Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Modelagem e Construção de Mecanismos de Coordenação em Ambientes Computacionais

Autor: Dennis Guimarães Pelluzi
Orientador: Prof. Dr. Léo Pini Magalhães

Dissertação de mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de Concentração: **Engenharia de Computação.**

Banca Examinadora

Prof. Dr. Léo Pini Magalhães.....DCA/FEEC/UNICAMP
Prof. Dr. Alberto Barbosa Raposo.....Tecgraf/PUC-RIO
Prof. Dr. Ivan Luiz Marques Ricarte.....DCA/FEEC/UNICAMP
Prof. Dr. Rafael Santos Mendes.....DCA/FEEC/UNICAMP
Prof. Dr. Fernando Antonio Campos Gomide.....DCA/FEEC/UNICAMP

Campinas, SP
Julho/2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA – BAE – UNICAMP

P367m

Pelluzi, Dennis Guimarães
Modelagem e construção de mecanismos de coordenação em ambientes computacionais / Dennis Guimarães Pelluzi. -- Campinas, SP: [s.n.], 2007.

Orientador: Léo Pini Magalhães
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de petri. 2. Fluxo de trabalho. 3. Engenharia de software. 4. UML (Linguagem de modelagem padrão). I. Magalhães, Léo Pini. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Modeling and construction of coordination mechanism in computational environment.
Palavras-chave em Inglês: Coordination, Computational processes, Petri net, Workflow.
Área de concentração: Engenharia de Computação.
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Alberto Barbosa Raposo, Ivan Luiz Marques Ricarte, Rafael Santos Mendes e Fernando Antonio Campos Gomide.
Data da defesa: 23/07/2007
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Dennis Guimarães Pelluzi

Data da Defesa: 23 de julho de 2007

Título da Tese: "Modelagem e Construção de Mecanismos de Coordenação em Ambientes Computacionais"

Prof. Dr. Léo Pini Magalhães (Presidente): _____

Prof. Dr. Alberto Barbosa Raposo: _____

Prof. Dr. Ivan Luiz Marques Ricarte: _____

Prof. Dr. Rafael Santos Mendes: _____

Prof. Dr. Fernando Antônio Campos Gomide: _____

Resumo

Este trabalho aborda o problema da coordenação de atividades interdependentes em processos computacionais. As interdependências entre atividades podem ser temporais, de recursos, de bloqueio e de mútua exclusão, entre outras. Exemplos de processos computacionais com atividades interdependentes são ambientes multimídias, ferramentas de suporte ao trabalho colaborativo, *workflows*, vídeo games, animação computacional e composição de *web services*. Uma das dificuldades em projetar o mecanismo de coordenação é garantir que tal mecanismo seja consistente com a especificação do processo. Vários trabalhos sugerem o uso de ferramentas de modelagem como redes de Petri ou linguagens de coordenação para construir um mecanismo de coordenação. Um desse trabalhos apresenta a metodologia Grafo de Relações (GR) que trata das dependências temporais entre atividades. Esta dissertação apresenta uma extensão da metodologia GR para lidar com dependências de recursos, uma ferramenta para automatizar a modelagem do mecanismo de coordenação e uma proposta para o coordenador (componente de software responsável pela coordenação das atividades).

Palavras-chave: coordenação, processos computacionais, redes de Petri, modelagem.

Abstract

This work approaches the problem of coordination of activities in computational processes. The coordination necessity appears when there are interdependencies among activities. The type of interdependencies can be temporal, of resources, blockage and mutual exclusion, among others. Processes with interdependent activities occur in multimedia environments, collaborative work support tools, workflows, video games, computational animation and web services composition. One of the difficulties in designing coordination mechanisms is to guarantee that such mechanisms are consistent with the specification of the process. Some works suggest the use of modeling tools such as Petri Nets or coordination languages to construct a coordination mechanism. One of them presents the methodology of Relations Graph which deals with the temporal dependencies among activities. This work presents an extension for the methodology of GR to deal with dependencies of resources, a tool which automates the modeling of the coordination mechanism and proposes a coordinator (software component which coordinates activities).

Keywords: coordination, computational processes, Petri Net, modeling.

Aos meus pais, Marcos e Dalva.
Aos meus avós paternos, Francisco e Geralda Pelluzi,
e maternos, José e Pedrolina (in memoriam).

Agradecimentos

Ao meu orientador, Prof. Dr. Léo Pini Magalhães, pela oportunidade do aprendizado e pela orientação durante o curso.

À minha família, pelo apoio à realização deste mestrado.

Aos colegas Alex Seehagen, Atílio Hsu, Gabriel Tambur, Jurandy Nascimento e William Sbordoni, pelo companheirismo durante o meu mestrado.

Aos demais colegas que me ajudaram ao longo da realização deste trabalho.

Aos professores que compartilharam suas experiências e seus conhecimentos.

Aos funcionários da FEEC por manterem a boa infra-estrutura da faculdade.

À CAPES, pelo suporte financeiro através da bolsa de mestrado.

À RNP, pelo suporte ao Projeto Giga e seus subprojetos.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

CDE	Collaborative Development Environment
CPN	Coloured Petri Net
CVE	Collaborative Virtual Environment
DCWPL	Describing Collaborative Work Programming Language
ESM	Extended State Machine
GR	Grafo de Relações
MAMC	Modelador Automático de Mecanismo de Coordenação
MC	Mecanismo de Coordenação
MCL	Mecanismo de Coordenação Local
MCL-a	Mecanismo de Coordenação Local da Atividade
MCL-ra	MCL das relações de um rótulo de atividade
MCL-r	MCL de relação entre duas atividades
MCL-rt	MCL das relações especificadas em um rótulo de aresta não unitário.
MCM	Método de Controle de Movimento
PEV	Processo-Estado-Evento
PN	Petri Net
PNML	Petri Net Markup Language
PNTD	Petri Net Type Definition
SVG	Scalable Vector Graphics
UPS	Unidade Provedora de Serviços
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
WS	Web Service
XML	Extensible Markup Language

Trabalho Publicado pelo Autor

1. A. A. Cruz, L. P. Magalhães, A. B. Raposo, R. S. Mendes & D. G. Pelluzi. Coordinating Multi-task Environments through the Methodology of Relations Graph. *Lecture Notes in Computer Science, vol. 4715/2007. International Workshop on Groupware, 13th* (CRIWG 2007). Bariloche, Argentina, setembro de 2007.

Capítulo 1

Introdução

Um processo computacional é composto por um conjunto de atividades executadas através de computadores e que podem ou não estar relacionadas entre si. Este conjunto de atividades é responsável por procedimentos que definem o objetivo do processo computacional. Se há relações de dependência entre as atividades, por exemplo, dependências temporais, então há a necessidade de um mecanismo de coordenação. Este mecanismo de coordenação tem como objetivo garantir a execução das atividades de acordo com as dependências existentes.

Podemos citar como exemplos de processos computacionais com atividades interdependentes, sistemas multimídias, ferramentas de suporte ao trabalho colaborativo, sistemas de *workflows*, vídeo games e animação computacional. Alguns processos computacionais requerem interações entre pessoas. Este é o caso de processos existentes em um ambiente de desenvolvimento colaborativo (CDE – *Collaborative Development Environment*). CDE é um ambiente virtual onde os envolvidos em um projeto podem discutir, compartilhar informações e executar tarefas mesmo estando distribuídos geograficamente e no tempo (assíncronos). Como em qualquer ambiente colaborativo, existe a necessidade de um coordenador gerenciar as atividades, o compartilhamento de recursos, o fluxo de informação, entre outras coisas. De forma geral, o papel do coordenador é desempenhado por uma pessoa, embora existam várias ferramentas de suporte ao trabalho colaborativo (p.e., Microsoft Outlook e IBM LotusNotes).

A Internet vem sendo utilizada como um CDE. Há várias ferramentas que auxiliam o trabalho colaborativo como *e-mails*, comunicadores instantâneos, fóruns de discussões, editores *on-line* colaborativos (por exemplo, Google Docs & Spreadsheets) e aplicações *wiki*[1] (como o

1 Uma aplicação *wiki* permite que uma página web seja editada colaborativamente por várias pessoas.

Wikipedia). Entretanto, não existe uma ferramenta que integre e coordene todas as ações de trabalho.

Considerando ainda o ambiente da Internet, temos a computação orientada a serviço. Nesta arquitetura, o software é visto como um serviço e uma aplicação *web* é composta de vários serviços. Em outras palavras, cada serviço é visto como um bloco básico de construção de uma aplicação (composição de serviços). A composição de serviços permite a colaboração inter-organizacional entre atividades através de *workflow* inter-organizacional. Para isto, é necessário que as atividades sejam coordenadas.

Em outros processos computacionais, como uma animação, queremos que os atores exerçam suas atividades automaticamente conforme um *script*. O projetista, neste caso é o animador, especifica as ações dos atores e atribui a um coordenador (software) a tarefa de gerenciar as atividades.

1.1 Motivação e objetivos

Uma das dificuldades de projetar processos computacionais com atividades interdependentes é coordenar a execução das atividades. O projetista deve conhecer a priori o comportamento das atividades e as relações de dependência direta e indireta entre elas. Alguns autores ([Malone, 94], [Crowston, 94], [Schimidt, 96], [Raposo, 00]) já abordaram o processo de gerenciar dependências entre atividades em diferentes aplicações.

O trabalho de A. Cruz [Cruz, 04] propôs uma metodologia de construção de mecanismo de coordenação independente da aplicação. Esta metodologia, denominada Grafo de Relações (GR), lida com interdependências temporais entre as atividades. A metodologia GR trabalha em três níveis de abstração. No primeiro nível (nível de especificação), definimos as atividades e as relações temporais entre elas usando um grafo acíclico, dirigido e rotulado. No segundo nível (nível de coordenação), temos um algoritmo que constrói o mecanismo de coordenação a partir do grafo especificado no primeiro nível. Por fim, o terceiro nível (nível de execução) é responsável pela construção do coordenador que implementa o mecanismo de coordenação.

O mecanismo de coordenação construído através da metodologia GR é livre de inconsistências temporais. Ou seja, o mecanismo garante que todas as atividades serão executadas sem violar as restrições temporais impostas. Entretanto, falta à metodologia GR a

capacidade de lidar com outras dependências como o compartilhamento de recursos, mútua exclusão, bloqueio e número de execuções. Além disto, também falta uma especificação da construção do coordenador (nível de execução) e de como ele irá interagir com as atividades.

Os objetivos deste trabalho são desenvolver uma ferramenta de modelagem de mecanismos de coordenação baseada na extensão da metodologia GR que possibilite analisar o comportamento de um processo computacional e propor uma especificação para a construção e para o funcionamento do coordenador (componente de software). A extensão da metodologia GR proposta neste trabalho permite incluir o uso e a produção de recursos por parte das atividades. Esta extensão é feita no nível de especificação e no nível de coordenação.

A ferramenta desenvolvida neste trabalho, denominada Modelador Automático de Mecanismo de Coordenação (MAMC) trabalha nos dois primeiros níveis da metodologia GR. A partir da especificação das atividades e das interdependências, temos um modelo de mecanismo de coordenação baseado em redes de Petri coloridas (CPN – *Coloured Petri Net*). Este modelo, descrito em um arquivo, pode ser usado em um ferramenta de simulação de redes de Petri com o intuito de analisar o comportamento do sistema.

1.2 Organização do trabalho

Este trabalho está organizado da seguinte maneira: no capítulo 2, discutimos coordenação de diferentes processos computacionais. Apresentamos uma revisão de trabalhos correlatos e duas abordagens de coordenação de atividades interdependentes; redes de Petri e linguagens de coordenação. Discutimos também a questão de coordenação em *web services* e definimos os conceitos empregados ao longo deste trabalho.

No capítulo 3, apresentamos de forma geral a metodologia grafo de relações. Esta metodologia lida com a coordenação de atividades com interdependências temporais. Mostramos o modelo do mecanismo de coordenação e o algoritmo de construção deste utilizados pela metodologia GR. Apresentamos também duas técnicas de verificação de inconsistências temporais na especificação do grafo de relações.

No capítulo 4, começamos com a extensão da metodologia GR. Descrevemos como incluir o uso e a produção de recursos por parte das atividades no nível de especificação e no nível de coordenação. Em seguida, apresentamos o protótipo desenvolvido neste trabalho e a proposta de

especificação do coordenador.

No capítulo 5, apresentamos dois exemplos de uso da ferramenta de modelagem. O primeiro exemplo é um sistema de animação computacional enquanto o segundo, um sistema de *workflow*. A idéia do capítulo 5 é mostrar o uso da ferramenta MAMC e do coordenador na coordenação de processos computacionais.

Por fim, apresentamos a conclusão e as sugestões de trabalhos futuros no capítulo 6.

Capítulo 2

Coordenação de processos computacionais

O trabalho de coordenação surge da necessidade de garantir a correta execução de atividades interdependentes. Processos computacionais com atividades interdependentes ocorrem, por exemplo, em aplicações de multimídia, em ambiente virtual colaborativo (CVE – *Collaborative Virtual Environment*) e em sistema de gerenciamento de workflow (WfMS – *Worflow Management System*). Exemplos de interdependências entre as atividades são temporal, de recursos, de bloqueio e de mútua exclusão.

O objetivo deste capítulo é expor alguns dos esforços em coordenar atividades interdependentes de processos computacionais.

Vários autores apresentam técnicas e modelos de coordenação baseados em redes de Petri [Raposo, 00], [Hsu *et al*, 03] e [Gonsalves *et al*, 04]. A opção por modelos que utilizam redes de Petri (clássicas ou suas extensões) [Murata, 89], se deve ao seu suporte matemático para análise e simulação do comportamento das atividades. Alguns destes trabalhos propõem mecanismos de coordenação específicos para uma aplicação, por exemplo, multimídia [Hsu *et al*, 03]. Outros propõem mecanismos de coordenação independentes da aplicação [Raposo,00].

Outra opção para coordenar atividades é usar linguagens de coordenação como abordado nos trabalhos de Cortes [Cortes, 00] e Ciobanu [Ciobanu, 05]. Tais linguagens são específicas do problema de interesse (coordenação de processos concorrentes ou construção de aplicações colaborativas) e visam auxiliar o programador. Por isso, as linguagens de coordenação têm sintaxe semelhantes às linguagens de programação.

A seção 2.1 apresenta os conceitos gerais de coordenação. Na seção 2.2, abordamos os diferentes tipos de interdependências que podem ocorrer em processos computacionais. Nas

seções seguintes, discutimos alguns trabalhos de coordenação de processos computacionais.

2.1 Conceitos gerais de coordenação

De modo intuitivo, coordenação é todo esforço para garantir que as várias partes que compõem um processo atuem conjuntamente, sem que haja conflitos, de modo a atingir um determinado objetivo. Para ilustrar essa definição intuitiva, podemos citar, por exemplo, o trabalho de fazer um filme. É necessário que haja uma coordenação entre as ações de todos os indivíduos envolvidos na produção (diretor, atores, técnicos) para que se consiga alcançar o objetivo final, que é a realização do filme.

Uma definição simples e precisa de coordenação é apresentada por Malone [Malone, 94] como sendo o gerenciamento de dependências entre atividades. Ainda segundo Malone, se não há dependências entre atividades então não há necessidade de coordenação. Embora, o conceito apresentado anteriormente seja aplicável a qualquer tipo de processo (industrial, computacional, biológico e outros), estaremos tratando neste trabalho a coordenação em ambientes computacionais[2].

Sendo que o trabalho de coordenação trata dependências entre atividades, devemos identificar a priori quais os tipos de dependências envolvidas para, então, determinarmos meios de gerenciar estas dependências. Raposo [Raposo, 00] separa as dependências entre atividades em duas classes: dependências temporais e de gerenciamento de recursos. Ambas são independentes, isto é, uma pode ocorrer sem que haja a ocorrência da outra. Esta classificação abrange uma gama bastante ampla de situações que podem ocorrer em um processo computacional como, por exemplo, sincronização de vídeo, compartilhamento de recursos e controle de fluxo de informação (*workflow*). Além destas dependências, podemos ter dependências de bloqueio, de mútua exclusão e quanto ao número de execuções da atividade.

Uma vez identificados os tipos de dependência, resta definir o que é uma atividade. No contexto deste trabalho, uma atividade representa uma parte funcional de um processo. Toda atividade tem um intervalo de duração que corresponde ao seu tempo de execução. Este intervalo, em geral, é não-determinístico, pois depende de fatores externos, como uma intervenção do usuário ou uma resposta de outra atividade. Sob o ponto de vista da teoria de Sistemas

2 Notemos que existem processos computacionais que requerem também interação entre pessoas.

Dinâmicos a Eventos Discretos (SDED) [Cassandras, 93], este intervalo corresponde ao tempo entre a ocorrência dos eventos de início e de fim da atividade. Eventos, por definição, são instantâneos. Uma abordagem de SDED se preocupa com os instantes de ocorrência dos eventos início e fim da atividade e não com a sua execução.

Definimos atividade como uma ação atômica realizada em um processo computacional durante um intervalo finito e não nulo cuja execução é autorizada de acordo com um conjunto de restrições previamente definido.

Antes de implementar um mecanismo de coordenação, é interessante testá-lo a fim de verificar se o mecanismo é consistente, isto é, se ele atende aos requisitos impostos pelo projetista. Um modelo[3] formal, que descreve o mecanismo de coordenação bem como as atividades e suas dependências, é importante para verificarmos o comportamento do sistema e validarmo-no. Na próxima seção, apresentaremos uma visão geral de interdependência entre atividades.

2.2 Interdependências entre atividades

No contexto deste trabalho, atividades são ditas interdependentes quando estão relacionadas entre si, de modo que a execução de uma atividade dependa de alguma forma da execução de outra(s). Se a execução das atividades leva a um objetivo comum, as atividades são ditas colaborativas.

Em geral, a relação de interdependência entre atividades é expressa através de um grafo [Diestel, 05] (Fig. 2.1-(a)). As atividades são representadas pelos vértices do grafo e as relações entre elas são representadas pelas arestas. Uma atividade A é interdependente de uma atividade B se os vértices que as representam forem adjacentes. Notemos que esta relação é transitiva. Por exemplo, se A depende de B e B é depende de C, então A também depende de C.

3 Um modelo é definido como um dispositivo teórico que, de alguma maneira, descreve o comportamento de um sistema [Cassandras, 93].

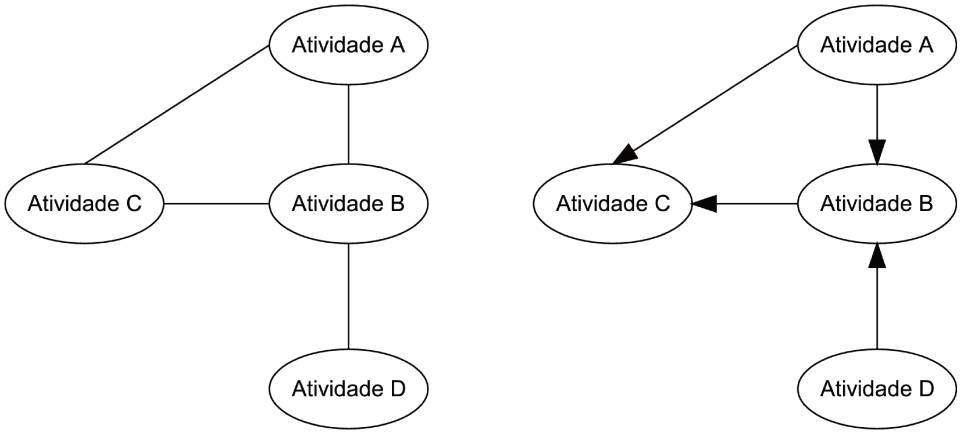


Fig. 2.1: Grafo de interdependências entre atividades. (a) relação simétrica. (b) relação não simétrica.

Para o caso de relação de dependência não simétrica, usa-se um grafo dirigido (dígrafo). Neste caso, o sentido da relação é indicado pelo sentido da aresta (Fig. 2.1-(b)).

2.2.1 Interdependência temporal

A interdependência temporal diz respeito à ordem de execução de duas ou mais atividades. A execução de uma atividade requer um determinado tempo e possui dois eventos associados: o evento de início da atividade (instante inicial) denotado por i e o evento de fim da atividade (instante final) denotado por f. Obviamente, $i < f$.

J. F. Allen [Allen, 83] mostrou que existem sete relações temporais primitivas e mutuamente exclusivas através das quais dois intervalos podem estar relacionados. A Fig. 2.2 mostra essas primitivas para dois intervalos.

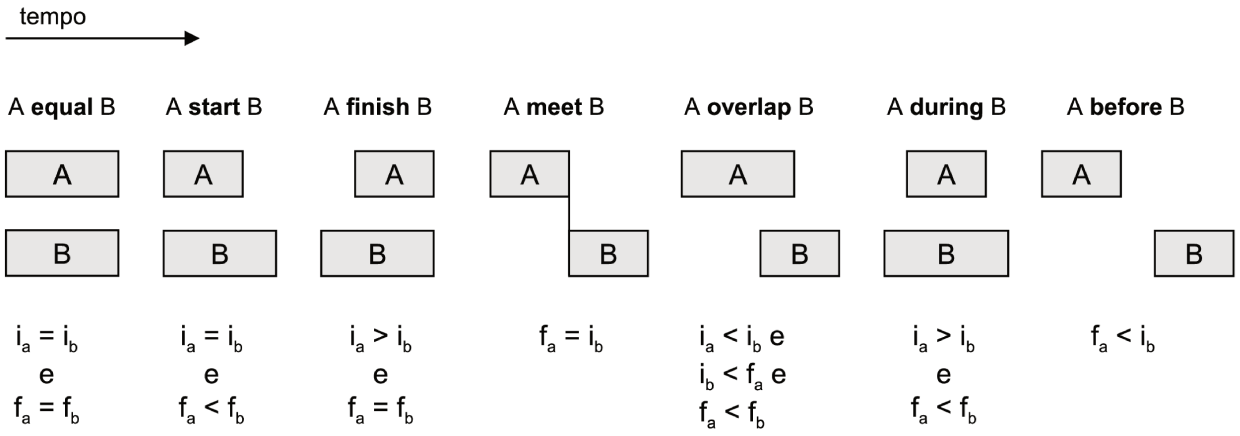


Fig.2.2: Relações temporais entre dois intervalos A e B [Allen, 83].

Esse conjunto de relações temporais (**equal**, **start**, **finish**, **meet overlap**, **during** e **before**) se aplica apenas a intervalos, e não a instantes de tempo. Entretanto, este conjunto pode ser utilizado sem problemas em coordenação de atividades, pois como definimos anteriormente, as atividades são executadas em intervalo finito e não nulo.

Uma questão levantada por Raposo [Raposo, 02] é que as relações temporais de Allen permitem diversas interpretações, principalmente no contexto de coordenação. Isto se deve ao fato de que o conjunto de relações temporais da Fig. 2.2 é meramente descritivo. Consideremos, por exemplo, duas atividades, A e B, e a relação A **equal** B. Sabemos que as atividades descritas deverão ocorrer simultaneamente. O que deveria fazer o coordenador quando ocorrer o fim da atividade A e a atividade B ainda não estiver concluída? Uma interpretação possível seria cancelar a atividade B imediatamente. Outra interpretação seria aguardar o fim da atividade B para, então, autorizar o fim da atividade A. O mesmo problema ocorre no início das duas atividades.

Raposo [Raposo, 02] denominou a primeira interpretação de ativa e a segunda de passiva. Para lidar com essas duas interpretações ele definiu dois operadores: *enables* (habilita) e *forces* (força). O operador *enables* representa a interpretação passiva, pois apenas permite que a atividade inicie ou termine. O operador *forces* representa a interpretação ativa, pois força o início ou fim de uma atividade. Esses operadores são aplicados aos instantes inicial e final de cada atividade. Adicionalmente, cada atividade possui dois estados: *ready* (pronto) e *concluded* (concluído). O estado *ready* indica que uma atividade está pronta para começar (ou terminar)

enquanto o estado *concluded* indica que uma atividade já iniciou (ou terminou). Esses dois estados são usados juntamente com os operadores *enables* e *forces*.

Para exemplificar, consideremos a atividade A "compilar programa" e a atividade B "exibir erros da compilação" com os instantes iniciais e finais i_a, i_b, f_a e f_b, respectivamente. Consideremos ainda a relação "exibir erros da compilação" **equal** "compilar programa". Para esta relação temporal podemos ter as seguintes especificações, entre outras.

1. i_a (concluded) forces i_b AND f_a (concluded) forces f_b.

Esta sentença indica uma situação ativa. O início (fim) da atividade A "compilar programa" força o início (fim) da outra.

2. i_a (concluded) enables i_b.

Esta sentença indica uma situação passiva. Aqui o início da atividade B "exibir erros da compilação" só estará habilitada se a outra atividade já tiver sido iniciada.

2.2.2 Interdependência de recursos

Outra classe de interdependência entre atividades que pode ocorrer em um processo computacional é a de recursos. Este tipo de interdependência lida com a distribuição de recursos entre as atividades. De acordo com Raposo [Raposo, 00], há três tipos básicos de gerenciamento de recursos: compartilhamento (*sharing*), simultaneidade (*simultaneity*) e volatilidade (*volatility*).

No caso de compartilhamento, um número limitado de recursos deve ser compartilhado entre várias atividades. Um exemplo comum é o que ocorre quando várias pessoas desejam simultaneamente usar uma impressora ou editar um documento.

Em simultaneidade, um recurso só está disponível se um determinado número de atividades o requisitarem simultaneamente. É o caso de um canal de comunicação que só estará aberto se dois usuários o requisitarem.

A volatilidade de recurso indica se um recurso está disponível após sua utilização. Um exemplo típico de recurso volátil é a folha de papel em uma impressora. A impressora, por sua vez, é um recurso não volátil (em situações normais, está disponível após ser utilizada).

Os três tipos de gerenciamento de recursos citados requerem parâmetros que indiquem o número de instâncias disponíveis, o número de atividades que utilizam o recurso e o número de

vezes que um recurso pode ser utilizado. Além disso, essas dependências podem ser combinadas. Por exemplo, um recurso pode ser volátil e também estar sendo compartilhado entre várias atividades.

No caso de uma relação produtor-consumidor, isto é, quando a execução de uma ou mais atividades dependente da disponibilidade de recursos produzidos por outra(s) atividade(s), podemos usar a relação temporal **meet** ou **before** para representar a relação produtor-consumidor (Fig. 2.3). Entretanto, caso o recurso não seja produzido por uma atividade, esta forma de representação não será a mais adequada. Neste caso a execução da atividade que utiliza um determinado recurso não dependente da execução de outra(s), não sendo possível usar algum relacionamento temporal.

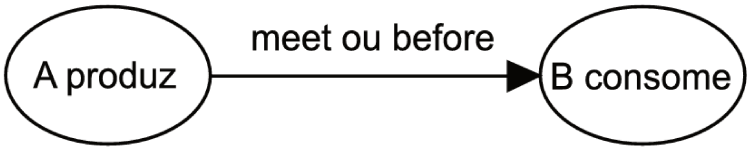


Fig. 2.3: Relação produtor-consumidor.

2.2.3 Interdependência de bloqueio

Pode haver situações em que a execução de uma atividade bloqueia o início ou interrompe a execução de outras atividades. Este bloqueio é proposital e temporário. Nesta classe de interdependência, queremos garantir que uma ou mais atividades não sejam executadas enquanto outra estiver em execução, mas que ao final da execução as atividades bloqueadas possam ser executadas.

2.2.4 Interdependência de mútua exclusão

Mútua exclusão indica que a execução de uma atividade inibe a execução de outra atividade. Podemos estender esta interpretação para um conjunto de atividades, onde apenas uma atividade será escolhida para ser executada. Diferentemente da interdependência de bloqueio, o término da atividade escolhida não habilita a execução das outras atividades. Esta interdependência equivale ao operador lógico ou-exclusivo.

2.2.5 Interdependência quanto ao número de execuções.

Em alguns casos, é necessário indicar quantas vezes uma atividade deve ser executada. Geralmente esta restrição está associada à interdependência temporal. Por exemplo, a atividade A é executada 3 vezes antes da atividade B ou a atividade A é executada 2 vezes durante a atividade B. Se requeremos somente dizer quantas vezes uma atividade deve ser executada sem estar relacionada a nenhuma outra atividade, então temos apenas uma dependência quanto ao número de execuções.

2.3 Modelos de coordenação

Para lidar com as dependências descritas na seção anterior, existem ferramentas que nos auxiliam na construção de um mecanismo de coordenação. Nesta seção, descrevemos como algumas destas ferramentas são usadas na construção ou modelagem de mecanismos de coordenação em ambientes de multimídia, *groupware* e *web services*.

2.3.1 Modelos baseados em redes de Petri

rede de Petri (PN – *Petri Net*) é uma ferramenta de modelagem que descreve o comportamento de um sistema em termos de seus estados e suas transições [Murata, 89]. Uma característica da rede de Petri é a capacidade de representar sistemas com concorrência e sincronismo. Além do modelo clássico [Petri, 62], há várias extensões como as rede de Petri temporizadas [Ghezzi, 89], coloridas [Jensen, 97], estocásticas [Haas, 02] e nebulosas [Pedrycz, 94]. A opção por modelos para coordenação baseados em PN se deve, principalmente, a seu suporte matemático para análise e simulação do comportamento do sistema, além de sua notação gráfica que facilita a compreensão do sistema.

Para a área de aprendizagem colaborativa apoiada por computador, temos um exemplo de utilização de mecanismos de coordenação baseados em redes de Petri no trabalho de Olguín *et al* [Olguín *et al*, 02]. Nesse trabalho, tais mecanismos são utilizados para gerenciar os registros de um grupo de discussão. O objetivo principal é “estabelecer, antecipadamente, requisitos e momentos adequados para o desenvolvimento de atividades de discussão (...).” O modelo de

coordenação utilizado, baseado no trabalho de Raposo [Raposo, 02], separa as atividades de suas interdependências. Isto facilita alterações na política de coordenação, o que torna o modelo flexível.

Outro exemplo de uso de redes de Petri pode ser encontrado em sistemas de workflows e já foi abordado por van der Aalst *et al* [van der Aalst *et al*, 94]. A alocação de recursos e a seqüência de execução são facilmente tratadas com redes de Petri clássicas. O trabalho apresentado por Raposo [Raposo, 00], por exemplo, propõe o uso de tais redes para coordenar interdependências temporais e gerenciar recursos entre atividades. Este trabalho baseia-se em três níveis hierárquicos para o modelo de coordenação: nível de workflow, nível de coordenação e nível de execução de atividades. No nível de workflow, são estabelecidas as interdependências entre as atividades do ambiente. O nível de coordenação é construído a partir do nível de workflow, inserindo os mecanismos de coordenação correspondentes. O modelo do ambiente é analisado e simulado neste nível. O nível de execução, por sua vez, lida com a execução das atividades no ambiente.

O trabalho de A. Cruz [Cruz, 04] [Cruz *et al*, 07] utiliza redes de Petri coloridas (CPN – Coloured Petri Net) para modelar interdependências temporais entre atividades. Essa metodologia, chamada de grafo de relações (GR), gera um modelo de coordenação livre de inconsistências, isto é, o modelo gerado garante que todas as atividades serão executadas de acordo com as restrições temporais estabelecidas. A metodologia GR abstrai do projetista o conceito de rede de Petri, pois o projetista utiliza um grafo dirigido (dígrafo) rotulado para especificar as dependências entre as atividades. Um algoritmo é usado para gerar o modelo em CPN a partir do grafo. Uma característica da metodologia GR é a possibilidade de especificar dependências e atividades alternativas, isto é, é possível escolher em tempo de execução qual atividade e/ou qual dependência temporal será executada. Além disso, a metodologia GR independe da aplicação a ser modelada, sendo aplicável a várias áreas. Entretanto, a metodologia GR trata apenas da dependência temporal, não levando em consideração outros tipos de interdependência, como o compartilhamento de recursos. A metodologia GR será discutida com mais detalhes no capítulo 3.

Em animação por computador, Magalhães *et al*. [Magalhães *et al*, 98] apresentaram o uso de PN como ferramenta de modelagem e análise. No trabalho de Bicho [Bicho, 01], foi abordado o uso de PN para o controle da animação. Neste último, o uso da PN é feito num nível de abstração

mais elevado (nível de controle global), deixando para o controle do nível local o uso de alguma outra ferramenta de animação. A utilização de PN permite descrever um ambiente de animação em termos de ocorrências de eventos. Sendo assim, dada uma seqüência de eventos, é possível analisar e prever o comportamento de uma animação.

No contexto de sistemas colaborativos de engenharia, Gonsalves *et al*. [Gonsalvez *et al*, 04] utilizou redes de Petri para modelar serviços distribuídos. Um sistema colaborativo de engenharia é composto por diversos colaboradores que juntos oferecem diferentes tipos de serviços. Podemos citar, por exemplo, sistemas de suporte ao desenvolvimento de software onde um grupo de pessoas trabalha de forma distribuída[4]. Nesse ambiente, um grupo que oferece um serviço (produto da atividade realizada), que é utilizado por outro(s) grupo(s), é denominado Unidade Provedora de Serviços (UPS). O uso de simulação do modelo cliente-servidor em redes de Petri, segundo Gonsalves *et al* [Gonsalves *et al*, 04], permite diagnosticar deficiências e melhorar a performance do sistema. Um serviço é dito distribuído se ele é solicitado por diferentes clientes. Por exemplo, em um sistema de suporte de desenvolvimento colaborativo de software, várias equipes de programadores (clientes) podem necessitar de bibliotecas fornecidas por outros programadores (servidores). Notemos que uma UPS também pode ser cliente de outra UPS. O modelo em rede de Petri permite modelar atividades paralelas e concorrentes (comuns em um sistema colaborativo de engenharia). A Fig. 2.4 ilustra o modelo cliente-servidor em redes de Petri.

4 Há basicamente três dimensões de distribuição: geográfica, temporal e de especialização (conhecimento) [Park, 04]

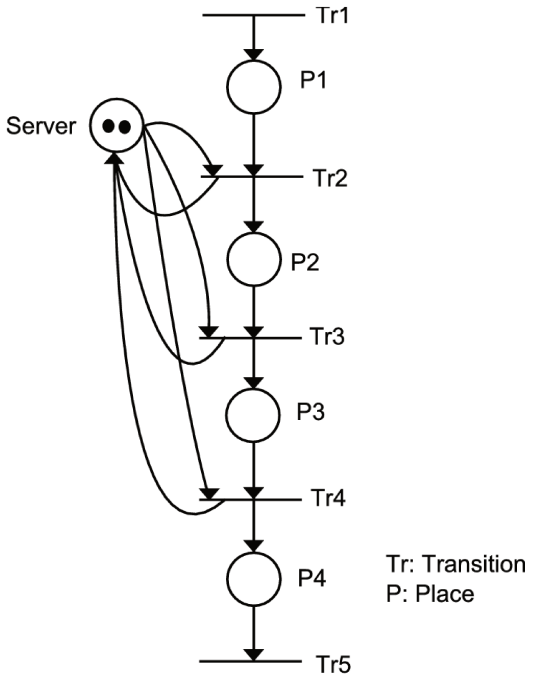


Fig. 2.4: Modelo cliente-servidor [Gonsalves *et al*, 04].

2.3.2 Linguagens de coordenação

Uma outra abordagem para gerenciar atividades interdependentes é usar linguagens de coordenação. Linguagens de coordenação são complementares às linguagens de programação de uso geral e descrevem o mecanismo de coordenação através de uma sintaxe própria. Em geral, uma linguagem de coordenação é específica a um determinado domínio.

Cortes [Cortes, 00] apresentou o projeto e a implementação de uma linguagem de coordenação, denominada Linguagem de Programação de Descrição de Trabalho Colaborativo (DCWPL - *Describing Collaborative Work Programming Language*). DCWPL é uma linguagem de programação textual projetada para facilitar o desenvolvimento de programas colaborativos. A DCWPL suporta múltiplos cenários de colaboração para uma dada aplicação computacional. Cada cenário de colaboração é especificado por um programa DCWPL, estabelecendo as regras de interação entre os participantes, permitindo ao programador construir aplicações de *groupware*. Usando a DCWPL, o programador descreve, por exemplo, como os usuários compartilham informações. As principais características da DCWPL são:

1. Especificação da interação entre grupos através de regras.
2. Separação entre os aspectos funcionais da aplicação e de coordenação.
3. Reengenharia de aplicações não colaborativas.

Além das características citadas acima, a DCWPL permite ao programador especificar diferentes políticas de coordenação para cada cenário de colaboração sem mudar as funcionalidades do programa. Consideremos, por exemplo, um editor de texto compartilhado que permite aos usuários editar o texto e fazer anotações, entre outras coisas. Podemos ter vários grupos de trabalho usando esta aplicação, sendo que para cada grupo, há políticas de coordenação diferentes. As funcionalidades do aplicativo não mudam (como a edição de texto), mas regras podem variar entre os grupos de trabalho (como, por exemplo, quem pode usar o editor e quando).

Para a especificação de coordenação, a DCWPL utiliza artefatos. Um artefato descreve como objetos são compartilhados entre os membros de um grupo. A definição de um artefato é similar a de uma classe em programação orientada a objetos. Um artefato possui atributos e funções. Um atributo de um artefato representa informações de controle enquanto que uma função especifica as condições de execução que devem ser satisfeitas para executarmos uma função computacional. Usando os atributos e as funções de um artefato, o programador pode escrever protocolos, políticas e privilégios de controle de acesso aos objetos. Não há um mapeamento um-para-um entre os artefatos e as classes computacionais.

Como dito anteriormente, DCWPL permite alterar a política de coordenação sem alterar as funcionalidades do aplicativo, mas é necessário um conhecimento da linguagem e das funções do programa coordenado. Isto dificulta a alteração das políticas de coordenação por parte do usuário final, cabendo esta operação ao programador.

A DCWPL é diferente de linguagens de coordenação como Linda [Carriero, 92] e Manifold [Arbab, 93], que foram projetadas para aplicações com processos paralelos e/ou concorrentes. A DCWPL é voltada para ambientes de trabalho colaborativo entre pessoas.

Coordenação de atividades concorrentes é um aspecto importante em linguagens de programação orientadas a objetos concorrentes e *softwares* baseados em componentes. Normalmente, estas linguagens oferecem pouco suporte para sincronização de atividades concorrentes. Os principais problemas são a não separação entre o código de coordenação e o

modelo de coordenação, a ausência de abstração e a dificuldade para o programador implementar a coordenação desejada.

Para tentar solucionar estes problemas, Ciobanu e Lucanu [Ciobanu, 05] propuseram uma linguagem de especificação onde atividades são descritas através de objetos, a coordenação é definida como um processo e a integração entre eles é feita através de um envoltório (*wrapper*). A sintaxe desta linguagem de especificação é próxima da sintaxe das linguagens de programação orientadas a objetos.

O processo de coordenação e as atividades coordenadas são independentes. Um mesmo processo de coordenação pode ser usado com diferentes objetos concorrentes e os mesmos objetos podem ser usados com coordenadores distintos. A descrição de colaboração entre as atividades define a política de interação entre os objetos. Em outras palavras, um processo de coordenação provê uma descrição em alto nível da interação entre objetos. Um processo é descrito como uma seqüência de declarações (ações globais e ações locais) seguida por um conjunto de equações que definem regras de comportamento.

Limniotes *et al* [Limniotes *et al*, 02] abordaram a coordenação dirigida a eventos de componentes de tempo real em sistemas distribuídos e paralelos. Eles mostraram uma extensão para a linguagem de coordenação Manifold com coordenação em tempo real e aplicaram o modelo proposto em sistemas multimídia distribuídos.

Manifold é uma linguagem de coordenação dirigida a eventos. Nesta linguagem, existem dois tipos de processos: gerentes (ou coordenadores) e trabalhadores. Um processo gerente é responsável por configurar e cuidar das necessidades de comunicação de um grupo de processos trabalhadores. Um processo trabalhador, por outro lado, ignora quem precisa de seus resultados e de onde recebe os dados que processa. Os processos gerentes são escritos em Manifold, enquanto os processos trabalhadores são escritos também em Manifold ou alguma linguagem de programação (tipicamente a linguagem C).

Uma atividade em Manifold é dirigida a eventos. Um processo coordenador espera pela ocorrência de um evento específico (geralmente disparado por um processo trabalhador) quando então passa a um determinado estado e executa algumas ações. O processo se mantém no mesmo estado até observar a ocorrência de outro evento.

O trabalho de Limniotes *et al* melhora o gerenciador de eventos da linguagem Manifold acrescentando a habilidade de expressar restrições de tempo real associadas ao disparo de eventos

e também reações com limite de tempo de quem os observa. Desta forma, enquanto na linguagem Manifold clássica o disparo de um evento *e* por um processo *p* e sua subseqüente observação por outro processo *q* são assíncronos, na extensão proposta por Limniotes *et al*, é possível impor restrições temporais, como por exemplo, quando um processo *p* deve disparar um evento *e* e quando o processo *q* deve reagir a este evento. Além disto, eles propuseram mecanismos de alocação de recursos em tempo real.

2.3.3 Modelos de coordenação em Web Services

Computação orientada a serviço (software como um serviço) vem se tornando uma área de interesse crescente em engenharia de software. Serviços podem ser executados em plataformas diferentes e distribuídas. Cada serviço é visto como um bloco básico de construção de uma aplicação (composição de serviços). *Web service* é a principal realização do conceito de computação orientada a serviço. Cada *web service* (WS) provê serviços especializados para outros WS na rede. Assim, em uma arquitetura orientada a serviços, WS são considerados blocos de construção de novas aplicações.

Composição de serviços é o interesse principal quando se trata de desenvolvimento de aplicações sobre *web services*. A composição de serviços permite a colaboração inter-organizacional entre atividades (*workflow* inter-organizacional). Tais atividades precisam ser coordenadas. Conforme apontando por van der Aalst *et al* [van der Aalst *et al*, 03], há várias linguagens de composição de WS como WSFL, WSCI, WS-Coordination, BPML, XLANG e BPEL4WS. Entretanto, estas linguagens provêem diferentes técnicas de composição de *web services* sem uma sólida teoria de coordenação [Prasad, 05].

Kumar e Wainer [Kumar, 04] apresentaram uma metodologia para coordenar exceções em um *workflow*. Uma exceção é uma situação especial que ocorre com pouca freqüência em um *workflow*. As exceções podem ser previstas ou não. Para lidar com as exceções, Kumar e Wainer introduzem o conceito de *meta-workflow* e de regras de processo-estado-evento (PEV). Um *meta-worflow* é um processo de controle de alto nível que consiste de cinco comandos: iniciar, terminar, suspender, continuar e esperar. O *meta-workflow* é diferente do *workflow* base, que corresponde a um processo específico. Uma regra PEV executa um *meta-workflow* quando ocorre um evento em um certo estado de um *workflow* base. Um *meta-workflow* é usado apenas para

controle de *workflows* bases e consiste dos comandos citados anteriormente. Podemos dizer que o *meta-workflow* juntamente com regras PEV formam um mecanismo de coordenação para lidar com exceções em sistemas de *workflows*.

A metodologia de Kumar e Wainer pode ser aplicada a vários tipos de *web services*. Em *web services*, *meta-workflows* podem ser usados para facilitar a interoperacionalidade entre múltiplos serviços que devem ser integrados. Assim, *meta-workflows* servem como uma ferramenta de modelagem e construção de múltiplos *workflows* e de tratamento dos vários tipos de exceções que possam ocorrer.

Prasad e Balasooriya [Prasad, 05] utilizam o conceito de *web bonds* (ligações *web*) para modelagem e desenvolvimento de aplicações colaborativas baseadas em *web services*. *Web bonds* têm o poder de modelagem das redes de Petri Estendidas[5] e permitem que aplicações criem contratos entre entidades forçando interdependências e restrições. Entidades podem ser objetos ou componentes de softwares, mas neste contexto são consideradas *web services*.

Há dois tipos de *web bonds*: subscrição e negociação. Ligações de subscrição permitem o fluxo automático de informação e de controle de uma entidade fonte para outras entidades que a subscrevem. Ligações de negociação forçam dependências e restrições entre entidades. As regras de negociação e subscrição são definidas abaixo.

- **Subscrição**: se a atividade A é executada, então execute as atividades B_i que subscrevem A.
- **Negociação**: execute A somente se todas as atividades B_i que negociam com A podem ser executadas.

Web bonds podem ser empregados tanto na modelagem de sistemas de *workflows* quanto na de comunicação síncrona (requisição-reposta) e assíncrona (difusão).

2.4 Conclusão

Este capítulo apresentou a coordenação de atividades interdependentes de processos computacionais. A necessidade de coordenação surge para garantir a correta execução das

5 rede de Petri Estendida é uma rede de Petri com arcos especiais denominados inibidores.

atividades, isto é, garantir que as relações de interdependência sejam satisfeitas.

A seção 2.2 mostrou as principais interdependências existentes entre atividades de processos computacionais. A existência de uma classe de interdependência (temporal, de recursos, de bloqueio, mútua exclusão) depende do contexto da aplicação e não implica na exclusão de outra classe. Podemos ter processos com diferentes classes de interdependências entre as atividade, por exemplo um processo de workflow.

É interessante que o mecanismo de coordenação seja testado antes de sua implementação a fim de verificar se ele atende aos requisitos impostos. Conforme visto na seção 2.3, modelos baseados em redes de Petri são bastantes utilizados para modelar tais mecanismos em diferentes áreas. A simulação da rede de Petri permite analisar as propriedades do mecanismo de coordenação, corrigir eventuais erros e testar mudanças impostas na especificação do sistema. Entretanto, os modelos em redes de Petri estão distantes da implementação. Por outro, lado as linguagens de coordenação são usadas conjuntamente com as linguagens de programação, facilitando a implementação do mecanismo de coordenação. Mas tais linguagens não provêem uma visão geral do modelo de coordenação, possuem um baixo nível de abstração se comparadas a modelos em redes de Petri e não possuem suporte para análise.

Apresentamos também neste capítulo que a composição de WS com a finalidade de criar *workflow* inter-organizacional necessita de um mecanismo de coordenação. Existem várias linguagens de composição de WS, mas nenhuma delas provê um modelo formal de coordenação.

No capítulo seguinte discutiremos com mais detalhes a metodologia grafo de relações e seu mecanismo de coordenação.

Capítulo 3

Metodologia Grafo de Relações

Quando projetamos um sistema que envolve atividades interdependentes, uma dificuldade encontrada é garantir que todas as restrições impostas para a execução das atividades sejam respeitadas. Essa dificuldade ocorre devido a restrições indiretas derivadas da transitividade das relações existentes entre as atividades. Para lidar com essa questão sob o aspecto temporal, Cruz [Cruz, 04] propôs uma metodologia, denominada grafo de relações (GR). Essa metodologia utiliza redes de Petri coloridas para modelar um mecanismo de coordenação que garanta a execução das atividades sem violar as restrições impostas. Além disso, a metodologia GR utiliza um algoritmo de complexidade linear para automatizar o processo de modelagem do mecanismo de coordenação.

O propósito deste capítulo é apresentar a metodologia grafo de relações, que é a base da ferramenta desenvolvida neste trabalho. Essa ferramenta, apresentada com mais detalhes no capítulo seguinte, implementa o algoritmo da metodologia GR e permite modelarmos mecanismos de coordenação a partir da especificação dos relacionamentos entre as atividades.

A primeira seção deste capítulo descreve resumidamente a metodologia GR. Em seguida, discutimos métodos de verificação de inconsistência temporal na especificação do sistema.

3.1 A metodologia Grafo de Relações

A metodologia grafo de relações permite expressar analítica e graficamente relações de interdependências temporais entre atividades de um processo computacional. A partir da

especificação das atividades e de seus relacionamentos, obtemos um modelo de mecanismo coordenação.

A metodologia GR trabalha com três níveis de abstração para a modelagem de sistemas (Fig. 3.1). No nível de especificação (N1), definimos o comportamento do sistema por meio das relações temporais entre as atividades. No nível de coordenação (N2), construímos os mecanismos de coordenação (MC). Mecanismos de coordenação são estruturas responsáveis por garantir o comportamento dinâmico do sistema conforme a especificação estabelecida no nível N1. Por fim, no nível N3 (nível de execução) um programa denominado coordenador deve implementar o MC construído no nível N2.

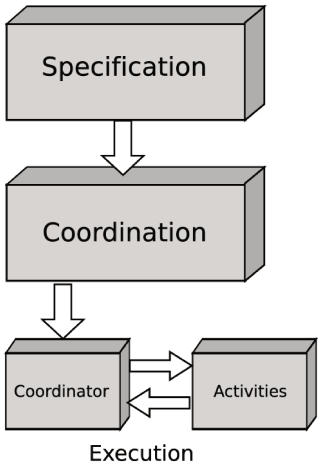


Fig. 3.1: Níveis de abstração da metodologia GR.

O sistema a ser modelado é descrito em termos das relações temporais entre as atividades, mais especificamente, das relações entre os tempos de execução das atividades. As relações temporais entre dois intervalos usadas pela metodologia GR são as mesmas relações apresentadas por Allen [Allen, 83], conforme discutido na seção 2.2.1. Definimos o conjunto de relações temporais D como:

D = {e, s, d, f, o, m, b}

As letras *e*, *s*, *d*, *f*, *o*, *m* e *b* representam, respectivamente, as relações **equal**, **start**, **during**, **finish**, **overlap**, **meet** e **before** (ver Fig. 2.2).

3.1.1 Modelagem do nível de especificação

A especificação de um sistema S é feita através de uma expressão E(A, R, F, G). Uma expressão E(A, R, F, G) é um grafo direcionado, rotulado e conexo onde:

- A é o conjunto de atividades (vértices);
- $R \subset A \times A$ é o conjunto de relações (arestas);
- $F:R \rightarrow \wp(D)$ é a função que associa a cada aresta um subconjunto de D = {e, s, d, f, o, m, b} diferente de vazio;
- $G:A \rightarrow \wp(A)$ é a função que associa a cada elemento do conjunto A um subconjunto de A.

A função F, também chamada função rotuladora de arestas, define o(s) tipo(s) de dependência temporal entre duas atividades. Se o conjunto do rótulo da aresta tiver dois ou mais elementos, então temos dependências alternativas entre as atividades. Por exemplo, $F(a_i, a_j) = \{s, b\}$ indica que a relação entre as atividades a_i e a_j podem ser ou **start** ou **before**, sendo a escolha por uma das duas opções feita em tempo de execução.

O sentido das arestas indica a ordem do relacionamento entre as atividades. Por exemplo, $F(a_i,a_j) = \{s\}$ significa que a_i **start** a_j, o que é diferente de a_j **start** a_i.

A função G, chamada de função rotuladora de atividades, define um conjunto de atividades onde apenas uma delas deve ser selecionada. Por exemplo, $G(a_i) = \{a_1, a_2, a_3\}$ significa que apenas uma atividade do conjunto $\{a_1, a_2, a_3\}$ será selecionada para relacionar-se com a_i. A escolha de uma das atividades é feita em tempo de execução.

A função G deve satisfazer as seguintes propriedades:

1. Se *b* está no rótulo de *a*, então existe uma relação entre *a* e *b*, isto é, $(a, b) \in R$;
2. Se *b* está no rótulo de *a*, então *a* não está no rótulo de *b*;
3. O rótulo de uma atividade se diferente de vazio possui pelo menos dois elementos.

O propósito da função G é possibilitar comportamentos alternativos na dinâmica do sistema modelado.

A expressão E1(A, R, F,G), definida abaixo, é um exemplo de expressão da metodologia GR. A sua representação gráfica é ilustrada na Fig. 3.2.

- $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$
- $R = \{(a_3,a_1), (a_3,a_2), (a_3,a_4), (a_4,a_5), (a_4,a_6)\}$
- $F(a_3, a_1) = \{s\}$ $F(a_3, a_2) = \{b\}$ $F(a_3, a_4) = \{m, b\}$ $F(a_4, a_5) = \{e\}$ $F(a_4, a_6) = \{e\}$
- $G(a_3) = \{a_1, a_2\}$ e $G(a_1) = G(a_2) = G(a_4) = G(a_5) = G(a_6) = \emptyset$

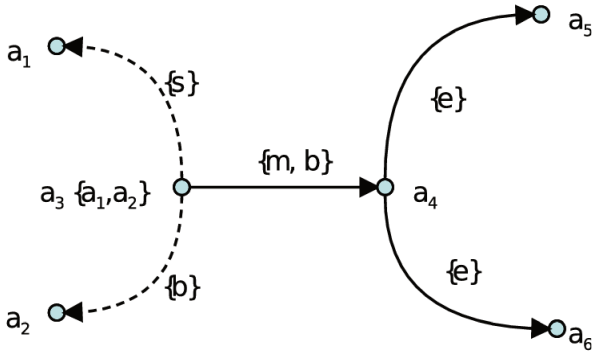


Fig. 3.2: Representação gráfica de expressão E1.

Uma observação importante a respeito da metodologia GR é que o algoritmo de modelagem do mecanismo de coordenação (discutido mais adiante) é aplicável apenas em expressões cujo grafo seja uma árvore[6]. Uma expressão que atende a este requisito é dita acíclica.

Segundo Cruz [Cruz, 04], a justificativa de se utilizar apenas expressões cujo grafo seja uma árvore (expressões acíclica) é que assim o sistema especificado é livre de inconsistências temporais, isto é, todas as restrições temporais são satisfeitas. Entretanto, uma expressão cujo grafo não seja uma árvore pode ou não conter inconsistências temporais. Neste caso, para saber se o sistema é consistente com relação às dependências temporais é necessário uma análise mais criteriosa da expressão. A seção 3.2 abordará a verificação de inconsistências temporais em tais expressões.

3.1.2 Modelagem do mecanismo de coordenação

Após a especificação do comportamento temporal de um sistema por meio de uma expressão E, fazemos a modelagem do mecanismo de coordenação, abreviado por MC. Para isso, utilizamos redes de Petri coloridas, abreviadas por CPN [Jensen, 97].

Um MC é formado por mecanismos de coordenação local (MCL). Um MCL é responsável pela coordenação de uma atividade ou de uma relação. Os MCL são classificados em:

6 Uma árvore é um grafo conexo cujo número de arestas é igual ao número de vértices menos um [Diestel, 05].

1. Mecanismo de coordenação local da atividade (MCL-a);
2. Mecanismo de coordenação local da relação entre duas atividades (MCL-r);
3. Mecanismo de coordenação local das relações temporais alternativas entre duas atividades (rótulo de aresta não unitário) (MCL-rt);
4. Mecanismo de coordenação local das relações da atividade a_i com as atividades especificadas em $G(a_i) \neq \emptyset$ (MCL-ra).

O diagrama da Fig. 3.3 mostra como é formado um MC. O MCL-a é específico de um determinada atividade. Um MCL-r é composto por exatamente dois MCL-a`s, pois corresponde ao relacionamento entre duas atividades. Um MCL-rt e um MCL-ra são compostos por dois ou mais MCL-r`s, já que coordenam relações alternativas, onde apenas um MCL-r é escolhido por vez. A diferença é que um MCL-rt coordena as dependências temporais alternativas entre duas atividades (rótulo de aresta não unitário) e um MCL-ra coordena as relações alternativas de uma atividade a_i com as atividades do conjunto do seu rótulo ($G(a_i) \neq \emptyset$).

Dependendo da especificação do sistema, o MC pode ou não conter MCL-rt e MCL-ra.

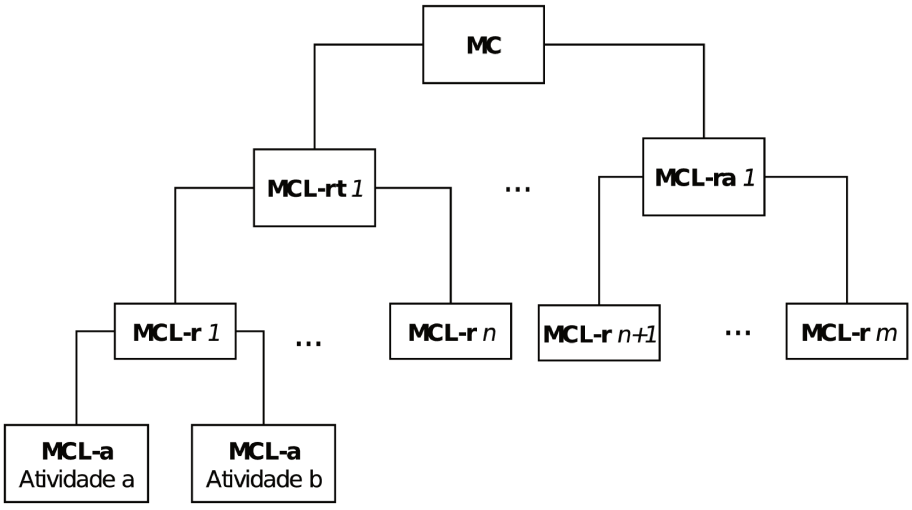


Fig. 3.3: Diagrama de blocos de um MC.

Modelo em rede de Petri do MCL de atividade (MCL-a)

O MCL de atividade, MCL-a, é responsável pelo controle de execução da atividade de

interesse. O modelo em PN deste mecanismo é ilustrado na Fig. 3.4. A transição t_{Ia} (t_{Fa}) representa o evento de início (fim) da atividade. Ela só estará habilitada se todas as condições impostas à execução da atividade forem satisfeitas.

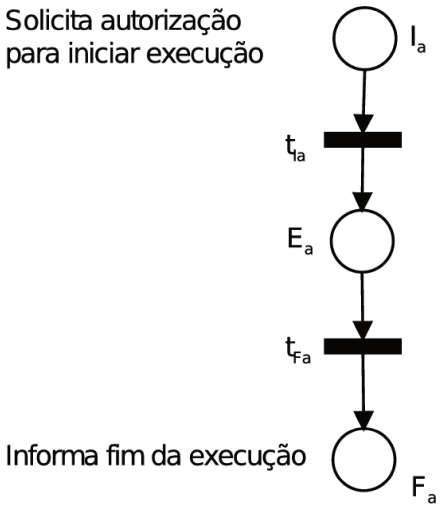


Fig. 3.4: PN do MCL de atividade (MCL-a).

O lugar E_a encapsula a execução propriamente dita da atividade. A Fig. 3.5 ilustra o modelo de execução, encapsulado por E_a, para uma atividade qualquer. Nesse modelo, a transição R é temporizada, representando o tempo de execução da atividade.

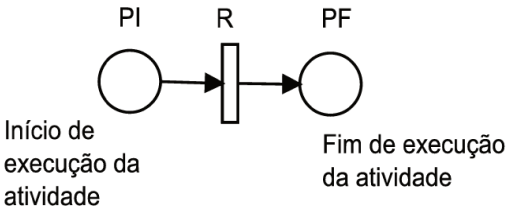


Fig. 3.5: Modelo de execução de uma atividade qualquer - E_a.

Modelo do MCL de relação (MCL-r)

O MCL de relação, MCL-r, é responsável pela coordenação do relacionamento temporal entre duas atividades. O modelo em PN de uma relação $r \in D$ entre duas atividades a_i e a_j é ilustrado na Fig. 3.6.

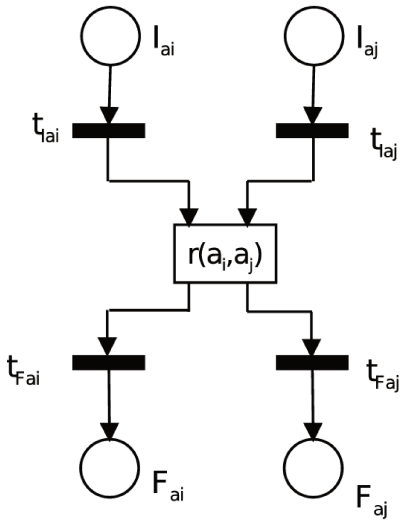


Fig. 3.6: PN do MCL de relação (MCL-r).

A "caixa preta" $r(a_i,a_j)$ contém os modelos de execução das atividades a_i e a_j (Fig. 3.6).

Para que o modelo da Fig. 3.6 represente uma das relações do conjunto D, é necessário fazermos algumas modificações de acordo com as restrições temporais (detalhes em [Cruz, 04]). A representação das relações temporais é feita inserindo restrições ao disparo das transições t_{Iai} (t_{Fai}) e t_{Iaj} (t_{Faj}). Uma restrição temporal é definida por equações e inequações envolvendo os instantes inicial e final das atividades. Essas equações e inequações são inseridas no modelo de rede de Petri de acordo com as regras 1 e 2.

1. Regra 1. Inequações do tipo $x < y$ ou $x > y$, onde x e y representam os instantes inicial e final das atividade, são representadas em rede de Petri através da inclusão de um arco ligando a transição que corresponde à variável de menor valor ao lugar P_z, $z \in \mathbb{N}$, e de um arco ligando P_z à transição que representa a variável de maior valor. Por exemplo, na relação **before**, temos $f_{a1} < i_{a2}$, que significa incluir um arco de t_{Fa1} ao lugar P_1 e um arco de P_1 à transição t_{Ia2}.

2. Regra 2. Adicionar uma equação na rede de Petri consiste em fazer uma operação de fusão[7] nas transições que representam as variáveis da equação.

7 A operação de fusão entre duas transições é efetuada transferindo-se para uma delas todos os arcos de entrada e todos os arcos de saída da outra [Cruz, 04].

Modelo do MCL das relações temporais alternativas entre duas atividades (rótulo de aresta não unitário) (MCL-rt)

Para o caso das relações temporais alternativas especificadas em rótulo de aresta não unitário, temos o modelo em CPN do MCL-rt ilustrado na Fig. 3.7. Notemos que o MCL-rt contém dois ou mais MCL-r`s, sendo que cada um corresponde a uma relação temporal alternativa entre as atividades a_i e a_j.

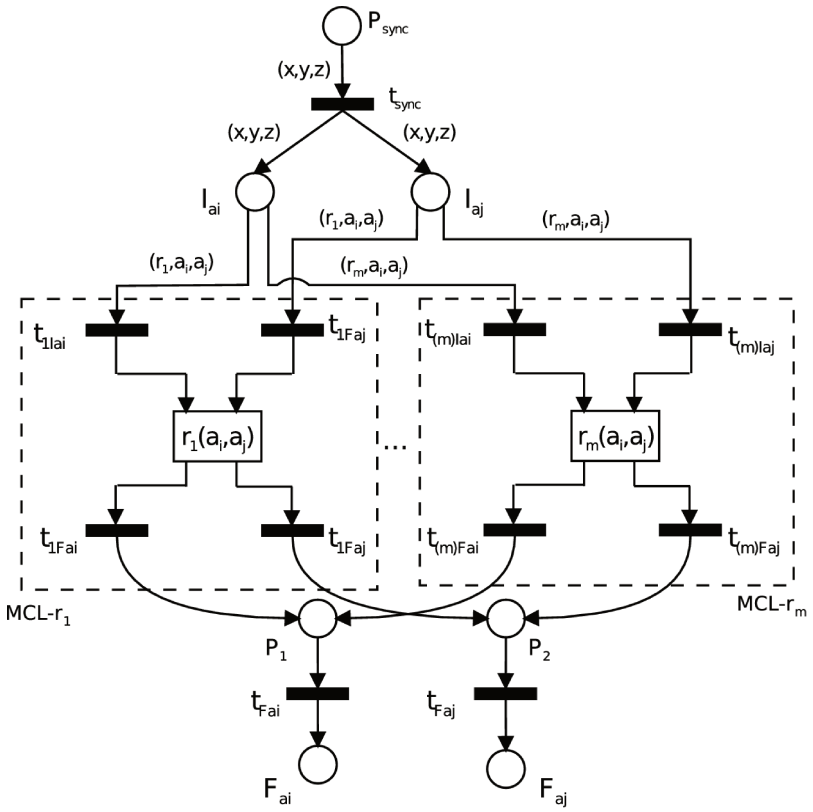


Fig. 3.7: CPN do MCL das relações com rótulo de aresta não unitário (MCL-rt).

O terno ordenado (x, y, z) define a cor da ficha que habilita o disparo da transição. A variável x pertence ao conjunto de relações D, enquanto as variáveis y e z pertencem ao conjunto de atividades A.

Por simplicidade de notação, convencionamos que a ausência de rótulo nos arcos significa a remoção de uma ficha do lugar de entrada e a adição de uma ficha da mesma cor da ficha removida no lugar de saída.

A escolha da relação r_i (i = 1, 2, ... , m) é feita inserindo uma ficha de cor apropriada no lugar

P_{sync}. Este modelo garante que apenas uma das m (m = 2, 3, 4, ...) relações alternativas seja executada.

Modelo do MCL das relações da atividade a_i com as atividades especificadas em $G(a_i)$ (MCL-ra)

Temos na Fig. 3.8 o modelo em CPN do MCL para o caso de uma atividade com rótulo diferente de vazio (MCL-ra). O MCL-ra contém dois ou mais MCL-r`s, sendo que cada um corresponde à relação da atividade a_i com a atividade $b_i \in G(a_i)$. A escolha de uma relação é feita de modo análogo ao MCL-rt.

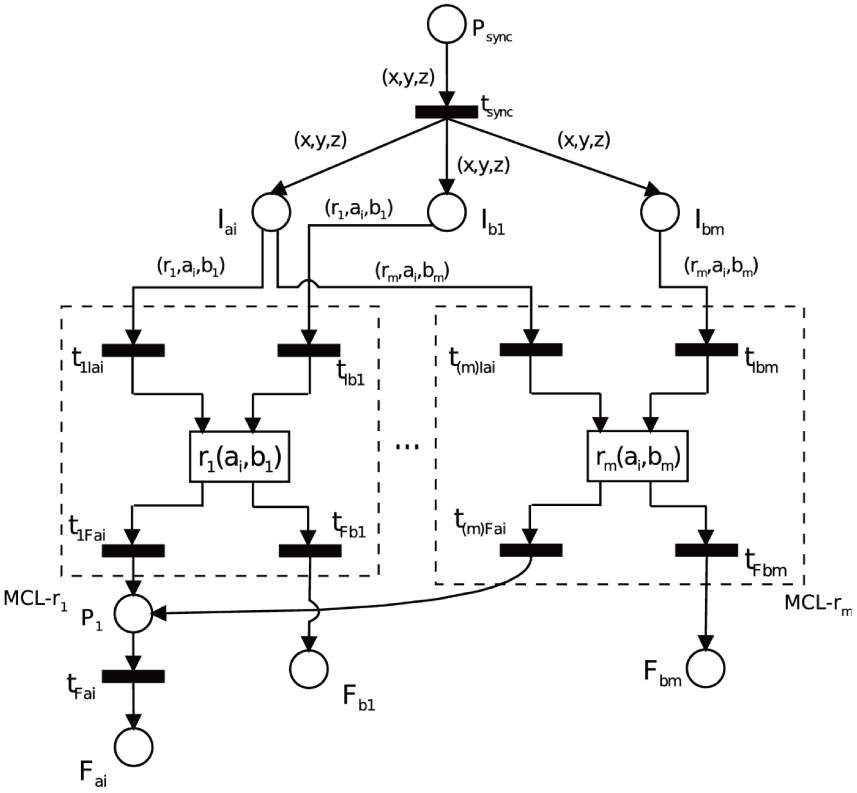


Fig. 3.8: CPN do MCL das relações de um rótulo de atividade (MCL-ra).

As relações indicadas nas figuras 3.7 e 3.8 são genéricas, isto é, representam alguma das dependências temporais do conjunto D.

3.1.3 Algoritmo de modelagem

O próximo passo depois da especificação do sistema é construir o modelo completo de coordenação. Para isso, a metodologia GR utiliza um algoritmo de complexidade O(n) [Cruz, 04], sendo n o número de atividades.

Como dito anteriormente, o grafo de uma expressão consistente é uma árvore. Com base nesta propriedade, podemos dividir a atividade em diferentes níveis como a seguir (Fig. 3.9).

Seja o grau de uma atividade a, denotado por $\partial(a)$, o número de suas relações diretas. Em outras palavras, o número de vértices adjacentes à atividade.

Algoritmo 1: Particionamento do grafo.
Seja I_k um conjunto de atividades, $k \in \mathbb{N}$;
$k \leftarrow 0$;
Enquanto houver atividades não pertencentes a um conjunto I_k, faça:
    Retirar do grafo todas atividades com grau igual a um e adicioná-las ao conjunto I_k;
    Atualizar o grau das atividades que restaram no grafo;
    $k \leftarrow k + 1$;
Fim enquanto
Fim algoritmo

No final, temos um conjunto de partições das atividades, denotado por P(A), cujos elementos são os conjuntos $I_0, I_1, ..., I_k, k \geq 0$

Uma característica desse conjunto P(A), é que atividades de um mesmo nível não estão relacionadas entre si e que as atividades dos níveis I_k, para $k \neq 0$, estão relacionadas com pelo menos uma atividade do nível I_{k-1}. A Fig. 3.9 mostra um exemplo de particionamento.

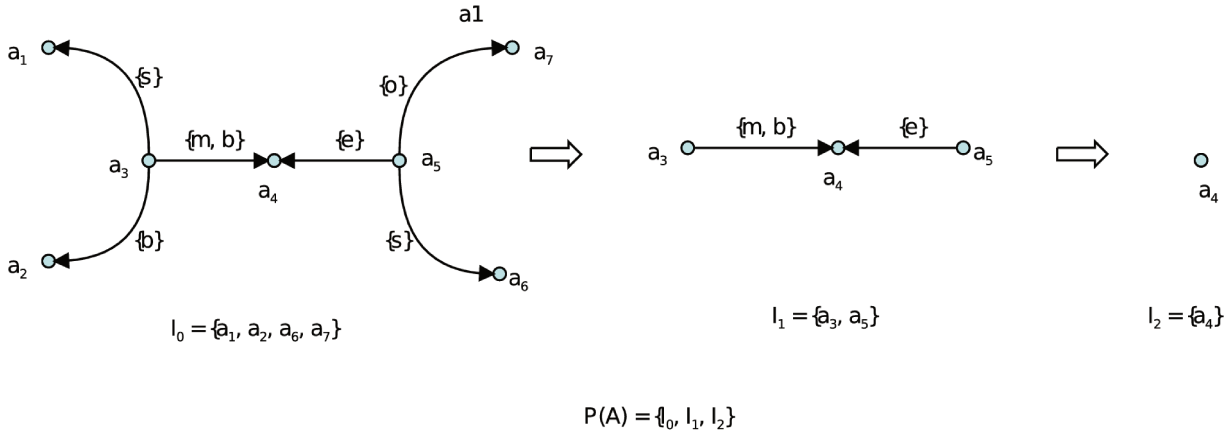


Fig. 3.9: Exemplo de particionamento P(A).

O algoritmo de modelagem do MC é definido a seguir.

Algoritmo 2: Identificação e modelagem
Dada uma expressão E(A,R,F,G);
Obter uma partição P(A) do conjunto de atividades;
Para cada subconjunto I_k, $k \geq 1$, da partição P(A) faça
    Para cada elemento do subconjunto I_k selecionado faça
        Identificar e modelar as restrições temporais obtendo seu MCL;
        Conectar o MCL da atividade a_i com o MCL apropriado da atividade $a_j \in I_{k-1}$;
    Fim para
    Se o último subconjunto I tiver duas atividades, então conectar os MCL de a_i e de a_j;
Fim para
Fim algoritmo

O procedimento de conectar significa juntar dois MCL que tenham uma atividade em comum utilizando a fusão e/ou fissão nas transições apropriadas[8]. Para exemplificar a operação de conexão, utilizaremos, por exemplo, a atividade a_5 da Fig. 3.9. Na Fig. 3.10, temos os MCL das relações (a_5, a_7) e (a_5, a_6). A operação de conexão, denotada pelo símbolo $\oplus$, entre estes dois mecanismos é feita realizando a fusão das transições que autorizam o início e o término da atividade em comum, neste caso é a atividade a_5. Na Fig. 3.11, temos o MC resultante da

8 A operação de fissão da transição t nas transições t' e t" consiste em atribuir todos os arcos de entrada de t para t' e todos os arcos de saída de t para t" [Cruz, 04].

conexão. De modo análogo, fazemos a conexão dos MCL das relações das atividades a_3 e a_4.

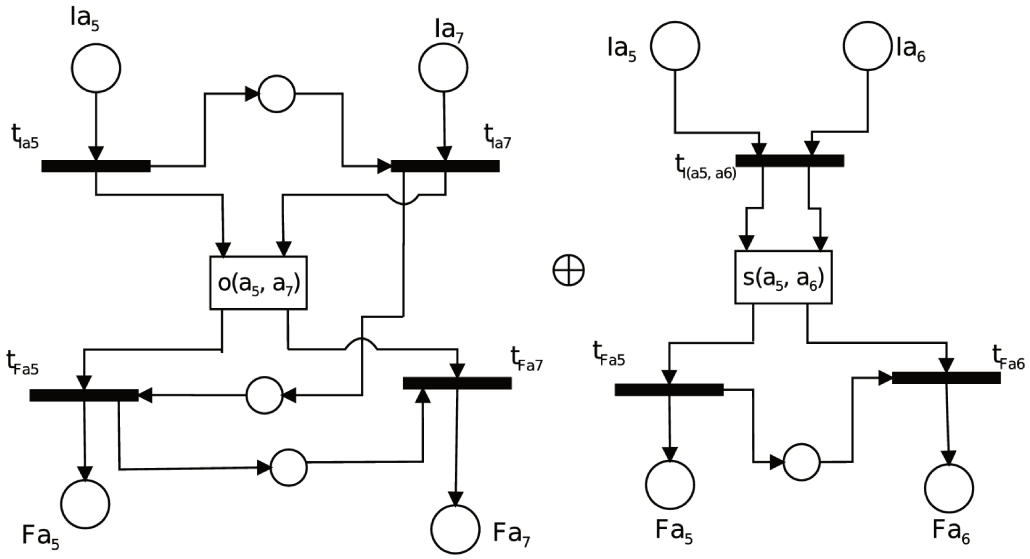


Fig. 3.10: MCL das relações (a_5, a_7) e (a_5, a_6).

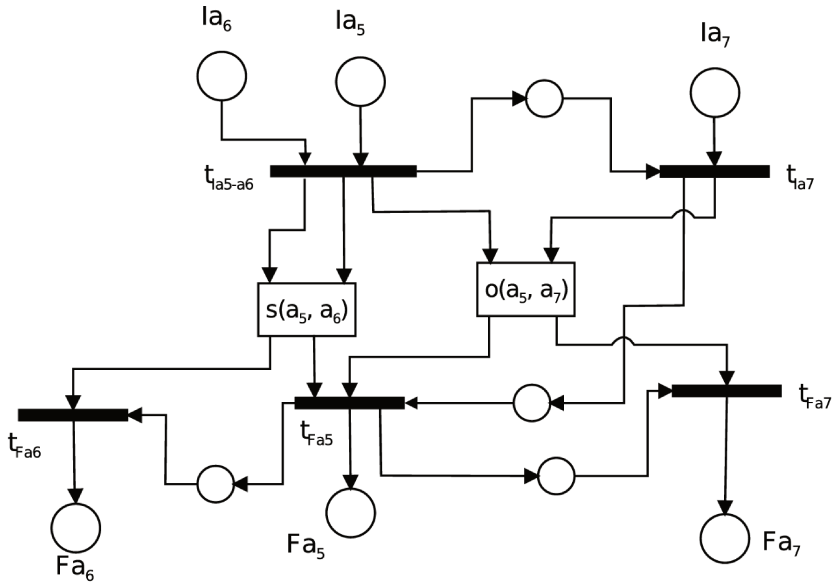


Fig. 3.11: MC da atividade a5 após fusão.

3.2 Verificação de inconsistências temporais

Conforme demonstrado por [Cruz, 04], toda expressão acíclica é consistente. No entanto, podemos ter uma expressão E cujo grafo de relação não seja um árvore (grafo fechado) e tal

expressão pode ou não conter inconsistências temporais. Nesta seção, discutimos o problema de verificação de inconsistências temporais em expressões com grafo fechado.

Uma expressão E é inconsistente se existir pelo menos um par de restrições na forma (a), (b) ou (c), onde as variáveis x e y com valores no conjuntos dos reais $\Re$ representam os tempos de início ou fim dos intervalos X e Y respectivamente.

(a) $x < y$ e $x = y$
(b) $x < y$ e $x > y$
(c) $x = y$ e $x > y$

A verificação de inconsistências pode ser feita através de um *layout* de consistência ou através de um dígrafo representando os instantes de início e fim das atividades. A primeira abordagem permite uma fácil visualização enquanto a segunda permite uma implementação computacional mais fácil.

Para ilustrar esses métodos de verificação, utilizaremos a expressão E2 dada abaixo e ilustrada na Fig. 3.12. Notemos que o grafo da expressão é fechado (não é uma árvore) e acíclico, isto é, não contém circuitos[9]. Conforme veremos a seguir, se escolhermos a relação **during** para o par (a_4, a_3) então a expressão E2 é consistente. Por outro lado, se escolhermos a relação **before** para o par (a_4, a_3) então a expressão é inconsistente.

E2(A, R, F)
$A = \{a_1, a_2, a_3, a_4\}$
$R = \{(a_1, a_2), (a_2, a_3), (a_1, a_4). (a_4, a_3)\}$
$F(a_1, a_2) = \{s\}$, $F(a_2, a_3) = \{s\}$, $F(a_1, a_4) = \{b\}$ e $F(a_4, a_3) = \{d, b\}$

9 Em um grafo dirigido, um circuito é um caminho fechado seguindo o sentido das arestas.

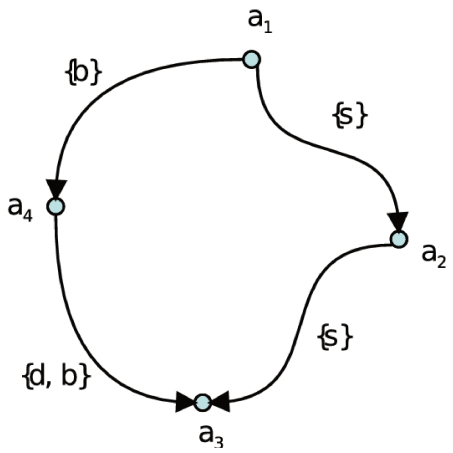


Fig. 3.12: Grafo da expressão E2(A,R,F).

O *layout* de consistência consiste em uma representação gráfica onde cada intervalo é representado por um segmento de reta posicionado sobre o eixo do tempo, segundo sua relação temporais com os demais intervalos. A Fig. 3.13 mostra os *layouts* de consistência para os dois casos.

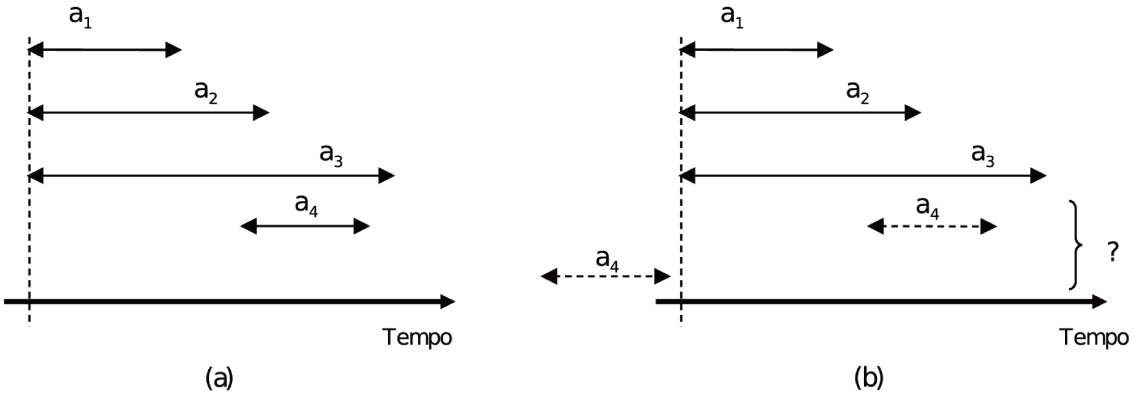


Fig. 3.13: Layout de consistência de E2.

Na Fig. 3.13 (b) temos uma inconsistência temporal. A aresta (a_4, a_3) indica que a atividade a_4 deve ser executada antes de a_3. Por outro lado, a aresta (a_1, a_4) indica que a atividade a_1 deve ser executada antes de a_4. Mas as atividades a_1 e a_3 devem iniciar juntas. Conforme ilustra a Fig. 3.13 (b), não há uma única posição para o segmento de reta da atividade a_4 que satisfaça todas as restrições.

Para construir o *layout* de consistência devemos analisar as relações temporais de cada intervalo com os demais intervalos para o correto posicionamento dos segmentos de reta. Esta

construção torna-se muito trabalhosa caso a expressão tenha muitas relações temporais. Além disso, a existência de rótulos de arestas com mais de uma opção pode tornar a expressão consistente para algumas escolhas e inconsistente para outras, como no exemplo anterior.

O outro método para verificar inconsistências temporais é utilizar um grafo dirigido representando os pontos de início e fim dos intervalos conforme o trabalho de Zaidi [Zaidi, 99]. Neste tipo de representação, cada vértice representa o ponto de início ou fim das atividades e o sentido das arestas representa a relação (<, >) entre os vértices segundo ilustrado na Fig. 3.14.

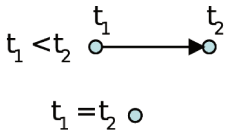


Fig. 3.14: Representação em grafo dirigido dos instantes de início/fim

Sendo t_1 e t_2 dois vértices que representam os instantes de início ou fim de um intervalo, a aresta terá o sentido t_1 para t_2 se $t_1 < t_2$ e terá o sentido de t_2 para t_1 se $t_2 < t_1$. Se dois ou mais instantes de início ou fim forem iguais (ocorrem ao mesmo tempo), eles serão representados por um mesmo vértice. A partir destas considerações podemos montar o dígrafo para cada relação temporal do conjunto de primitivas D conforme a Fig. 3.15.

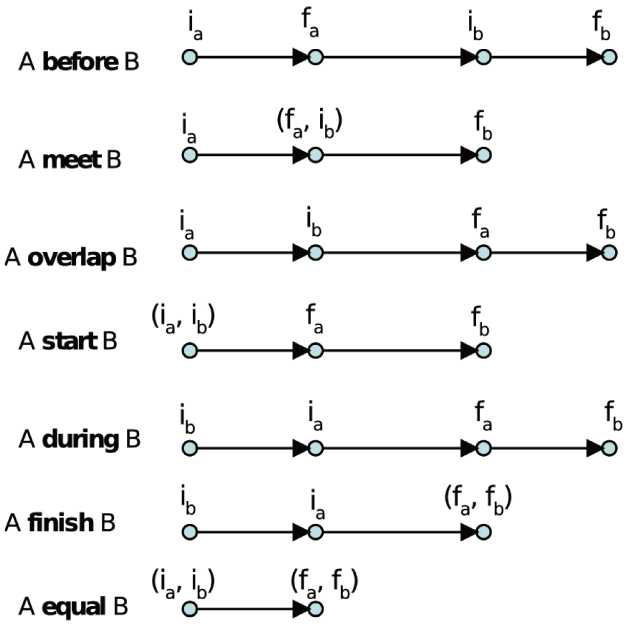


Fig. 3.15: Representação em grafo dos instantes de início/fim dos intervalos A e B.

Na Fig. 3.15, temos dois intervalos não nulos (A e B) e suas relações temporais possíveis sendo que i_a, f_a, i_b e f_b representam os instantes de início e de fim dos intervalos, respectivamente.

Dada um expressão E(A, R, F, G), podemos construir um grafo para os intervalos das atividades. Essa construção é feita aplicando para cada relação temporal entre duas atividades um grafo da Fig. 3.15. A partir daí, é feita a combinação desses grafos resultando num único grafo dirigido. Esta combinação é feita juntando os vértices que possuem pelos menos um valor em comum em seus rótulos. Por exemplo, se um vértice representa os instantes (i_a, f_b) e outro representa os instantes (i_c, i_a), então estes dois vértices resultarão num único vértice com o rótulo (i_a, f_b, i_c). Este vértice resultante terá todas as arestas incidentes dos vértices que a formaram[10]. Esse processo continua até que não existam dois ou mais vértices que representem um mesmo instante.

Zaidi demonstrou que um conjunto de sentenças temporais é consistente se, e somente se, a representação em grafo for acíclica, isto é, não tiver circuitos. Aplicando essa definição numa expressão E(A, R, F, G), podemos afirmar que ela é consistente se, e somente se, a representação em dígrafo dos instantes de início e fim das atividades for acíclica.

Para exemplificar esse método de verificação de inconsistência, temos na Fig. 3.16 os grafos

10 Arestas paralelas, que possuem pontas em comum, podem ser suprimidas em uma só pois representam uma mesma informação.

de instantes das atividades da expressão E2 para os casos (a) a_4 **during** a_3 e (b) a_4 **before** a_3.

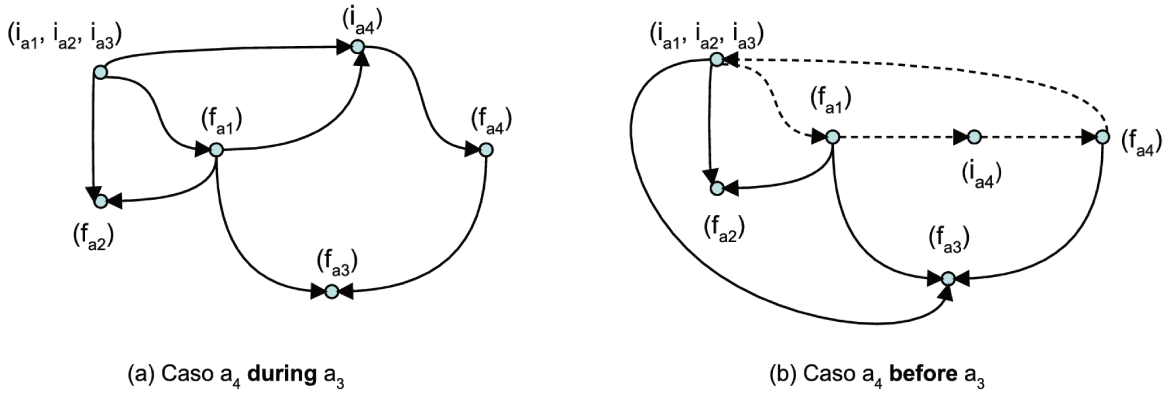


Fig. 3.16: Representação em grafo dos instantes de início/fim das atividades de E2.

Podemos verificar, através deste tipo de representação, que a expressão E2 é inconsistente para o caso da relação a_4 **before** a_3, pois existe um circuito no grafo da Fig. 3.16 (b) indicado pelos arcos tracejados. Uma vantagem deste tipo de verificação de inconsistência em relação ao *layout* de consistência é que na construção do grafo não temos que analisar as relações temporais de cada atividade com todas as outras. Outra vantagem é que este processo de verificação pode ser implementado em um programa possibilitando automatizar a análise. A verificação de existência de ciclos no dígrafo pode ser feita utilizando o algoritmo de busca em profundidade (DFS – *Deep First Search*) [Diestel, 05].

3.3 Conclusão

Este capítulo fez uma breve apresentação da metodologia grafo de relações proposta por Cruz [Cruz, 04]. O principal objetivo da metodologia GR é prover um algoritmo para automatizar a modelagem de mecanismos de coordenação livres de inconsistências temporais. Além disto, a metodologia permite a definição de relações alternativas, isto é, a possibilidade de selecionarmos em tempo de execução um tipo de dependência, dentro de um conjunto de opções, para um par de atividades. A metodologia permite também escolhermos se uma atividade dentro um subconjunto de atividades será ou não executada.

Esta metodologia possui três níveis de abstração (especificação, coordenação e execução) que utilizam modelos bem definidos. O nível de especificação utiliza grafos, enquanto os níveis de coordenação e execução usam redes de Petri.

Os métodos de verificação discutidos na seção 3.2 permitem analisar a especificação das relações temporais em busca de inconsistências antes da modelagem do MC. Isto garante que o modelo do MC satisfaz as restrições temporais impostas.

No próximo capítulo, apresentaremos o protótipo de uma ferramenta de modelagem de mecanismos de coordenação. Esta ferramenta utiliza a metodologia GR apresentada neste capítulo.

Capítulo 4

Ferramenta de suporte à modelagem de mecanismos de coordenação

O objetivo desse capítulo é apresentar o protótipo de uma ferramenta de suporte à modelagem de mecanismos de coordenação, além de propor uma especificação para o coordenador (nível N3 da metodologia GR). Esta ferramenta usa a metodologia GR, discutida no capítulo anterior, acrescentada de uma extensão. A extensão aqui proposta permite incluir o uso de recursos por parte das atividades em conjunto com a especificação das relações temporais. Esta extensão permite à metodologia GR abranger um número maior de sistemas.

A partir da especificação das atividades e das relações de dependências (temporal e de recursos), a ferramenta de modelagem de mecanismos de coordenação gera um arquivo no formato XML com a descrição do modelo em redes de Petri. Este arquivo é utilizado por um simulador de redes de Petri para verificação e validação do modelo e, posteriormente, é utilizado para construção do coordenador.

O coordenador é um software que interage com a aplicação (atividades) e é responsável por gerenciar as dependências entre as atividades.

Na primeira seção deste capítulo, apresentamos uma extensão para a metodologia GR. Nas demais seções, apresentamos a ferramenta de modelagem.

4.1 Extensão para a metodologia GR

A metodologia GR provê um mecanismo de coordenação que gerencia interdependências

temporais entre atividades. Alguns tipos de interdependências não-temporais também podem ser modelados usando a metodologia GR, por exemplo, uma relação de produtor-consumidor. Neste caso, podemos usar a relação temporal **meet** ou **before** para representar esse tipo de interdependência. Entretanto, se o recurso não for produzido por uma atividade, então essa forma de representação não é a mais adequada.

Conforme discutido na seção 2.2, um outro tipo de dependência bastante comum entre atividades de processos computacionais é a dependência de recursos. Há, basicamente, três tipos de dependência de recursos: compartilhamento, simultaneidade e volatilidade [Raposo, 00]. Com exceção da simultaneidade, que pode ser representada através da relação temporal **equal**, os outros dois tipos de gerenciamento de recursos não são cobertos pela metodologia GR.

O propósito desta seção é sugerir uma extensão para a metodologia GR (tanto no nível de especificação quanto no nível de coordenação) a fim de lidar também com dependências de recursos.

4.1.1 Especificação de recursos.

Recursos utilizados pelas atividades são representados por vértices, diferentes daqueles que representam as atividades. Associamos aos vértices de recursos um rótulo (t, n), onde t representa o tipo do recurso quanto a sua volatilidade e n, o número de instâncias disponíveis inicialmente.

Um recurso é dito volátil (v) se ele não está mais disponível após sua utilização e é dito não-volátil (nv) caso contrário. O sentido das arestas que ligam uma atividade a um recurso indica se a atividade utiliza aquele recurso (sentido recurso-atividade) ou se atividade produz o recurso (sentido atividade-recurso). Essas arestas possuem um valor numérico que indica quantas instâncias do recurso são produzidas ou consumidas pela atividade associada a ele.

Portanto, agora uma expressão do GR pode ser redefinida como uma expressão E(A, R, F, G, S, P, w, u) onde:

- A é o conjunto de atividades (vértices);
- $R \subset A \times A$ é o conjunto de relações temporais entre duas atividades;
- $F:R \rightarrow \wp(D)$ é a função que associa a cada elemento de R um subconjunto de D diferente de vazio;

- D = {e, s, f, d, o, m, b} é conjunto de relações temporais primitivas.
- G:A → $\wp$(A) é a função que associa a cada elemento de A um subconjunto de A;
- S é o conjunto de recursos (vértices);
- P ⊂ (A × S ∪ S × A) é conjunto das relações de dependência entre atividades e recursos;
- w:P → $\mathbb{N}$ é a função que associa cada elemento de P um número inteiro positivo e não nulo.
- u:S → T × $\mathbb{N}$ é a função que associa a cada elemento de S um par ordenado (t, n) onde t ∈ {nv, v} representa o tipo do recurso e n ∈ $\mathbb{N}$ representa o número inicial de instância.

A Fig. 4.1 mostra a representação gráfica da expressão E3(A, R, F, G, S, P, w, u) definida abaixo. Para facilitar a compreensão do grafo, os vértices do conjunto S (recursos) são representados por quadrados e os vértices do conjunto A (atividades), por círculos.

$A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$

$R = \{(a_2, a_3), (a_2, a_4), (a_1, a_5), (a_5, a_6), (a_7, a_6)\}$

$F(a_2, a_3) = F(a_2, a_4) = \{e\}$, $F(a_1, a_5) = \{b\}$, $F(a_5, a_6) = \{e, s\}$, $F(a_7, a_6) = \{f\}$

$G(a_2) = \{a_3, a_4\}$, $G(a_1) = G(a_3) = G(a_4) = G(a_5) = G(a_6) = G(a_7) = \varnothing$

$S = \{s_1, s_2\}$

$P = \{(a_1, s_1), (s_1, a_2), (s_2, a_5), (s_2, a_7)\}$

$w(a_1,s_1) = w(s_2, a_5) = 2$, $w(s_2, a_7) = w(s_1, a_2) = 1$

$u(s_1) = (nv, 2)$, $u(s_2) = (v, 5)$

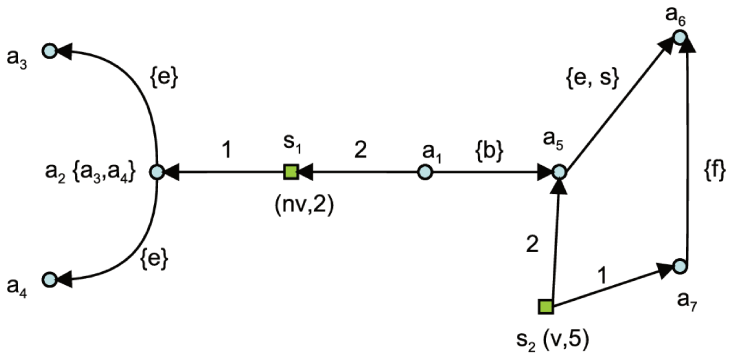


Fig. 4.1: Grafo da expressão E3.

Notemos na Fig. 4.1 que a atividade a_1 produz duas instâncias do recurso s_1 que por sua vez é

utilizado pela atividade a_2. Temos neste caso uma relação de produtor-consumidor. O recurso s_2, que é compartilhado entre duas atividades (a_5 e a_7), não é produzido por nenhuma atividade. Sua disponibilidade é feita por um agente externo à aplicação. Por exemplo, em um software de editoração a opção de imprimir um documento só é possível se a impressora (recurso) estiver disponível com um número de folhas suficiente.

4.1.2 Extensão para o mecanismo de coordenação

O modelo do mecanismo de coordenação da metodologia GR é estendido através da inclusão de lugares e arcos para incluir a utilização e a produção de recursos por parte das atividades.

Cada recurso é representado por um lugar (P_s). O número de fichas no lugar P_s representa o número de instâncias disponíveis. Se uma atividade utiliza um recurso então há um arco ligando o lugar P_s à transição que autoriza o início da atividade (t_{Ia}). O peso deste arco indica o número de instâncias do recurso necessário para o início da atividade. Caso a atividade produza um recurso, então há um arco ligando a transição que autoriza o fim da atividade (t_{Fa}) ao lugar P_s. O peso desse arco indica o número de instâncias do recurso produzidas pela atividade.

Se um recurso é não-volátil, então existe também um arco ligando a transição que autoriza o fim da atividade (t_{Fa}) que utilizou esse recurso ao lugar P_s. Esse arco, cujo peso é o número de instâncias de recursos utilizadas, permite que as fichas que estavam no lugar P_s antes do início da atividade voltem para o mesmo lugar após o fim da atividade, indicando que o recurso está novamente disponível. Caso contrário, isto é, se o recurso é volátil então não existe esse arco “de volta”. Desta forma, as fichas do lugar P_s consumidas no início da atividade não serão repostas ao final da execução. A Fig. 4.2 ilustra alguns exemplos de MCs com dependências de recursos.

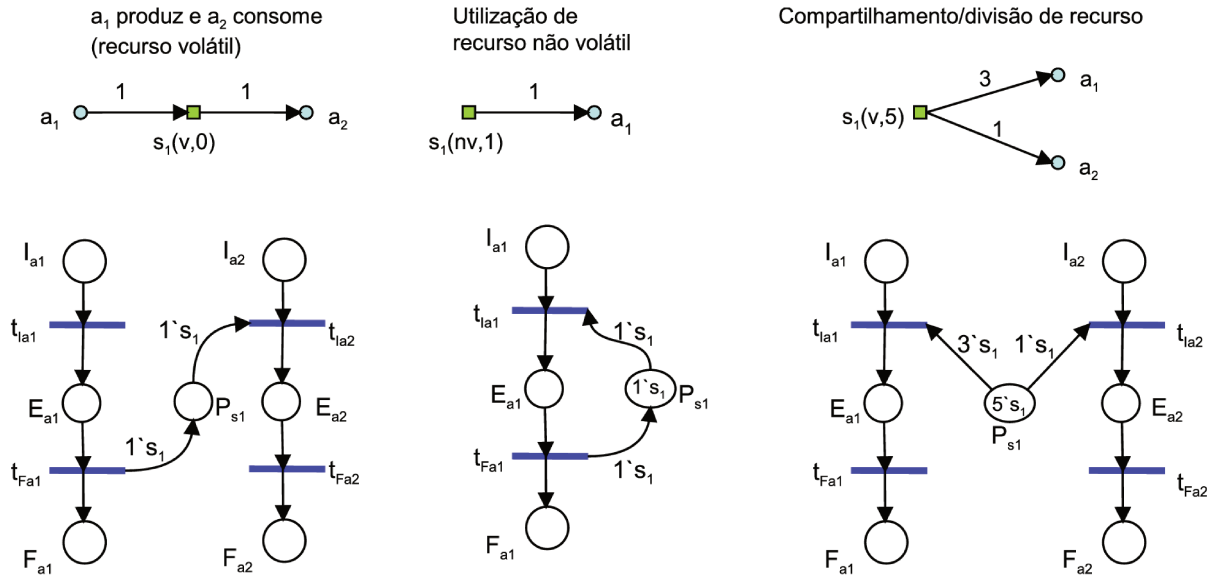


Fig. 4.2: Exemplos de MCs com dependências de recursos.

A inscrição n`s nos arcos e nos lugares da Fig. 4.2, sendo n um número inteiro positivo e s ∈ S, representa as n fichas requeridas pelo arco ou contidas no lugar.

Na Fig. 4.3, temos, por exemplo, um recurso s_1 volátil compartilhado por três atividades, um recurso s_2 volátil produzido por uma atividade e utilizado por outras duas e um recurso s_3 não-volátil compartilhado entre duas atividades. Na Fig. 4.4, temos o modelo do mecanismo de coordenação em redes de Petri deste exemplo.

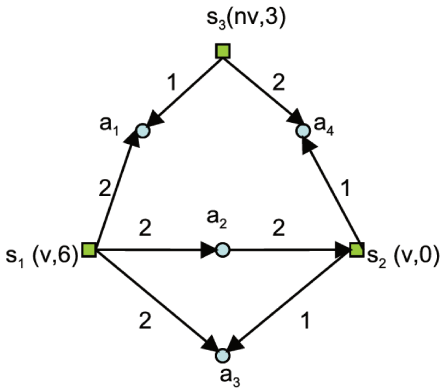


Fig. 4.3: Exemplo de especificação de dependência de recursos entre atividades.

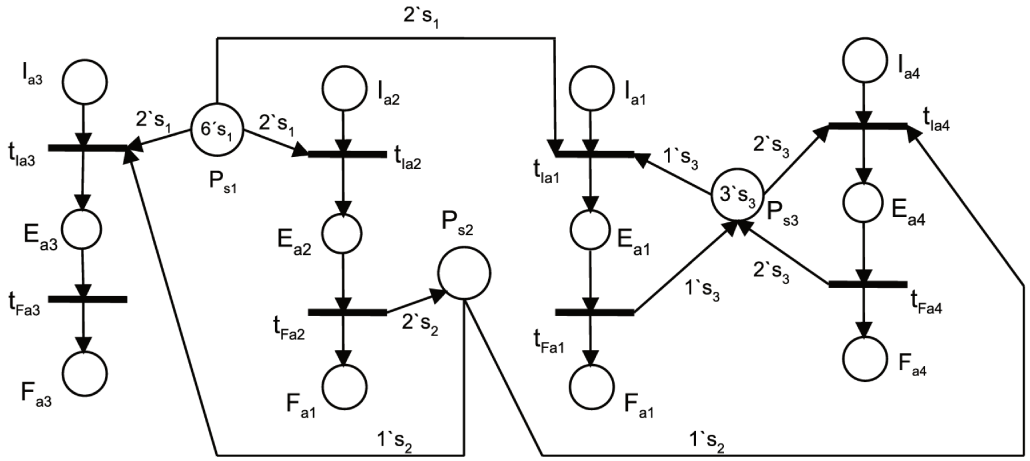


Fig. 4.4: Mecanismo de coordenação da expressão da Fig. 4.3.

O exemplo anterior ilustra apenas interdependências de recursos entre as atividades. Como citamos anteriormente, é possível colocar em um mesmo modelo de mecanismo de coordenação as dependências temporais e de recursos. Assim, garantimos a correta execução das atividades segundo suas restrições temporais e também segundo a disponibilidade de recursos.

A Fig. 4.5 mostra um exemplo de mecanismo de coordenação de três atividades com interdependências temporal e de utilização de recursos.

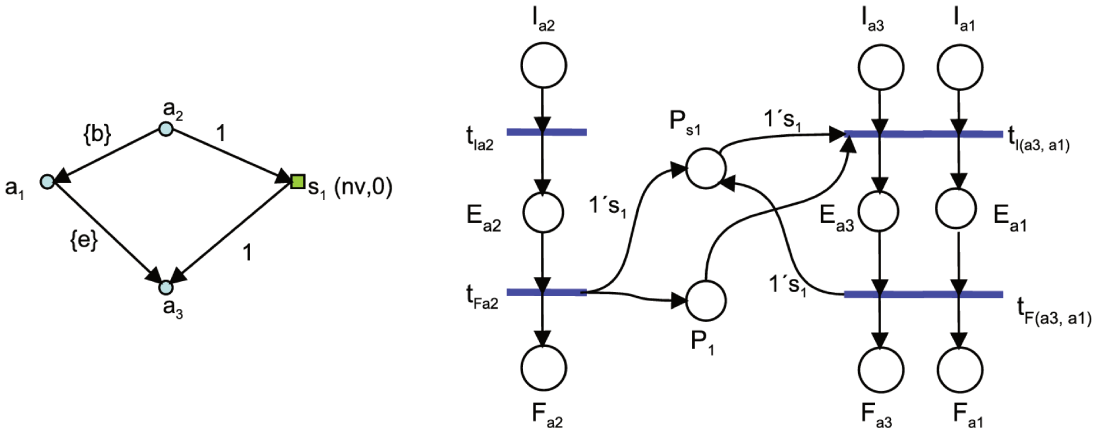


Fig. 4.5: Exemplo de MC com dependências temporal e de recursos.

A inclusão no mecanismo de coordenação dos lugares P_s e dos arcos associados a estes lugares é feita após a inserção das restrições temporais. Primeiro segue a identificação das

restrições temporais e a construção do mecanismo de coordenação conforme o algoritmo 2. Em seguida, para cada recurso s ∈ S é inserido um lugar P_s com a marcação inicial igual ao número de instâncias indicado no rótulo do seu vértice no grafo de relações. Depois, inserimos os arcos ligando os lugares P_s às transições t_{Iai} e/ou t_{Fai} das atividades que dependem desses recursos de acordo com a sua volatilidade, como descrito anteriormente. Ao final teremos um modelo em rede de Petri do mecanismo de coordenação de atividades com interdependências temporais e de recursos.

4.2 Implementação da ferramenta de modelagem

Nesta seção, apresentamos o protótipo da ferramenta de modelagem de MCs baseado na metodologia GR estendida. A finalidade é automatizar o processo de construção do mecanismo de coordenação. O aplicativo, denominado MAMC (Modelador Automático de Mecanismos de Coordenação), gera, a partir da especificação em grafos de relações, uma descrição da rede de Petri que modela o mecanismo de coordenação das atividades. Essa descrição é feita através de um arquivo no formato XML [XML, 06] utilizando a linguagem *Petri Net Markup Language* (PNML) [Billington *et al*, 03]. A idéia principal é que a partir do modelo em redes de Petri, o projetista possa simular e analisar o comportamento do sistema antes de implementar o modelo.

O diagrama da Fig. 4.6 mostra de modo geral a seqüência de passos para gerar o MC no MAMC. A entrada de dados (especificação do grafo de relações) no MAMC é feita através de uma interface gráfica. O usuário constrói o grafo de relações inserindo em uma área de trabalho as atividades e os recursos que serão utilizados, indicando os relacionamentos entre eles.

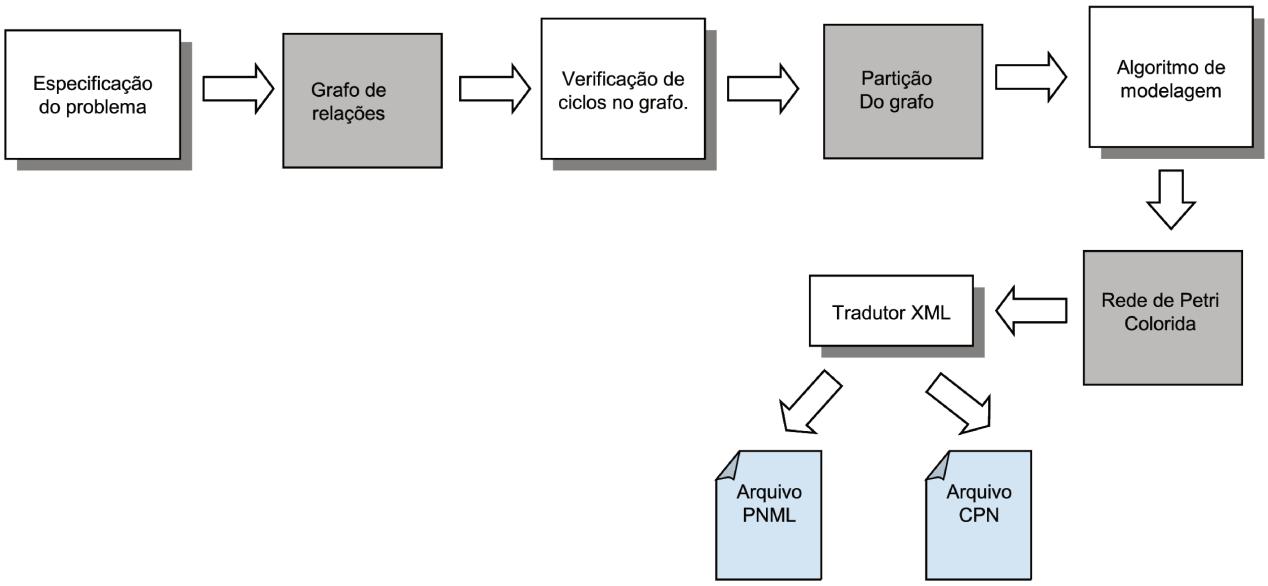


Fig 4.6: Seqüência de passos para gerar o modelo do MC no MAMC.

A interface gráfica foi feita utilizando a biblioteca Jgraph [Jgraph, 06] versão 5.8.2.2. Essa biblioteca possibilita o desenho e a edição do grafo em um ambiente de janelas. A biblioteca Jgraph utiliza por sua vez componentes do Java Swing [Loy *et al*, 02]. A Fig. 4.7 mostra a interface do programa com um exemplo de utilização.

A opção por uma interface gráfica é para facilitar o uso da ferramenta. Poderíamos fornecer o grafo de relações por outros meios, por exemplo, através de um arquivo.

Uma vez que fornecemos o grafo, o MAMC verifica a existência de ciclos que podem indicar inconsistências nas restrições temporais. Se houver algum, será pedido ao projetista rever as relações temporais entre as atividades.

Feita a verificação de ciclos e não havendo nenhum, o MAMC gera o particionamento do grafo (conforme descrito na seção 3.1.3) que será utilizado pelo algoritmo de modelagem. Este algoritmo constrói a rede de Petri inserindo os MCL (seção 3.1.2), as restrições temporais (através das operações de fusão e fissão) e as dependências de recursos.

O tradutor XML é responsável por gerar o arquivo que descreve a rede de Petri. O tradutor XML é extensível, podendo ser utilizado para gerar diferentes formatos de XML para redes de Petri. Nesta fase de desenvolvimento, são gerados arquivos nos formatos PNML (seção 4.3) e CPN, usado no simulador CPN Tools [CPN, 07] (ver Anexo A).

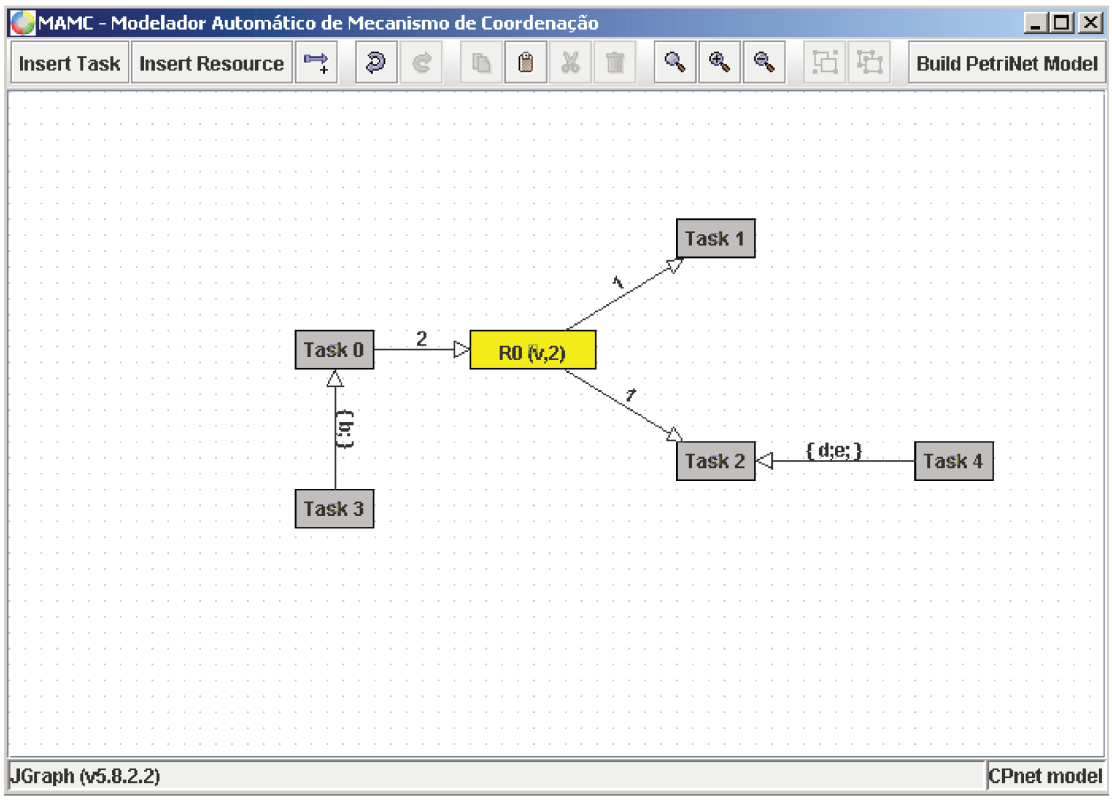


Fig. 4.7: Interface gráfica do programa MAMC

A utilização do MAMC é a primeira fase do processo de construção do coordenador. O processo completo é ilustrado na Fig. 4.8. O aplicativo MAMC é responsável pela especificação do problema e pela modelagem do coordenador. O arquivo PNML do modelo é utilizado por um simulador de redes de Petri para que possamos fazer a verificação e validação do modelo. Através da simulação, podemos verificar se o modelo corresponde ao problema de interesse. Se houve alguma inconsistência, então devemos voltar ao MAMC e rever a especificação do problema de interesse. O arquivo PNML depois é usado para criar o coordenador, conforme discutido na seção 4.4.

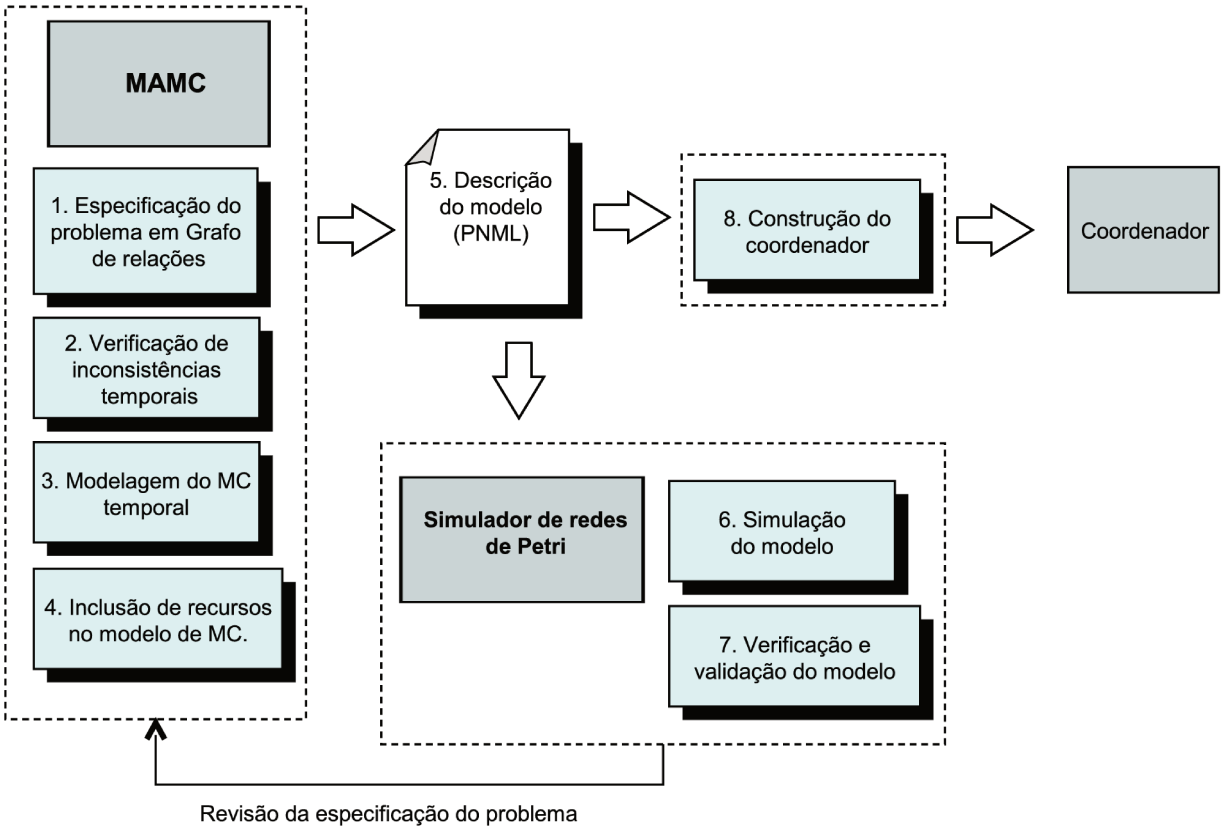


Fig. 4.8: Processo de construção do coordenador.

4.2.1 Modelo de classes

O diagrama conceitual das classes que implementam um grafo de relações é mostrado na Fig. 4.9.

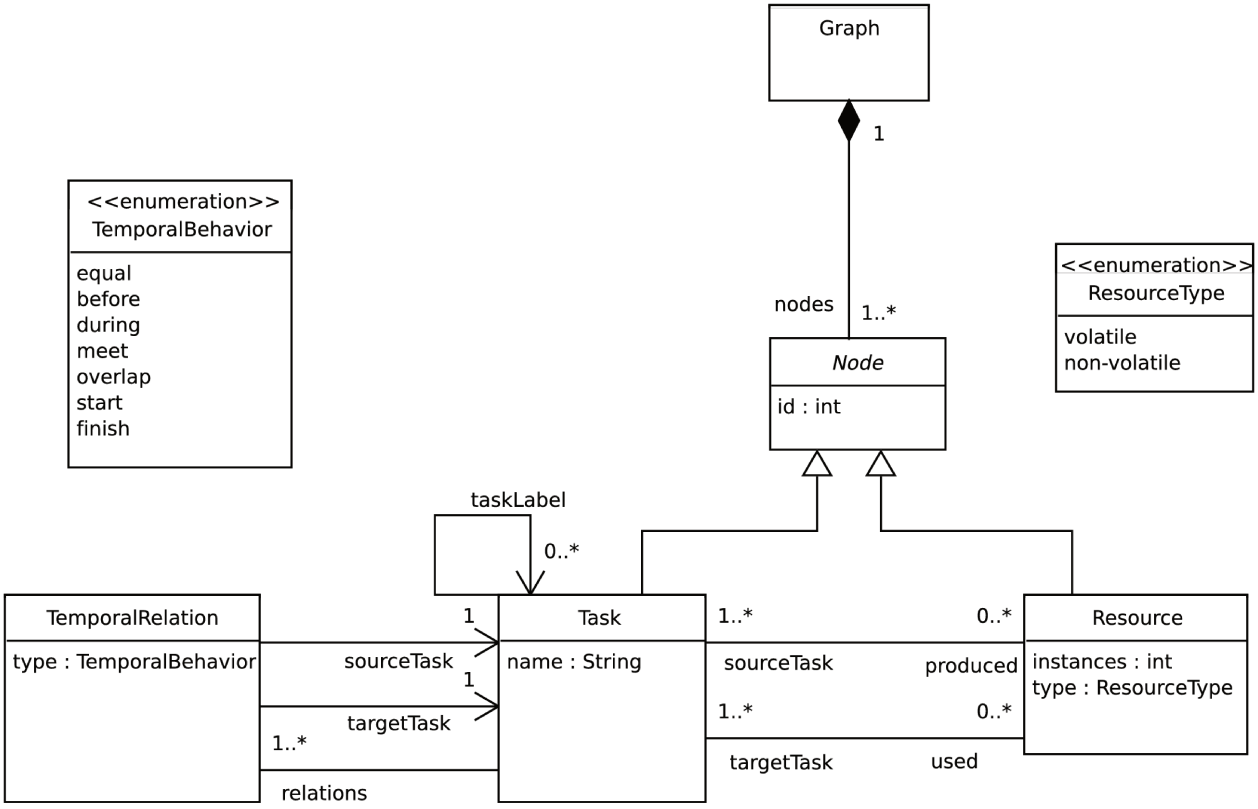


Fig. 4.9: Diagrama de classes que implementam um grafo de relações.

A classe *Graph* (grafo) é composta pela classe *Node* (nó ou vértice). A classe *Node* é uma classe abstrata, portanto, não pode ser instanciada. Assim, para representar um vértice usamos ou a classe *Task* (tarefa) ou a classe *Resource* (recurso), que são subclasses de *Node*.

A classe *Task* está associada à classe *TemporalRelation* (relação temporal), ou seja, cada atividade possui uma ou mais relações temporais com outra atividade. Cada objeto da classe *TemporalRelation* contém exatamente duas atividades, que são os extremos de uma aresta (*sourceTask* e *targetTask*). A classe *Resource* possui o atributo *type* (tipo) que indica se o objeto *resource* é volátil (*volatile*) ou não volátil (*non-volatile*). As associações entre as classes *Task* e *Resource* indicam quais recursos um objeto atividade produz e/ou consume. Este modelo de classes não permite que tenhamos uma aresta do tipo recurso-recurso, sendo que esse tipo de relação não é especificado pela metodologia GR estendida.

Esse modelo de classes do grafo de relações é utilizado no MAMC para realizar as etapas 1 e 2 da Fig. 4.8.

O diagrama de classes que implementam uma rede de Petri é mostrado na Fig. 4.10.

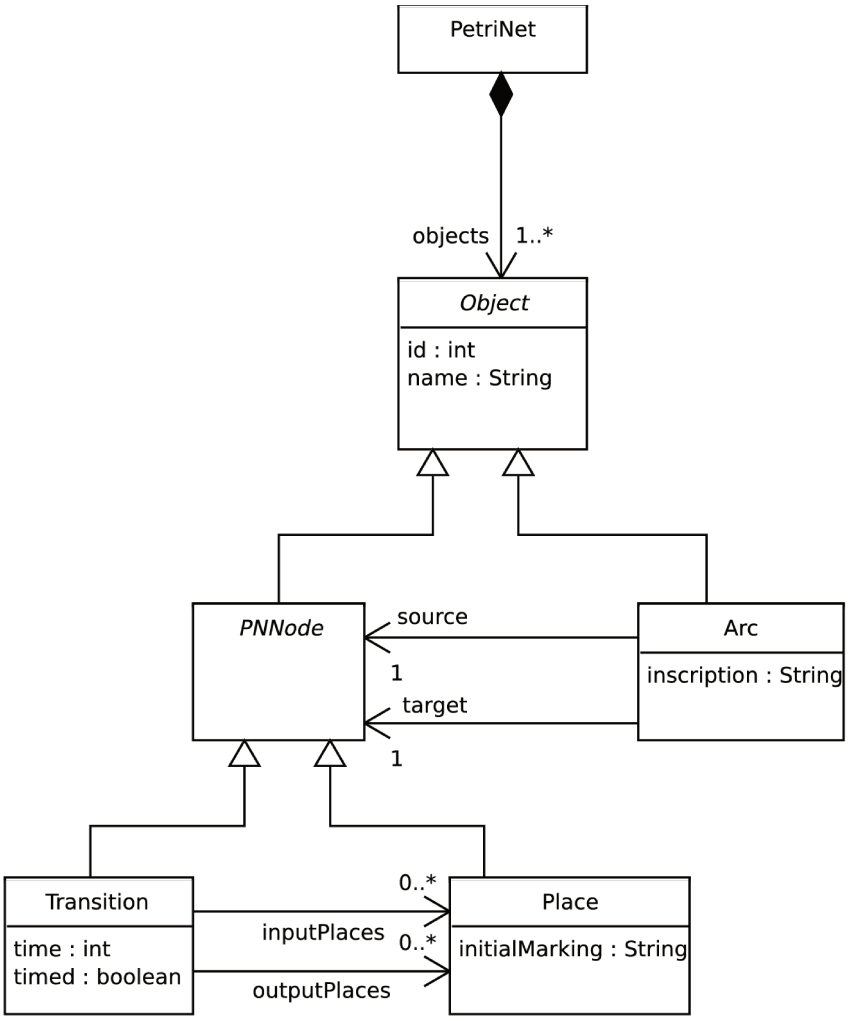
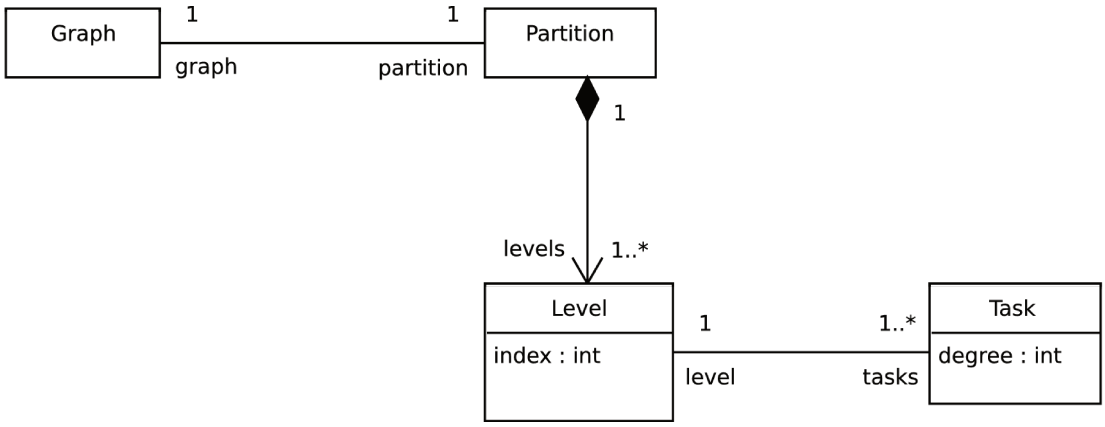


Fig. 4.10: Diagrama de classes que implementam uma rede de Petri.

As classes *Object* e *PNNode* são abstratas, sendo que suas instanciações são feitas através das classes filhas *Arc* (arco), *Transition* (transição) ou *Place* (lugar). Aqui, a rede de Petri é vista como um grafo dirigido sendo que os nós (*node*) são ou do tipo transição (*transition*) ou do tipo lugar (*place*). Essa representação é semelhante à utilizada pela especificação do formato PNML, a ser discutido na seção seguinte.

Antes da construção do mecanismo de coordenação, é necessário dividir as atividades em níveis como discutido na seção 3.1.3. Temos então uma partição do grafo de relações composto

por n níveis onde cada um contém uma ou mais atividades (Fig. 4.11). É a partir dessa partição que o algoritmo de modelagem trabalha (etapa 3 na Fig. 4.8).



Fig, 4.11: Diagrama de classes que implementam uma partição do GR.

4.3 Estrutura do arquivo PNML

A fim de facilitar a transferência de modelos de rede de Petri entre diversas ferramentas, foi necessário definir um padrão de formato de arquivo de transferência. Uma proposta inicial para estabelecer um padrão internacional para redes de Petri de alto nível ocorreu em 1995 [ISO, 95]. No ano de 2000, surgiram as primeiras propostas baseadas em XML como a *Petri Net Markup Language* (PNML) [Jüngel *et al*, 00]. O padrão PNML vem sendo adotado por diversas ferramentas que usam redes de Petri, pois é flexível o suficiente para integrar diferentes tipos de redes de Petri e é aberto a extensões futuras [Billington *et al*, 03].

O padrão PNML considera uma rede de Petri como um grafo dirigido e rotulado onde toda informação da rede é representada em rótulos. Um rótulo pode estar associado a um nó, a um arco ou à rede em si [ISO, 05]. A Fig 4.12 mostra a estrutura básica do PNML. Notemos que a representação em diagrama UML [Rumbaugh, 99] da Fig. 4.12 não define a sintaxe XML para documentos PNML.

Um documento que atende aos requisitos do PNML é chamado de *Petri Net Document* (PetriNetDoc). Este pode conter várias redes de Petri sendo que cada uma possui um identificador

próprio e um tipo definido. A definição dos rótulos da rede de Petri utiliza a sintaxe do Documento de Convenções que permite a uma ferramenta reconhecer a estrutura da rede de Petri.

Conforme mostra a Fig. 4.12 cada objeto possui um rótulo (*label*) que pode ser ou uma anotação ou um atributo. Um rótulo do tipo anotação é usado para exibir uma informação junto ao objeto, enquanto um rótulo do tipo atributo define uma característica do objeto. Cada objeto e anotação possuem também informações referentes à sua posição espacial, e opcionalmente, a sua forma, cor, tipo de fonte e tamanho.

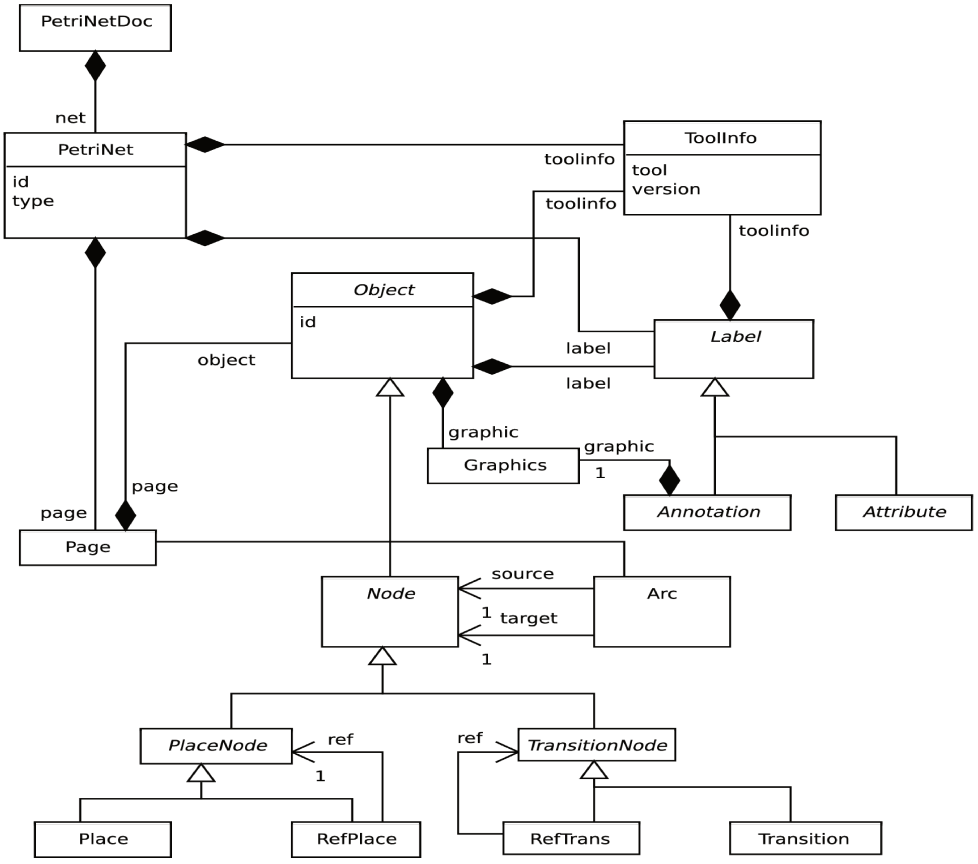


Fig. 4.12: Diagrama UML da estrutura básica do PNML.

O padrão PNML permite que uma rede seja separada em diferentes páginas. Uma página é um objeto que contém outros objetos, inclusive outras páginas. Uma restrição quanto ao uso de várias páginas para representar uma rede é que um arco só pode conectar nós (lugar ou transição) em uma mesma página.

A linguagem de esquema utilizada para a implementação do formato PNML é a RELAX NG [Clark, 01], sendo que esta é modular e tem suporte à validação. Usando a gramática RELAX NG, cada elemento do meta-modelo PNML é traduzido para o XML conforme a Tab. 4.1.

Tab. 4.1: Elementos do meta-modelo PNML

Classe	Elemento XML	Atributo XML
PetriNetFile	<pnml>	
PetriNet	<net>	id: ID type: anyURI
Place	<place>	id: ID
Transition	<transition>	id: ID
Arc	<arc>	id: ID source: IDRef (Node) target: IDRef (Node)
Page	<page>	id: ID
RefPlace	<referencePlace>	id: ID ref: IDRef (Place or RefPlace)
RefTrans	<referenceTransition>	id: ID ref: IDRef (Transition or RefTrans)
ToolInfo	<toolspecific>	tool: string version: string
Graphics	<graphics>	

A estrutura do elemento <graphics> depende do tipo de elemento ao qual ele pertence. A Tab. 4.2 mostra os elementos que podem ocorrer na subestrutura de <graphics>. O elemento <position> define uma posição absoluta e é requerido por todos os nós, enquanto que o elemento <offset> define uma posição relativa e é requerido pelas anotações. Os demais sub-elementos de <graphics> são opcionais. Para um arco, uma sequência de <position> define seus pontos intermediários.

Tab. 4.2: Sub-elementos de <graphics>

Elemento mestre	Sub-elementos de <graphics>
Node, Page	<position> (requerido) <dimension> <fill> <line>
Arc	<position> (zero ou mais) <line>
Annotation	<offset> (requerido) <fill> <line> <font>

Quanto aos rótulos, sua definição está incluída no Documento de Convenções, bem como na definição do tipo de rede de Petri (*Petri Net Type Definition - PNTD)* . O Documento de Convenções consiste de uma seqüência de definição de rótulos. A cada rótulo é designado um único nome. A Tab. 4.3 mostra alguns exemplos de rótulos definidos no Documento de Convenções. Notemos que alguns deles possuem um mesmo elemento XML. Isto é possível, pois PTMarking e HLMarking são utilizados em tipos distintos de rede de Petri. Se o rótulo for uma estrutura de dado simples, o valor é indicado através do elemento <text>. Para rótulos indicados como estruturado, são utilizados os elementos <text> e <structure>, criando assim uma árvore XML.

A visualização da rede de Petri é tarefa da transformação do documento PNML para o formato SVG (*Scalable Vector Graphics*) [Ferraiolo *et al*, 03]. O SVG é um padrão para descrição de gráficos vetoriais de duas dimensões também baseado em XML.

Tab. 4.3: Elementos XML para rótulos

Nome do rótulo	Elemento XML	Domínio
Name	<name>	String
PTMarking	<initialMarking>	Inteiro não negativo
HLMarking	<initialMarking>	Estruturado
PTCapacity	<capacity>	Inteiro não negativo
HLSort	<sort>	Estruturado

4.4 Construção do coordenador

Uma vez gerado o arquivo PNML que representa o modelo do MC, iniciamos a construção do coordenador. A construção do coordenador é feita traduzindo o modelo do mecanismo de coordenação obtido em um componente de software capaz de se comunicar com as atividades da aplicação. Esta comunicação é feita através de uma interface que possibilita abstrair do coordenador detalhes de execução das atividades. A separação entre o mecanismo de coordenação e as atividades permite, por exemplo, alterarmos o modo de execução das atividades sem ter que mudar o mecanismo de coordenação. Podemos também alterar as dependências entre as atividades, alterando o MC, sem ter que alterar as atividades em si. Essa separação permite que o coordenador seja independente do tipo da aplicação coordenada[11]. A Fig. 4.13 ilustra em um nível de abstração alto o sistema de coordenação.

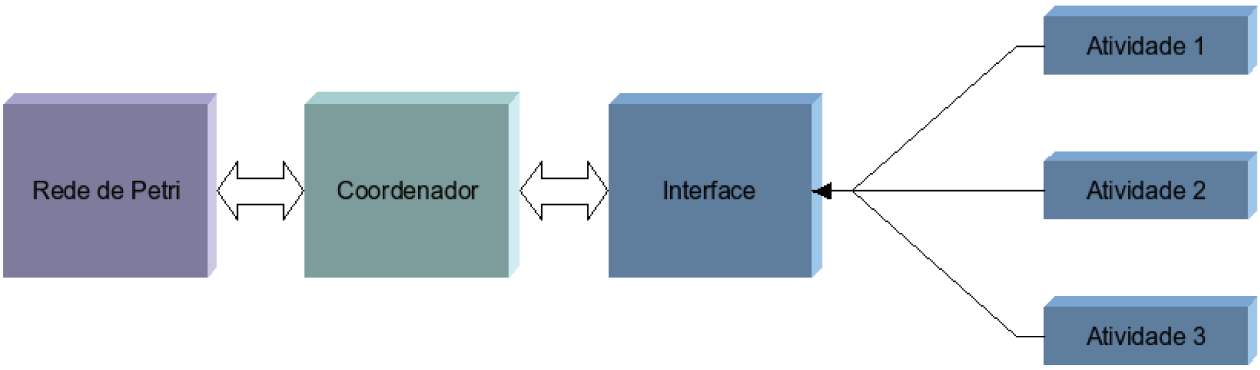


Fig. 4.13: Sistema de coordenação

O coordenador gerencia as dependências das atividades com base na rede de Petri gerada pelo MAMC. A cada atividade está associada uma transição que indica seu início (t_{Iai}). Se esta transição t_{Iai} estiver habilitada, então a atividade a_i está autorizada a iniciar. Por exemplo, na Fig. 4.14, a atividade a_1 está autorizada a iniciar sua execução pois a transição t_{Ia1} está habilitada. Entretanto, a atividade a_2 não poderá ser executada enquanto a transição t_{Ia2} não estiver habilitada. As atividades são unidades de execução interdependentes em uma aplicação e necessitam de autorização do coordenador para serem executadas.

11 Essa abordagem usando componentes de softwares que tratam as dependências e as atividades separadamente foi apresentada por Raposo [Raposo, 01].

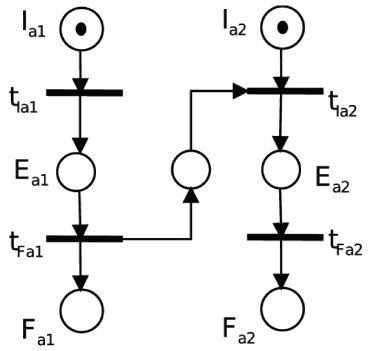


Fig. 4.14: MC da relação a_1 **before** a_2.

4.4.1 Comunicação coordenador – atividade

Para a comunicação entre o coordenador e as atividades definimos um protocolo. Este possibilita que as atividades enviem e recebam mensagens do coordenador. Os sinais usados na comunicação estão descritos na Tab. 4.4.

Tab. 4.4: Sinais trocados entre as atividades e o coordenador

Sinal	Característica
REQ_START	A atividade solicita autorização ao coordenador para iniciar.
TASK_FINISHED	A atividade indica que finalizou sua execução.
START_TASK	O coordenador autoriza o início da atividade.
FINISH_TASK	O coordenador autoriza o fim da atividade.
UNBLOCK_TASK	O coordenador desbloqueia a atividade.
WAIT	O coordenador indica que a atividade deve aguardar por uma autorização mantendo seu estado atual.

O sinal REQ_START é enviado pela atividade ao coordenador assim que o usuário ou o sistema solicita o início da atividade. Neste caso, o coordenador verifica no modelo de rede de Petri, se a transição de início está habilitada. Se estiver, então o coordenador envia o sinal START_TASK autorizando o início da execução da atividade Caso contrário o coordenador envia o sinal WAIT indicando que a atividade deve manter seu estado atual. Se a transição em questão também for responsável pelo início de outras atividades (caso das relações **equal** e **start**),

então o coordenador também envia o sinal START_TASK para estas atividades.

Quando uma atividade termina, ela envia o sinal TASK_FINISHED ao coordenador. O coordenador pode, se necessário, finalizar ou iniciar automaticamente uma atividade enviando o sinal FINISH_TASK ou START_TASK.

Os sinais que o coordenador envia para as atividades, mudam o estado delas (exceção do sinal WAIT). Os estados que as atividades podem assumir são indicados na Tab. 4.5 e seu diagrama de transição está na Fig. 4.15. Inicialmente, todas as atividades estão no estado Blocked (bloqueadas) e o coordenador verifica quais as transições de início (t_{Ia}) estão habilitadas. Para aquelas atividades que estão habilitadas, o coordenador envia o sinal UNBLOCK_TASK.

Tab. 4.5: Estados possíveis da atividade

Estado	Características
Blocked	A atividade está bloqueada pois existe pelo menos uma restrição para sua execução (dependência temporal e/ou falta de recurso).
Ready	Não há restrições para o início da atividade. Ela se encontra pronta para execução.
Executing	A atividade esta sendo executada.
Finished	A atividade está finalizada.

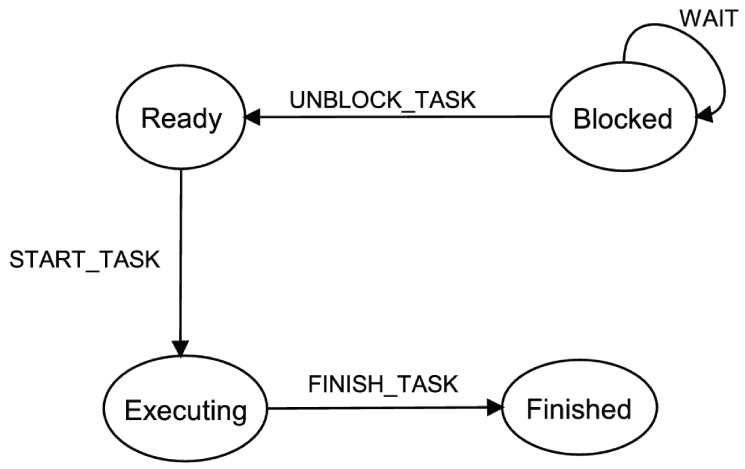


Fig. 4.15: Diagrama de estado da atividade.

Em resumo, o coordenador trabalha observando os eventos que ocorrem nas atividades (solicitação de início, término de execução), isto é, ele acompanha a dinâmica de execução das atividades.

4.4.2 Comunicação coordenador – rede de Petri.

Como citamos anteriormente, o coordenador utiliza o modelo em rede de Petri para gerenciar as dependências entre as atividades. A comunicação entre o coordenador e a rede é feita diretamente usando os sinais descritos na Tab. 4.6.

Tab. 4.6: Sinais trocados entre o coordenador e a rede de Petri.

Sinal	Característica
VERIFY_TRANS	O coordenador verifica na rede se uma determinada transição está habilitada ou não.
FIRE_TRANS	O coordenador solicita o disparo de uma transição habilitada.
IS_ENABLED	Indica ao coordenador que a transição em questão está habilitada.
NOT_ENABLED	Indica ao coordenador que a transição em questão não está habilitada.

Quando uma atividade solicita uma autorização de início, o coordenador verifica se a transição t_{Ia} está habilitada enviando o sinal VERIFY_TRANS à rede de Petri. Se a transição não estiver habilitada, então o coordenador recebe o sinal NOT_ENABLED e não autoriza o início da atividade. Caso contrário, o coordenador recebe o sinal IS_ENABLED, autorizando o início da atividade e em seguida envia o sinal FIRE_TRANS para disparar a transição em questão. Na Fig. 4.16, temos o diagrama de comunicação entre o coordenador e a rede de Petri. O módulo Leitor PNML é responsável em converter o arquivo PNML no objeto de rede de Petri com o qual o coordenador se comunica.

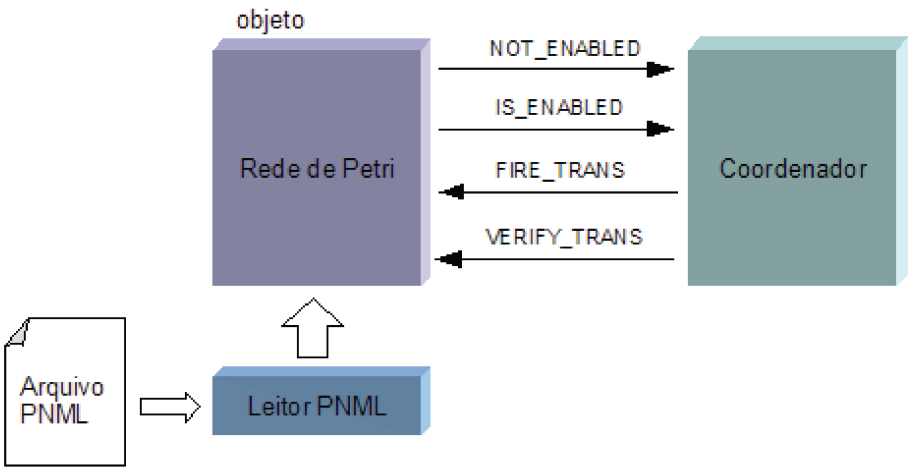


Fig. 4.16: Comunicação entre o coordenador e a rede de Petri.

4.5 Conclusão

Este capítulo apresentou o protótipo de uma ferramenta de modelagem de mecanismos de coordenação, uma extensão para a metodologia GR e uma proposta inicial de especificação do coordenador. O processo de construção do mecanismo de coordenação é dividido em três fases: especificação, simulação e construção do coordenador. O aplicativo MAMC desenvolvido neste trabalho converte a representação das dependências entre as atividades em forma de grafo (fase de especificação) no modelo de coordenação em redes de Petri. Optamos por utilizar um padrão aberto de representação de rede de Petri a fim de facilitar o intercâmbio de arquivos entre diferentes aplicações que façam uso das redes de Petri, embora haja algumas diferenças nos arquivos PNML utilizados pelas aplicações de diferentes desenvolvedores, pois o padrão PNML ainda encontra-se em desenvolvimento.

Na fase de simulação, optamos por utilizar um simulador disponível livremente, neste caso utilizamos o CPN Tools [CPN, 07]. Existem várias opções de simuladores de RP, gratuitos ou não, que atendem diferentes necessidades. O projetista pode utilizar a ferramenta que achar mais adequada.

A última fase é a construção do coordenador. Apresentamos um modelo de construção do coordenador. Esse modelo descreve o funcionamento básico do coordenador e de como ele faz o mapeamento entre o modelo de rede Petri e as atividades.

No próximo capítulo, discutiremos dois exemplos de uso da ferramenta de modelagem.

Capítulo 5

Exemplos de aplicação

O propósito deste capítulo é apresentar exemplos de utilização da ferramenta de modelagem apresentada no capítulo anterior. Os exemplos apresentados serão os seguintes:

- Exemplo 1: Animação computacional.
- Exemplo 2: Sistema de workflow.

Ambos os exemplos são processos computacionais compostos por atividades interdependentes. Desejamos construir um mecanismo de coordenação que garanta a execução correta das atividades.

A idéia é, dado um conjunto de atividades e suas relações de interdependências, criar um modelo de coordenação que possa ser testado (simulado) antes de sua implementação. Uma vez que tenhamos testado o modelo e verificado todos os requisitos, podemos construir o coordenador conforme discutido no capítulo anterior.

Neste capítulo, daremos ênfase à fase de especificação e modelagem de problemas de coordenação usando o MAMC.

5.1 Animação computacional

Um sistema de animação computacional contém, entre outros componentes, um mecanismo de controle responsável em gerar as ações desejadas. Há diversas técnicas de controle de

animação encontradas na literatura. N. M. Thalmann e D. Thalmann [Thalmann, 91] propõem três categorias de Método de Controle de Movimento (MCM) baseadas nas informações fornecidas para o controle de movimento dos atores[12]: geométrico, físico e comportamental.

O MCM comportamental especifica o movimento de um ator em termos de um comportamento desejado. Este método permite ao animador especificar a ação desejada e deixar para o sistema de animação o trabalho de criar a animação. Podemos usar esse método de controle com atividades simples e usá-las conjuntamente para criar uma animação mais complexa. Neste exemplo, desejamos criar o movimento de andar de uma figura humana a partir de animações mais simples como os movimentos dos braços e das pernas.

No trabalho de Camargo [Camargo, 95] foi apresentado o controle de uma figura bípede que se assemelha a um ser humano. A idéia deste trabalho é prover um mecanismo de controle global orientado a eventos responsável pelas interações entre os membros. Este controle é modelado usando Máquina de Estados Estendida (ESM – Extended State Machine).

O sistema de animação do exemplo atual utiliza um MCM comportamental baseado na proposta de Camargo [Camargo, 95] juntamente com a metodologia GR.

A partir da idéia de como uma figura humana caminha, o animador descreve esse movimento em termos de relações temporais entre as atividades de movimentar as pernas e os braços.

A seguir temos a especificação do processo usando uma expressão de grafo de relações.

$A = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$;

$R = \{(a_0, a_1), (a_4, a_1), (a_1, a_2), (a_6, a_2), (a_2, a_3), (a_5, a_3), (a_3, a_7)\}$;

$F(a_0, a_1) = \{b\}$, $F(a_4, a_1) = \{e\}$, $F(a_1, a_2) = \{b\}$, $F(a_6, a_2) = \{e\}$, $F(a_2, a_3) = \{b\}$, $F(a_5, a_3) = =\{e\}$, $F(a_3, a_6) = \{b\}$;

As atividades a_0 ... a_7 estão descritas na Tab. 5.1 e a descrição do processo de animação é feita graficamente usando o MAMC. A Fig. 5.1 ilustra o grafo de relações temporais.

12 Ator é o objeto de cena que se deseja animar.

Tab. 5.1: Descrição das atividades referentes à animação.

Atividade	Descrição
a_0	Levantar perna direita.
a_1	Deslocar e reposicionar perna direita.
a_2	Levantar perna esquerda.
a_3	Deslocar e reposicionar perna esquerda.
a_4	Move braço esquerdo para frente.
a_5	Mover braço direito para frente.
a_6	Mover braço esquerdo para trás.
a_7	Mover braço direito para trás.

O caminhar humano é conceitualmente bem conhecido. Neste tipo de movimento, as pernas se movimentam para frente e para trás, provendo suporte ao corpo, e os braços acompanham o movimento das pernas. As relações temporais entre as atividades reproduzem a seqüência de passos deste movimento. Algumas restrições devem ser impostas como não permitir que ambas as pernas estejam no ar ao mesmo tempo e que uma mesma perna são seja levantada mais de uma vez consecutivamente.

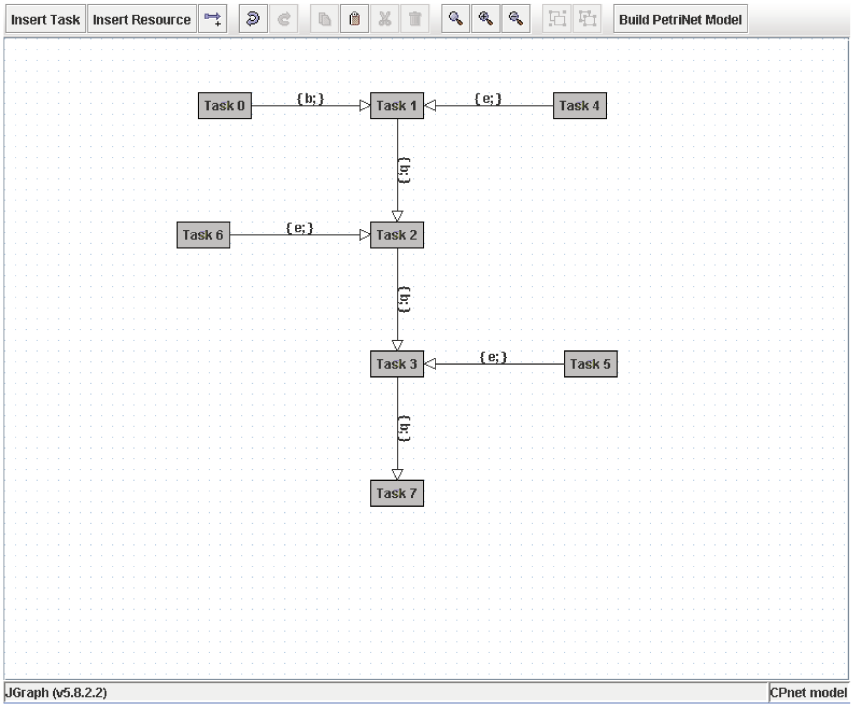


Fig. 5.1: Grafo de relações temporais do exemplo de animação.

Uma vez que o usuário tenha inserido o grafo utilizando a interface do MAMC conforme a Fig. 5.1, o programa constrói o modelo do mecanismo de coordenação e o salva em um arquivo denominado *mc_model.cpn*. Este arquivo contém a descrição da rede de Petri usando o formato CPN, linguagem de marcação (XML) específica do programa CPN Tools.

Usando esse arquivo, podemos simular o mecanismo de coordenação. A Fig. 5.2 ilustra o modelo do processo de animação em seu estado inicial. Nesta figura, a inscrição 1`a ao lado dos lugares indica a existência de uma ficha da cor "a". Cada atividade é representada pelos lugares I_{ai}, E_{ai}, e F_{ai} (i = 0, 1, ..., 7). Uma ficha no lugar E_{ai} indica que a atividade a_i está sendo executada.

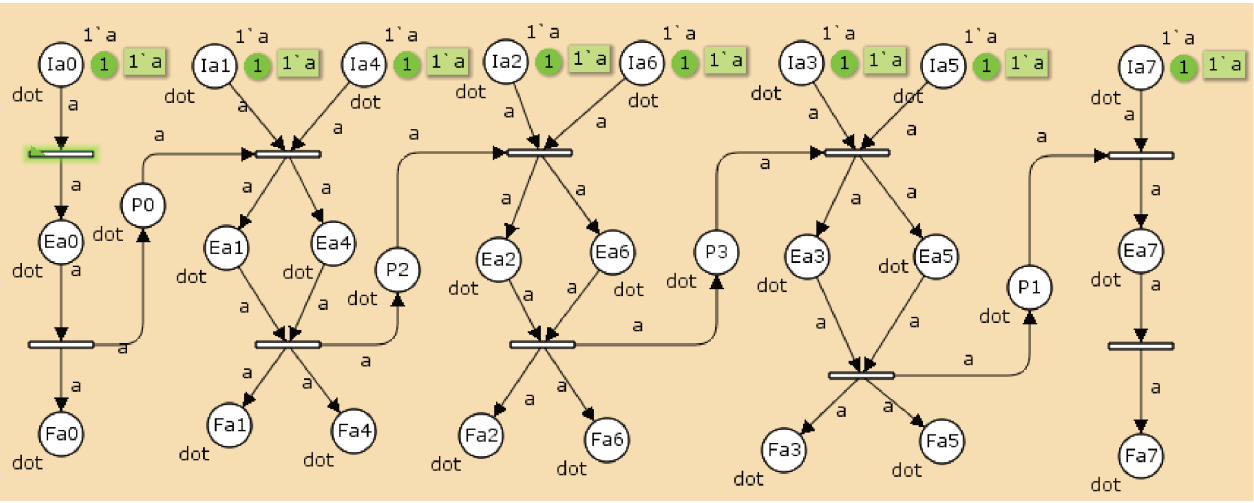


Fig. 5.2: RP do sistema de animação com a marcação inicial.

Inicialmente, temos um ficha "a" (1`a) em todos os lugares I_{ai}, indicando que as atividades estão requisitando autorização de início. Entretanto, somente a atividade a_0 está autorizada a iniciar, pois a transição que precede o lugar E_{a0} é a única habilitada. Somente após o fim de a_0, é que a_1 e a_4 estarão autorizadas a iniciar e assim por diante. A simulação indica que as atividades são executadas respeitando as relações temporais definidas no grafo. Por exemplo, não há uma situação em que as atividades a_0 e a_2 aconteçam ao mesmo tempo (situação na qual as duas pernas estão levantadas).

O processo pára quando todas as atividades estão finalizadas corretamente. Neste caso, um recurso de *timeout* poderia ser utilizado para reiniciar o processo.

Esse exemplo ilustra os passos percorridos para desenvolver um mecanismo de coordenação

para uma animação computacional. As atividades podem ser reutilizadas para outros tipo de animação como a ação de pular ou correr, bastando ao animador mudar os relacionamentos entre as atividades. A vantagem deste tipo de abordagem é que o animador pode a partir de um conjunto de animações simples criar diferentes cenas sem se preocupar em recriar todas as atividades.

Na próxima seção, abordaremos o uso da ferramenta de modelagem em um sistema de workflow.

5.2 Sistema de workflow

Nesta seção, mostraremos um exemplo de uso do MAMC que inclui a utilização de recursos conjuntamente com relações temporais. Utilizaremos aqui um exemplo de sistema de workflow.

A metodologia GR (tanto a versão orginal quanto a estendida) não é um modelo completo de especificação de *workflow*, pois não descreve todos os padrões de *workflow* [van der Aalst *et al*, 02]. Entretanto, alguns destes padrões podem ser implementados através da metodologia GR. A Fig. 5.3 mostra alguns exemplos de padrões de *workflow* usando grafo de relações.

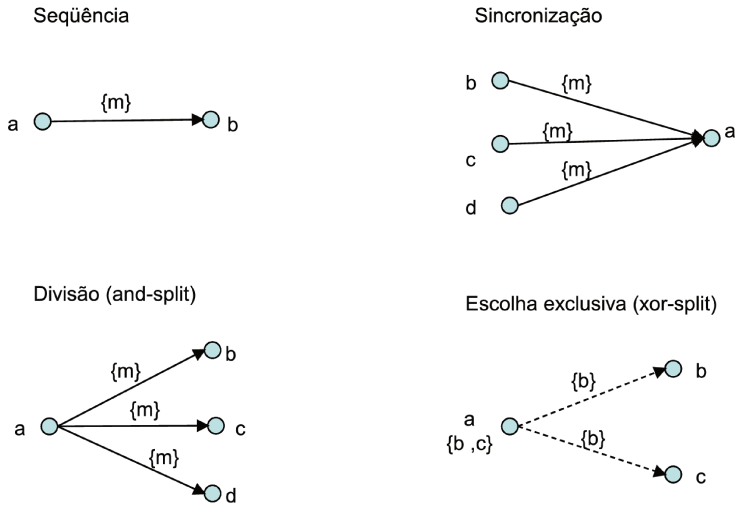


Fig. 5.3. Exemplos de padrões de workflow.

Considerando o Modelo de Referência da Workflow Management Coalition [WfMC, 95], o uso da metodologia GR e da ferramenta MAMC corresponde ao processo de definição. Isto é, são

ferramentas usadas para descrever a lógica do processo em uma notação de alto nível de abstração. O coordenador desenvolvido corresponde ao serviço de execução do *workflow*, responsável pela administração do processo, distribuição e invocação das atividades.

Com base nestes conceitos, podemos utilizar a ferramenta MAMC e o coordenador para criar um sistema de *workflow*. Para ilustrar essa idéia, consideremos um sistema de loja virtual (*e-commerce*). As atividades que compõem o sistema estão descritas na Tab. 5.2. Este sistema é inter-organizacional pois envolve diferentes organizações como a loja, a operadora de cartão de crédito, o fornecedor do produto e o agente responsável pela entrega.

Tab. 5.2: Descrição das atividades do sistema de *e-commerce*.

Atividade	**Descrição**
a_0	Fazer pedido
a_1	Fazer pagamento
a_2	Verificar crédito
a_3	Cancelar compra
a_4	Confirmar compra
a_5	Informar ao cliente o cancelamento do pedido
a_6	Agendar entrega
a_7	Informar confirmação do pagamento
a_8	Encaminhar produto

O sistema inicia com a solicitação de compra (a_0) que é seguida do pagamento do produto (a_1). Esta última atividade inicia a atividade de verificação do crédito (a_2). Se o crédito for aprovado, então é feita a confirmação do pedido (a_4). Caso contrário, a compra é cancelada (a_3) e uma comunicação é enviada ao cliente (a_5). A atividade a_6 (agendar entrega) só poderá ser executada se houver recursos disponíveis (s_0). Neste caso, s_0 seria o produto em estoque.

O peso do arco (s_0, a_6) indica a quantidade do produto solicitado. Se não houver quantidade suficiente de recursos (s_0), então a atividade a_6 não deverá ser executada. A expressão E(A, R, F, G, S, P, w, t), que define o sistema, é dada abaixo e o grafo de relações é ilustrado na Fig. 5.4.

$A = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$;

$R = \{(a_0, a_1), (a_1, a_2), (a_2, a_3), (a_3, a_5), (a_2, a_4), (a_4, a_6), (a_4, a_7), (a_6, a_8)\}$;

$F(a_0, a_1) = \{m\}$, $F(a_1, a_2) = \{s\}$, $F(a_2, a_3) = \{b\}$, $F(a_3, a_5) = \{m\}$, $F(a_2, a_4) = \{b\}$, $F(a_4, a_6) = \{m\}$, $F(a_4, a_7) = \{m\}$, $F(a_6, a_8) = \{b\}$;

$G(a2) = \{a3, a4\}$, $G(a_1) = G(a_3) = G(a_4) = G(a_5) = G(a_6) = G(a_7) = G(a_8) = \varnothing$;

$S = \{s_0\}$, $E = \{(s_0, a_6)\}$, $w(s_0, a_6) = 2$, $t(s_0) = (v, 10)$.

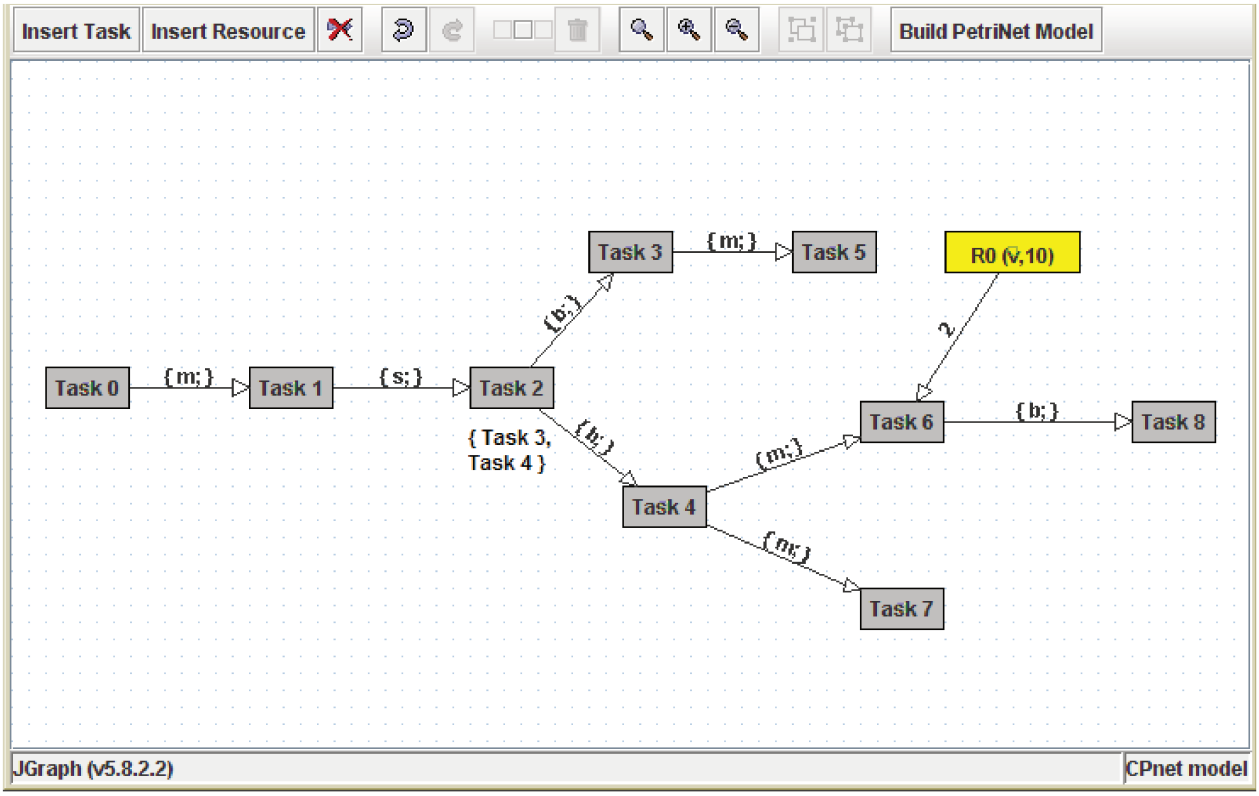


Fig. 5.4: Grafo de relações do sistema de workflow

A Fig. 5.5 mostra o modelo do sistema gerado pela ferramenta MAMC e simulado no programa CPN Tools.

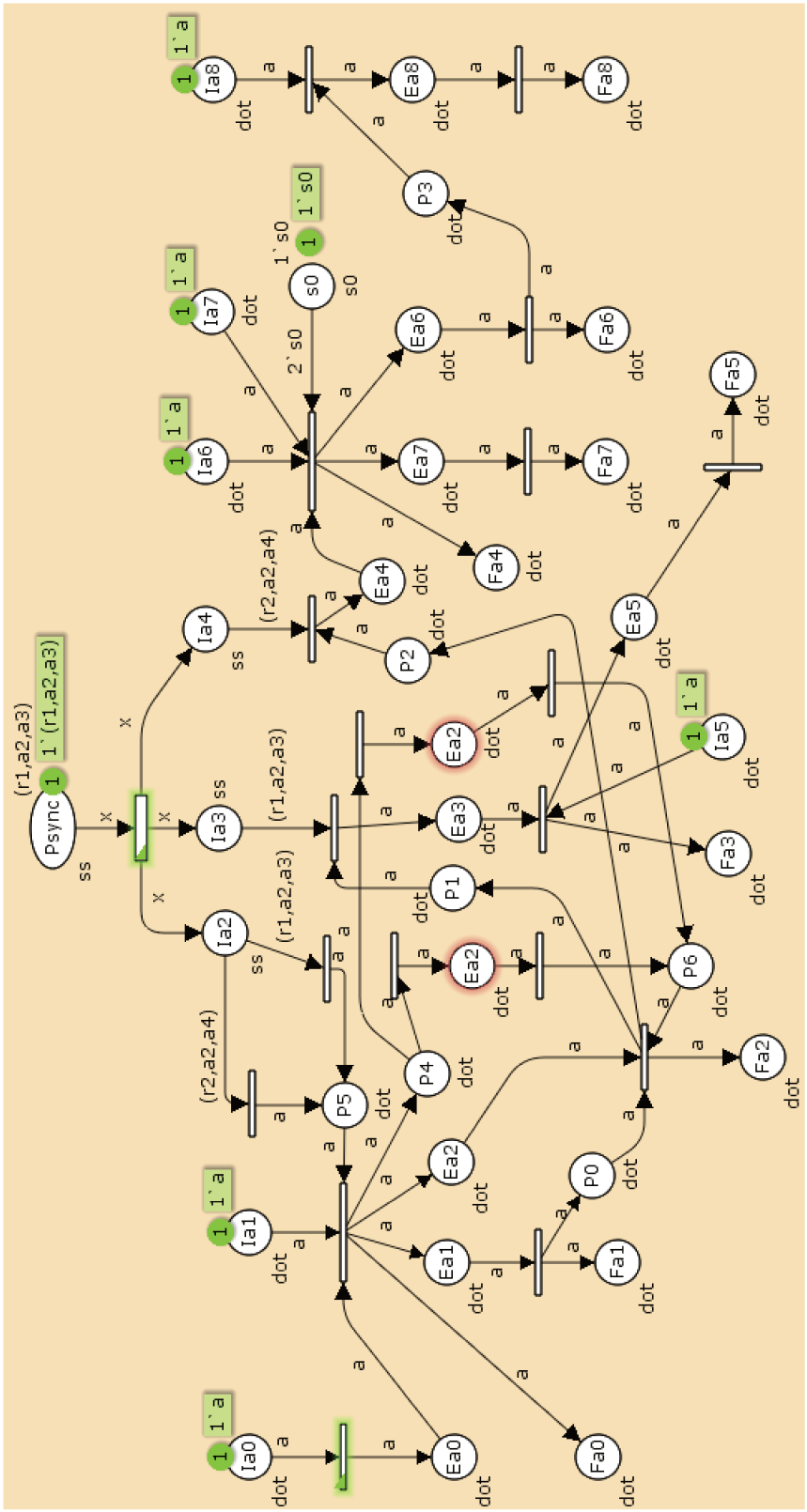


Fig. 5.5: PN do sistema de *e-commerce* com a marcação inicial.

A simulação da rede foi feita em duas situações. Na primeira, escolhemos a relação a_2 **before** a_3 (caso o crédito não tenha sido aprovado), enquanto na segunda situação, escolhemos a relação a_2 **before** a_4. A escolha da relação é feita colocando uma ficha de cor apropriada no lugar P_{sync}. Para a primeira opção, colocamos a ficha (r_1, a_2, a_3) e para a segunda, (r_2, a_2, a_4).

No caso da atividade a_2 relacionar-se com a atividade a_3, as atividades a_4, a_6, a_7 e a_8 não serão executadas. Isto garante que a entrega do produto não seja realizada caso o crédito não seja aprovado. Por outro lado, a seleção da relação a_2 **before** a_4 impede que as atividades a_3 e a_5 sejam executadas. Isto impede que o pedido seja cancelado caso o crédito seja aprovado.

A rede no estado inicial mostra que apenas a atividade a_0 está habilitada. Todas as demais atividades estão bloqueadas. Isto indica que o processo só começa com a solicitação do pedido de compra.

O processo de workflow pára em duas situações: quando as atividades terminam sua execução corretamente ou quando quantidade de recursos s_0 disponíveis não é suficiente para a execução da atividade a_6. Neste último caso, a_6 fica impedido de iniciar e, conseqüentemente, as atividades a_7 e a_8 não são executadas. Isto é, não haveria o agendamento da entrega (a_6) e conseqüentemente também não ocorreria o encaminhamento do produto (a_8), o que era esperado, pois não há o produto disponível. Entretanto, haveria a necessidade de reiniciar o processo, o que poderia ser feito usando um *timeout*. O mesmo recurso de *timeout* pode ser utilizado para reiniciar o sistema quando as atividades terminam corretamente. O tratamento de exceções no processo de *workflow*, como a falta de recursos, está fora do escopo deste trabalho, mas podemos citar o trabalho de Kumar e Wainer [Kumar, 04] que trata deste assunto.

5.3 Conclusão

Este capítulo demonstrou como a ferramenta MAMC facilita a construção do mecanismo de coordenação. Não é necessário para o projetista conhecer em detalhes a metodologia GR. A partir da especificação do relacionamento das atividades usando uma interface gráfica, o MAMC livra o projetista de construir manualmente o modelo de coordenação.

O uso do simulador de redes de Petri permitiu analisar o comportamento das atividades. Vimos nos dois exemplos apresentados que a execução das atividades atende aos requisitos impostos. Isto significa que as atividades são executadas segundo os critérios estabelecidos pelo

projetista.

Os dois exemplos analisados mostram que a ferramenta apresentada neste trabalho é aplicável a diferentes processos computacionais. A ferramenta de modelagem pode ser utilizada não apenas para verificar inconsistências em quaisquer processos que possuam atividades interdependentes mas também para auxiliar na construção do coordenador.

Notemos que os exemplos citados são meramente ilustrativos e didáticos, isto é, visam facilitar o entendimento da ferramenta.

Capítulo 6

Conclusão

Este trabalho apresentou uma ferramenta de suporte à modelagem de mecanismos de coordenação (MC), bem como uma extensão para a metodologia Grafo de Relações e uma proposta inicial para a construção do coordenador. A metodologia GR lida com a coordenação de dependências temporais entre atividades interdependentes de processos computacionais. Para isto, a metodologia GR trabalha em três níveis de abstração: especificação (N1), coordenação (N2) e execução (N3), conforme ilustra a Fig. 3.1. No nível N1, especificamos através de um grafo as atividades e as relações temporais (baseadas nas relações de Allen [Allen, 83]). No nível N2, um algoritmo de complexidade linear modela as relações temporais usando redes de Petri Coloridas. O nível N3 implementa o mecanismo de coordenação gerado no nível N2 através de um coordenador (componente de software).

Em muitos processos computacionais, temos atividades que competem por recursos ou que produzem recursos que serão utilizadas por outras atividades. Nestas situações, também é necessário ter um mecanismo de coordenação que gerencie o uso de recursos. Para isto, apresentamos uma extensão para a metodologia GR. Esta extensão, adicionada tanto no nível de especificação quanto no nível de coordenação (modelagem), permite gerenciar dependências de recursos. Isto permite à metodologia GR abranger uma quantidade maior de processos. Os recursos adicionados podem ou não ser produzidos pelas atividades e são classificados em voláteis ou não-voláteis. Um recurso é volátil se não estiver mais disponível após seu uso, caso contrário ele é não-volátil.

A especificação das relações temporais entre as atividades e dos recursos é feita através de um único grafo. No final, temos um único modelo de coordenação que gerencia as dependências

temporais e de recursos. Uma observação sobre o modelo de coordenação com uso de recursos é que atividades relacionadas com uma atividade que necessita de recursos podem não ser executadas se não houver recursos disponíveis. Isto obriga o projetista a alocar previamente os recursos necessários para a execução de uma atividade.

Usando a metodologia GR estendida, apresentamos um protótipo de ferramenta de suporte à modelagem de mecanismos de coordenação. Esta ferramenta, denominada MAMC (Modelador Automático de Mecanismos de Coordenação) implementa o algoritmo da metodologia GR para gerar o modelo de MC em redes de Petri (algoritmo 2). O modelo gerado é descrito em um arquivo no formato PNML. Este arquivo pode ser utilizado em um simulador de redes de Petri com intuito de verificarmos e validarmos o modelo de coordenação.

Uma das justificativas em utilizar redes de Petri é justamente a possibilidade de realizar simulações e análises. Com os resultados destas, o projetista do sistema pode avaliar se as atividades serão executadas segundo as restrições impostas.

Apresentamos também uma proposta de implementação do coordenador. Este componente interage com as atividades através de uma interface. Esta interface possibilita abstrair do coordenador detalhes de execução das atividades. Todas as atividades necessitam de autorização do coordenador para iniciar sua execução. O coordenador baseia-se no modelo em rede de Petri para decidir se uma atividade pode ou não ser executada. A comunicação entre o coordenador e as atividades é feita através de sinais que são enviados entre eles seguindo um protocolo. A seleção de comportamentos alternativos é feita por algum agente externo ao processo enviando ao coordenador um sinal que indica quais atividades serão escolhidas e o tipo de relacionamento temporal. O coordenador representa o nível N3 da metodologia GR.

Desta forma, alcançamos os objetivos deste trabalho os quais eram estender a metodologia GR original, implementá-la através de uma ferramenta computacional e propor um modelo inicial para a implementação do coordenador.

6.1 Sugestões e trabalhos futuros

O MAMC é capaz de verificar inconsistências na especificação das relações temporais. Entretanto, o uso de recursos pode conter restrições à execução das atividades. Por exemplo, consideremos três atividades: a_1, a_2 e a_3 e as relações temporais a_1 **before** a_2 e a_2 **before** a_3. Nesta

situação as atividades serão executadas na seguinte ordem: a_1, a_2, a_3 (Fig. 6.1) Se especificarmos que a atividade a_1 necessita de um determinado recurso produzido apenas pela atividade a_3, teremos um impasse (*deadlock*) e nenhuma atividade será executada. O MAMC não detecta tal situação. Entretanto, é possível fazer essa análise explorando a técnica de Zaidi [Zaidi, 99], descrita na seção 3.2, incluindo a produção e o uso de recursos.

Assim uma possível sugestão de trabalho futuro seria estudar e desenvolver uma solução para a verificação de inconsistência no uso de recursos considerando as restrições temporais no nível de especificação.

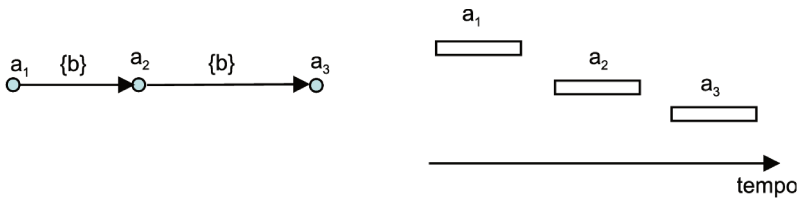


Fig. 6.1: Atividades com interdependência temporal **before**

A especificação do coordenador apresentada no capítulo 4 não possui uma implementação sendo um modelo inicial. Falta definir mais detalhes da especificação criando uma arquitetura que dê suporte ao coordenador, às atividades e aos recursos gerenciados. Adicionalmente, outro trabalho futuro seria fazer a especificação completa do coordenador e testá-lo em uma aplicação real.

O mapeamento entre o modelo em rede de Petri e o grafo de relações é feita pelo projetista. Ele deve analisar os resultados da simulação da rede e verificar se está consistente com a especificação em grafo. Isso obriga o projetista conhecer o domínio das redes de Petri e do grafo de relações. Uma possível melhora seria fazer o mapeamento da rede de Petri para o grafo de relação automaticamente através da ferramenta MAMC.

Referências Bibliográficas

[van der Aalst *et al*, 94] 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. of the 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, p. 31 - 50. 1994.

[van der Aalst *et al*, 02] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B and. Barros,. A.P. Workflow Patterns. *QUT Technical report, FIT-TR-2002-02*, Queensland University of Technology, Brisbane, 2002.

[van der Aalst *et al*, 03] van der Aalst, W.M.P., Dumas, M. & ter Hofstede, A.H.M. Web Service Composition Languages: Old Wine in New Bottles? *Proc. of the 29th EUROMICRO Conf. on New Waves in System Architecture*, Los Alamitos, CA, 2003.

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

[Arbab, 93] Arbab, F., Herman, T. & Spilling, P. An overview of Manifold and its implementation. *Concurrency: Practice and Experience*. Vol. 5 (1), fevereiro, 1993.

[Bicho, 01] Bicho, A. M. Controle de Animações por Computador utilizando redes de Petri. Dissertação de mestrado, Unicamp, setembro 2001.

[Billington *et al*, 03] Billington, J. *et al*. The Petri Net Markup Language: Concepts, Technology, and Tools. *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003. Lecture Notes in Computer Science*. Vol. 2679, p. 483 – 505, 2003

[Camargo, 95] Camargo, J. T. F. Animação modelada por computador - técnicas de controle de movimento em animação. Tese de Doutorado, Unicamp, 1995.

[Carriero, 92] Carriero, N. & Gelernter, D. Coordination languages and their significance. *Communications of the ACM*. Vol. 35 (2), fevereiro, 1992.

[Cassandras, 93] Cassandras, C. G. *Discrete Event Systems: Modeling and Performance Analysis*. Aksen Associates. 1993.

[Ciobanu, 05] Ciobanu, G. & Lucanu, D. A Specification Language for Coordinated Objects. International Conference on Software Engineering. *Proc. of the 2005 conference on Specification and verification of component-based systems*, ACM Press, 2005.

[Clark, 01] Clark, J. & Murata, M. (eds.). RELAX NG specification. http://www.oasis-open.org/committees/relax-ng/spec-20011203.html. Dezembro 2001.

[Cortes, 00] Cortes, M. A Coordination Language For Building Collaborative Applications. *Computer Supported Cooperative Work*, Kluwer Academic Publishers, 2000.

[Courtiat, 96] Courtiat, J. P., Diaz, M., Oliveira, R. C. & Senac, P. Formal Models for the Description of Timed Behaviors of Multimedia and Hypermedia Distributed Systems. *Computer Communications* , Vol.19, p. 1134 – 1150, 1996.

[CPN, 07] CPN Tools. Computer Tool for Colored Petri Nets. Disponível em http://wiki.daimi.au.dk/cpntools/cpntools.wiki. Junho, 2007.

[Crowston, 94] Crowston, K.. A Taxonomy Of Organizational Dependencies and Coordination Mechanisms. *Center for Coordination Science at the Massachusetts Institute of Technology*. http://ccs.mit.edu/papers/CCSWP174.html. Agosto, 1994.

[Cruz, 04] Cruz, A. J. Grafo de Relações: Uma Metodologia Para Coordenar Dependências Entre Atividades Em Ambientes Computacionais. Tese de doutorado, Unicamp, outubro 2004.

[Cruz *et al*, 07] Cruz, A. J., Magalhães, L. P., Raposo, A. B., Mendes, R. S. & Pelluzi, D. G. Coordinating Multi-task Environments through the Methodology of Relations Graph. *International Workshop on Groupware, 13th* (CRIWG 2007). Bariloche, Argentina, setembro de 2007.

[Diestel, 05] Diestel, R. *Graph Teory*. 3 ed. Springer-Verlag, Nova Iorque, 2005.

[Ferraiolo, 03] Ferraiolo, J., Jun, F., & Jackson, D. (eds.). Scalable Vector Graphics (SVG) 1.1 Specification. http://www.w3.org/TR/SVG/, 2003.

[Gamma *et al*, 95] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing, 1995.

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

[Gonsalves *et al*, 04] Gonsalves, T., Itoh & K. Kawabata, R. Use of Petri Nets in the Performance Design and Improvement of Collaborative Engineering Systems. *Integrated Design and Process Technology, IDPT*, julho 2004

[Haas, 02] Haas, P. J. *Stochastic Petri Nets: Modeling, Stability, Simulation*.Springer Series in Operations Research. Springer, 2002.

[Hsu *et al*., 03] Hsu, P., Chang, Y & Cheg, Y. STRPN: A Petri-Net Approach for Modeling Spatial-Temporal Relations between Moving Multimedia Objects. *IEEE Transactions on Software Engineering*, Vol 29, No. 1, January 2003.

[ISO, 95] ISO/IEC/JTC1/SC7 N1441. Subdivision of project 7.19 for a Petri net standard, Outubro 1995.

[ISO, 05] ISO/IEC JTC1/SC7 N3298. Software and Systems Engineering, High-level Petri Nets - Part 2: Transfer Format. Julho 2005.

[Jensen, 97] Jensen, K. A Brief Introduction to Coloured Petri Nets. In: E. Brinksma (ed.): Tools and Algorithms for the Construction and Analysis of Systems. *Proc. of the TACAS'97 Workshop*, Enschede, The Netherlands 1997. Lecture Notes in Computer Science Vol. 1217, Springer-Verlag 1997, 203 – 208.

[Jgraph, 06] Java Graph Visualization and Layout. http://www.jgraph.com/jgraph.html. Julho 2006.

[Jüngel *et al*, 00] Jüngel, M., Kindler, E. & Weber, M. The Petri Net Markup Language. *Petri Net Newsletter.*Vol. 59: p. 24 – 29, 2000.

[Kumar, 04] Kumar, A. & Wainer, J. Meta Workflows as a Control and Coordination Mechanism for Exception Handling in Workflow Systems. *Decision Support Systems*, Vol. 40 p. 89 – 105, 2004.

[Limniotes *et al*, 02] Limniotes, T., Mourlas, C. & Papadopoulos, G. A. Event-Driven Coordination of Real-Time Components. *Proc. of the 22nd International Conference on Distributed Computing Systems Workshops* (ICDCSW'02). IEEE Computer Society, 2002.

[Loy *et al*, 02] Loy, M., Eckstein, R., Wood, D., Elliott, J. & Cole, B. *Java Swing*. 2 ed. O`Relly, novembro 2002.

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

[Malone, 94] Malone, T. W. & Crowston, K. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*,Vol. 26(1), p. 87 – 119, 1994.

[Murata, 89] Murata, T. Petri Nets: properties, analysis and applications, *Proceedings. of the IEEE*, Vol. 77(4) p. 542 – 580 1989.

[Olguín *et al*, 02] Olguín, C. J. M., Raposo, A. B. & Ricarte, I. L. M. Coordenação de Atividades

em Ambientes de Aprendizagem Colaborativos. *XIII Simpósio Brasileiro de Informática na Educação – SBIE 2002,* p. 410 - 419. São Leopoldo, Brasil, 2002

[Park, 04] Park, H. An Intelligent Collaborative Engineering System over the Internet. *Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on.* Vol. 2, p. 473 – 476, maio 2004.

[Pedrycz, 94] Pedrycz, W. and Gomide, F. A Generalized Fuzzy Petri Net Model. *IEEE Transactions On Fuzzy Systems*, Vol 2(4), 1994

[Prasad, 05] Prasad, S. K. & Balasooriya, J. Fundamental Capabilities of Web Coordination Bonds: Modeling Petri Nets and Expressing Workflow and Communication Patterns over Web Services. *Proc. of the 38th Hawaii International Conference on System Sciences,* 2005

[Raposo, 00] Raposo, A. B. Coordenação em Ambientes Colaborativos Usando redes de Petri. Tese de doutorado, Unicamp, outubro 2000.

[Raposo, 01] Raposo, A.B., da Cruz, A. J. A., Adriano, C. M. & Magalhães, L. P. Coordination Components for Collaborative Virtual Environments. *Computers & Graphics*, Vol. 25(6), p. 1025 - 1039. Dezembro 2001. Special Issue on. Artificial Life: Towards A New Generation of Computer Animation. Pergamon Press, Holland. 2001.

[Raposo, 02] Raposo, A. B. & Fuks, H. Defining Task Interdependencies and Coordination Mechanisms for Collaborative Systems. In: Blay-Fornarino, M., Pinna-Dery, A. M., Schmidt, K. & Zaraté, P., *Cooperative Systems Design* (vol. 74 of Frontiers in Artificial Intelligence and Applications), pp. 88-103, IOS Press, Amsterdam, 2002.

[Rumbaugh, 99] Rumbaugh, J., Jacobson, I. & Booch, G. *The unified modeling language user guide*. Addison Wesley Publication, 1999.

[Schimidt, 96] Schimidt, K. & Simone, C. Coordination Mechanisms: Towards a Conceptual Foundation of CSCW Systems Design. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, Vol. 5(2-3), p. 155 – 200, 1996.

[Thalmann, 91] Thalmann, N. M. & Thalmann, D. Complex Models for Animating Synthetic Actors. *IEEE Computer Graphics & Applications*, p. 32 - 44. September, 1991.

[WfMC, 95] Workflow Management Coalition (WfMC) *The Workflow Reference Model – TC00 – 1003 v1.1.* http://www.wfmc.org/standards/referencemodel.htm. Janeiro 1995.

[XML, 06] W3C Recommendation, *Extensible Markup Language (XML) 1.0*, fourth edition edition, http://www.w3.org/TR/REC-xml. Agosto 2006

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

Anexo A

CPN Tools

CPN Tools é uma ferramenta de edição, simulação e análise de redes de Petri coloridas. É desenvolvida e mantida pelo grupo CPN da Universidade de Aarhus, Dinamarca [CPN, 07].

Este anexo descreve resumidamente a linguagem de descrição usada pelo programa CPN Tools bem como a representação dos objetos da rede de Petri.

A-1 Linguagem CPN ML

Para declarar as cores das fichas e fazer inscrições na rede de Petri, CPN Tools utiliza a linguagem CPN ML.

A-1.1 Declaração de conjunto de cores.

Um conjunto de cores é formado por uma ou mais cores que são representadas por símbolos ou valores. Uma ficha pode assumir apenas uma das cores de um conjunto de cores. Os conjuntos de cores podem ser simples ou compostos. Os conjuntos simples podem ser do tipo *unit*, *boolean*, *integer*, *string*, *enumerated* ou *index*.

O conjunto do tipo *unit* é formado por um único elemento.

Sintaxe:

colset nome = **unit** [**with** elemento];

Se omitido o parâmetro **with** elemento, então a cor desse conjunto unitário é representada por ( ). Caso contrário, a cor é representada pelo identificador “elemento”.

Um identificador é uma seqüência alfanumérica, incluindo apóstrofes e sublinhas, começando obrigatoriamente com uma letra.

Exemplos:

- **colset** U = **unit**;
- **colset** E = **unit with** e;

O conjunto do tipo *boolean* possui apenas dois elementos, *true* (verdadeiro) e *false* (falso).

Sintaxe:

colset nome = **bool** [**with** (new_false, new_true)];

A opção **with** permite renomear os elementos do conjunto que representam *false* e *true*.

Exemplos

- **colset** B = **bool**;
- **colset** Resposta = **bool with** (nao, sim);

O conjunto do tipo *integer* contém números inteiros. A princípio, esse conjunto é infinito. Entretanto, podemos restringir o tamanho do conjunto com a opção **with**.

Sintaxe:

colset nome = **int** [**with** int1 .. int2];

A restrição do tamanho do conjunto é feita pelo intervalo [int1, int2] (int1 ≤ int2).

Exemplos

- **colset** INT = **int**;
- **colset** N = **int with** 1 .. 10;

Os elementos do conjunto tipo *string* são seqüências de caracteres ASCII entre aspas duplas.

Sintaxe:

colset nome = **string** [**with** str1 .. str2 [**and** int1 .. int2]];

A opção **with** restringe o uso de caracteres enquanto que a opção **and** restringe o tamanho da seqüência.

Exemplos

- **colset** S = **string**;
- **colset** SmallString = **string with** “a” .. “d” **and** 3 .. 5;

O conjunto do tipo *enumerated* é formado por um ou mais elementos cujos identificadores são livres.

Sintaxe:

colset nome = **with** id0 | id1 | ... | idn;

Exemplo:

- **colset** Dia = **with** seg | ter | qua | qui | sex | sab | dom;

Conjuntos de cores compostos são formados por dois ou mais conjuntos simples citados anteriormente. Os tipos de conjunto composto são **product**, **record**, **list**, **union**, **subset** e **alias**.

Um conjunto do tipo **product** é formado pelo produto cartesiano entre dois ou mais conjuntos simples. Seus elementos são pares ordenados.

Sintaxe:

colset nome = **product** conj1 * conj2 * ... * conjn;

Exemplo:

- **colset** P = **product** U * I;

Para trabalharmos com redes de Petri temporizadas, deve haver pelo menos um conjunto de cores temporizado. Um conjunto de cores é temporizado se adicionarmos a palavra **timed** ao final da declaração.

Sintaxe:

colset nome = ... **timed**;

A-1.2 Declaração de variáveis

Variáveis são identificadores que podem mudar de valor durante a execução de uma rede.

Sintaxe:

var id1, id2, ... , idn : nome_conjunto;

Exemplos:

- **var** i, j, k : INT;
- **var** intList : IntList;

A-1.3 Constantes

Para declararmos uma constante, usamos a seguinte sintaxe:

Sintaxe:

val nome = valor;

Exemplos:

- **val** buffer_size = 20;
- **val** palavra = “Uma string constante”;
- **val** tupla = (1, true, “abc”);

Além de cores e variáveis, a linguagem CPN ML também permite declarar funções.

A-2 Representação de fichas

Fichas são representadas por um número seguido do símbolo ` (acento grave) e da sua cor. Por exemplo:

- 1`e
- 3`true
- 1`(a, b, 2)
- 2`10
- 3` “abcxyz”

O número à frente do símbolo ` indica o número de fichas de uma determinada cor. Para representarmos fichas de cores diferentes mas de um mesmo conjunto de cores usamos o sinal + +. Por exemplo:

- 3`true ++ 2`false

A-3. Inscrições

Inscrições são informações colocadas junto a arcos, lugares e transições.

A-3.1 Inscrições em lugares.

Há três tipos de inscrições que podem estar associados a um lugar.

1. Tipo de conjunto de cores;

2. Marcação inicial;

3. Nome do lugar.

O primeiro é obrigatório e os outros dois são opcionais. A inscrição do tipo de conjunto de cores define quais cores que as fichas presentes nesse lugar podem ser. A Fig. A.1 ilustra alguns exemplos.

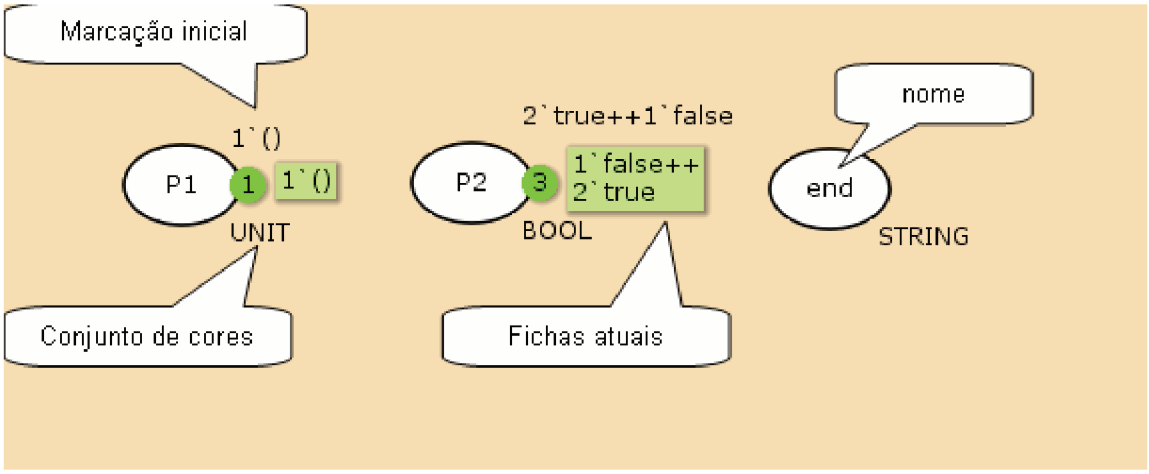


Fig. A.1: Inscrições em lugares.

A-3.2. Inscrições em transições

Podemos associar a uma transição quatro inscrições, todas opcionais.

1. Nome da transição;

2. Guarda;

3. Tempo;

4. Segmento de código.

A inscrição de guarda define uma expressão booleana que permite testar as variáveis dos arcos de entrada usando operadores relacionais (tipicamente <, >, =, <>).

A inscrição de tempo permite definir um atraso. Sua sintaxe é @+ tempo_atraso.

A inscrição de segmento de código define um código que será executado quando ocorrer o disparo da transição. A Fig. A.2 ilustra um exemplo de inscrições em uma transição.

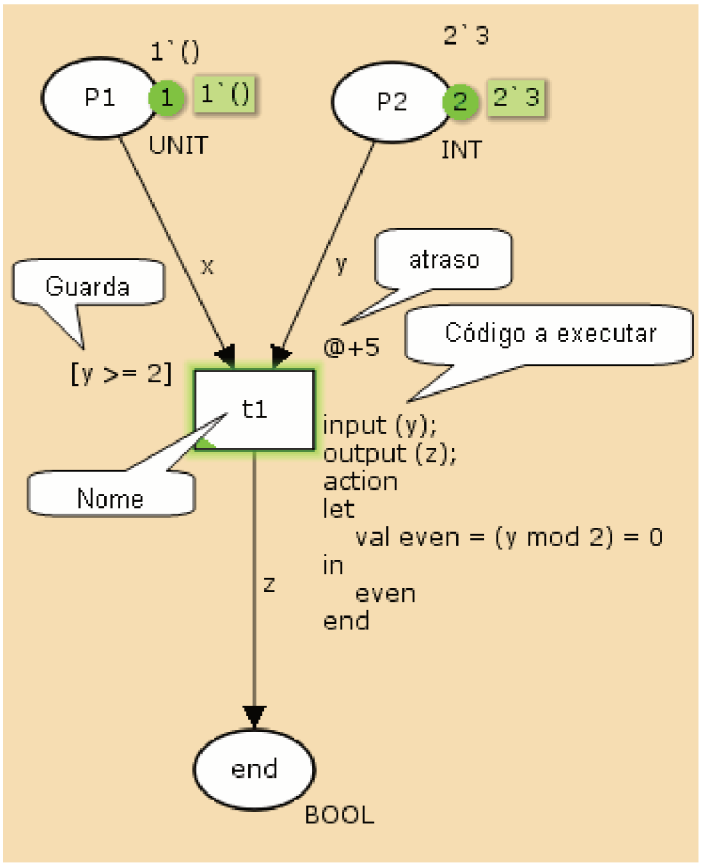


Fig. A.2: Inscrições em uma transição.

A-3.3. Inscrições em arcos

Arcos possuem apenas uma inscrição, tipicamente uma variável ou valor constante. Esta variável deve ser do mesmo conjunto de cores do lugar que precede o arco.

A-3.4. Comentários

Comentário é todo texto que começa com (* e termina com *) sendo ignorado pelo programa.