preferred to opt for extensibility instead of completeness. When a new kind of interdependency arises, a corresponding coordination mechanism can be modeled and inserted to the library, by adding a new “command” to the definition language and extending the program (Figure 8) to recognize the new dependency.

Finally, we reinforce our belief that the coordination of interdependent tasks in multi-workflow environments is a problem that should be addressed to ensure the effectiveness of the cooperation among organizations. The separation between activities and dependencies, and the utilization of reusable coordination mechanisms are steps towards this goal.

Acknowledgements. The first author is sponsored by FAPESP (Foundation for Research Support of the State of São Paulo), grant n. 96/06288-9. This work has also been supported by the SAPIENS project (FAPESP, grant n. 97/12807-1). We also would like to thank the School of Electrical and Computer Engineering (FEEC) – UNICAMP for the expressive support granted to this research.

References

1. **Anderson, M** Workflow Interoperability – Enabling E-Commerce, *WfMC White Paper* (April 1999). <http://www.aiim.org/wfmc/standards/docs/IneropChallPublic.PDF>
2. **Workflow Management Coalition** *Interface 4 – Interoperability Abstract Specification*, WFMC-TC-1012 (October 1996). <http://www.aiim.org/wfmc/standards/docs/if4-a.pdf>
3. **Malone, T W and Crowston, K** What is Coordination Theory and How Can It Help Design Cooperative Work Systems? *Proc. ACM Conf. on Computer Supported Cooperative Work* (1990) 357-370
4. **Edwards, W K** Policies and Roles in Collaborative Applications. *Proc. ACM Conf. on Computer Supported Cooperative Work* (1996) 11-20
5. **Petri, C A** *Kommunikation mit Automaten*. Schriften des IIM Nr. 3, Bonn: Institute für Instrumentelle Mathematik (1962)
6. **Murata, T** Petri Nets: properties, analysis and applications, *Proc. of the IEEE*, **77**(4) (1989) 541-580
7. **Ramamoorthy, C V and Ho, G S** Performance evaluation of asynchronous concurrent systems using Petri Nets, *IEEE Trans. Software Engineering*, **6**(5) (September 1980) 440-449
8. **van der Aalst, W M P** The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, **8**(1) (1998) 21-66
9. **Holt, A W** Coordination Technology and Petri Nets. In Rozenberg, G (Ed.) *Advances in Petri Nets, LNCS 222*, Springer-Verlag (1985) 278-296
10. **Holt, A W** Diplans: A New Language for the Study and Implementation of Coordination, *ACM Trans. Office Information Systems*, **6**(2) (April 1988) 109-125
11. **Genrich, H J** Predicate/Transition Nets. In Brauer, W, Reisig, W and Rozenberg G (Eds.) *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986, LNCS 254*, Springer-Verlag (1986) 207-247
12. **De Cindio, F, De Michelis, G and Simone, C** The Communication Disciplines of CHAOS. In *Concurrency and Nets*, Springer-Verlag (1988) 115-139
13. **Furuta, R and Stotts, P D** Interpreted Collaboration Protocols and their use in Groupware Prototyping. *Proc. ACM Conf. on Computer Supported Cooperative Work* (1994) 121-131
14. **Jensen, K** Coloured Petri Nets. In Brauer, W, Reisig, W and Rozenberg G (Eds.) *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986, LNCS 254*, Springer-Verlag (1986) 248-299
15. **Jablonski, S and Bussler, C** *Workflow Management: Modeling Concepts, Architecture and Implementation*, International Thomson Publishing (1996)
16. **Lawrence, P** (Ed.), *The Workflow Handbook 1997*, Workflow Management Coalition, John Wiley & Sons, Inc. (1997)
17. **Ellis, C A and Nutt, G J** Modeling and Enactment of Workflow Systems. In Marsan, M A (Ed.) *Application and Theory of Petri Nets 1993, LNCS 691*, Springer-Verlag (1993) 1-16
18. **van der Aalst, W M P, van Hee, K M and Houben, G J** Modelling and analysing workflow using a Petri-net based approach. *Proc. 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms* (1994) 31-50

19. **Camarinha-Matos, L M and Afsarmanesh, H** Flexible Coordination in Virtual Enterprises. *Proc. 5th Int. Workshop on Intelligent Manufacturing Systems* (1998) 43-48
20. **CACI** *SIMPROCESS* (2000). <http://www.simprocess.com>
21. **Infocosm** *METEOR (Managing Endo-To-End OpeRations)*. Commercial site <http://infocosm.com/html/products.html>. Academic site (LSDIS, Dept. of Computer Science, Univ. of Georgia) <http://lsdis.cs.uga.edu/proj/meteor/ meteor.html>
22. **Merz, M, Liberman, B and Lamersdorf, W** Using Mobile Agents to support Interorganizational Workflow Management, *Int. J. Applied Artificial Intelligence*, **11**(6) (September 1997)
23. **van der Aalst, W M P** Interorganizational Workflows – An approach based on Message Sequence Charts and Petri Nets, *Systems Analysis – Modelling – Simulation*, **34**(3) (1999) 335-367
24. **Allen, J F** Towards a General Theory of Action and Time, *Artificial Intelligence*, 23 (1984) 123-154
25. **Raposo, A B, Magalhães L P and Ricarte, I L M** *Petri Nets Based Coordination Mechanisms*. <http://www.dca.fee.unicamp.br/~alberto/pubs/IJCSSE/mechanisms/>
26. **Garbe, W** *Visual Simnet V.1.37 – Stochastic Petri-Net Simulator* (1997). <http://home.arcor-online.de/wolf.garbe/simnet.html>
27. **Starke, P H** *INA – Integrated Net Analyzer*. Institute für Informatik – Humboldt-Universität zu Berlin (1999). <http://www.informatik.hu-berlin.de/~starke/ ina.html>
28. **Raposo, A B, Magalhães L P and Ricarte, I L M** Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach. Accepted to the *4th World Multi-conference on Systemics, Cybernetics and Informatics*, Orlando, FL (July 2000)
29. **Nilsson, N J** *Principles of Artificial Intelligence*, Morgan Kaufmann Pubs. (1986)
30. **Magalhães L P, Raposo, A B and Ricarte, I L M** Animation Modeling with Petri Nets. *Computer & Graphics*, **22**(6) (1998) 735-743. Pergamon Press

Appendix – Other Coordination Mechanisms

In this appendix we shortly present some other coordination mechanisms of the library. The complete set of mechanisms is shown in [25].

The first mechanism presented is for dependency *Task1 before Task2* (Figure A1). The core of the model is transition *t1*, which determines the execution of Task2 and is inhibited if there is any token in *start_task1* or *P3* (Task1 is ready or being executed, respectively). The firing of *t1* sends a token to *P1*, inhibiting the firing of *t2* and, consequently, the execution of Task1. If Task2 has not been executed yet (no token in *P1*), *t2* can be fired, sending tokens to *P2*, which enables *task1*, and to *P3*, which inhibits *t1*. At the end of Task1 *t4* is fired, removing the token from *P3*. Transition *t3* enables other executions of Task2 (after the first one, which puts a token in *P1*), which can happen independently of the presence of tokens in *start_task1* (Task1 will not happen anymore). It is important to use a timeout to ensure that Task1 returns to its initial state when it cannot be executed anymore.

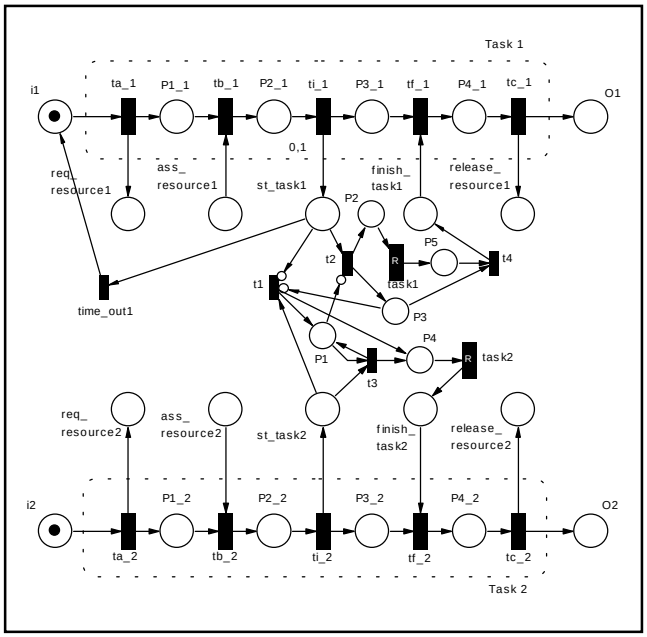

Figure A1: Coordination mechanism for *Task1 before Task2*.

Another mechanism presented here is for resource management dependency *sharing N*. The coordination mechanism for this dependency consists of a place *Pn* with N tokens representing the available resources. This place is the input place for a transition connecting *request_resource* to *assigned_resource*, defining whether there are available resources. At the end of the task, the token returns to *Pn* via *release_resource*. Figure A2 shows the model for N = 3.

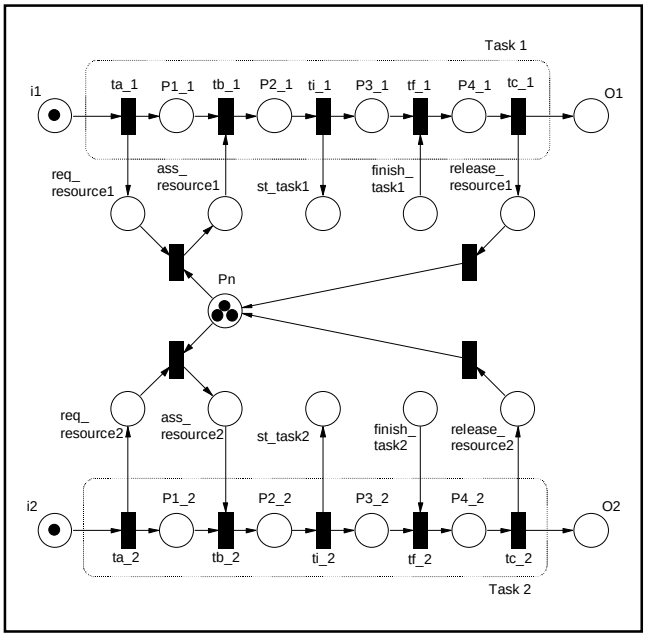

Figure A2: Resource manager for sharing N (N = 3) with two tasks able to request the resource.

Finally we show the coordination mechanism for the composite relation *simultaneity M with volatility N*. This mechanism is similar to that of *simultaneity M* (Figure 7). The difference is the inclusion of place *Pn* relative to the volatility. The timeouts presented in the figure are optional, but very important when the resource finishes. Figure A3 shows this mechanism for M = 2 and N = 4. It is important to note that N is the number of times the resource is going to be used, and not the number of groups of M tasks that is going to use it.

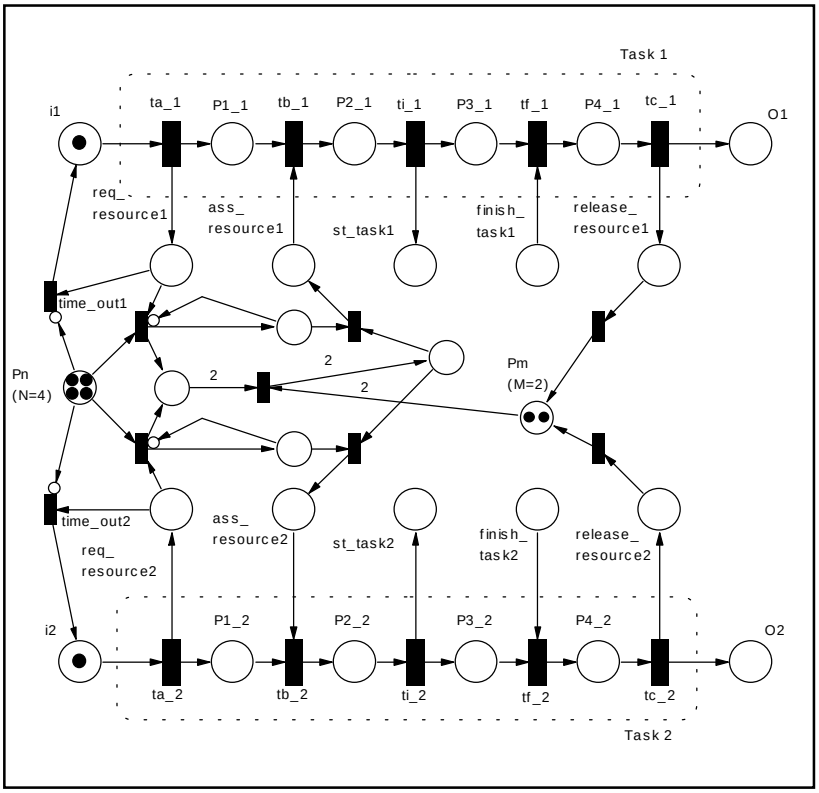

Figure A3: Resource manager for *simultaneity 2 with volatility 4*.

Mecanismos de Coordenação para Ambientes Colaborativos

Alberto B. Raposo, Léo P. Magalhães, Ivan L. M. Ricarte
Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
{alberto, leopini, ricarte}@dca.fee.unicamp.br

Resumo. A coordenação das interdependências entre atividades colaborativas é uma tarefa complexa, de difícil modelagem computacional. Neste trabalho é apresentado um conjunto de mecanismos de coordenação para a especificação e controle da interação entre tarefas colaborativas. Os mecanismos são modelados por redes de Petri e permitem avaliar o comportamento de um ambiente de suporte ao trabalho colaborativo antes de sua implementação.

Abstract. The coordination of interdependencies among collaborative activities is a complex task, not easily computer modeled. In this paper, a set of coordination mechanisms for the specification and control of interaction among collaborative tasks is presented. The mechanisms are modeled using Petri nets and enable to evaluate the behavior of a computer supported collaborative work system before its implementation.

1. Introdução

Trabalho colaborativo foi definido por Marx como “múltiplos indivíduos trabalhando juntos de maneira planejada no mesmo processo de produção ou em processos de produção diferentes, mas conectados” [Bannon 91]. No âmago desta definição está a noção de planejamento, responsável por garantir que o trabalho coletivo seja resultante do conjunto de tarefas individuais.

No entanto, algumas atividades envolvendo múltiplos indivíduos não exigem um planejamento formal. Atividades relacionadas às relações sociais são bem controladas pelo chamado protocolo social, caracterizado pela ausência de qualquer mecanismo de coordenação entre as atividades, confiando nas habilidades dos participantes de mediar as interações. Exemplos de atividades deste tipo em computador são os *chats* e as áudio- ou vídeo-conferências.

Por outro lado, atividades mais diretamente voltadas para o *trabalho* colaborativo (e não para as relações sociais) exigem sofisticados mecanismos de controle para evitar que os participantes se envolvam em tarefas conflitantes e/ou repetitivas. É este tipo de atividade que tem sido o principal alvo dos esforços de pesquisa em CSCW (*Computer Supported Cooperative Work*), uma área que estuda “a natureza e as características do trabalho cooperativo com o objetivo de projetar tecnologias computacionais adequadas” [Bannon 91].

A noção de planejamento presente na definição de Marx é materializada em CSCW pelo que foi chamado de trabalho de articulação, que 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 participantes, a distribuição das tarefas entre eles e a coordenação da realização das atividades.

Particularmente importante entre as atividades do trabalho de articulação é a coordenação, definida 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.

A coordenação de atividades em ambientes colaborativos auxiliados por computador é o principal foco deste artigo, que apresenta um conjunto de mecanismos de coordenação para uma série de interdependências que freqüentemente ocorrem entre tarefas colaborativas. A idéia é separar as atividades (tarefas) das dependências (controladas pelos mecanismos de coordenação), permitindo o uso de diferentes políticas de coordenação em um mesmo ambiente colaborativo, sendo necessário apenas trocar os mecanismos de coordenação. Além disso, torna-se possível o reuso dos mecanismos de coordenação em outros ambientes colaborativos. Para modelar os mecanismos de coordenação propostos, é usada uma abordagem baseada em redes de Petri (PNs – *Petri Nets*).

A próxima seção mostra uma visão geral de trabalhos relacionados à coordenação de atividades colaborativas. A Seção 3 faz uma breve introdução às PNs, e a Seção 4 apresenta os mecanismos desenvolvidos. Um exemplo de uso dos mecanismos de coordenação em uma situação de CSCW é mostrado na Seção 5. A Seção 6 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Apesar de sua reconhecida importância, mecanismos de coordenação não foram incluídos nos primeiros sistemas colaborativos. As tentativas de incluir mecanismos de coordenação neste tipo de sistema datam do final dos anos 80. Um dos mais significativos representantes da primeira geração de sistemas colaborativos com algum mecanismo de coordenação foi o *Coordinator* [Winograd 86], desenvolvido com base em teorias lingüísticas, com o objetivo de ajudar a estruturação de conversas por e-mail. O *Coordinator* segue padrões de conversação pré-estabelecidos para direcionar a realização das tarefas.

Desde aquela época há exemplos de uso de PNs para a coordenação de atividades em ambientes colaborativos [Holt 85], [De Cindio 88].

Na década de 90 começaram a surgir sistemas que tentam tornar os mecanismos de coordenação mais flexíveis e acessíveis aos usuários. Um destes sistemas é o Oval [Malone 95], que usa o conceito de EUP (*End User Programming*) para permitir que o usuário (não necessariamente um programador experiente) modifique o sistema sem sair do domínio da aplicação e ir para o domínio de uma linguagem de programação.

Inspirados pelas linguagens de coordenação [Gelernter 92], que propuseram a separação computação/coordenação para aplicações *multi-threaded*, surgiram sistemas colaborativos que separam a implementação dos componentes de coordenação das demais partes dos mesmos. Isso permite maior flexibilidade no uso de políticas de coordenação. O COCA (*Collaborative Objects Coordination Architecture*) [Li 98] e o Trellis [Furuta 94]

são exemplos de sistemas deste tipo. O Trellis, em particular, usa uma variação de PNs em um servidor para especificar protocolos de interação para um grupo de clientes.

PNs também têm sido usadas para modelar sistemas de *workflow*, definidos 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]. O COSM (*Common Open Service Market*) [Merz 97], por exemplo, usa agentes móveis para gerenciar *workflows* interorganizacionais, onde cada organização tem seu esquema interno de controle para a realização das tarefas (*workflow*), mas precisa colaborar com outras organizações [WfMC 96]. O comportamento destes agentes é controlado por uma máquina virtual modelada em PN.

PNs também foram usadas para o planejamento de animações interativas, de forma a prever seu comportamento antes de gerar as imagens [Magalhães 98].

O trabalho aqui apresentado é mais genérico que os descritos acima, pois define um conjunto de interdependências entre tarefas e mecanismos de coordenação associados (modelados por PNs) que podem ser usados em *workflows*, trabalho cooperativo, interação multiusuário, ambientes virtuais, etc. O objetivo não é implementar um sistema fechado, mas prover uma série de mecanismos que permitam ao projetista de um ambiente colaborativo prever e testar seu comportamento antes de implementá-lo, detectando as possíveis falhas da muitas vezes complexa cadeia de interação entre as atividades.

3. Redes de Petri

Rede de Petri 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 (*places);* $T = \{t_1, ..., t_n\}$ é um conjunto finito de transições; $F \subseteq (P \times T) \cup (T \times P)$ 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 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.

A notação gráfica de PNs é também muito utilizada (Figura 1). 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).

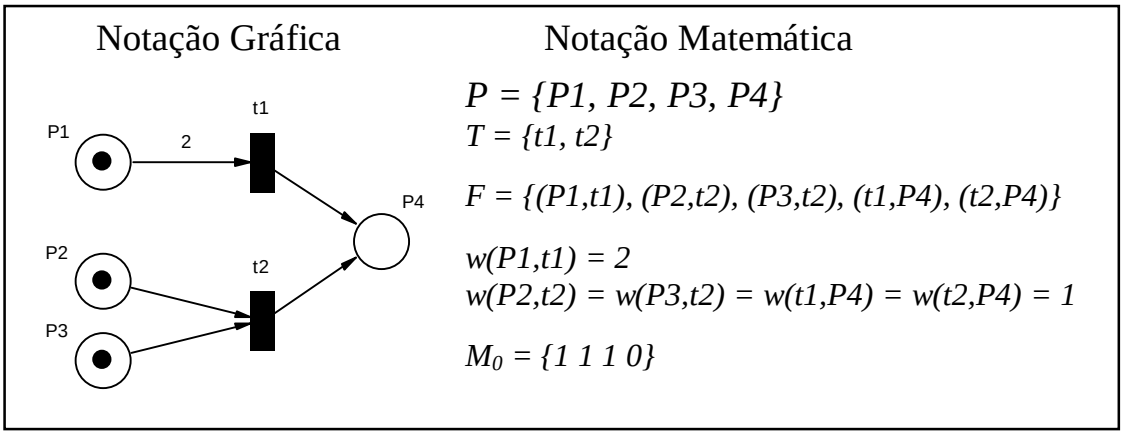

Figura 1: Notação gráfica e notação matemática de PNs.

Além do modelo básico, existem várias extensões de PN [Murata 89]. Duas extensões são utilizadas neste trabalho: arco inibidor e redes com tempo. O arco inibidor liga um lugar *P* a uma transição *t* e a habilita apenas se *P* estiver vazio. Na notação gráfica, arcos inibidores são representados com um círculo na extremidade. O modelo básico de PN não considera a noção de tempo. Uma maneira de incluir esta noção em uma PN é definir um tempo para o disparo das transições. Neste caso, os *tokens* são retirados dos lugares de entrada 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 também oferecem importantes ferramentas para análise do sistema modelado. Há três tipos possíveis de análise: verificação, validação e desempenho [van der Aalst 98]. A análise de verificação é realizada 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. A análise de validação testa se a rede funciona como esperado. Os testes são feitos por meio de simulação iterativa 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.

Em resumo, pela capacidade de modelagem, forte suporte teórico para a análise e grande número de técnicas de simulação, PNs são ferramentas adequadas para o planejamento e coordenação de atividades colaborativas. Por esta razão elas são usadas como ferramenta de modelagem dos mecanismos de coordenação apresentados na próxima seção.

4. Mecanismos de Coordenação

Este trabalho trata da inserção de mecanismos de coordenação para gerenciar interdependências entre tarefas e garantir que estas dependências não sejam violadas. A idéia é fazer com que o projetista do ambiente se preocupe apenas com a definição da rede que modela as tarefas e com a definição das interdependências entre estas tarefas, não com os mecanismos para gerenciar estas dependências.

No esquema proposto, o ambiente colaborativo é modelado em dois níveis hierárquicos distintos: nível de *workflow* e nível de coordenação. No nível de *workflow* são definidos o seqüenciamento das tarefas e as dependências entre elas. No nível de coordenação, as tarefas interdependentes são expandidas e os mecanismos de coordenação são inseridos entre elas.

Durante a passagem do nível de *workflow* para o nível de coordenação, cada tarefa (representada por uma transição no nível de *workflow*) que possuir uma interdependência com outra tarefa é expandida em um sistema como o da Figura 2 [van der Aalst 94]. Os cinco lugares associados às transições (*request_resource*, *assigned_resource*, *execute_task*, *finish_task* e *release_resource*) representam a interação com o gerenciador de recursos e com o agente que 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. Finalmente, o *release_resource* indica ao gerenciador que a tarefa foi encerrada e o recurso está novamente liberado.

O objetivo dos mecanismos aqui apresentados 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 2 e os mecanismos de coordenação reutilizáveis.

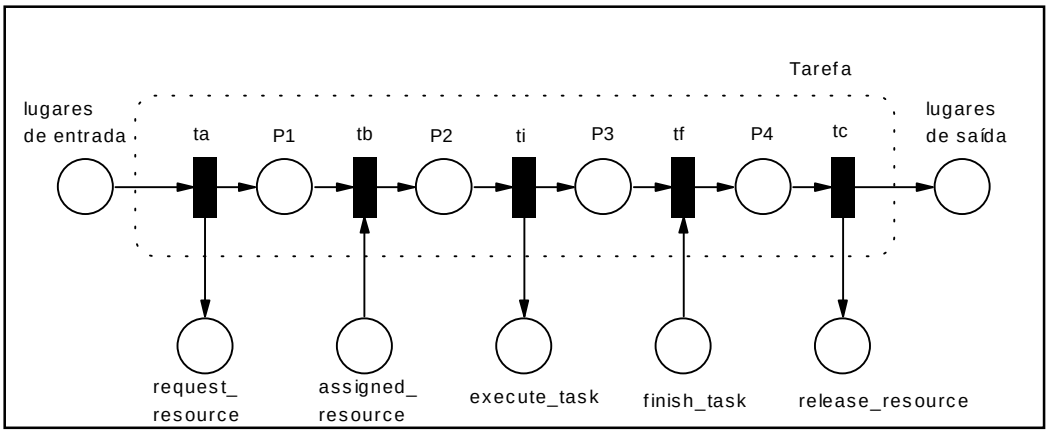

Figura 2: Estrutura de uma tarefa.

A partir desse modelo conclui-se que uma tarefa pode estar 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 (tem *execute_task* como lugar de entrada e *finish_task* como lugar de saída). Estas duas subredes estão diretamente relacionadas às duas grandes classes de interdependências entre tarefas apresentadas nas próximas seções: dependências temporais e dependências de gerenciamento de recursos.

4.1. Dependências Temporais

As dependências temporais servem para estabelecer o ordenamento no processo de execução de tarefas. Os mecanismos propostos são baseados nas relações temporais

definidas em um artigo clássico de lógica temporal [Allen 84], segundo o qual há um conjunto de relações primitivas e mutuamente exclusivas que podem ser aplicadas sobre intervalos de tempo. Estas relações entre intervalos de tempo foram adaptadas para a definição de dependências temporais entre tarefas, criando algumas variações das relações originalmente propostas e adicionando novas relações. As seguintes dependências temporais são definidas:

- *tarefa 1 equals tarefa 2*: esta dependência estabelece que duas tarefas devem ser executadas simultaneamente.
- *tarefa 1 starts tarefa 2*: a definição original de Allen estabelece que as tarefas devem começar juntas, e a tarefa 1 deve terminar antes (*startsA*). Retirando a restrição sobre qual tarefa deve terminar antes, criou-se uma nova dependência (*startsB*).
- *tarefa 1 finishes tarefa 2*: a definição original estabelece que as tarefas devem terminar juntas, mas a tarefa 1 deve começar depois da tarefa 2 (*finishesA*). É também possível eliminar a restrição sobre qual tarefa deve começar antes e criar uma nova relação (*finishesB*).
- *tarefa 2 after tarefa 1*: a tarefa 2 só pode ocorrer após a execução da tarefa 1. É possível definir duas variações para esta dependência. No primeiro caso (*afterA*), cada execução da tarefa 1 dá direito a uma execução da tarefa 2. No segundo caso (*afterB*), uma execução da tarefa 1 permite habilitar várias execuções da tarefa 2.
- *tarefa 1 before tarefa 2*: do ponto de vista da lógica temporal, esta relação pode ser vista como oposta à anterior, mas gera um mecanismo de coordenação totalmente diferente. Essencialmente, a restrição ocorre na execução da tarefa 1, que só pode ocorrer se a tarefa 2 ainda não tiver iniciado sua execução. A tarefa 2 não espera pela execução da tarefa 1, como ocorre na relação *tarefa 2 after tarefa 1*.
- *tarefa 1 meets tarefa 2*: a execução da tarefa 2 inicia imediatamente após o término da tarefa 1.
- *tarefa 1 overlaps tarefa 2*: a definição original estabelece que a tarefa 2 deve se iniciar antes do término da tarefa 1, que por sua vez deve terminar antes (*overlapsA*). Relaxando a restrição de que a tarefa 1 deve terminar antes, foi criada uma segunda relação (*overlapsB*).
- *tarefa 2 during tarefa 1*: possui duas variações. Na primeira delas (*duringA*), a tarefa 2 pode ser executada apenas uma vez durante a execução da tarefa 1. Na outra variação (*duringB*), a tarefa 2 pode ser executada várias vezes enquanto a tarefa 1 estiver sendo executada.

Para ilustrar o modelo dos mecanismos de coordenação para as dependências temporais, a Figura 3 mostra a PN que representa o mecanismo para a relação *tarefa 1 meets 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 3, esta relação é satisfeita colocando o lugar *execute_task2* como entrada da transição que representa a tarefa 1 (transição não instantânea com reserva de *tokens*, como indicado pela letra “R” no desenho). 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*).

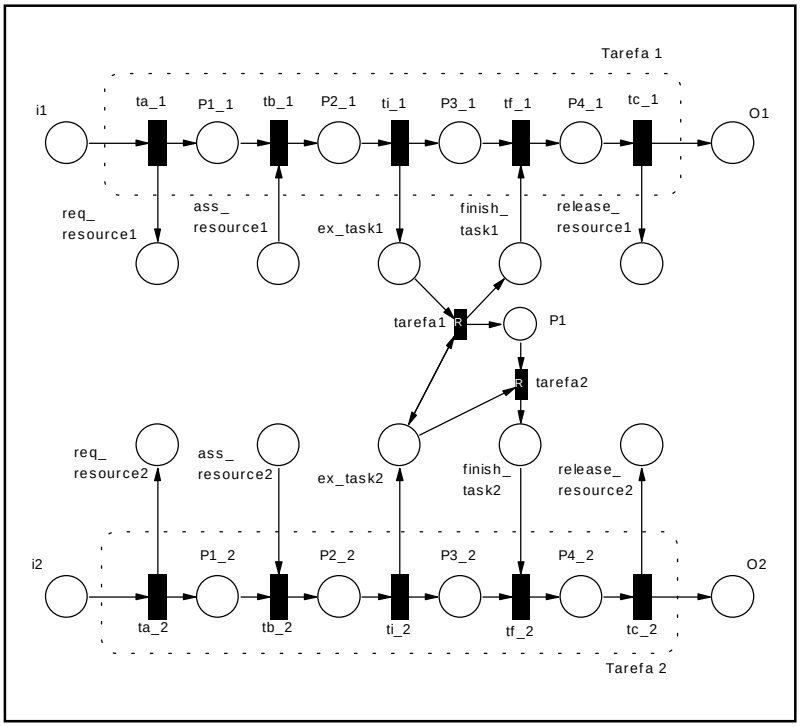

Figura 3: Mecanismo de coordenação para a dependência *tarefa 1 meets tarefa 2*.

Uma alternativa para evitar *deadlocks* é acrescentar mecanismos de *timeout*. Um tipo de *timeout* utilizado (*timeoutA*) define uma tarefa alternativa a ser executada se a original não se iniciar após um certo tempo de espera. Outro tipo de *timeout* (*timeoutB*) retorna *tokens* aos lugares de entrada da tarefa após um certo tempo de espera, de modo que a colaboração possa seguir um caminho alternativo que não passe pela tarefa bloqueada.

4.2. Dependências de Gerenciamento de Recursos

Os mecanismos de coordenação para o gerenciamento de recursos são complementares aos apresentados na seção anterior. Podem ser usados de forma independente, pois eles utilizam os lugares *request_resource*, *assigned_resource* e *release_resource* do modelo expandido de tarefa (Figura 2), não utilizados nos mecanismos da seção anterior.

Este tipo de mecanismo de coordenação lida com a distribuição dos recursos entre as tarefas. Há três mecanismos básicos:

- *Divisão de recursos*: um número limitado de recursos precisa ser compartilhado entre várias tarefas. É um caso muito 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 não é um recurso volátil, mas uma folha de papel é.

Neste contexto, o termo “recurso” está sendo usado de maneira mais ampla que o normalmente usado em *workflows*, referindo-se não apenas ao agente que realiza a tarefa, mas também a qualquer artefato necessário à realização da tarefa.

As dependências de gerenciamento de recursos, diferentemente das temporais, não são relações binárias. É possível, por exemplo, que mais de duas tarefas compartilhem um recurso. Além disso, estas dependências requerem um parâmetro para indicar o número de recursos compartilhados, o número de tarefas que devem solicitar um recurso simultaneamente, ou o número de vezes que um recurso pode ser utilizado (volatilidade).

A Figura 4 ilustra o modelo para o gerenciador de recursos da divisão por 1 (duas tarefas compartilhando uma única instância do recurso – exclusão mútua). O modelo consiste em um lugar (*Pn*) com um *token*, representando o recurso. *Pn* 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*.

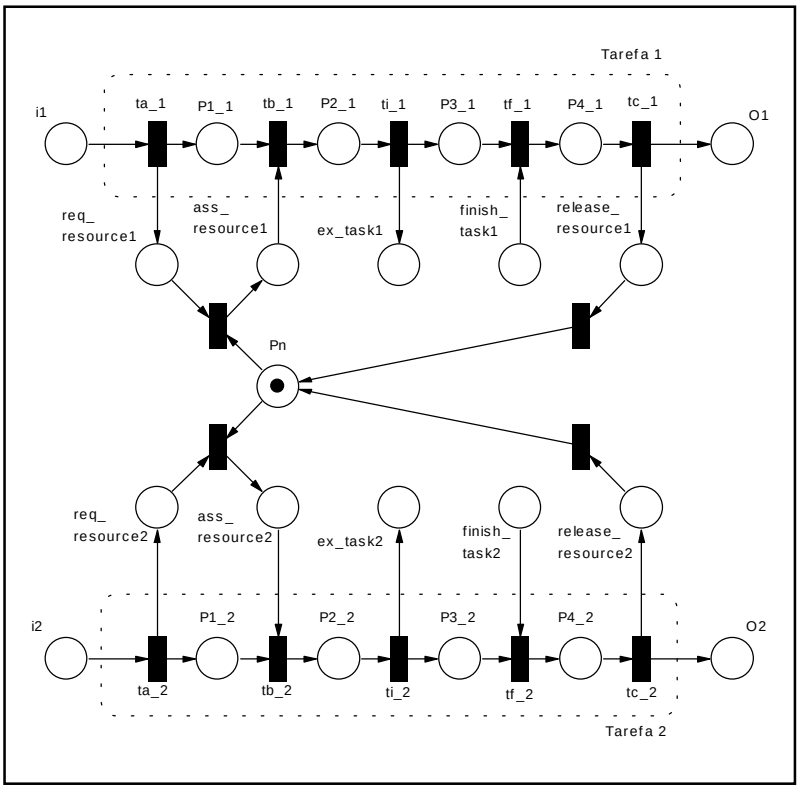

Figura 4: Mecanismo de coordenação para a exclusão mútua (divisão por 1).

Também são definidos mecanismos compostos, a partir das três dependências básicas apresentadas acima, tais como *divisão por M com simultaneidade N* (até M grupos de N tarefas podem compartilhar o recurso) e *simultaneidade M com volatilidade N* (o recurso é disponibilizado para grupos de M tarefas, o que pode ser feito por até N vezes).

4.3. Implementação

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 de coordenação, é proposto um esquema com três componentes: uma 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 linguagem para a definição das dependências estabelece, uma a cada linha, todas as dependências existentes entre as tarefas de uma PN pré-modelada com a ferramenta de simulação. As dependências são definidas de acordo com a seguinte sintaxe `<nome da dependência> [parâmetros] “<nome da tarefa1>” “<nome da tarefa2>” [“<nomes das demais tarefas>”] [<timeouts>]`, onde os parâmetros são necessários para as dependências de gerenciamento de recursos, e os *timeouts* são opcionais.

A primeira implementação usa a ferramenta *freeware* para simulação de PNs Visual Simnet [Garbe 97], que possui editor gráfico para a modelagem das PNs, vários recursos para análise e um formato textual bastante simples para a definição das PNs que permite a exportação para outras ferramentas.

A partir do arquivo que define as dependências e do modelo inicial (nível de *workflow*) criado no Visual Simnet, foi implementado um programa para gerar uma nova PN no nível de coordenação, cujo comportamento pode ser analisado para a detecção de possíveis problemas nas interdependências entre as tarefas.

5. Exemplo

Para ilustrar o uso dos mecanismos de coordenação, é apresentado nesta seção um exemplo modelando uma situação de autoria colaborativa. 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.

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 5. Apesar deste cenário não tratar em detalhe o funcionamento de um sistema de autoria colaborativa, ele mostra como os mecanismos de coordenação podem ser utilizados em uma situação prática.

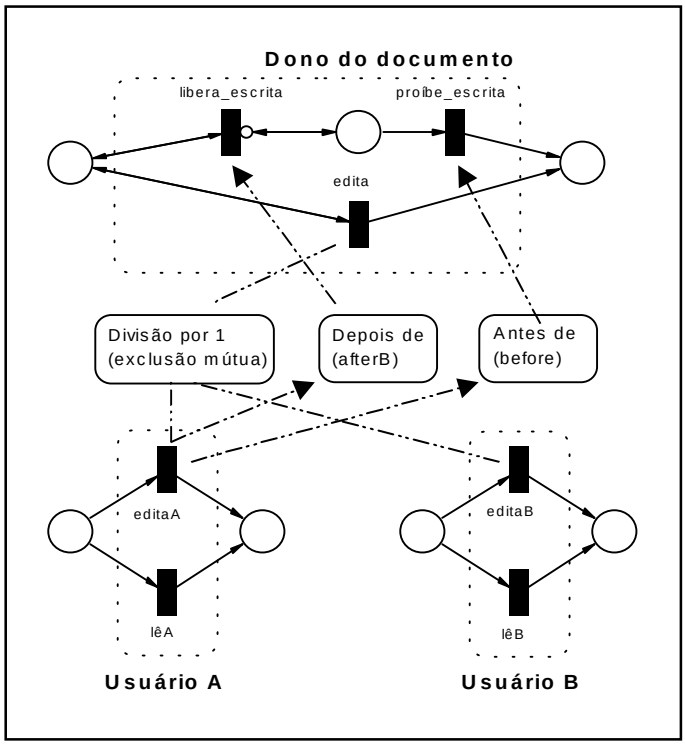

Figura 5: Exemplo modelado no nível de *workflow*.

A construção do nível de coordenação deste exemplo usando a aplicação apresentada na seção anterior é feita com o seguinte arquivo de dependências:

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

O modelo no nível de coordenação é mostrado na Figura 6. Apesar de parecer complexa, a rede mostrada nesta figura é bastante modular e facilmente montada a partir da rede anterior (Figura 5) e dos mecanismos de coordenação pré-definidos. Com relação à expansão das tarefas, deve ser observado que nem todas as tarefas são expandidas exatamente como mostrado na Figura 2. O motivo é que esta expansão pode ser simplificada se uma tarefa possuir apenas dependências temporais ou de gerenciamento de recursos. Apenas as tarefas que possuírem os dois tipos de dependências são expandidas por completo (por exemplo, a tarefa “editaA”).

Para a PN do nível de coordenação, 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.

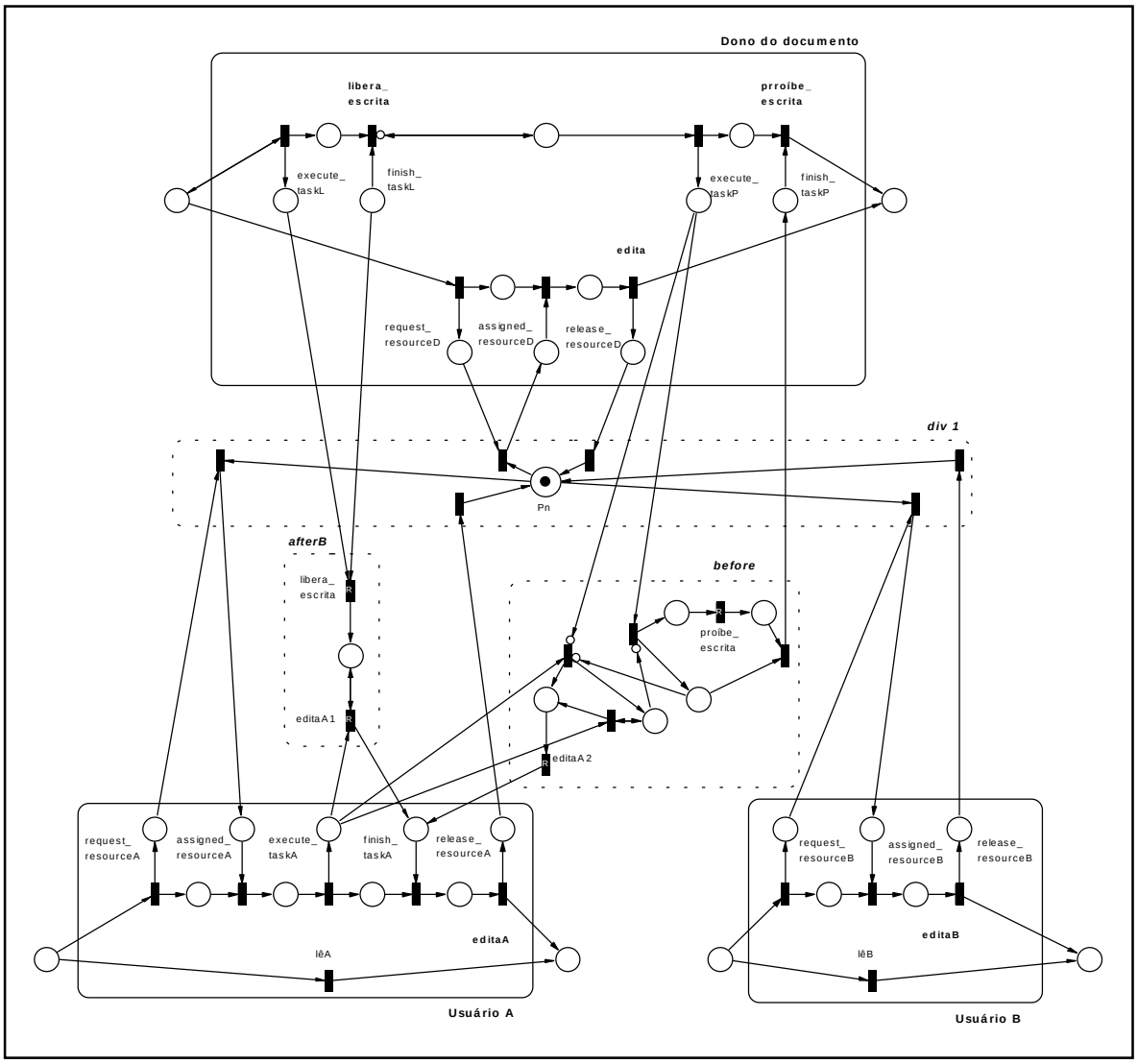

Figura 6: Exemplo modelado no nível de coordenação.

6. 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. Além disso, a uniformidade do modelo facilita a padronização, desde que os elementos necessários sejam incorporados aos modelos de referência (por exemplo, [WfMC 98] e [WfMC 99]).

O conjunto de mecanismos de coordenação apresentado neste artigo não pretende abranger todas as possíveis dependências entre tarefas. A partir do uso, surgirão naturalmente novas dependências e mecanismos de coordenação que poderão ser agregados a este conjunto.

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* e coordenação). 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 [Murata 89] podem simplificar os mecanismos de coordenação para minimizar este problema. Também pretende-se estudar a possibilidade de aplicação de metodologias como CPM (*Critical Path Method*) e PERT (*Program Evaluation and Review Technique*) [Nilsson 86] para concentrar o esforço de coordenação nas atividades do caminho crítico.

Devido à sua generalidade, os mecanismos propostos têm se mostrado adequados para uma série de sistemas colaborativos, desde *workflows* interorganizacionais [Raposo 00] (pretende-se disponibilizar estes mecanismos em sistemas de *workflow* atuais, tais como o Meteor [Meteor 99]) até ambientes virtuais colaborativos. Atualmente os mecanismos de coordenação estão sendo implementados para a utilização em ambientes virtuais colaborativos, onde usuários remotos estão simultaneamente presentes em um ambiente virtual 3D e podem interagir com objetos desse mundo e com os outros usuários [Benford 95]. 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.

Agradecimentos. O primeiro autor é bolsista de doutorado da FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), processo número 96/06288-0. Além da FAPESP, os autores agradecem ao DCA – FEEC – Unicamp pelo apoio expressivo dado à realização deste trabalho.

Referências

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

[Bannon 91] L. J. Bannon and K. Schmidt. *CSCW: Four Characters in Search of a Context*. In J. M. Bowers and D. Benford (Eds.). *Computer Supported Cooperative Work*, pp. 3-16. North-Holland, 1991.

[Benford 95] S. Benford et al. Networked Virtual Reality and Cooperative Work. *Presence*, 4(4): 364-386. MIT Press, Fall 1995.

[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.

[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. Springer-Verlag, 1993.

[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.

[Gelernter 92] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2): 97-107. February 1992.

[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.

[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.

[Magalhães 98] 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.

[Malone 95] T. W. Malone, K-Y. Lai and C. Fry. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *ACM Transaction on Information Systems*, 13(2): 177-205. April 1995.

[Merz 97] M. Merz, B. Liberman and W. Lamersdorf. Using Mobile Agents to support Interorganizational Workflow Management. *International Journal of Applied Artificial Intelligence*, 11(6). September 1997.

[Meteor 99] LSDIS – Dept. of Computer Science – Univ. of Georgia. *METEOR$_2$ (Managing End-To-End OpeRations)*. <http://lsdis.cs.uga.edu/proj/meteor/meteor.html>

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

[Nilsson 86] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Pubs, 1986.

[Raposo 00] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments. Aceito para o *Int. J. of Computer Systems Science & Engineering, Special Issue on Flexible Workflow Technology Driving the Networked Economy*. CRL Publishing, September 2000.

[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.

[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.

[WfMC 96] Workflow Management Coalition**.** *Interface 4 – Interoperability Abstract Specification*, WFMC-TC-1012 (October 1996). <http://www.aiim.org/wfmc/standards/ docs/if4-a.pdf>

[WfMC 98] Workflow Management Coalition**.** *Interface 5 – Audit Data Specification – Version 1.1*, WFMC-TC-1015 (September 1998). <http://www.aiim.org/wfmc/standards/ docs/if4v11b.pdf>

[WfMC 99] Workflow Management Coalition**.** *Interface 1 – Process Definition Interchange Process Model – Version 1.1*, WFMC-TC-1016-P (October 1999). <http://www.aiim.org/ wfmc/standards/docs/if19910v11.pdf>

[Winograd 86] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, 1986.

Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach

Alberto B. Raposo, Léo P. Magalhães and **Ivan L. M. Ricarte**
State University of Campinas (UNICAMP)
School of Electrical and Computer Engineering (FEEC)
Department of Computer Engineering and Industrial Automation (DCA)
CP 6101 – 13083-970 – Campinas, SP, Brazil
{alberto, leopini, ricarte}@dca.fee.unicamp.br

ABSTRACT

The coordination of interdependencies among activities in collaborative environments is a very important and difficult task. In this paper we present a set of coordination mechanisms for the specification and control of interaction among collaborative activities. To model these mechanisms, we use high level Petri nets, which have proven to be an adequate approach to evaluate the behavior of a computer supported collaborative system before its implementation.

Keywords: Coordination, Collaborative Environments, Petri Nets, Computer Supported Cooperative Work, Multiuser Interaction.

1. INTRODUCTION

Some activities involving multiple individuals do not require a formal planning. Activities associated to the social relations are generally well coordinated by the "social protocol", which is characterized by the absence of any coordination mechanism among activities, trusting the participants' abilities to mediate interactions. Examples of computer supported activities of this kind are the chats and videoconferences.

On the other hand, activities related to cooperative *work* (not social relations) require sophisticated coordination mechanisms to avoid that participants get involved in conflicting or repetitive tasks.

This paper focuses on the coordination of activities in computer supported collaborative environments, defining a set of interdependencies that frequently occur among collaborative tasks and presenting coordination mechanisms for these dependencies. The idea is to separate activities (tasks) from dependencies (controlled by the coordination mechanisms), enabling the use of different coordination policies in the same collaborative environment, by changing only the coordination mechanisms. Moreover, the coordination mechanisms can be reused in other collaborative environments.

This paper is organized as follows. In the next section we introduce high level Petri nets, which is the tool we use to model the coordination mechanisms. Then, in Section 3, we make a brief overview of works related to coordination in Computer Supported Cooperative Work. In Section 4 the dependencies and coordination mechanisms are presented, and in Section 5 an example of use of the mechanisms is shown. The conclusions and future work are discussed in Section 6.

2. HIGH LEVEL PETRI NETS

Petri nets (PNs) are a modeling tool applicable to a variety of fields and systems, specially suitable for systems with concurrent events. Murata [11] presents a very good introduction to the theme. Formally, a PN is defined as a 5-tuple (P, T, F, w, M_0), where: $P = \{P_1, ..., P_m\}$ is a finite set of places; $T = \{t_1, ..., t_n\}$ is a finite set of transitions; $F \subseteq (P \times T) \cup (P \times T)$ is a set of arcs; $w: F \rightarrow \{1, 2, ...\}$ is a weight function; $M_0: P \rightarrow \{0, 1, 2, ...\}$ is the initial marking; with $(P \cap T) = \varnothing$ and $(P \cup T) \neq \varnothing$.

In a PN model, states are associated to places and tokens, and events to transitions. A transition t is said to be enabled if each input place $P_i \in \bullet t$ is marked with at least $w(P_i, t)$ tokens, where $w(P_i, t)$ is the weight of the arc between P_i and t. Once enabled, a transition will fire when its associated event occurs. Firing transition t, $w(P_i, t)$ tokens are removed from each input place P_i and $w(t, P_o)$ tokens are added to each output place $P_o \in t\bullet$. Here, $\bullet t$ and $t\bullet$ means, respectively, the set of input and output places of transition t.

A useful notation for PNs is the graphical notation, where circles represent places, rectangles represent transitions, dots represent tokens and arrows represent the arcs, with weights above. By definition, an unlabeled arc has weight 1.

In addition to the basic PN model, several extensions appears in the literature. In this paper we use three of them: inhibitor arcs, nets with time and high level nets. An inhibitor arc connects a place P with a transition t and enables t only if P has no tokens. In the graphical notation, inhibitor arcs are represented with a circle on the edge. One way to include the notion of time in PNs is to associate it with transitions firing. In this case, tokens are removed from input places of a transition and some time later (firing time) are added to the output places. This kind of non-instantaneous firing is called firing with token reservation.

High level PNs include, among other types, predicate/transition nets and colored PNs [4]. The most important characteristic of a high level PN is the distinction of tokens (called colored tokens). The arcs have labels defining variables (or constants) that dictate how many and which kind of tokens will be removed from or added to the places. The same variable appearing in the incoming and outgoing arcs of a transition denotes the same token type. A transition is enabled if there is at least one possibility of consistent substitution of variables into typed tokens. In the example of Figure 1, transition *t1* is enabled because there are two tokens of type *a* in *P1*, being possible to substitute variable <*x*> for type *a*. After the firing of *t1*, *P1* remains with token *b* and *P2* receives a token of type *a*.

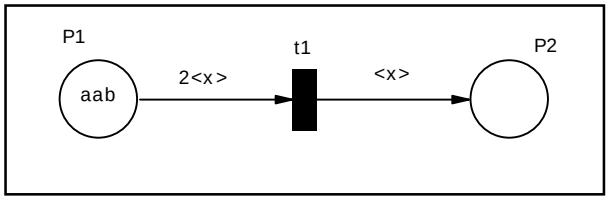

Figure 1: Example of a high level PN.

Besides their modeling capabilities, PNs have also a strong theoretical support for analysis and a number of simulation techniques. There are three kinds of analysis applicable to a PN model; verification, validation and performance analysis [2]. The verification analysis is used to guarantee that the net is correctly defined. It is verified whether the net has deadlocks, whether there are dead transitions, whether it reaches any undesired state, among others. In the validation analysis, it is checked whether the model works as expected. Tests are made via iterative simulations of fictitious cases to ensure that the model treats them correctly. Finally, the performance analysis evaluates the capacity of the system to achieve requisites such as average waiting time, throughput times, resource use, and so on.

3. RELATED WORK

In spite of their recognized importance, coordination mechanisms have not been included in the first collaborative systems. Only in the second half of the 80's the first systems with some kind of coordination mechanism were developed. One of the most representative system of that generation was the Coordinator [13], which has been developed based on linguistic theories and whose goal was to help email communication. Since that time, there has been examples of the use of PNs to coordinate activities in collaborative environments [6], [9].

In the 90's, systems have been constructed with more flexible and accessible coordination mechanisms. Inspired by coordination languages [8], which proposed the separation of computation from coordination for multi-threaded applications, some collaborative systems separates the implementation of coordination components from their other parts. This allows more flexibility in the use of coordination policies. COCA (Collaborative Objects Coordination Architecture) [10] and Trellis [7] are examples of systems of this kind. The Trellis, in particular, uses a variation of PNs in a server to specify interaction protocols for a group of collaborative clients.

Our work is more generic than those presented above. We define a set of interdependencies among tasks and associated coordination mechanisms (modeled by high level PNs) that can be used in workflows, multiuser interaction, virtual environments, etc. The main goal is not to implement a closed system, but to provide a set of mechanisms to enable the collaborative system designer to preview and test its behavior before implementing it, detecting possible problems.

4. COORDINATION MODELS

This work intends to provide mechanisms to manage interdependencies among collaborative tasks and guarantee that these dependencies will not be violated. The idea is that the designer of a collaborative environment be concerned only with the definition of tasks and their interdependencies, and not with the management of those dependencies.

In the proposed schema, an environment is modeled in two distinct levels, *workflow* and *coordination*. In the *workflow* level, the tasks and their interdependencies are defined. In the *coordination* level, interdependent tasks are expanded and the adequate coordination mechanisms are inserted among them.

During the passage from the workflow to the coordination level, each task which has an interdependency with another is expanded according to the model of Figure 2 [1]. The five places associated to the expanded tasks represent the interface with the resource manager and the agent that executes the tasks. Place *request_resources* indicates to the resource manager that the task needs a resource. After the assignment of the resource, the manager puts a token in place *assigned_resources* to continue the task. Places *start_tasks* and *finish_tasks* indicates, respectively, the beginning and the end of the task. Finally, *release_resources* indicates to the resource manager that the task has finished and the resource is available again.

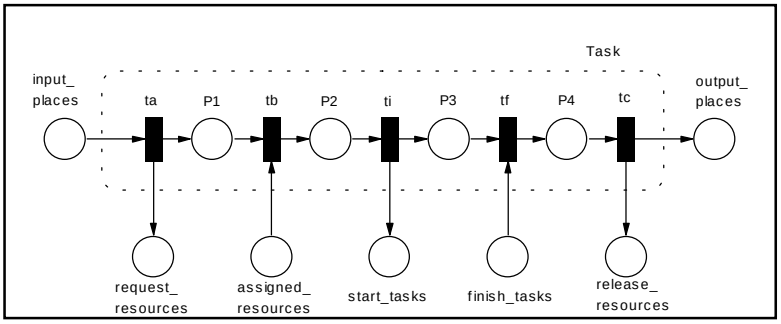

Figure 2: An expanded task in the coordination level.

The main goal of our work is to construct the coordination level from the workflow level, because once the interdependencies are defined, the expansion of tasks according to the model of Figure 2 and the insertion of coordination mechanisms can be automated.

The interdependencies presented below are divided into two classes: temporal and resource management.

Temporal Dependencies

Temporal dependencies establish the execution order of tasks. Coordination mechanisms associated to this kind of dependency have *start_tasks* as input place and *finish_tasks* as output place (Figure 2).

The proposed mechanisms are based on temporal relations defined in a classic paper of temporal logic [3]. In this paper, a set of primitive and mutually exclusive relations between time intervals is defined. We adapted these relations for the definition of temporal dependencies between tasks in collaborative environments, adding a couple of new relations and a few variations of those originally proposed. The following temporal dependencies are defined.

- *task 1 equals task 2*: both tasks must be executed simultaneously.
- *task 1 starts task 2*: Allen's original definition establishes that both tasks must start together and task 1 must finish before task 2 (we called this relation *startsA*). We also relaxed the second part of the definition, creating a variation of the relation in which it does not matter which task finishes before (*startsB*).
- *task 1 finishes task 2*: the original definition establishes that both tasks must finish together, but task 1 has to start after task2 (*finishesA*). As in the previous case, we created a new dependency relaxing the restriction on which task must start before (*finishesB*).
- *task 2 after task 1*: task 2 may only be executed after the execution of task 1. Two variations are possible for this relation. In the first one (*afterA*) each execution of task 1 enables a single execution of task 2. In the second variation, several executions of task 2 are enabled after a single execution of task 1.
- *task 1 before task 2*: from the temporal logic point of view, this dependency can be seen as the opposite of the previous one, but it generates a totally different coordination mechanism. Essentially, the difference is because in this case, the restriction occurs in the execution of task 1, which may not be executed anymore if task 2 has already been executed. Here, task 2 does not wait for the execution of task 1, which was the case for *task 2 after task 1*.
- *task 1 meets task 2*: task 2 starts immediately after the end of task 1.
- *task 1 overlaps task 2*: the original definition establishes that task 2 must start before the end of task 1, which must finish before task 2 (*overlapsA*). Relaxing the restriction that task 1 must finish first, we defined a variation of this relation (*overlapsB*).
- *task 2 during task 1*: two variations are possible. In the first one (*duringA*), task 2 can be executed only once during the execution of task 1. In the second one (*duringB*), task 2 can be executed several times.

Since the goal of the coordination mechanisms is to deal with relations that sometimes belongs to complex procedures, it is interesting to add mechanisms to avoid frequent deadlocks. One of such mechanisms is the use of timeouts. We defined two kinds of timeouts. In the first one (*timeoutA*), an alternative task is defined if the original one does not start after a certain waiting time. The other kind of timeout (*timeoutB*) returns the tokens to the input places of the task after a waiting time, enabling the following of alternative paths that do not execute the blocked task.

Resource Management Dependencies

Coordination mechanisms for resource management are complementary to those presented in the previous section and can be used in parallel to them. This kind of coordination mechanism deals with the distribution of resources among the tasks, and have request_resources and release_resources as input places and assigned_ resources as output place (Figure 2). We define three basic mechanisms for resource management:

- Sharing: a limited number of resources needs to be shared among several tasks.
- Simultaneity: a resource is available only if a certain number of tasks requests it simultaneously.
- Volatility: indicates whether, after the use, the resource is available again.

We have also defined composite mechanisms from the basic ones discussed above. For example, sharing M + volatility N indicates that up to M tasks may share the resource, which can be used N times only.

Differently from temporal dependencies, resource management dependencies are not binary relations. It is possible, for instance, that more than two tasks share a resource. Moreover, each of the above mechanisms requires a parameter indicating the number of resources to be shared, the number of tasks that must request a resource simultaneously, or the number of times a resource can be used (volatility).

A Language for the Definition of Interdependencies

In order to define the interdependencies among tasks, we have created a language that describes, one at each line, all the dependencies of a collaborative environment. The interdependencies are defined according to the following syntax `<dependency name> [parameters] "<task1 name>" "<task2 name>" ["<taskn name>"] [timeouts]`, where parameters are needed for resource management dependencies and the list of timeouts is optional.

Modeling Coordination Mechanisms using High Level Petri Nets

Initially, we have modeled coordination mechanisms for all dependencies discussed above using ordinary PNs [12]. However, a typical problem in the use of ordinary PNs is the state explosion, which can occur in our context when the number of interdependencies increases. High level PNs reduce this problem because they generate simpler models, with less places and transitions. Therefore, we remodeled them using high level PNs.

To illustrate one of the coordination mechanisms, Figure 3 shows the mechanism for simultaneity 2 (a resource is available only if two tasks request it simultaneously). In the figure, the arcs with expression *<x>+<y>* ensure that two different tasks (tokens of different colors) which are requesting the resources (tokens *r*) are going to receive them if they are available in place *Pn*.

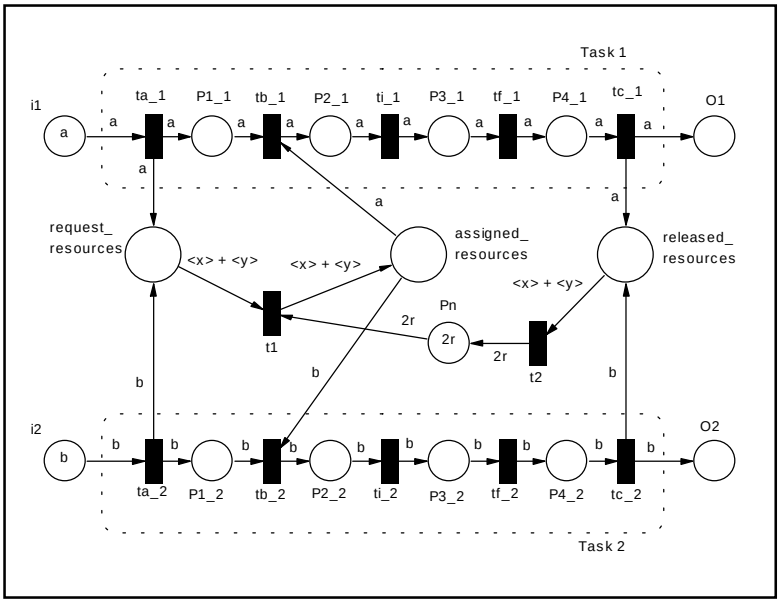

Figure 3: Coordination mechanism for *simultaneity 2*.

5. EXAMPLE

To illustrate the use of the coordination mechanisms, we present an example describing a typical situation in multiuser interaction. In our hypothetical environment, two users interact by means of a whiteboard. The workflow level of this environment is shown in Figure 4. As can be seen in the model, each user can "enter" the whiteboard, then write on it several times before "leaving" it. The following interdependencies appears in the model. The whiteboard is available only if both uses request it simultaneously (simultaneity 2); only one user may write on the whiteboard at each time (sharing a resource, e.g., a pen); and when user A leaves the whiteboard, user B must follow him/her (meets).

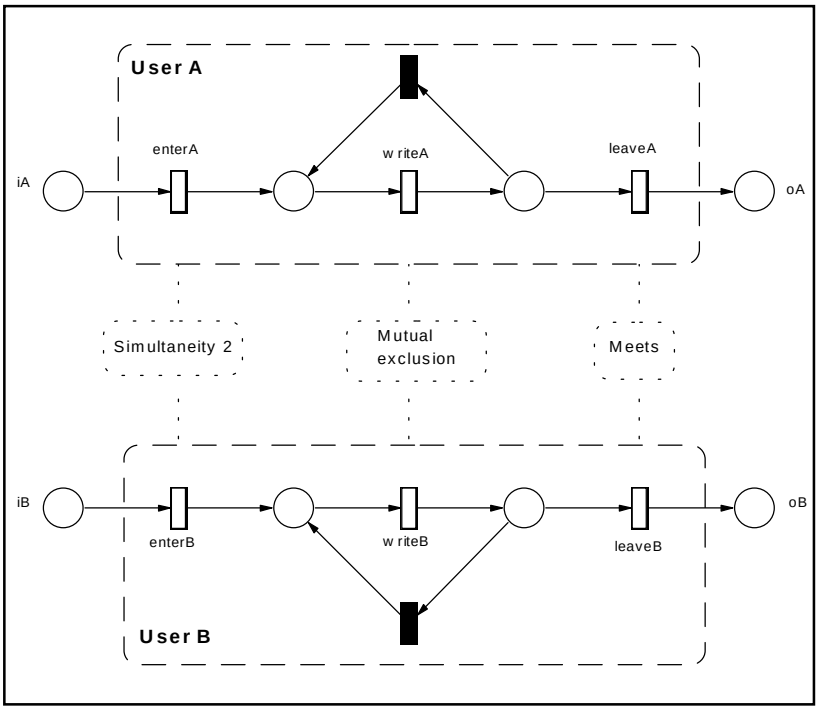

Figure4: Workflow level of the example.

Using the language for the definition of interdependencies, the following file is written for the example above:

```
sim 2 "enterA" "enterB" time_outB time_outB
div 1 "writeA" "writeB" time_outB time_outB
meets "leaveA" "leaveB"
```

The model of the coordination level for this example is shown in Figure 5. At first glance, this model may seem complicate, but it is highly modular and easily built from the model of Figure 4, by expanding interdependent tasks (open rectangles in Figure 4) and inserting the predefined coordination mechanisms.

It is necessary to observe that interdependent tasks of Figure 5 are not expanded exactly according to the model of Figure 2. The reason is that in the case of temporal dependencies, only places *start_tasks* and *finish_tasks* are used, and in the case of resource management dependencies, only *request_resources*, *assigned_resources* and *release_resources* are used. Therefore, we used simplified versions of the model.

The coordination level model of the environment can be simulated and analyzed with any tool that supports high level PNs (see [5] for a list of PN simulation tools).

Verification and validation analysis indicated that our examples has eight final states. Seven of them can be eliminated by the correct use of timeouts (i.e., with adequate waiting times). The eightieth final state is the correct one (tokens in *oA* and *oB*). Performance analysis could be realized, for instance, to measure average waiting times of an user, given the rates with which the other requests the resources.

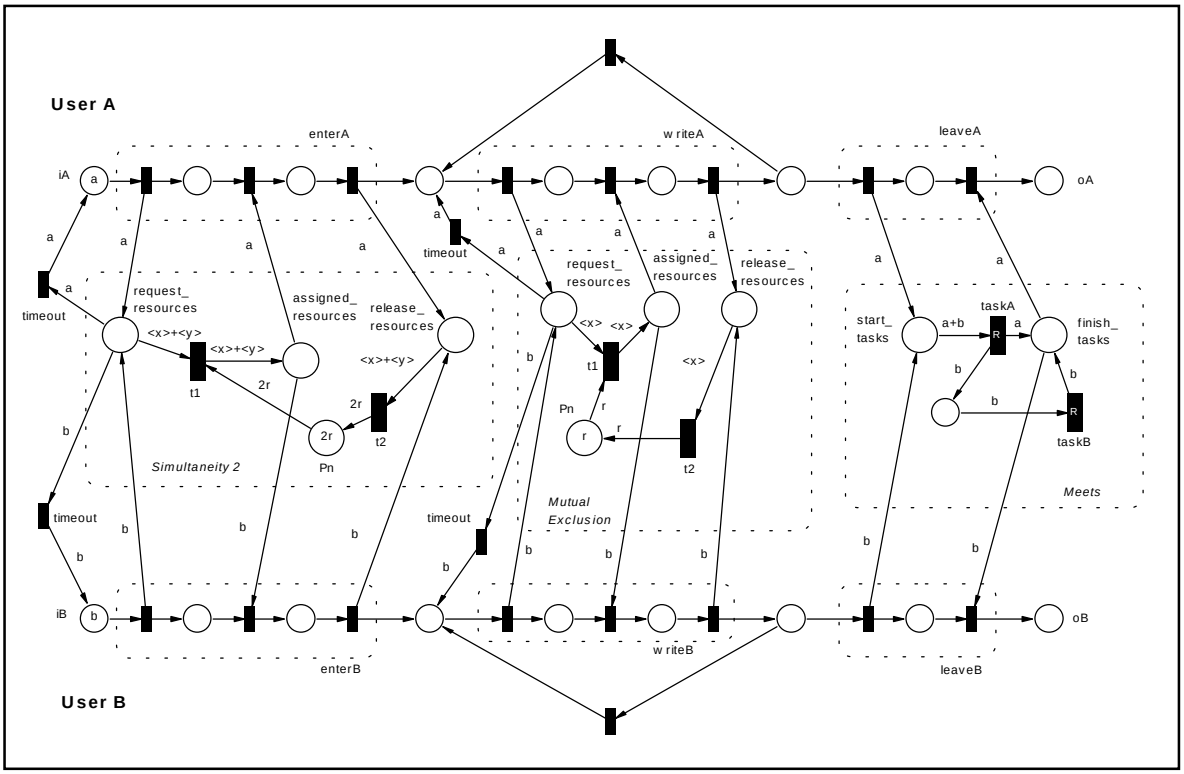

Figure 5: Coordination level of the example.

Finally, it is necessary to reinforce that the example represents just a hypothetical situation. We did not stress all details for the modeled scenario. Its main goal was to show how the coordination mechanisms can be used in a practical situation.

6. CONCLUSION

The coordination of interdependent activities in collaborative environments is a problem that should be addressed to ensure the effectiveness of the cooperation. The separation between activities and dependencies, and the utilization of reusable coordination mechanisms are steps towards this goal.

Petri nets, due to their support for modeling, simulation and analysis, have proven to be a powerful tool for verifying the correctness and validating the effectiveness of collaborative environments before their actual implementation [2], [12]. Furthermore, the hierarchical description of PNs showed an appropriate way to define the coordination structure in different abstraction levels (workflow and coordination levels). In particular, high level PNs also reduces the problem of state explosion.

The set of interdependencies presented in this paper does not claim to be complete. It would be very difficult to establish a framework of all possible interdependencies. For that reason, we opted for an extensible approach. When a new kind of interdependency arises, a corresponding coordination mechanism can be modeled and easily inserted between corresponding tasks.

One of the next steps of this work is to automate the passage from the workflow to the coordination level of models in a high level PN simulation tool. We have already done this for ordinary PN models [12].

Due to their generality, the presented coordination mechanisms are adequate to a wide range of collaborative systems, from interorganizational workflows to virtual environments. Presently, we are implementing the mechanisms to be used in the development of collaborative virtual environments. The coordination of activities will facilitate the use of this kind of environment for the realization of tasks that cannot be controlled by the social protocol.

7. REFERENCES

[1] W. M. P. van der Aalst. Modelling and analysing workflow using a Petri-net based approach. *Proc. 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pp. 31-50. 1994.

[2] 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.

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

[4] W. Brauer, W. Reisig and G. Rozenberg (Eds.). *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986*. Lecture Notes in Computer Science, 254. Springer-Verlag, 1986.

[5] CPN Group, Univ. of Aarhus, Denmark. *The World of Petri Nets – Tools on the Web*. 2000. <http://www.daimi.aau.dk/~petrinet/tools>

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

[7] 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.

[8] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2): 97-107. February 1992.

[9] 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.

[10] 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.

[11] T. Murata. Petri Nets: properties, analysis and applications. *Proc. of the IEEE*, 77(4): 541-580. 1989.

[12] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments. To be published in the *Int. J. of Computer Systems Science & Engineering*, September 2000.

[13] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, 1986.

Acknowledgements: The first author is sponsored by FAPESP (Foundation for Research Support of the State of São Paulo), process number 96/06288-9. We also would like to thank DCA – FEEC – Unicamp for the support granted to this research.

Coordination Mechanisms for Collaborative Virtual Environments

Alberto Barbosa Raposo, Léo Pini Magalhães, Ivan L. Marques Ricarte

DCA – FEEC - UNICAMP
{alberto, leopini, ricarte}@dca.fee.unicamp.br

Abstract. *In order to be effectively used as collaborative work tools, developers of virtual environments should invest, among other aspects, in the coordination of users' activities. The goal of this work is to present coordination mechanisms that may be reused in different implementations of collaborative virtual environments (CVEs).*

1. Introduction

In CVEs, users are simultaneously present and can interact with objects and other users. Currently, the development of CVEs has been dominated by leisure activities, enabling basically navigation through virtual scenarios and communication with remote users [Frécon 98]. This kind of activity is well coordinated by the "social protocol", characterized by the absence of any coordination mechanism, trusting the participants' abilities to mediate interactions. However, activities related to cooperative work require sophisticated coordination mechanisms to avoid that participants get involved in conflicting or repetitive tasks.

This paper focuses on the coordination of activities in CVEs, defining a set of interdependencies that frequently occur among collaborative tasks and presenting coordination mechanisms for them. The idea is to separate activities from dependencies (controlled by the coordination mechanisms), enabling the use of different coordination policies in the same CVE by changing only the coordination mechanisms. Moreover, these mechanisms are generic and can be reused in other CVEs.

2. Coordination Mechanisms

Coordination is "the act of managing interdependencies between activities performed to achieve a goal" [Malone 90]. Therefore, this work started with the definition of a set of frequent interdependencies between cooperative tasks. The next step was the modeling of coordination mechanisms to guarantee that those dependencies will not be violated. The final step is the implementation of those mechanisms in CVEs. The idea is that the designer of a CVE be concerned only with the definition of tasks and their interdependencies, and not with the management of those dependencies.

Interdependencies were divided into two main classes: temporal and resource management. Temporal dependencies establish the execution order of tasks, while resource management ones deal with the distribution of resources needed to the execution of a task. A total of 20 dependencies were defined [Raposo 00].

Petri nets (PNs) were used to model the coordination mechanisms. The graphical representation of PNs is simple and offers an adequate hierarchical description to define the coordination structure in different abstraction levels. PNs, due to their support for modeling, simulation and analysis, are a powerful tool for verifying the correctness and validating the effectiveness of CVEs before their actual implementation. Using this approach, it is possible to predict the behavior of a CVE, avoiding undesired situations. A similar approach were used for computer animations [Magalhães 98].

After the environment has been tested with the PN model, it is necessary to implement the CVE by including the coordination mechanisms to control the tasks' execution. One way to do this is by a "manager program" that interacts with the environment. Blaxxun platform [Blaxxun 00], a system to create CVEs offers the resources for this implementation.

3. Example

To illustrate the use of the coordination mechanisms, consider the hypothetical situation depicted in Figure 1. The figure shows two PNs, each one representing the sequence of tasks of a specific CVE user. During the environment specification, the developer detects interdependencies among tasks. For example, task3B must occur during the execution of task2B and task5A shares a resource with task4B (mutual exclusion).

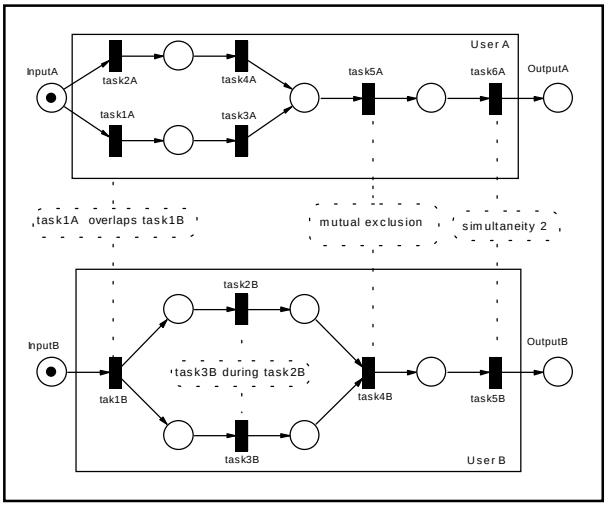

Figure 1. Tasks and interdependencies in a hypothetical CVE.

After the modeling phase, the PN model, with coordination mechanisms included, is tested to verify its correctness. In Figure 1, if user A follows, for example, the path that executes task2A instead of task1A, user B will be blocked in the execution of task1B, which depends on task1A. This indicates that the model should be revised.

When the model is approved, it should be "translated" to the CVE. This is achieved by indicating how tasks are mapped to the CVE (i.e., which events are associated to the tasks) and including the coordination mechanisms of a predefined library.

4. Conclusion

The coordination of activities in CVEs is a problem that should be addressed to facilitate the use of this kind of environment for the realization of tasks that cannot be controlled by the social protocol. The separation between activities and dependencies and the use of coordination mechanisms are steps towards this goal.

The developed coordination mechanisms also revealed generic enough to be used in a wide range of collaborative systems, including workflow systems [Raposo 00].

References

Blaxxun Interactive, Inc. (2000) "Blaxxun Community Platform 4", http://www.blaxxun.com.

Frécon, E. and Nöu, A. A. (1998) "Building Distributed Virtual Environments to Support Cooperative Work", *Proc. VRST'98*, p. 105-119.

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

Malone, T. W. and Crowston, K. (1990) "What is Coordination Theory and How Can It Help Design Cooperative Work Systems?", *Proc. CSCW'90*, p. 357-370.

Raposo, A. B., Magalhães, L. P. and Ricarte, I. L. M. (2000) "Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments", accepted for the *Int. J. of Computer Systems Science & Engineering*.

Pergamon

Comput. & Graphics, Vol. 22, No. 6, pp. 735–743, 1998

PII: S0097-8493(98)00094-6
Printed in Great Britain
0097-8493/98/$ - see front matter

Technical Section

ANIMATION MODELING WITH PETRI NETS

LÉO P. MAGALHÃES†, ALBERTO B. RAPOSO and IVAN L. M. RICARTE

State University of Campinas (UNICAMP), School of Electrical and Computer Engineering (FEEC), Department of Computer Engineering and Industrial Automation (DCA), CP 6101-13083-970, Campinas SP, Brazil

Abstract—This paper introduces the use of Petri Nets as a modeling and analysis tool for animation environments. Firstly, the original formulation for Petri Nets is applied in two animation situations, one modeled as a state machine and another exploring interdependent transitions. Increasing the complexity level, some modeling extensions are discussed and more sophisticated animation examples are studied.

Key words: animation planning, computer modeled animation, Petri Nets, behavioral animation, animation modeling and analysis.

1. INTRODUCTION

Control modeling is one of the most important aspects in computer animation. It specifies how characters acting in an animation should move and how they interact with the animation environment. Movements of characters in an animation can be defined using parametrical interpolations, kinematic and dynamic equations (direct and inverse), genetic models, etc. [1, 3, 18, 20, 22]. Characters interacting with the environment can be controlled from an antecipative point of view (e.g., interpolation) or detected on the fly using for example collision detection techniques [6, 21].

At a more abstract level one can consider characters subject to emotions, having a reactive behavior, or being oriented towards a specific task or goal. In these cases, control can be based on strategies such as logical description of behaviors [8, 16], or systems of kinematic or dynamic equations [7].

This paper focuses on the support of animation modeling at this more abstract level using a formal framework based on Petri Nets theory. Besides being useful as a tool for modeling animation environments, Petri Nets also offer a powerful contribution for animation analysis.

This paper is structured as follows: Section 2 will introduce Petri Nets. Section 3 will present some initial examples of their use. Section 4 will address the use of Petri Nets in more sophisticated animation problems. The last sections will present the conclusions, next issues and related bibliography.

2. PETRI NETS: FUNDAMENTALS

Petri Nets [17] (from here on, PN) are a modeling tool applicable to a variety of fields and systems, specially suited to systems with concurrent events. Murata [15] presents a very good introduction to the theme. Formally, a PN can be defined as a 5-tuple (P, T, F, w, M_0), where: $P = \{P1, \ldots, P_m\}$ is a finite set of places; $T = \{t_1, \ldots, t_n\}$ is a finite set of transitions; $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs; w: $F \rightarrow \{1,2,\ldots\}$ is a weight function; M_0: $P \rightarrow \{0,1,2,\ldots\}$ is the initial marking; with $(P \cap T) = \varnothing$ and $(P \cup T) \neq \varnothing$.

In a PN model, states are associated to places and marks, and events to transitions. A transition t is said to be enabled if each input place P_i of t is marked with at least $w(P_i, t)$ tokens, where $w(P_i, t)$ is the weight of the arc between P_i and t. Once enabled, a transition will fire when its associated event occurs. Firing the transition t removes $w(P_i, t)$ tokens from each input place P_i of t, and adds $w(t, P_o)$ tokens to each output place P_o of t.

A very useful notation for PN is the graphic notation (Fig. 1) which will be used in the examples throughout this paper. In this notation, circles represent places, bars represent transitions, dots the marks (also called tokens), and arrows the arcs, with weights above. By definition, an unlabeled arc has weight 1.

For example, in the PN of Fig. 1, only transition t_2 is enabled; t_1 is not enabled because it would require two marks in P_1 to fire, since $w(P_1, t_1) = 2$. When t_2 is fired, the marks in P_2 and P_3 are removed and P_4 receives one mark. It should be observed that the number of marks in a PN is not necessarily conserved.

Besides the graphical notation used in this paper, a matrix is also adequate to indicate the possible changes of states in a net. This matrix, C, has dimension $m \times n$, with position $C_{i,j}$ indicating how many tokens place P_i will receive (positive value) or lose (negative value) when transition t_j is fired. Representing a state by a vector marking

†Corresponding author. Tel.: +55-19-788-3706; Fax: +55-19-289-1395; E-mail: leopini@dca.fee.unicamp.br.

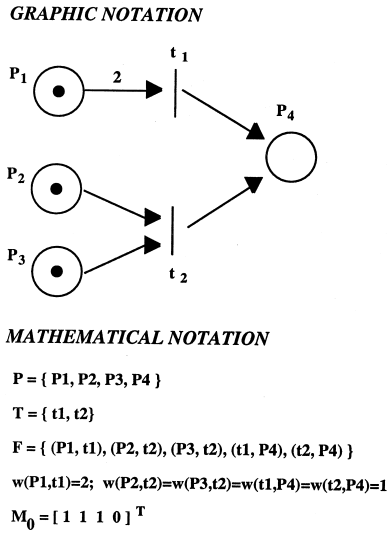

Fig. 1. Petri Nets graphic and mathematical notation

$M_i=[q_1,q_2 \ldots q_m]^T$, where q_i indicates the quantity of tokens in place P_i, the next state after firing transition t_j is given by $M_{i+1} = M_i + C \times e_j$, where e_j is the characteristic vector for transition t_j, a column vector with 1 in position j and 0 in the remaining positions.

Summarizing, the behavior of a system using PN is described in terms of its states and their changes [15]. States are modeled by *places* and *marks*, which define the current state of the system. *Transitions* (firing rules) model the dynamic behavior of the system. *Arcs* indicate the sequence of possible transitions between states and they can be *weighted* meaning the quantity of *marks* necessary to fire a transition.

Besides the modeling capabilities of PN, their support for analysis is very important and useful. This analysis is based on the properties of the mathematical model of PN. Some of these properties are:

Reachability: is there a sequence of firing that reaches a given state?

Boundness: will a place be overloaded? A PN can be defined as k-bounded if the number of tokens in each place does not exceed k.

Liveness: is there any state or sequence of states which will not be reached anymore, indicating a possible deadlock?

Reversibility: is it possible to return to a defined initial state M_0?

Persistence: is the firing of any pair of enabled transitions interdependent, i.e., the firing of one will disable the other?

Synchronic Distance: what is the relationship between two transitions? This metric is related to the degree of mutual dependence between transitions.

3. MODELS FOR SIMPLE ANIMATIONS

The correct modeling and use of PN properties for analysis allow an animator to preview the behavior of an animation even before starting any shot. *Reachability* can be used to detect modeling problems related to defined animation states which will never be reached. *Boundness* is related, e.g., to a number of actors wished in a state. *Liveness* can be used to find states which will never happen if an specified state is reached. *Reversibility* allows to test whether an initial state can be reached from another state. *Persistence* and *synchronic* distance test the interdependence between animation events.

The following two sections introduce the powerful characteristics of PN applied to animation problems by means of simple but clear examples. The benefits that can be achieved in more complex environments will be discussed in Section 4.

3.1. *Single sphere example*

The example of Fig. 2 shows two buttons and a sphere which can follow two different trajectories depending on which button was chosen. One button is associated to an internal trajectory of the sphere and the other with an external one.

For the above example the event of pressing one button could be associated to a warning signal (e.g., button 1 danger, button 2 no problems), signalizing the trajectory to be followed.

The animation is modeled in PN as follows:

Place 1 (P_1) is associated to trajectory A1
Place 2 (P_2) is associated to trajectory A2
Transition 1 (t_1) is associated to the press of button 1
Transition 2 (t_2) is associated to the press of button 2.

Formally, the PN for this example is a 5-tuple (P, T, F, w, M_0), where: $P = \{P_1, P_2\}$ is the set of places; $T = \{t_1, t_2\}$ is the set of transitions; $F = \{(P_1, t_2), (P_2, t_1), (t_1, P_1), (t_2, P_2)\}$ is the set of arcs; $w(f) = 1$ **for all** $f \in F$; $M_0=[1\ 0]^T$ is the initial marking.

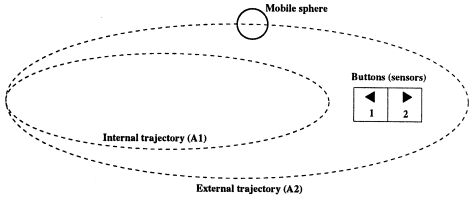

Fig. 2. Basic example

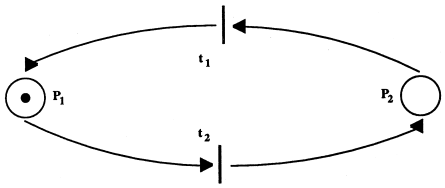

Fig. 3. PN representation of the basic example

Figure 3 shows the graphical representation of the PN model for this animation. The main character of the animation, the sphere, is represented by the mark (dot token) in the Fig. 3.

The graph gives the reader the following direct information. First, there are two stationary states in the environment. When the mark is in place P_1, the sphere follows trajectory A1. Otherwise, when the mark is in place P_2, the sphere follows trajectory A2. Second, the effect of pressing a button is dependent on the current trajectory. For example, if the sphere is following A1 (token in P_1) the press of button 1 has no effect, since t_1 is not enabled. Finally, the movement will remain forever, since the graph does not present any state where the sphere is not moving.

Information can also be obtained by means of an informal analysis of the graph based on the properties of PN:

- All possible states can be reached. The initial state $M_0=[1\ 0]^T$ after the firing of t_2 will be transformed into $M_1=[0\ 1]^T$, which after the firing of t_1 will return to M_0, and so on.
- There are no deadlocks, i.e., the graph is alive. All the possible states (M_0 and M_1) can be reached.
- Whichever the initial state is, it is possible to return to it.
- The transitions are mutually dependent, i.e., it is not possible to fire one of them twice without firing the other. Even though one can press the same button two times sequentially, the second time has no effect in the animation.

This very simple example illustrates the basic mapping between situations in an animation and a corresponding PN model. In the following section the complexity of the presented example will be increased in order to stress the use of PN for the analysis of an animation behavior.

3.2. *Two spheres example*

This section will detail some additional modeling and analysis aspects of PN taking a more complex environment.

The basic example, Fig. 2, will be modified as shown in Fig. 4. Now there are two spheres, each one travelling in one of the two possible trajectories (internal or external) controlled by a button. The animation has a behavioral restriction defined by the rule that only one of the spheres can be at its external trajectory at a time in order to avoid collisions.

Figure 5 introduces the PN model for this example. The following characteristics are modeled:

- There are four places defining the two possible trajectories for each sphere, $P = \{A_1, A_2, B_1, B_2\}$.
- There are four transitions defining button press events, $T = \{t_1, t_2, t_3, t_4\}$,where t_1 and t_4 are associated to the press of the buttons that put the spheres A and B, respectively, in their internal trajectories, and transitions t_2 and t_3 are associated to the press of the buttons that put the spheres A and B, respectively, in their external trajectories.
- All arcs have unit weight.

Let the initial marking be $M_0=[1\ 0\ 1\ 0]^T$. It can easily be seen that in this state only transitions t_2 and t_3 are enabled, and that firing t_2 will lead to a state $M_1=[0\ 1\ 1\ 0]^T$ whereas firing t_3 will lead to $M_2=[1\ 0\ 0\ 1]^T$. From M_1, the only possible transition, t_1, takes the net back to M_0. The same happens from M_2, for which the only possible transition is t_4, which also takes the net back to the initial marking when fired. Therefore, the complete set of states for this net is $M = \{M_0, M_1, M_2\}$.

A powerful tool for the analysis of PN is the coverability graph which offers a vision of the complete sequence of transitions and states in a PN. Figure 6 illustrates the coverability graph for this example, considering the initial state M_0.

Based on the coverability graph and taking into account the PN properties, the following can be stated:

> **Reachability**: Starting at one of the states of M the system never goes to the forbidden state $[0\ 1\ 0\ 1]^T$, which might cause a collision, or to the impossible states $[1\ 1\ 0\ 0]^T$ and $[0\ 0\ 1\ 1]^T$, in which one sphere would follow two or no trajectories.

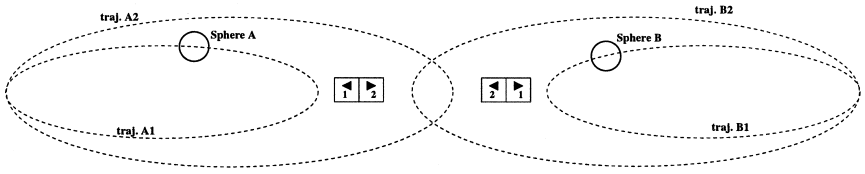

Fig. 4. Two spheres example

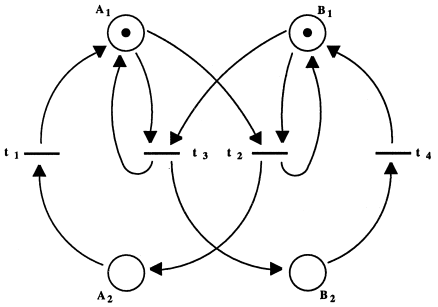

Fig. 5. PN for the example with 2 spheres

All other states can be obtained by a given firing sequence.

Boundness: the PN is 1-bounded, i.e., no place will have more than one mark at a time. A PN with this property is called *safe*.

Liveness: each valid state has a following valid state for a firing sequence, i.e., there are no deadlocks. In addition, all transitions appear in the coverability graph, meaning there are no *dead* transitions.

Reversibility: it is always possible to return to an initial state by a firing sequence.

Persistence: This PN is not persistent. The firing of t_2 will inhibit t_3 and vice versa. This expresses the mutual exclusion relationship between both events.

Synchronic distance: this characteristic expresses the level of mutual dependence between two transitions. Considering σ a firing sequence at any marking in M and $\sigma(t_i)$ the number of times t_i fires in σ, $d_{i,j} = \max|\sigma(t_i) - \sigma(t_j)|$.

Regarding the synchronic distance, $d_{1,2} = d_{3,4} = 1$ shows that these pairs of transitions are interdependent—in fact, the transitions of each of these pairs are associated to events of the same sphere. On the other hand, $d_{1,4} = d_{2,3} = \infty$ shows that these pairs of transitions are associated to independent events, which should be true since they are associated to events of different spheres.

It can be concluded from the above points that:

1. The developed model has no undesired states. It conforms with the initial animation description.
2. There are no deadlocks, since the control statements will always put the animation in a valid state.
3. The animation will run forever, since there is no final state.
4. There are two mutually excludent transitions, confirming the behavioral restriction.
5. The only mutually dependent transitions, with $d_{i,j} = 1$, are associated to events of the same object, either sphere 1 or sphere 2.

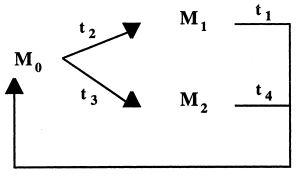

Fig. 6. Coverability graph for the example

The PN model for these animations associates transitions with user intervention (pressing buttons). If autonomous behavior were required, PN extensions would have to be used, as described in the next section.

4. PN EXTENSIONS FOR COMPLEX ANIMATION ENVIRONMENTS

The previous section introduced the use of PN in animation environments. The first example presented a state-machine PN which is a subclass of PN for which each transition has exactly one input and one output. The second example introduced interdependent transitions in a more complex environment.

In this section, the reuse of PN models will be emphasized by means of an example where the simple model of Section 3.1 is reused in a typical computer animation situation. After that, the notion of time is introduced in PN by the use of a PN extension.

4.1. *Dancer example*

A classical example in the computer animation field is the control of the walking movement of a biped structure such as a human being. The walking movement must satisfy the constraint that both legs cannot be out of the ground at the same time. This movement can be modeled by the PN of Fig. 5 considering that each ball now represents a leg, the internal trajectory represents the leg on the ground, and the external trajectory represents the leg off the ground. However, the walking movement also requires the legs to be raised alternately. (It does not make sense to raise the same leg two times sequentially.) Due to this additional restriction, the walking movement of a biped can be now easily modeled by a state machine similar to that of Fig. 3, with four states: left leg up, left leg down, right leg up, and right leg down.

A more challenging example is to define the movements of a ballet dancer. In this model, the dance consists of two basic and nonsequential movements: walking and jumping. In addition, the dancer can tiptoe or have the feet in the normal position. The supposed choreography requires that the dancer jumps only when tiptoeing.

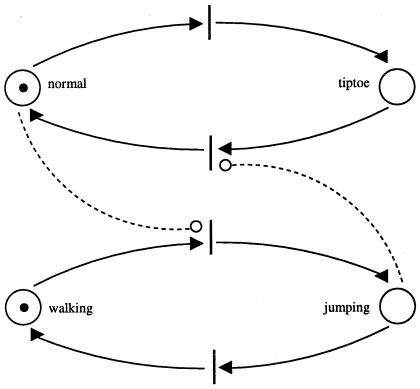

Fig. 7. PN graph for the dancer example

The above situation can be modeled using two PN similar to that of Fig. 3, one modeling the feet position and another modeling the dancer's movement. The PN are interconnected by inhibitor arcs, a tool to synchronize interdependent events [10, 13]. In Fig. 7, inhibitor arcs are represented by dashed lines with a circle at the end. This kind of arc enables the inhibited transition only when its input place becomes empty. The inhibitor arcs connecting the PN guarantee that the dancer will jump only when tiptoeing.

This example illustrates not only the reuse of a basic PN model, but also the capacity of encapsulation offered by PN. The graph of Fig. 7 hides the details of the walking and jumping movements, enforcing a hierarchical description model. The movement of walking, for example, is modeled by a PN state machine as commented before. A further step in this hierarchical description could be the definition of the relationship among the various dancers in a ballet performance, hiding the control of each dancer, as discussed in the next section.

The models presented up to this point do not use the notion of time, e.g., it is not possible to control the duration of the walking or jumping movements. The notion of time is introduced by a PN extension presented in the next section.

4.2. *On stage example*

Generalized Stochastic Petri Nets are derived from PN by associating fire rates to transitions and partitioning the set of transitions into two subsets [13]. This technique enlarges the class of animations that can be modeled by PN introducing the notion of time associated to a transition fire, as will be seen next.

4.2.1. *Generalized stochastic PN*. First of all the PN definition of Section 2 is extended by introducing the set of firing rates, possibly marking-dependent, associated with the PN transitions. For a PN with s transitions the set $R = \{r_1,\ldots,r_s\}$ is defined, where r_i is the rate associated with the firing of transition t_i. This formulation defines the Stochastic Petri Nets (SPN) [2, 5, 14].

Generalized Stochastic Petri Nets (GSPN) [13] are obtained by generalizing SPN, subdividing the set T (see Section 2) into timed and immediate transitions, reducing R to s' elements, where s' is the number of timed transitions. Immediate transitions fire in zero time after enabled, while timed transitions fire after a random exponentially distributed enabling time, introducing an additional and powerful tool for modeling animation environments.

The following statements apply to GSPN:

- If the set of simultaneously enabled transitions H comprises only timed transitions, the enabled timed transition t_i fires with probability $r_i/\Sigma_{k\in H} r_k$.
- If the set of simultaneously enabled transitions comprises both kind of transitions only the immediate ones can fire. In that case if there are more than one immediate transition enabled, it is necessary to associate a probability function to define which one will fire.

In this paper double bars represent timed transitions and immediate transitions are represented by a single bar as in the basic PN graphical representation.

4.2.2. *Modeling*. This section explores some of the additional modeling features offered by GSPN to animation environments.

The following example presents a dance performance with several dancers, each one modeled as described in Section 4.1, sharing a limited number of costumes. Dancers enter and leave the stage at random rates. Two dancers cannot share a costume at the same time and, if all costumes are in use, a dancer has to wait until one of the dancers currently performing finishes in order to get a costume.

Figure 8 is the PN graph modeling the described behavior. The model presents the following states and transitions:

P_1 dancers that are going to perform

P_2 dancers ready to perform

P_3 costumes that are currently available

P_4 dancers on stage, each one with a different costume

P_5 dancers ready to perform but waiting for a costume

t_1 enables dancer

t_2 dancer starts performing

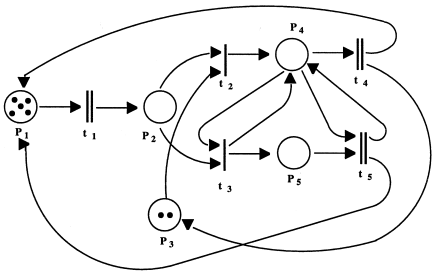

Fig. 8. PN graph for the dancers and costumes example

t_3 dancer has to wait for a costume

t_4 dancer has finished performing

t_5 dancer has finished and is substituted by another one that was waiting for his/her costume.

It is important to note that t_2 and t_3 are immediate transitions firing when their preconditions are fulfilled. On the other hand t_1, t_4, and t_5 are timed transitions with associated random exponentially distributed firing rates.

The initial state is $M_0 = [5\ 0\ 2\ 0\ 0]^T$ defining an environment composed by five dancers and two costumes.

Complementing the ideas discussed by the examples of Section 2, the current example will be analysed by means of a simulation, studying the effect of varying the firing rates in the animation behavior.

4.2.3. *Simulation analysis*. The presented example will be simulated to verify a condition to be avoided: $P_3 > 0$ and $P_5 > 0$. This condition represents what can be considered a bad system behavior, because there is at least one dancer waiting for performing ($P_5 > 0$), although there is (are) available costume(s) ($P_3 > 0$).

For this simulation the following parameters will be considered. λ_1 is the average value of an exponentially distributed random variable that determines the interval between firings of t_1, weighted by q_1 (the number of tokens in P_1). λ_4 and λ_5 are the average values of exponentially distributed random variables that control the interval between successive firings of t_4 and t_5, respectively. The value λ_i defines an average firing rate $1/\lambda_i$.

In the simulation, when both immediate transitions (t_2 and t_3) are enabled, it is defined that t_2 will fire.

In order to demonstrate one of the possible analyses to predict the system behavior, some simulation data will be graphically shown.

Figure 9 shows the behavior of the system for λ_1, λ_4, and λ_5 varying between 0.25 and 2.0 time units. The best system behavior of the current simulation is obtained for $\lambda_4 = 2.0$ and $\lambda_5 = 0.25$. This figure suggests that the rate λ_4/λ_5 defines part of the system behavior. However, the system is also sensitive to the variation of λ_1 as shown by the dashed curves.

This lead up to a further simulation presented in Fig. 10, now including the influence of the rate λ_1/λ_5. For this simulation, the values of λ_4/λ_5 and λ_1/λ_5 are between 0.125 and 8.0. This figure allows a more precise analysis of the system behavior. It can be seen that the system has a worst behavior peak in the region where λ_4/λ_5 and λ_1/λ_5 are smaller than one—for $\lambda_4/\lambda_5 = 0.125$ and $\lambda_1/\lambda_5 = 0.5$—and good behavior in the regions where at least one of these rates is high.

Looking closely the obtained simulation data for the peak region, the following relationships were obtained: $\lambda_4/\lambda_5 < 1$, $\lambda_1/\lambda_5 < 1$, and $\lambda_5 > \lambda_4 > \lambda_1/q_1$ (the firing rate of t_1 is weighted by q_1). This means that, in the region of interest, the rate which causes dancers to become ready to perform (q_1/λ_1) is higher than the one which causes dancers to leave the stage ($1/\lambda_4$). This contributes to put dancers in P_5 (waiting to start performing). Furthermore, the

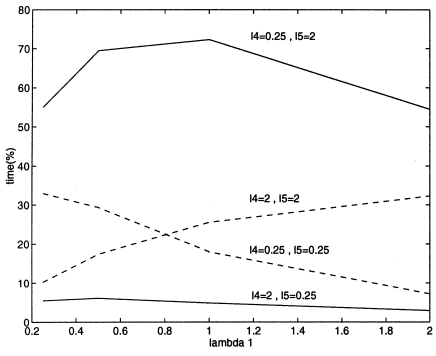

Fig. 9. Percentage of time where $P_3 > 0$ and $P_5 > 0$. In the figure *li* symbolizes λ_i

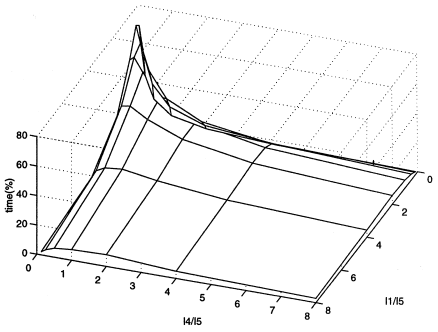

Fig. 10. 3D view of simulation results

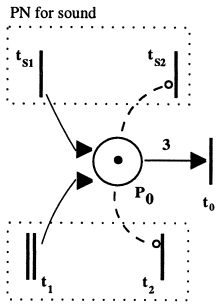

Fig. 11. Sound synchronization example

firing rate of t_4 is higher than that of t_5. This causes dancers in P_5 to remain there although costumes became available, because t_5 substitutes a dancer for another in the performance, while t_4 simply removes dancers.

This example illustrates the use of PN to model situations involving concurrency, synchronization and conflict resolution. These characteristics are also present in many other situations commonly analysed through PN, such as in multiprocessor systems [13].

4.3. *Other PN extensions*

In order to show other modeling tools, the example of the previous section will be enhanced by establishing that the beginning of the dance has to simultaneously start a sound track. This can be done by using inhibitors arcs, as shown in Fig. 11. In the example, after t_1 sends its first token (indicating that the first dancer is ready) and the sound PN signalizes sound is ready (i.e., t_{S1} fires), then P_0 has three tokens and t_0 fires. After that, both sound (t_{S2}) and movement (t_2) are allowed to proceed.

A second example also derived from that of the previous section is defined by redirecting the arcs from t_4 and t_5 (previously going to P_1) to a new node and then synchronizing the beginning of the next dance with the arrival of the fifth token (or dancer). In this case, the next performance may begin only when all the dancers of the previous one have finished dancing. Figure 12 shows the representation of this sequencing mechanism.

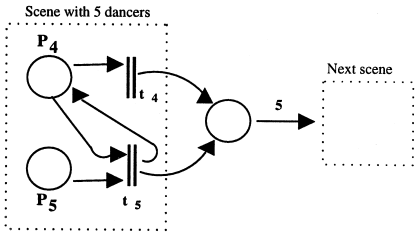

Fig. 12. Sequencing example

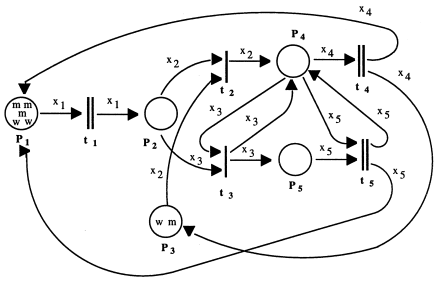

Fig. 13. Example using a Predicate/Transition Net

This second example may exhibit a deadlock situation if a last mark remains in P_5, because there will not be another mark in P_4 to enable t_5. A simulation similar to that of the previous section would be essential to avoid such situation.

A third example using once more the example of the previous section presents the use of Predicate/Transition Nets [11] which allows the differentiation of tokens, defining a type for them. The arcs have labels defining variables that dictate how many and which kind of tokens will be removed from or added to the places. The same variable appearing in the incoming and outgoing arcs of a transition denotes the same token type. A transition is enabled if there is at least one possibility of consistent substitution of variables into typed tokens.

So, in the example (Fig. 13) it is possible to distinguish male from female dancers, respectively tokens labeled *m* and *w*. The same is valid for the costumes, that can be masculine (*m*) or feminine (*w*). It is important to note that transition t_2 is enabled only if there is an adequate costume for the waiting dancer, which is indicated by the same value for label x_2, *m* or *w*, in both incoming arcs. The same is valid for t_5, that can substitute a dancer only for another of the same sex. The different indexes used for arcs leaving or entering a transition express that the token type is defined each time a transition fires. For instance, after t_1 fires twice (*m* and *w*), the firing of t_2 can get from P_2 *m* or *w* depending exclusively on t_2.

Another way to include the notion of time in PN, besides SPN and GSPN, is by the use of timed PN [19]. This PN extension defines a holding time for tokens in a place before the enabling of its output transition(s). It can be considered a simplification of SPN where the firing rates are constants and not exponentially distributed random variables.

5. CONCLUSION

This paper introduced the use of PN and some of its extensions for animation modeling and analysis. Another of the few attempts to apply PN in Computer Graphics uses them in an interface environment [4].

As other modeling techniques, PN fits very well when modeling the different aspects of an animation behavior. The problems faced when modeling an animation environment are quite similar to those where PN techniques are broadly used, they present common characteristics such as concurrency, synchronization, and conflict resolution.

PN also offers some additional mechanisms not present in most computer animation modeling techniques which enhance the process of animation development:

- The graphic representation encapsulates non-desired details, offering a very clear hierarchical description model.
- PN offers a very strong theoretical support for process analysis. Section 2 has shown some of the possible analysis that can be performed based on a graph model. Additional techniques not discussed in this paper are also available, such as state equations and priority net [15].
- Simulation techniques as presented in Section 4 complement the system analysis.

Several other tools and extensions related to PN have been developed. Some of them are: Decomposition and Aggregation (to treat the explosion of states in a SPN and GSPN) [23], colored PN [12], and other strategies allowing firing time of transitions to be specified by an arbitrary time distribution function [9].

As seen by the presented examples, PN favours the reuse of animation models. For instance, the first example (Fig. 3) represents the class of all animations in which the animated object can assume one of two possible states—as in the case of the two-orbit sphere or the feet position of a walking biped. Furthermore, the use of hierarchical PN supports encapsulation, hiding animation control details of lower level models.

Such features of animation model by PN motivates the creation of libraries with primitive PN blocks (graphs) that may be used by the animators to build the scripts that control the behavior of the animation. By using these primitives, animation modelers would not have to build the PN model for each animation from scratch, but rather they would identify animation patterns and interconnect them to define the desired behavior, as done in the case of a single dancer (Fig. 7). Furthermore, the use of PN simulators could help the animator to analyse and probe the animation model.

More interestingly, PN offers the possibility to anticipate and test animation behavior even before a single frame is shot.

For all of that, the authors believe that the introduction of PN and its extensions can bring to the field of Computer Graphics a large and well established set of techniques for animation modeling and analysis.

Acknowledgements—Part of this research was developed during the stay of the first two authors in the University of Waterloo—Computer Graphics Laboratory, Computer Science Department. The authors would like to thank the following institutions for the expressive support granted to this research: UNICAMP (State University of Campinas), FAPESP (Foundation for Research Support of the State of São Paulo) and the University of Waterloo. Thanks also to J. T. F. de Camargo and J. L. E. Campos for their valuable comments and contributions.

REFERENCES

1. Badler, N. I., Computer Animation Techniques. In *Introduction to Computer Graphics, Course Notes for SIGGRAPH 95*, Vol. 21. ACM-SIGGRAPH, 1995.
2. Balbo, G., On the Success of Stochastic Petri Nets. In *Proc. 6th Int. Workshop on Petri Nets and Performance Models*. IEEE, 1995. ISBN 0-8186-7210-2, pp. 2–9.
3. Barzel, R., *Physically-Based Modeling for Computer Graphics: a Structured Approach*. Academic Press, Inc., 1992. ISBN 0-12-079880-8.
4. Bastide, R. and Palanque, P., A Petri-Net based environment for the design of event-driven interfaces. *Lecture Notes in Computer Science*, 1995, **935,** 66–83.
5. Bause, F. and Kritzinger, P.S., Stochastic Petri nets: an introduction to the theory. *Advanced Studies in Computer Science*. Verlag Vieweg, 1996. ISBN 3-528-05535-9.
6. Camargo, J. T. F., Magalhães, L. P. and Raposo, A. B., Modeling motion simulation with DEDS. In *Proceedings of IFIP 94—13th: Congress of the International Federation of Information Processing*, 1994. http: //www.dca.fee.unicamp.br/ projects/prosim/publiPS.html, pp. 162–167.
7. Camargo, J. T. F., Magalhães, L. P. and Raposo, A. B., Foundations of Computer Modeled Animation. In *SIBGRAPI 95—Brazilian Symposium of Computer Graphics and Image Processing*, 1995. http://www.dca.fee.unicamp.br/ projects/prosim/ publiPS.html (in Portuguese).
8. Costa, M. and Feijó, B., Agents with emotions in behavioral animation. *Computer & Graphics*, 1996, **20**(3), 377–386.
9. Dugan, J.B., *Extended Stochastic Petri Nets: Applications and Analysis*. Ph.D. Thesis, Dept. of Electrical Engineering, Duke University, 1984.
10. Dugan, J. B., Trivedi, K. S., Geist, R. M. and Nicola, V.F., Extended Stochastic Petri Nets: Application and Analysis. In *Proc. PERFORMANCE '84*, Paris, France, 1984.
11. Genrich, H. J., Predicate/Transition Nets. In *High-level Petri-Nets: Theory and Application*. Springer-Verlag, 1991. ISBN 3-540-54125X, pp. 3–43.
12. Jensen, K., Coloured Petri Nets: a High Level Language for System Design and Analysis. In *High-level Petri-Nets: Theory and Application*. Springer-Verlag, 1991. ISBN 3-540-54125X, pp. 44–119.
13. Marsan, M. A., Conte, G. and Balbo, G., A class of generalized stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 1984, **2**(2), 93–122.
14. Molloy, M. K., Performance analysis using stochastic Petri Nets. *IEEE Transactions on Computers*, 1982, **31**(9), 913–917.
15. Murata, T., Petri Nets: properties, analysis and applications. *Proceedings of the IEEE*, 1989, **77**(4), 541–580.

16. Perlin, K. and Goldberg, A., Improv: a System for Scripting Interactive Actors in Virtual Worlds. In *Proceedings of SIGGRAPH '96*, 205–216, 1996.
17. Petri, C. A., *Kommunikation mit Automaten*. Schriften des IIM Nr. 3, Bonn: Institut fuer Instrumentelle Mathematik, 1962.
18. Preston, M. and Hewitt, W. T., Animation using NURBS. *Computer Graphics Forum*, 1994, **4**(13), 229–241.
19. Ramamoorthy, C. V. and Ho, G. S., Performance evaluation of asynchronous concurrent systems using Petri Nets. *IEEE Transactions on Software Engineering*, 1980, **6**(5), 440–449.
20. Sims, K., Evolving Virtual Criatures, In *Proceedings of SIGGRAPH '94*, 15–22, 1994.
21. Thalmann, N. M. and Thalmann, D., Complex models for animating synthetic actors. *IEEE Computer Graphics and Applications*, 1991, **11**(5), 32–44.
22. Wyvill, B., A Computer Animation Tutorial. In *Computer Graphics Techniques: Theory and Practice* (eds D.F. Rogers and R.A. Earnshaw), pp. 235–282. Springer-Verlag, 1990.
23. Ziegler, P. and Szczerbicka, H., A Structured Based Decomposition Approach for GSPN. In *Proc. 6th Int. Workshop on Petri Nets and Performance Models*, pp. 261–270. IEEE, 1995. ISBN 0-8186-7210-2.