PADRÕES DE FLUXOS DE PROCESSOS EM BANCO DE DADOS RELACIONAIS

Kelly Rosa Braghetto

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO DE MESTRE
EM
CIÊNCIA DA COMPUTAÇÃO

ÁREA DE CONCENTRAÇÃO: CIÊNCIA DA COMPUTAÇÃO

ORIENTADOR: Prof. Dr. João Eduardo Ferreira

Durante a elaboração deste trabalho a autora recebeu auxílio financeiro do CNPq

São Paulo, Julho de 2006

PADRÕES DE FLUXOS DE PROCESSOS EM BANCO DE DADOS RELACIONAIS

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Kelly Rosa Braghetto e aprovada pela Comissão Julgadora.

São Paulo, 17 de Julho de 2006.

Banca Examinadora:

Prof. Dr. João Eduardo Ferreira (orientador) – IME/USP

Prof. Dr. Marcelo Finger – IME/USP

Prof^ª. Dr^ª. Claudia Maria Bauzer Medeiros – IC/UNICAMP

Aos meus pais, Aparecida e Laerte.

Agradecimentos

Ao meu orientador, Prof. Dr. João Eduardo Ferreira, por toda a atenção, paciência e entusiasmo dispensados a este trabalho e pela generosidade com que sempre compartilhou seus inestimáveis conhecimentos. Mais do que um orientador, para mim ele é um exemplo a ser seguido.

Aos professores Marcelo Finger, Roberto Marcondes Cesar Junior e Claudia Maria Bauzer Medeiros, pelos comentários feitos no meu exame de qualificação e na defesa da dissertação, os quais foram de grande valia para a finalização deste trabalho.

Aos meus pais, Aparecida e Laerte, e meu irmão Anderson, pelo apoio incondicional que sempre dedicaram a mim.

Aos companheiros do Laboratório de Banco de Dados do IME - USP, por toda a receptividade com a qual me acolheram e pela troca de experiências sobre pesquisa científica. Aos membros do grupo de Análise e Modelagem de Dados, pelos estudos em conjunto que conduziram ao tema deste mestrado. Especialmente aos amigos Daniel de Angelis Cordeiro, David da Silva Pires, Giuliano Mega, Márcio Katsumi Oikawa, Marcos Eduardo Bolelli Broinizi e Pedro Losco Takecian, pelas várias contribuições diretas e indiretas a este trabalho.

Às minhas amigas (e irmãs de coração) Silvia Cavaliunas Ferreira e Eliane Matsuda e aos amigos Felipe Godói Rosário, Gustavo Tadeu Halasi, Jonatas de Moraes Júnior e Wlisses Alves C. Oliveira, por estarem sempre ao meu lado, me incentivando em todas as circunstâncias, e pela paciência com que muito me ouviram falar sobre este mestrado.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio financeiro a esta pesquisa.

Resumo

A representação e execução de processos de negócio têm gerado importantes desafios na área de Ciência da Computação. Um desses desafios é a escolha do melhor arcabouço formal para a especificação dos controles de fluxo padrões. Algumas linguagens defendem o uso de redes de Petri ou álgebras de processos como base formal. O uso de redes de Petri para especificar *workflows* clássicos é uma abordagem bastante conhecida. Entretanto, pesquisas recentes vêm difundindo o uso de novas extensões da álgebra de processos como uma alternativa para a especificação formal de *workflows*.

A principal contribuição deste trabalho é a definição da *Navigation Plan Definition Language* (NPDL). A NPDL foi implementada como uma extensão da linguagem SQL. Ela é uma alternativa para a representação de *workflows* que utiliza a álgebra de processos como arcabouço formal. A NPDL promove uma separação explícita entre o ambiente de especificação e o ambiente de execução de um *workflow*. Esta separação propicia o reaproveitamento de passos de negócio e o uso das propriedades da álgebra de processos não só na modelagem, mas também no controle da execução dos processos. Após a especificação de um *workflow* por meio da NPDL, a execução dos passos que o definem é controlada pela ferramenta *NavigationPlanTool*. Essa ferramenta é a segunda contribuição deste trabalho de pesquisa.

Abstract

The representation and execution of business processes have generated some important challenges in Computer Science. An important related concern is the choosing of the best formal foundation to represent control-flow patterns. Some of the workflow languages advocate the Petri nets or process algebra as formal foundation. The use of Petri nets is a famous approach to support classic workflows. On the other hand some researches are introducing modern process algebra extensions as an alternative formal foundation for representing workflows.

The first contribution of this research is the definition of the *Navigation Plan Definition Language* (NPDL). NPDL was implemented as an extension of SQL language. It is an alternative to represent business processes using process algebra as formal foundation. NPDL provides the explicit separation between specification and execution workflow environment. This separation allows reusing of business steps and usage of process algebra properties in the process modeling and execution controlling tasks. After the definition of a workflow using NPDL, the business steps execution is carried out and controlled by a tool called *NavigationPlanTool*. This tool is the second contribution of this research.

Índice

1 Introdução	1
1.1 Contextualização do Trabalho.....	2
1.2 Objetivos.....	4
1.3 Contribuições	4
1.4 Organização do texto.....	5
2 Fundamentos	6
2.1 Teoria de Processos	8
2.2 Redes de Petri.....	10
2.2.1 Árvore de Cobertura	14
2.2.2 Equações Matriciais	15
2.3 Álgebra de Processos.....	18
2.4 Álgebra de Processos $\times$ Redes de Petri	26
2.5 Plano de Navegação	29
2.5.1 A Arquitetura RiverFish.....	30
2.5.2 Definição Formal de Plano de Navegação	32
2.6 Padrões de Controle de Fluxo	32
2.7 Conclusão.....	33
3 Navigation Plan Definition Language (NPDL).....	35
3.1 Definição da NPDL	35
3.2 Especificação dos Padrões de Controle de Fluxo em NPDL.....	39
3.2.1 Padrões Básicos de Controle de Fluxo	39
3.2.2 Padrões Avançados de Ramificação e Sincronização	44
3.2.3 Padrões Estruturais	50
3.2.4 Padrões de Múltiplas Instâncias	52
3.2.5 Padrões Baseados em Estados.....	55
3.2.6 Padrões de Cancelamento	59
3.3 Conclusão: Usando NPDL para especificar um exemplo de <i>workflow</i>	60
4 Implementação da NPDL.....	63

4.1 A <i>NavigationPlanTool</i>	64
4.2 O Interpretador da Linguagem NPDL	65
4.2.1 Estruturas relacionais de dados criadas pelo Interpretador NPDL.....	66
4.3 O Serviço de Instanciação de Processos.....	68
4.4 O Serviço Monitor de Execução de Instâncias de Processos.....	68
4.4.1 A navegação pelo plano	69
4.5 Conclusão: Exemplo de execução de plano de navegação	80
5 Conclusão	88
5.1 Resumo	88
5.2 Contribuições	89
5.3 Limitações.....	91
5.4 Trabalhos Futuros.....	91
5.4.1 Representação em NPDL dos dados envolvidos nos processos.....	91
5.4.2 Associação da NPDL a ferramentas gráficas de modelagem.....	92
5.4.3 Desenvolvimento de mecanismos de análise de processos para as expressões em NPDL	93
5.4.4 “Reconfiguração” de expressões de processos.....	93
Referências Bibliográficas	94
Apêndice A - Sintaxe da <i>Navigation Plan Definition Language</i>	100
Apêndice B - A Estrutura Relacional de Dados da <i>NavigationPlanTool</i>.....	104
Apêndice C - Síntese dos Padrões de Controle de Fluxo em NPDL	112

Índice de Figuras

Figura 2.1: Exemplos de grafos de processos	9
Figura 2.2: Rede de Petri para o processamento de reclamações (Fonte: [6])	12
Figura 2.3: Rede de Petri para um sistema Produtor/Consumidor (Fonte: [32])	16
Figura 2.4: Árvore de cobertura da rdP do sistema Produtor/Consumidor.....	16
Figura 2.5: Grafo de processo simplificado da Aquisição de Item de Acervo.....	25
Figura 2.6: Exemplo de modelagem de sincronização usando redes de Petri	28
Figura 2.7: Modelo da arquitetura <i>RiverFish</i> (Fonte: [21])	31
Figura 3.1: Seqüência (Fonte: [42]).....	39
Figura 3.2: Divisão Paralela (Fonte: [42])	40
Figura 3.3: Sincronização (Fonte: [42]).....	41
Figura 3.4: Escolha Exclusiva (Fonte: [42])	41
Figura 3.5: Junção Simples (Fonte: [42]).....	43
Figura 3.6: Escolha Múltipla (Fonte: [42])	44
Figura 3.7: Junção Sincronizada (Fonte: [42]).....	45
Figura 3.8: Junção Múltipla (Fonte: [42]).....	47
Figura 3.9: Discriminador (Fonte: [42])	49
Figura 3.10: Ciclo Arbitrário (Fonte: [6]).....	51
Figura 3.11: Escolha Postergada (Fonte: [42]).....	55
Figura 3.12: Roteamento Paralelo Entrelaçado (Fonte: [42])	56
Figura 3.13: Marco	57
Figura 3.14: Marco envolvendo duas linhas paralelas (Baseado em [42])	58
Figura 3.15: Sistema de processamento de reclamações (Fonte: [6])	60
Figura 4.1: Características de um sistema de <i>workflow</i> (Fonte: [28])	64
Figura 4.2: Diagrama Entidade-Relacionamento	66
Figura 4.3: Árvore de navegação do processo $P = ((a+b) . c) \mid \mid ((a+c) . d)$	70
Figura 4.4: Árvore de navegação do processo $P = (a . b) + (a . c)$	78
Figura 4.5: Estados dos nós na árvore de navegação do processo	
$P = ((a+b) . c) \mid \mid ((a+c) . d)$	79

Figura 4.6: Exemplo de replicação de ramo em uma árvore de navegação.....	79
Figura 4.7: Estado inicial da árvore de navegação para o processo Aquisição.....	82
Figura 4.8: Árvore de navegação após a substituição de P_1	82
Figura 4.9: Árvore de navegação após o mapeamento dos operadores “ *”	83
Figura 4.10: Árvore de navegação após a execução da ação a_2	83
Figura 4.11: Árvore de navegação após a execução da ação a_1	84
Figura 4.12: Árvore de navegação após a substituição de P_1	84
Figura 4.13: Árvore de navegação após a substituição dos operadores “ *”	85
Figura 4.14: Árvore de navegação após a execução da ação a_3	85
Figura 4.15: Árvore de navegação após o mapeamento do operador “?*”	86
Figura 4.16: Árvore de navegação final do processo de Aquisição	87
Figura C.1: Padrões de controle de fluxo em NPDL (Parte 1).....	112
Figura C.2: Padrões de controle de fluxo em NPDL (Parte 2).....	114

Índice de Tabelas

Tabela 2.1: Algoritmo para a construção da árvore de cobertura.....	15
Tabela 2.2: Regras de transição para a ABP	20
Tabela 2.3: Axiomas para a ABP	20
Tabela 2.4: Regras de transição para o entrelaçamento.....	21
Tabela 2.5: Regras de transição para o entrelaçamento envolvendo comunicação.....	22
Tabela 2.6: Regras de transição para o entrelaçamento à esquerda e para o entrelaçamento com comunicação	23
Tabela 2.7: Axiomas para o entrelaçamento, o entrelaçamento à esquerda e o entrelaçamento com comunicação	23
Tabela 2.8: Axiomas para o <i>deadlock</i>	24
Tabela 3.1: Precedências dos operadores da NPDL	38
Tabela 4.1: Possíveis estados para passos.....	67
Tabela 4.2: Procedimento obtenhaPróximosPassos(nóAtual, passosVálidos).....	72
Tabela 4.3: Procedimento atualizaEstadoÁrvore(nó).....	76
Tabela A.1: Regras de produção da NPDL.....	101
Tabela B.1: Comandos SQL para a criação das relações.....	104
Tabela B.2: Relação np_process.....	107
Tabela B.3: Relação np_step	107
Tabela B.4: Tipos possíveis de passos.....	107
Tabela B.5: Relação np_navigation_plan	109
Tabela B.6: Tipos possíveis de componentes.....	109
Tabela B.7: Relação np_required_process	110
Tabela B.8: Relação np_instance_log.....	110
Tabela B.9: Possíveis estados para passos	111

Siglas

ACP	Algebra of Communicating Processes
APB	Álgebra de Processos Básica
API	Application Programming Interface
BPEL4WS	Business Process Execution Language for Web Services
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
CCS	Calculus of Communication Systems
CSP	Communicating Sequential Processes
GPN	Gerenciamento de Processos de Negócio
JDBC	Java DataBase Connectivity
NPD	Navigation Plan Definition Language
PN	Plano de Navegação
rdP	Rede de Petri
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SOC	Service Oriented Computing
SQL	Structured Query Language
STR	Sistemas de Transições Rotuladas
WfMC	Workflow Management Coalition
WFMS	Workflow Management System
WSFL	Web Services Flow Language
XLANG	Web Services for Business Process Design
XML	eXtensible Markup Language
YAWL	Yet Another Workflow Language

Capítulo 1

Introdução

Um **processo de negócio** é descrito por um ou mais procedimentos que, em conjunto, realizam um objetivo de negócio. A execução de um processo de negócio possui condições muito bem definidas de início e término, e pode combinar procedimentos automáticos e manuais [41].

Um conceito bastante difundido antes mesmo do surgimento dos sistemas gerenciadores de processos de negócio é o de *workflow*¹. Um *workflow* segundo a definição da WfMC (*Workflow Management Coalition*) [28] é “*a automação de um processo de negócio, em todo ou parte, por meio da qual documentos, informações ou tarefas são passadas de um participante ao outro por ações, de acordo com um conjunto de regras procedurais*”. Também segundo a WfMC, um **sistema de gerenciamento de workflow** (WFMS, de *Workflow Management System*) é um “*sistema que define, cria e gerencia a execução de workflows por meio do uso de software*”.

O **Gerenciamento de Processos de Negócio** (GPN²) envolve métodos, técnicas e ferramentas para apoiar o projeto, a execução, o gerenciamento e a análise operacional dos processos de negócio [30]. Enquanto as definições tradicionais de *workflow* priorizam a execução dos processos operacionais, o GPN também apóia a fase de diagnóstico, permitindo que os processos sejam analisados, objetivando a detecção de falhas e possíveis melhorias no projeto.

¹ Neste texto, o termo inglês *workflow* não é substituído pela sua tradução “fluxo de trabalho” por ser amplamente utilizado e reconhecido na comunidade científica.

² GPN é a tradução de *Business Process Management* (BPM).

Conforme Aalst, Hofstede e Weske apontam em [5], atualmente existem três tendências de desenvolvimento de software que justificam a atenção que vem sendo dada ao GPN:

- 1) A “montagem” de componentes de software;
- 2) O enfoque nos processos e não só nos dados;
- 3) A valorização do re-projeto e crescimento orgânico em detrimento de projetos estáticos.

Essas tendências foram confirmadas com o paradigma da **computação orientada a serviços**, o SOC (*Service-Oriented Computing*) [24], apresentado em 2003.

A escolha de arcabouços formais³ para a representação de processos de negócio sempre gerou discussões na área acadêmica. Os formalismos mais utilizados para a modelagem de processos de negócio são as **lógicas temporais**, as **redes de Petri** e as **álgebras de processos**, sendo que esses dois últimos serão apresentados nas seções 2.2 e 2.3 deste texto.

Na área de composição de serviços *web*, existem muitas iniciativas associadas à definição de processos de negócio. O problema da composição de serviços *web* é abordado por vários trabalhos recentes, como as linguagens WSFL (*Web Services Flow Language*) [31], a XLANG (*XML Language*) [40] e a BPEL4WS (*Business Process Execution Language for Web Services*) [8]. A maior parte desses trabalhos se concentra nos mecanismos para a composição em camadas abstratas, dependentes de outros padrões como o XML⁴ [19].

1.1 Contextualização do Trabalho

A WSFL (*Web Services Flow Language*) é uma linguagem XML, proposta pela IBM, para a descrição da composição de serviços *web*. A WSFL trata de dois tipos de composições: o primeiro especifica o padrão de uso apropriado de uma coleção de serviços

³ Neste texto, os termos “formal” e “formalismo” referem-se a “método matemático rigoroso”.

⁴ A XML (*eXtensible Markup Language*) é uma linguagem de marcação definida pela W3C [19]. Ela é um subtipo da linguagem padrão de marcação generalizada, a SGML (*Standard Generalized Markup Language*), capaz de descrever diversos tipos de dados. O objetivo principal da XML é facilitar o compartilhamento de informações na Internet.

web, de modo que sua composição resultante seja uma descrição de como alcançar um determinado objetivo de negócio; o segundo especifica o padrão de interação de uma coleção de serviços *web*.

A XLANG (*XML Language*), criada pela Microsoft, pode ser considerada uma extensão da WSFL. Por possuírem as mesmas características e objetivos, a XLANG e a WSFL foram combinadas e geraram a BPEL4WS. A BPEL4WS (*Business Process Execution Language for Web Services*) é uma linguagem baseada em XML, utilizada para descrever processos de negócio. Amparada por empresas como a Microsoft, IBM, Siebel Systems, BEA e SAP, a BPEL4WS vem sendo aclamada como linguagem padrão para a definição de processos de negócio baseados em serviços *web*.

A BPMN (*Business Process Modeling Notation*) [17], definida pela BPMI (*Business Process Management Initiative*) [16], é um conjunto de convenções gráficas para descrever processos de negócio. A notação da BPMN foi especificamente projetada para coordenar seqüências de processos e a troca de mensagens existente entre processos participantes de um conjunto de atividades relacionadas. A intenção de um modelo de processo em BPMN é capturar detalhes suficientes para permitir que ele seja utilizado como fonte de uma descrição executável de um processo. Por esse motivo, é possível traduzir especificações em BPMN para BPEL4WS.

A BPMI, em 2005, anunciou sua integração ao grupo OMG (*Object Management Group*) e tem em sua ampla lista de membros empresas influentes como a IBM, a Oracle, a SAP e a Adobe Systems.

As especificações de processos de negócio desenvolvidas pela BPMI e a linguagem XLANG (precursora da BPEL4WS) declaram ter π -Calculus - uma extensão da álgebra de processos - como arcabouço formal, mas nenhuma delas descreve em sua especificação como esse formalismo é utilizado. Uma das poucas linguagens que define de forma clara sua relação com um formalismo para modelagem de *workflows* é a YAWL (*Yet Another Workflow Language*) [2].

A YAWL é uma linguagem gráfica que utiliza redes de Petri como arcabouço formal. Ela aumenta o seu poder de expressividade na modelagem de processos de negócio por meio de mecanismos que apóiam, de forma mais direta e intuitiva, a representação de padrões de controle de fluxo, superando limitações das redes de Petri. Além disso, a

YAWL não se restringe aos serviços *web*, como ocorre com as linguagens apresentadas anteriormente.

Puhlmann e Weske discutem em [37] a aplicação de π -Calculus para descrever aspectos comportamentais de *workflows*. A principal contribuição desse artigo é a utilização da semântica de execução e dos conceitos de comunicação e troca de π -Calculus para definir precisamente o comportamento de cada um dos padrões de controle de fluxo definidos em [4].

1.2 Objetivos

Os trabalhos para representar *workflows* são tradicionalmente baseados em redes de Petri. Mais recentemente, algumas iniciativas usam extensões da álgebra de processos. Uma importante iniciativa [37] ilustra a deficiência associada à maioria das extensões de álgebra de processos: *a dificuldade de expressar aspectos definidos em tempo de execução de processos*.

A abordagem proposta pelo presente trabalho se identifica com a do artigo [37], por se basear em uma extensão da álgebra de processos para especificar padrões de controle de fluxo. Entretanto, este trabalho adiciona mecanismos à álgebra de processos que remediam a deficiência descrita no parágrafo anterior.

O objetivo principal deste trabalho é criar uma linguagem para facilitar a representação das principais estruturas de controle de fluxo utilizadas na definição de processos de negócio.

1.3 Contribuições

A contribuição mais expressiva deste trabalho é a linguagem de definição de plano de navegação – a NPDL (*Navigation Plan Definition Language*). A NPDL é uma linguagem de definição de processos de negócio baseada em álgebra de processos, desenvolvida como uma extensão da linguagem SQL.

Uma segunda contribuição deste trabalho é o desenvolvimento da ferramenta *NavigationPlanTool*, fundamentada pela NPD. Essa ferramenta cria e manipula as estruturas de dados que representam processos em um modelo de dados relacional. Ela apóia as fases de projeto, instanciação e execução de processos de negócio, podendo ser aplicada na composição de serviços básicos, como os especificados pelo paradigma SOC.

1.4 Organização do texto

No Capítulo 2 deste texto são apresentados os fundamentos empregados ao longo deste trabalho, tais como: os conceitos envolvidos na **Teoria de Processos**, a comparação entre os formalismos **redes de Petri** e **álgebra de processos**, a definição de **plano de navegação** e a descrição dos **padrões de controle de fluxo**. A linguagem de definição de processos NPD, desenvolvida neste trabalho, é descrita no Capítulo 3. Ainda no Capítulo 3, é mostrado como a álgebra de processos e a NPD são utilizadas na especificação dos padrões de controle de fluxo.

O Capítulo 4 detalha a implementação da ferramenta *NavigationPlanTool*, que além de ter um interpretador para a NPD, oferece serviços de apoio à instanciação e ao controle de execução de processos. No Capítulo 4, também são descritas as estruturas de dados em gerenciadores de dados relacionais e os algoritmos em memória utilizados pela *NavigationPlanTool*. Por fim, o Capítulo 5 destaca os resultados alcançados, apontando propostas para trabalhos futuros.

Capítulo 2

Fundamentos

O comportamento de sistemas paralelos e distribuídos, os chamados **sistemas concorrentes**, é um tópico bastante abordado pela literatura da área da computação. Uma prova disto é o crescente número de **linguagens** e **semânticas formais** para a descrição e análise de comportamento de sistemas concorrentes [9].

Uma **linguagem formal** é um conjunto de cadeias de tamanho finito, formadas a partir de um alfabeto. Uma linguagem geralmente é gerada a partir de uma gramática, que é um conjunto de regras que determinam quais sentenças pertencem à linguagem. **Semânticas Formais** é a área de Teoria da Computação que trata do estudo matemático do significado das linguagens de programação e modelos computacionais. A semântica formal de uma linguagem é expressa por um modelo matemático que representa as possíveis operações descritas pela linguagem. O objetivo de uma semântica formal é criar um arcabouço preciso e não-ambíguo para o raciocínio sobre sistemas.

O comportamento de sistemas concorrentes é frequentemente representado por meio das *ações* que o sistema pode executar e a *ordenação* destas ações. Essa visão abstrata sobre o comportamento de sistemas concorrentes é denominada **processo** [9]. Sistemas geralmente são caracterizados por processos e dados. Processos são os mecanismos de controle para a manipulação dos dados. Enquanto os processos são dinâmicos e ativos, os dados são estáticos e passivos. O comportamento dos sistemas tende a ser composto por vários processos que são executados concorrentemente. Processos trocam dados entre si e desta forma influenciam o comportamento de outros processos e têm o seu próprio comportamento influenciado por outros processos [23].

O comportamento de sistemas concorrentes geralmente pode ser definido por meio de expressões em alguma linguagem formal. Duas expressões descrevem o mesmo sistema se, e somente se, elas correspondem a processos equivalentes na mesma semântica, sendo que a equivalência de processos é chamada **equivalência semântica**.

A classificação de equivalências semânticas é alvo de várias discussões nesta área de pesquisa. Segundo Baeten e Basten, em [9], duas discussões importantes são as comparações entre semânticas de **tempo linear** e semânticas de **tempo ramificado**, e entre semânticas de **ordem total** e semânticas de **ordem parcial**.

Em semânticas de tempo linear, dois processos cuja ordem das ações é igual são considerados equivalentes. Entretanto, esses processos podem diferir em sua estrutura de ramificação. A estrutura de ramificação de um processo é determinada pelos momentos nos quais são feitas as escolhas entre ramos alternativos de comportamento.

As ações de um processo em uma semântica de ordem total são sempre completamente ordenadas, ao passo que, nas semânticas de ordem parcial, as ações podem ocorrer simultaneamente ou com independência causal entre si. A propriedade fundamental de semânticas de ordem total é que nelas o conceito de **concorrência** é equivalente a **não-determinismo**. Concorrência em um conjunto de ações representa a situação em que essas ações são executadas paralelamente. O não-determinismo em um conjunto de ações representa a escolha “não previsível” de uma entre todas as possibilidades de ordenação dessas ações.

Os dois formalismos mais utilizados para a especificação de sistemas concorrentes são as redes de Petri e as álgebras de processos. A semântica das redes de Petri é de ordem parcial e de tempo ramificado. A maior parte das álgebras de processos possui semântica de ordem total e de tempo ramificado [9]. Detalhes sobre redes de Petri e álgebra de processos são fornecidos nas seções 2.2, 2.3 e 2.4.

A próxima seção apresenta a **Teoria de Processos**, com o objetivo de definir alguns termos que serão usados neste texto.

2.1 Teoria de Processos

A **Teoria de Processos** é o estudo voltado principalmente a duas atividades relacionadas a processos: **modelagem** e **verificação** [25]. A modelagem consiste na representação dos processos, geralmente por meio de estruturas matemáticas ou expressões em uma linguagem descritiva de sistemas. Verificação é a atividade de provar enunciados sobre processos, baseando-se na semântica de equivalência do formalismo empregado na especificação.

Segundo [9] e [23], um modo natural e direto de representar o comportamento de um processo em um sistema concorrente é a utilização de um **grafo de processo**. Um grafo de processo é um grafo em um **Sistema de Transições Rotuladas** (STR). Um STR é um grafo direcionado, no qual as arestas, denominadas **transições**, são rotuladas por nomes de um alfabeto de ações. Os vértices do grafo são chamados de **estados**. Uma ação pode ser qualquer tipo de atividade que um sistema concorrente desempenha. O conjunto de estados em um STR é uma abstração de todos os possíveis estados de um sistema concorrente. STRs podem ser usados para definir tanto semânticas de ordem parcial quando semânticas de ordem total.

Em uma visão de ordem total, assume-se que ações são entidades atômicas sem estrutura interna. Um sistema pode executar uma única ação por vez; assim, as ações são totalmente ordenadas. Em uma visão de ordem parcial, as ações são conjuntos de ações atômicas, permitindo que o sistema execute de forma simultânea ações atômicas e casualmente independentes. Nesse caso, as ações atômicas não são mais totalmente ordenadas.

A noção básica do arcabouço dos STR usada nesta pesquisa é a chamada de **espaço de processos** [9]. Um espaço de processos é um STR estendido com um predicado de terminação nos estados. Na Teoria de Processos, um processo é um STR estendido com um predicado de terminação e um estado inicial diferenciado. O predicado de terminação define os estados nos quais o processo pode terminar com sucesso. Denomina-se **deadlock** o estado a partir do qual um processo não pode executar uma ação nem terminar com sucesso.

Um espaço de processos pode ser definido pela quádrupla $(P, A, \rightarrow, \downarrow)$, sendo P um conjunto de estados de processos, A um conjunto de ações, $\rightarrow \subseteq P \times A \times P$ uma relação de transição ternária, e $\downarrow \subseteq P$ um predicado de terminação. Se o sistema de transições contém uma relação de transição $s \xrightarrow{a} s'$ (representada por uma aresta no grafo do processo), então o processo pode evoluir do estado s para o estado s' com a execução da ação a . No grafo de processo da Figura 2.1(a), as seguintes relações de transição podem ser observadas: $s \xrightarrow{a} s'$, $s' \xrightarrow{b} s''$ e $s' \xrightarrow{c} s''$, sendo que s'' é um estado final para o processo, ou seja, $\downarrow s''$.

Grafos de processos podem ser comparados por meio de alguma equivalência comportamental [23]. Uma possível equivalência pode relacionar dois grafos de processos se, e somente se, eles possam executar as mesmas seqüências de ações – essa equivalência é denominada **equivalência por observação**. Outro tipo de equivalência, a mais refinada de todas, é a que requer que dois grafos de processos, além de poderem executar a mesma seqüência de ações, tenham a mesma estrutura de ramificação – preservando os momentos de escolha e os *deadlocks*. Esse segundo tipo de equivalência é denominada **equivalência por bissimulação**. Por exemplo, os grafos dos dois processos apresentados na Figura 2.1 são equivalentes comportamentalmente segundo a equivalência por observação, pois ambos os processos executam primeiro ação a ação a , seguida ou pela ação b ou pela ação c . Entretanto, os grafos são diferentes segundo a equivalência por bissimulação, pois enquanto o primeiro grafo indica que a escolha entre as ações b e c é feita no segundo passo da execução do processo, o segundo grafo modela um processo no qual esta escolha é feita no primeiro passo da execução.

Figura 2.1: Exemplos de grafos de processos

2.2 Redes de Petri

A teoria das redes de Petri é um dos exemplos mais conhecidos de teoria de ordem parcial para modelagem e análise de sistemas concorrentes. O conceito foi introduzido por Carl Adam Petri, em sua tese de doutorado em 1962, na Faculdade de Matemática e Física da Universidade de Darmstadt, Alemanha.

A popularidade das redes de Petri (rdPs) como ferramenta para estudo de sistemas se deve a sua representação gráfica de fácil compreensão e ao seu potencial matemático para a análise de processos. Essas análises incluem verificações de propriedades inerentes aos sistemas concorrentes, como relações de precedência entre eventos, sincronização e existência ou não de *deadlocks*.

Redes de Petri são formadas basicamente por dois tipos de componentes: a **transição**, que é um componente ativo correspondente a alguma ação ou evento realizado dentro do sistema, e o **lugar**, que é um componente passivo e relacionado a alguma variável de estado do sistema. Esses dois elementos são os vértices do grafo associado às rdPs.

Uma rdP clássica é um grafo bipartido⁵ e um estado inicial, chamado de **marcação inicial** ou M_0 . O grafo de uma rdP é orientado e seus arcos possuem pesos. A definição formal de uma rdP é dada pela quintupla $RP = (L, T, A, P, M_0)$ [34] em que:

- $L = \{l_1, l_2, \dots, l_m\}$ é um conjunto finito de lugares;
- $T = \{t_1, t_2, \dots, t_n\}$ é um conjunto finito de transições;
- $A \subseteq (L \times T) \cup (T \times L)$ é um conjunto de arcos;
- $P = A \rightarrow \{1, 2, 3, \dots\}$ é a função peso dos arcos;
- $M_0 = L \rightarrow \{0, 1, 2, 3, \dots\}$ é a marcação inicial da rede;
- $L \cap T = \emptyset$ e $L \cup T \neq \emptyset$.

Denota-se por R uma estrutura de rede de Petri $R = (L, T, A, P)$ sem especificação da marcação inicial.

⁵ Um **grafo bipartido** é aquele em que o conjunto de vértices V pode ser dividido em dois conjuntos V_1 e V_2 disjuntos, sendo que cada aresta do grafo conecta um vértice pertencente a V_1 a um vértice pertencente a V_2 .

Os conceitos associados a cada um dos itens de uma rdP são:

- **Lugar** (representado graficamente por um círculo): modela uma condição que deve ser satisfeita para que o disparo da ação seja realizado;
- **Transição** (representada graficamente por um retângulo ou barra): pode ser compreendida como uma ação/evento, ou um processamento de um sinal, ou ainda uma tarefa;
- **Arco orientado**: liga um lugar a uma transição ou vice-versa, encadeando condições e eventos;
- **Marca** ou **ficha**: representa um recurso disponível. O posicionamento destas fichas nos lugares do grafo constitui a **marcação** da rdP. Cada lugar pode possuir 0 ou mais fichas. A evolução da marcação permite modelar o comportamento dinâmico do sistema;
- **Peso**: cada arco possui um peso associado a ele; o peso indica quantas marcas uma transição consome de um lugar de entrada ou quantas marcas uma transição acrescenta em um lugar de saída. Quando um arco não possui um peso explicitamente indicado no grafo, considera-se que o seu peso é 1.

Uma rede de Petri em que todos os arcos possuem peso 1 é classificada como **ordinária**. Para ilustrar a modelagem de processos em uma rede de Petri ordinária, será utilizado como exemplo um sistema de processamento de reclamações, apresentado em [6]. A Figura 2.2 exibe a rede de Petri do processo que representa o sistema. A primeira ação nesse sistema é o registro de uma reclamação; após o registro, duas ações são executadas paralelamente: o envio de um questionário para o reclamante e a análise da reclamação. O questionário será processado se o reclamante retorná-lo dentro de um prazo de 2 semanas; expirado o prazo, o questionário é descartado. Baseado no resultado da avaliação, a reclamação pode ou não ser processada. O processamento da reclamação só acontece depois da ocorrência do processamento do questionário ou da expiração do prazo de 2 semanas. Após a realização do processamento da reclamação, uma verificação é feita e, caso o processamento esteja correto, a reclamação é arquivada. Caso haja problemas, o processamento deve ser refeito.

Uma transição está associada a um ou mais **lugares de entrada** e a um ou mais **lugares de saída**. Por exemplo, na rede da Figura 2.2, o conjunto dos lugares de entrada da

transição “Envio de Questionário” é $\{c_1\}$ (a transição só possui um lugar de entrada); o conjunto dos lugares de saída da transição “Registro de Reclamação” é $\{c_1, c_2\}$. Um lugar de entrada de uma transição geralmente expressa uma pré-condição, um dado/sinal de entrada, um recurso de hardware/software requerido ou ainda um *buffer*. Um lugar de saída pode ser compreendido como uma pós-condição, um dado/sinal de saída, um recurso liberado, uma conclusão ou um *buffer*.

Figura 2.2: Rede de Petri para o processamento de reclamações (Fonte: [6])

O disparo das transições (execução das ações) é controlado pelo número de marcas e pela distribuição destas nos lugares do grafo que representa a rede. Uma transição está habilitada se, e somente se, *para todo lugar de entrada da transição, o lugar possui um número de marcas superior ou igual ao peso do arco que liga o lugar à transição*. Esta regra habilita o disparo de uma transição, mas não o obriga.

O disparo de uma transição gera a subtração de marcas de todos os seus lugares de entrada (de acordo com o peso dos arcos que ligam os lugares de entrada à transição) e o acréscimo de marcas em todos os seus lugares de saída (de acordo com o peso dos arcos que ligam a transição aos seus lugares de saída). A marcação de uma rede de Petri muda a cada disparo de transição, permitindo simular o comportamento dinâmico do sistema e definir o seu estado em um dado momento.

As redes de Petri foram estendidas de diversas formas e aplicadas a vários tipos de problemas. Os primeiros trabalhos sobre o uso de rdPs para a modelagem e análise de *workflows* datam da década de 70 [5]. Essas extensões são as bases formais mais utilizadas para especificação de *workflows*. Em [35], os autores atribuem a popularidade das redes de Petri não somente às características decorrentes de sua representação formal, mas também à sua simplicidade gráfica. Essa simplicidade gráfica viabiliza uma linguagem comum entre os diferentes tipos de especialistas envolvidos na especificação de um sistema.

Dois tipos de propriedades podem ser estudadas a partir de um modelo em rdP:

- **Propriedades comportamentais** - dependentes da marcação inicial da rede;
- **Propriedades estruturais** - independentes da marcação inicial.

A propriedade fundamental para o estudo das propriedades dinâmicas de qualquer sistema descrito por uma rdP é uma propriedade comportamental conhecida por **alcançabilidade**.

O disparo de uma transição habilitada muda a marcação da rede de acordo com a regra de transição descrita anteriormente. Uma sequência de disparos resulta em uma sequência de marcações. Diz-se que uma marcação M_n é **alcançável** a partir de M_0 se existe uma sequência de disparos que transformam M_0 em M_n . O conjunto de todas as marcações alcançáveis a partir de M_0 em uma rede (R, M_0) é denotado por $A(R, M_0)$.

De acordo com [34], o **problema da alcançabilidade** para rdPs é o problema de verificar se $M_n \in A(R, M_0)$ para uma dada marcação M_n em uma rede (R, M_0) . Foi provado que o problema da alcançabilidade é *decidível*, no entanto consome espaço e tempo exponencial para verificar o caso geral.

Os métodos de análise para redes de Petri podem ser divididos em três abordagens distintas: (1) baseados em árvore de cobertura; (2) baseados em equações matriciais e (3) baseados em técnicas de redução e decomposição.

Os métodos do tipo (2) e (3) não fazem restrição ao tamanho da rdP em que serão aplicados. Entretanto, em muitos casos eles só são aplicáveis a subclasses ou situações especiais de rdPs. Já as abordagens baseadas em árvore de alcançabilidade envolvem essencialmente a enumeração das marcações alcançáveis. Embora sejam aplicadas em qualquer classe de rede, elas são restritas às redes pequenas devido à explosão do número de estados.

As abordagens (1) e (2) estão descritas nas duas próximas seções deste texto. A abordagem (3) pode ser representada por diferentes técnicas que não estão detalhadas aqui, pois estão fora do escopo desta pesquisa. Detalhes da abordagem (3) podem ser encontrados em [32].

2.2.1 Árvore de Cobertura

Dada uma rdP (R, M_0) , a partir da marcação inicial M_0 , é possível obter uma nova marcação para cada transição habilitada. A partir de cada nova marcação, novas marcações podem ser alcançadas. Esse processo resulta em uma árvore de marcações, denominada **árvore de cobertura**. Os nós representam marcações geradas a partir de M_0 (a raiz) e seus sucessores, e cada arco representa o disparo de uma transição, que transforma uma marcação em outra.

Entretanto, esta árvore irá crescer infinitamente se a rede for **ilimitada**. Uma rdP é dita **limitada** se o número de fichas em cada um de seus lugares não excede a um número finito k para cada marcação alcançável a partir de M_0 .

Para uma rdP limitada, a árvore de cobertura é chamada de **árvore de alcançabilidade**, já que ela possui todas as possíveis marcações alcançáveis. Algumas propriedades comportamentais podem ser extraídas de uma árvore de cobertura, mas somente uma árvore de alcançabilidade pode fornecer **todas** as propriedades.

É possível construir uma árvore finita de cobertura, utilizando o símbolo especial ω , que pode ser compreendido como “infinito” [34]. Para qualquer inteiro n , o símbolo ω possui as seguintes propriedades: $\omega > n$, $\omega \pm n = \omega$ e $\omega \geq \omega$. Os passos para construção da árvore de cobertura são os descritos na Tabela 2.1.

Aplicando os passos descritos na Tabela 2.1 sobre a rede de Petri da Figura 2.3, que representa um sistema produtor/consumidor, obtém-se a árvore de Figura 2.4.

Tabela 2.1: Algoritmo para a construção da árvore de cobertura

Passo	Instrução
1	Rotule a marcação inicial M_0 de "nova". Ela é a raiz da árvore de cobertura
2	Para cada marcação M rotulada como "nova", faça:
2.1	Se M é idêntica a alguma outra marcação existente no caminho entre a raiz e M , então rotule M de "antiga" e volte ao passo 2
2.2	Se nenhuma transição está habilitada a partir de M , rotule M de "final" e volte ao passo 2
2.3	Para cada transição t habilitada de M faça:
2.3.1	Obtenha a marcação M' resultante do disparo de t a partir de M
2.3.2	Considere $M(p)$ como sendo o número de marcas existentes no lugar p na marcação M . No caminho entre a raiz e M , se existir uma marcação M'' tal que $M'(p) \geq M''(p)$ para todo lugar p e $M' \neq M''$, ou seja, M'' está coberta , então substitua $M'(p)$ por ω em cada p que satisfizer a condição $M'(p) > M''(p)$
2.3.4	Inclua M' como um nó na árvore, ligue-o a M por meio de um arco rotulado como t , e rotule M' como "nova"

2.2.2 Equações Matriciais

É possível também representar o comportamento dinâmico de redes de Petri por meio de equações matriciais. As equações matriciais são aplicáveis às rdPs **puras**. Uma rdP pura é uma rede que não contém um par de lugar p e transição t em que p é, ao mesmo tempo, um lugar de entrada e um lugar de saída de t .

Figura 2.3: Rede de Petri para um sistema Produtor/Consumidor (Fonte: [32])

Figura 2.4: Árvore de cobertura da rdP do sistema Produtor/Consumidor

A partir da **matriz de incidência** e das **equações de estado**, é possível determinar se uma marcação pode ser atingida a partir de M_0 .

Para uma rede de Petri com n transições e m lugares, a matriz de incidência $A=[a_{ij}]$ é uma matriz $n \times m$ de inteiros e sua entrada típica é dada por $a_{ij} = a_{ij}^+ - a_{ij}^-$, no qual $a_{ij}^+ = w(i, j)$ é o peso do arco que sai da transição i para o seu lugar de saída j ; $a_{ij}^- = w(j, i)$ é o peso do arco que liga a transição i ao seu lugar de entrada j . Desta forma, a_{ij}^- , a_{ij}^+ e a_{ij} representam, respectivamente, o número de fichas removidas, adicionadas e modificadas no lugar j quando a transição i dispara uma vez.

Ao escrever as equações matriciais, usa-se um vetor coluna $m \times 1$ para escrever uma marcação M_k . A j -ésima entrada de M_k denota o número de fichas no lugar j imediatamente após o k -ésimo disparo de uma seqüência de disparos. O k -ésimo disparo ou **vetor de controle** u_k é um vetor coluna $n \times 1$ de $n-1$ zeros e uma entrada diferente de zero, um 1 na i -ésima posição indicando que a transição i dispara no k -ésimo disparo. Como a i -ésima linha da matriz de incidência A denota a mudança de marcação como o resultado do disparo da transição i , é possível escrever a seguinte equação de estados para uma rdP:

$$M_k = M_{k-1} + A^T u_k, \quad k = 1, 2, \dots \quad (i)$$

Suponha que uma marcação de destino M_d é alcançável a partir de M_0 por meio de uma seqüência de disparo $\{u_1, u_2, \dots, u_d\}$. Escrevendo as equações de estado (i) para $i = 1, 2, \dots, d$ e somando-as, obtém-se a **equação fundamental das redes de Petri**:

$$M_d = M_0 + A^T \sum_{k=1}^d u_k$$

que pode ser reescrita como

$$A^T x = \Delta M$$

sendo $\Delta M = M_d - M_0$ e $x = \sum_{k=1}^d u_k$. Aqui, x é um vetor coluna $n \times 1$ de números

inteiros não negativos, chamado de **vetor contador de disparos**. A i -ésima entrada de x denota o número de vezes que a transição i precisa disparar para transformar M_0 em M_d . Se M_d é alcançável a partir de M_0 , o vetor contador de disparos precisa existir, ou seja, há uma solução válida para $A^T x = \Delta M$.

A partir da equação fundamental das redes de Petri é possível verificar algumas propriedades comportamentais e derivar outros métodos para a análise de propriedades estruturais da rede [34].

2.3 Álgebra de Processos

Para o propósito do raciocínio matemático, é freqüentemente conveniente representar os grafos de processos algebricamente, na forma de termos. A álgebra de processos trata da especificação e manipulação de termos de processos por meio de uma coleção de operadores, constituindo um arcabouço para o raciocínio formal sobre processos e dados, com ênfase em processos que são executados concorrentemente. Ela pode ser usada para detectar propriedades indesejáveis e formalmente derivar propriedades desejáveis de uma especificação de sistema [23].

A álgebra de processos impõe uma lógica de equivalência sobre termos de processos, de modo que dois termos de processos só podem ser igualados se os seus grafos de processos são comportamentalmente equivalentes.

Uma álgebra de processos é composta por um conjunto de símbolos de ações (ou eventos), um conjunto de operações e um conjunto de axiomas descrevendo as propriedades dos operadores. O conjunto de axiomas (ou leis equacionais) podem também especificar quando dois processos são considerados iguais.

A maioria das álgebras de processos contém operadores básicos para a construção de processos finitos (composição seqüencial e composição alternativa), operadores de comunicação para a construção de processos concorrentes (composição paralela e sincronismo) e alguma notação de recursão para capturar comportamentos potencialmente infinitos. Uma álgebra de processos pode ser estendida com novos operadores para aumentar o seu poder de expressividade ou facilitar a especificação do comportamento de sistemas [11].

As bases de álgebra de processos foram desenvolvidas, independentemente, por Milner e Hoare. Suas bases são parcialmente enraizadas nas redes de Petri, na teoria de autômatos e nas linguagens formais. Milner desenvolveu a álgebra de processos CCS

(*Calculus of Communicating Systems*) [33], enquanto Hoare definiu a CSP (*Communicating Sequential Processes*) [27]. A álgebra de processos explorada por este trabalho é a ACP (*Algebra of Communicating Processes*), apresentada por Bergstra e Klop em 1984 [11], relacionada à CCS. Todas as definições, regras de transição e axiomas expostos a partir deste parágrafo até o final desta seção foram extraídos de [23].

Uma **assinatura** Σ consiste em um conjunto finito de símbolos de funções (ou operadores) $f, g, \dots$, em que cada símbolo de função f tem uma aridade $ar(f)$ que representa seu número de argumentos.

O conjunto $T(\Sigma)$ de termos sobre uma assinatura Σ é definido como o conjunto mínimo que satisfaça as condições:

- cada variável está em $T(\Sigma)$;
- se $f \in \Sigma$ e $t_1, \dots, t_{ar(f)} \in T(\Sigma)$, então $f(t_1, \dots, t_{ar(f)}) \in T(\Sigma)$.

A assinatura de uma álgebra de processos consiste em:

- um conjunto finito e não vazio A de ações atômicas, representando comportamentos indivisíveis;
- um operador binário “+”, chamado de **composição alternativa**. Se os termos fechados⁶ t_1 e t_2 representam os processos p_1 e p_2 , respectivamente, então o termo fechado $t_1 + t_2$ representa o processo que executa p_1 ou p_2 ;
- um operador binário “•”, chamado de **composição seqüencial**. Se os termos fechados t_1 e t_2 representam os processos p_1 e p_2 , respectivamente, então o termo fechado $t_1 \cdot t_2$ representa o processo que executa primeiro p_1 e depois p_2 .

Cada processo finito pode ser representado por um termo fechado construído a partir de um conjunto de ações atômicas A , do operador “•” e do operador “+”. Esses termos são denominados **termos de processos básicos**; a coleção de todos os termos de processos básicos é chamada de **álgebra de processos básica** (APB).

As regras de transição para a APB, que constituem a sua **semântica operacional**, são as descritas na Tabela 2.2. As variáveis x , x' , y e y' nestas regras assumem valores da coleção de termos básicos de processos, enquanto v assume valores do conjunto A de

⁶ Um *termo fechado* é um termo que não contém variáveis.

ações atômicas. A semântica operacional de uma linguagem descreve como uma sentença válida da linguagem deve ser interpretada em passos sequenciais.

Tabela 2.2: Regras de transição para a ABP

$\overline{v \xrightarrow{v} \surd}$			
$\frac{x \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd}$	$\frac{x \xrightarrow{v} x'}{x + y \xrightarrow{v} x'}$	$\frac{y \xrightarrow{v} \surd}{x + y \xrightarrow{v} \surd}$	$\frac{y \xrightarrow{v} y'}{x + y \xrightarrow{v} y'}$
$\frac{x \xrightarrow{v} \surd}{x . y \xrightarrow{v} y}$		$\frac{x \xrightarrow{v} x'}{x . y \xrightarrow{v} x' . y}$	

A primeira regra de transição da Tabela 2.2 diz que cada ação atômica v pode terminar com sucesso pela execução dela própria. As quatro regras seguintes expressam que $t + t'$ executa t ou t' . As duas regras restantes expressam que $t . t'$ executa t até que este seja terminado com sucesso e, após isso, inicia a execução de t' .

A Tabela 2.3 lista os axiomas para a APB. As variáveis x , y e z nos axiomas assumem valores da coleção de termos básicos de processos.

Tabela 2.3: Axiomas para a APB

A1	$x + y = y + x$
A2	$(x + y) + z = x + (y + z)$
A3	$x + x = x$
A4	$(x + y) . z = x . z + y . z$
A5	$(x . y) . z = x . (y . z)$

Neste texto, também são utilizados dois outros conceitos que não pertencem a APB: o **operador binário** “ $\parallel$ ” chamado **entrelaçamento**, da **ACP**, e as **expressões recursivas** da **ACP with guarded recursion** [23].

O “ $\parallel$ ” foi introduzido por Milner em [33]. O termo $s \parallel t$ indica que os termos de processos s e t serão executados em paralelo, ou seja, é possível executar uma transição inicial de s (i.e., $s \xrightarrow{a} s'$ ou $s \xrightarrow{a} \sqrt{}$) ou uma transição inicial de t . A formalização desse comportamento é feita pelas regras de transição da Tabela 2.4. As variáveis x , x' , y e y' nas regras assumem valores da coleção de termos básicos de processos, enquanto v assume valores do conjunto A de ações atômicas.

Tabela 2.4: Regras de transição para o entrelaçamento

$\frac{x \xrightarrow{v} \sqrt{}}{x \parallel y \xrightarrow{v} y}$	$\frac{x \xrightarrow{v} x'}{x \parallel y \xrightarrow{v} x' \parallel y}$
$\frac{y \xrightarrow{v} \sqrt{}}{x \parallel y \xrightarrow{v} x}$	$\frac{y \xrightarrow{v} y'}{x \parallel y \xrightarrow{v} x \parallel y'}$

Além disso, $s \parallel t$ também representa a possibilidade de comunicação entre duas transições iniciais de s e t . Para isso, existe a **função de comunicação** definida como $\gamma: A \times A \rightarrow A$, que produz para todo par de ações atômicas a e b , sua comunicação $\gamma(a, b)$. A função de comunicação é comutativa e associativa, ou seja, para a, b e $c \in A$:

$$\begin{aligned}\gamma(a, b) &\equiv \gamma(b, a) \\ \gamma(\gamma(a, b), c) &\equiv \gamma(a, \gamma(b, c))\end{aligned}$$

As regras de transição para o entrelaçamento, da Tabela 2.5, expressam o comportamento do operador envolvendo também a possibilidade de comunicação entre ações. As variáveis x , x' , y e y' das regras assumem valores da coleção de termos básicos de processos, enquanto v e w assume valores do conjunto A de ações atômicas.

Para que os axiomas da APB estendida com o operador de entrelaçamento fossem válidos e completos, Bergstra e Klop em [12] provaram a necessidade de outros dois operadores na extensão: o **entrelaçamento à esquerda** e o **entrelaçamento com comunicação**. Ambos capturam parte do comportamento do entrelaçamento.

Tabela 2.5: Regras de transição para o entrelaçamento envolvendo comunicação

$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x \parallel y \xrightarrow{\gamma(v,w)} \surd}$	$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x \parallel y \xrightarrow{\gamma(v,w)} y'}$
$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x \parallel y \xrightarrow{\gamma(v,w)} x'}$	$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x \parallel y \xrightarrow{\gamma(v,w)} x' \parallel y'}$

O entrelaçamento à esquerda $s \parallel t$ realiza uma transição inicial do termo de processo s , e depois se comporta como o operador entrelaçamento “ $\parallel$ ”. Deste modo, o entrelaçamento à esquerda possui o mesmo significado que o entrelaçamento, exceto pelo fato do processo da esquerda ter que executar a primeira ação.

O entrelaçamento com comunicação $s | t$ realiza como transição inicial a comunicação entre as transições iniciais dos termos de processos s e t , e depois se comporta como o “ $\parallel$ ”. O entrelaçamento com comunicação, assim como o entrelaçamento, também denota a execução paralela de seus operandos, com a restrição de que eles precisam se sincronizar em sua primeira ação.

As regras de transição para os dois operadores adicionais estão na Tabela 2.6 e os axiomas, na Tabela 2.7. As variáveis x, x', y, y' e z nas regras e axiomas assumem valores da coleção de termos básicos de processos; as variáveis v e w assumem valores do conjunto A de ações atômicas.

A constante especial “ δ ” da ACP representa o *deadlock* e não apresenta nenhum comportamento. O modo como o *deadlock* interfere nas expressões às quais ele pertence pode ser compreendido pelos axiomas da Tabela 2.8. A variável x que aparece nos axiomas assume valores da coleção de termos de processos.

Sistemas frequentemente apresentam comportamento ilimitado, ou seja, podem conter ciclos estruturados (com um único ponto de entrada e um único ponto de saída) ou arbitrários (com mais de um ponto de entrada ou mais de um ponto de saída). As equações recursivas da ACP *with guarded recursion* são utilizadas para a representação destas iterações finitas ou infinitas.

Tabela 2.6: Regras de transição para o entrelaçamento à esquerda e para o entrelaçamento com comunicação

$\frac{x \xrightarrow{v} \surd}{x \parallel y \xrightarrow{v} y}$	$\frac{x \xrightarrow{v} x'}{x \parallel y \xrightarrow{v} x' \parallel y}$
$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x \mid y \xrightarrow{\gamma(v,w)} \surd}$	$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x \mid y \xrightarrow{\gamma(v,w)} y'}$
$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x \mid y \xrightarrow{\gamma(v,w)} x'}$	$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x \mid y \xrightarrow{\gamma(v,w)} x' \parallel y'}$

Tabela 2.7: Axiomas para o entrelaçamento, o entrelaçamento à esquerda e o entrelaçamento com comunicação

M1	$x \parallel y = (x \parallel y + y \parallel x) + x \mid y$
LM2	$v \parallel y = v.y$
LM3	$(v.x) \parallel y = v.(x \parallel y)$
LM4	$(x + y) \parallel z = x \parallel z + y \parallel z$
CM5	$v \mid w = \gamma(v, w)$
CM6	$v \mid (w.y) = \gamma(v, w).y$
CM7	$(v.x) \mid w = \gamma(v, w).x$
CM8	$(v.x) \mid (w.y) = \gamma(v, w).(x \parallel y)$
CM9	$(x + y) \mid z = x \mid z + y \mid z$
CM10	$x \mid (y + z) = x \mid y + x \mid z$

Tabela 2.8: Axiomas para o *deadlock*

A6	$x + \delta = \delta$
A7	$\delta . x = \delta$
LM11	$\delta \parallel x = \delta$
CM12	$\delta \mid x = \delta$
CM13	$x \mid \delta = \delta$

Uma **especificação recursiva** é um conjunto de equações recursivas da forma

$$\begin{aligned} X_1 &= t_1(X_1, \dots, X_n) \\ &\vdots \\ X_n &= t_n(X_1, \dots, X_n) \end{aligned}$$

em que os lados esquerdos X_i são variáveis de recursão, e os lados direitos $t_i(X_1, \dots, X_n)$ são termos de processos na ACP com possíveis ocorrências das variáveis de recursão $X_1, \dots, X_n$. Por exemplo, o processo que executa sequencialmente as ações a e b infinita vezes pode ser representado pela seguinte especificação recursiva:

$$\begin{aligned} X &= a.Y \\ Y &= b.X \end{aligned}$$

Para ilustrar como processos são modelados por uma álgebra de processos, será apresentado um exemplo de modelagem feita para o processo de *Aquisição de Item de Acervo da Biblioteca do IME-USP*.

A aquisição de item de acervo da biblioteca é realizada a partir de uma lista de itens, que representa um pedido de compra. A primeira etapa da aquisição é a verificação dos dados do pedido. Caso os dados do pedido estejam válidos, é necessário o recebimento de uma verba para compra, antes que os procedimentos de aquisição do pedido possam ser realmente iniciados. Após o recebimento da verba, é necessário efetuar uma priorização dos itens do pedido, uma vez que a verba recebida nem sempre é suficiente para a aquisição de todos os itens da lista. Depois de priorizados, os itens são cotados. As etapas de priorização e cotação podem ser realizadas um número irrestrito de vezes. Após uma cotação, o pedido

pode ser liberado para compra ou a compra pode ser cancelada. No caso de cancelamento da compra, o pedido pode ser reutilizado em uma nova tentativa de compra. No caso da liberação do pedido, a compra é então efetuada. Mas é possível que ocorra problemas com a etapa de compra (por exemplo, a indisponibilidade temporária de um item do pedido na editora); neste caso, cada problema ocorrido deve ser registrado, até que a compra seja entregue ou cancelada definitivamente. Em qualquer ponto do processo de aquisição, é possível cancelar o pedido; um pedido cancelado não pode ser reaproveitado em uma nova tentativa de compra.

O conjunto de ações possíveis nesse sistema é definido como $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$, sendo que:

- a_1 verifica os dados do pedido;
- a_2 indica o recebimento de uma verba para compra;
- a_3 efetua a priorização (ordenação) dos itens do pedido;
- a_4 efetua cotação do pedido;
- a_5 libera o pedido para compra;
- a_6 indica a compra do pedido;
- a_7 registra um problema na compra do pedido;
- a_8 cancela o pedido de compra (encerra o processo);
- a_9 cancela a compra (mas permite o uso do mesmo pedido para uma nova tentativa de compra).

O grafo de processo simplificado de aquisição é mostrado na Figura 2.5.

Figura 2.5: Grafo de processo simplificado da Aquisição de Item de Acervo

Os vértices no grafo da Figura 2.5 representam os possíveis estados do processo de Aquisição; as arestas do grafo representam as ações que o processo pode executar.

Não há atividades que possam ser executadas de forma paralela no processo de Aquisição. Portanto, os vértices no grafo da Figura 2.5 que possuem mais de uma aresta de saída representam pontos de escolha no processo: no momento da execução do processo, somente **uma** ação entre as possíveis ações (representadas pelas arestas de saída do vértice) deverá ser executada, conduzindo o processo a um novo estado. Por exemplo, após a execução da ação a_1 (verificar os dados do pedido), o processo pode executar a ação a_2 (indicar o recebimento de uma verba para compra) **ou** a ação a_8 (cancelar o pedido de compra).

Pelo grafo do processo é possível notar que existem trechos no fluxo de Aquisição que podem ser executados repetidas vezes. Para representar esse comportamento em álgebra de processos, o processo é dividido em subprocessos⁷ definidos por expressões recursivas. A expressão algébrica do processo de aquisição (PA) correspondente à Figura 2.5 fica:

$$\begin{aligned} PA &= a_8 + a_1 \cdot (a_8 + P_1 \cdot (a_8 + P_2)) \\ P_1 &= a_2 \cdot (a_8 + a_3 \cdot (a_8 + P_3 \cdot (a_8 + a_5 + a_9 \cdot (a_8 + P_1)))) \\ P_2 &= a_6 + a_7 \cdot (a_8 + P_2) \\ P_3 &= a_4 + a_4 \cdot (a_8 + a_3 \cdot (a_8 + P_3)) \end{aligned}$$

2.4 Álgebra de Processos × Redes de Petri

As redes de Petri e a álgebra de processos compartilham duas características importantes: ambas são fundamentadas em descrições matemáticas precisas e são aplicadas à modelagem de sistemas concorrentes. Entretanto, rdPs e álgebra de processos são dois formalismos completamente diferentes. A principal diferença entre rdPs e álgebras de processos é que as rdPs são baseadas em grafos bipartidos, enquanto as álgebras são baseadas em uma descrição textual. Em ambas as áreas é possível encontrar uma expressiva

⁷ Neste trabalho, foi classificado de **subprocesso** um processo utilizado dentro da definição de outro processo.

quantidade de estudos já realizados. Inclusive, muitos conceitos desenvolvidos para rdPs foram traduzidos para álgebras de processos e vice-versa.

Best, Devillers e Koutny, em [13], enumeram as vantagens das álgebras de processos e das redes de Petri. Sobre as álgebras de processos, é possível ressaltar:

- Possibilitam o estudo de conectivos diretamente associados às linguagens de programação reais;
- São composicionais por definição, ou seja, possibilitam a composição de grandes sistemas a partir de sistemas menores de forma estruturada;
- Contam com uma grande variedade de leis algébricas que podem ser utilizadas para manipular os sistemas, tanto para refiná-los quanto para provar corretude com relação a alguma especificação.

Das rdPs, destacam-se as seguintes vantagens:

- Distinguem precisamente estados de atividades, sendo estas últimas definidas como mudanças de estados;
- Mesmo sendo formais, possuem uma representação gráfica que é fácil de ser compreendida e por isso possuem grande apelo prático;
- Por serem representadas como grafos bipartidos, as redes de Petri possuem ligações com a teoria de grafos, que podem ser exploradas para a verificação de sistemas.

Basten, em sua tese de doutorado [10], afirma que as rdPs possuem uma representação gráfica intuitiva, que é de fácil compreensão para “não-especialistas”. Um modelo em rdP descreve tanto os estados quanto as ações do sistema que está sendo construído. Existem várias técnicas disponíveis para a investigação de propriedades dos estados e também do comportamento dinâmico de um modelo em rdP. Entretanto, Murata, em [34], destaca como maior fraqueza das rdPs o problema da complexidade: modelos baseados em rdP tendem a ficar muito extensos para análise, mesmo para sistemas de pequeno porte.

Basten pondera que a álgebra de processos é um formalismo puramente simbólico, particularmente apropriado para manipulação computacional. Uma descrição de sistema em álgebra de processos enfoca o comportamento dinâmico. As técnicas de prova na álgebra de

processos geralmente têm como objetivo mostrar a igualdade de descrições comportamentais, sendo útil, portanto, para comparar descrições de sistemas.

A Figura 2.6 apresenta um exemplo de processo modelado em uma rdP clássica, com oito transições representando as ações executadas pelo processo. Nesse processo, a ação *a* é executada seguida pela *b* e *c* em paralelo; *b* é seguida pela *c*; entretanto, *f* precisa esperar pela execução de *c* e *e* para só depois ser executada. Após a execução de *f*, vem a execução de *g*. E a execução de *d* segue a de *c*. Quando *d* e *g* tiverem sido executadas, finalmente *h* é disparada.

Figura 2.6: Exemplo de modelagem de sincronização usando redes de Petri

Em uma primeira tentativa de transcrição para álgebra de processos, pode-se descrever o processo da Figura 2.6 pela seguinte expressão:

$$a.((b.c) \parallel e).(d \parallel (f.g)).h$$

No entanto, analisando rigorosamente a expressão, é possível notar que ela não preserva a relação causal existente entre a ação *c* e a ação *d* representada na rdP. No grafo da Figura 2.6, fica claro que, embora a ação *f* é habilitada para execução somente após a execução das ações *c* e *e*, a ativação da ação *d* depende somente da execução de *c*. Nesta primeira expressão algébrica apresentada, a execução de *d* também ficou condicionada à execução de *e*, o que não era desejado.

Para expressar a relação causal existente entre *c* e *d* que a rdP representou facilmente, será preciso expandir o termo $(b.c) \parallel e$, que representa a primeira situação de paralelismo dentro do processo. Este termo corresponde ao trecho em destaque na rede da

Figura 2.6 (delimitado por um retângulo com bordas tracejadas). A expansão foi feita utilizando os axiomas para o operador de entrelaçamento, considerando que a comunicação entre as ações resulta em *deadlock*. O resultado obtido é a expressão abaixo:

$$(b.c) \parallel e = b.c.e + b.e.c + e.b.c$$

Sobre esta expressão resultante, introduz-se a sequência de ações $f.g$, obtendo-se:

$$(b.c.e + b.e.c + e.b.c).f.g = b.c.e.f.g + b.e.c.f.g + e.b.c.f.g$$

Finalmente, é incluída na última expressão apresentada a relação de causalidade entre c e d . A expressão resultante é:

$$\begin{aligned} & b.c.(d \parallel (e.f.g)) + b.e.c.(d \parallel (f.g)) + e.b.c.(d \parallel (f.g)) = \\ & b.c.(d \parallel (e.f.g)) + (b.e.c + e.b.c).(d \parallel (f.g)) \end{aligned}$$

Para completar a expressão, falta introduzir a ação inicial a e a final h . A expressão final é:

$$a.(b.c.(d \parallel (e.f.g)) + (b.e.c + e.b.c).(d \parallel (f.g))).h$$

O exemplo mostrado pela Figura 2.6 foi proposto por Aalst em [1] com o objetivo de ilustrar a maior limitação das álgebras de processo quando comparadas às rdPs: a dificuldade na modelagem de casos específicos de sistemas que requerem sincronismo de ações de linhas distintas de controle de fluxo. O autor, com isso, afirma que a modelagem de sistemas complexos em álgebras de processos é uma tarefa para especialistas de negócio, enquanto que a modelagem desse tipo de sistema em redes de Petri é bastante natural.

2.5 Plano de Navegação

O **Plano de Navegação** (PN) foi definido por Ferreira *et al* em [20] e estendido em [21] como um conjunto de todos os processos de negócio exigidos em uma aplicação para se atingirem os objetivos de negócio. O Plano de Navegação mapeia todas as regras de consistência em função dos dados pertencentes às requisições dos serviços eletrônicos de todos os objetivos de negócio. O PN é o principal conceito associado à arquitetura *RiverFish*.

2.5.1 A Arquitetura RiverFish

RiverFish é uma arquitetura para a representação, o controle e a execução de processos de negócio. Segundo esta arquitetura, processos podem ser divididos em duas categorias: **atômicos** ou **compostos**. Processos de negócio atômicos são descritos em termos de **passos de negócio**, sendo que passos de negócio podem ser **ações simples** ou **pontos de verificação**. Processos de negócio compostos são aqueles obtidos a partir da composição de outros processos de negócio. Ações e pontos de verificação são utilizados para a manipulação dos dados.

Passos de negócio são as condições a serem satisfeitas pelos dados até que eles sejam armazenados numa base de dados. Cada passo de negócio realizado para um determinado dado significa uma mudança de estado desse dado. Requisições são solicitações de serviços num sistema de informação.

Uma instância do PN é aqui entendida como um vínculo entre uma requisição de um determinado usuário e os passos de negócio que responderão por ela. Cada nova requisição de processo gera uma nova instância de seu plano de navegação. A arquitetura *RiverFish* propõe o uso de instâncias do plano de navegação para controle da ordem de processamento dos passos das requisições.

A arquitetura *RiverFish* é composta por três módulos: o módulo de **controle unificado**, o módulo de **execução de instâncias do plano de navegação** e o módulo de **armazenamento de dados**. A Figura 2.7 mostra resumidamente o modelo da arquitetura *RiverFish*.

O controle unificado gerencia três bancos de dados: um para armazenar todas as informações de usuários, seus perfis e permissões de acesso; outro para armazenar todos os passos pertencentes aos formulários de requisições (os chamados **planos de navegação**); e o último para o armazenamento dos dados das instâncias de planos de navegação. Neste trabalho, nosso interesse se concentra na estrutura destes dois últimos bancos de dados.

Figura 2.7: Modelo da arquitetura *RiverFish* (Fonte: [21])

Um plano de navegação conecta uma requisição aos seus respectivos passos de negócio. Uma vez definido um processo de negócio por seu plano de navegação, cada nova requisição recebida implica na criação de uma nova **instância** de seu plano de navegação. Essa operação de instanciação é feita no módulo de execução, que também é responsável por interpretar e executar os passos descritos pelo plano de navegação associado à instância. Os passos executados e os resultados alcançados são armazenados no banco de dados de instâncias.

De acordo com [20], a definição formal de plano de navegação utiliza fundamentos da álgebra de processos.

2.5.2 Definição Formal de Plano de Navegação

Definição 1: Ação Simples. Uma ação simples é um conjunto de ações atômicas compostas usando os operadores de sequência e composição alternativa.

Definição 2: Ponto de verificação. Um ponto de verificação é um conjunto de ações atômicas compostas usando regras restritivas e condicionais.

Definição 3: Passo de Negócio. Um passo de negócio ou é uma ação simples ou um ponto de verificação.

Definição 4: Processo de Negócio. Um processo de negócio é um conjunto de passos de negócio compostos usando operadores básicos da álgebra de processos e suas extensões.

Definição 5: Plano de Navegação. Um plano de navegação é um conjunto de todos os processos de negócio exigidos em uma aplicação para atingir o objetivo de negócio.

Passos de negócio podem ser especializados em ações simples e pontos de verificação. Cada passo especializado implementa suas ações atômicas internas sobre um dado específico. Quando um plano de navegação é instanciado, um vínculo é criado entre os dados e os passos de negócio.

A possibilidade de reuso de um determinado passo de negócio na definição de outros processos de negócios e a manutenção dos dados de controle de execução das instâncias de planos e navegação em banco de dados relacionais são características importantes da arquitetura *RiverFish*.

2.6 Padrões de Controle de Fluxo

Aalst, Hofstede, Kiepuszewski e Barros, em [4], identificaram e definiram 20 padrões de controle de fluxos em *workflows*. O objetivo principal dos autores ao definir esses padrões era o de estabelecer critérios para a realização de comparações entre as diversas linguagens para especificação de *workflows* disponíveis tanto no ambiente acadêmico como no tecnológico. Eles sugerem que cada linguagem seja classificada de

acordo com a capacidade de expressar, por meio de sua sintaxe, os padrões de controle de fluxos.

Os padrões de controle de fluxo estão divididos em 6 categorias:

1. **Padrões básicos de controle de fluxo** – referem-se a construções básicas, presentes na maioria das linguagens de modelagem de *workflows*;
2. **Padrões avançados de ramificação e sincronização** – representam controles de fluxo mais avançados que os cobertos pelos padrões da categoria anterior;
3. **Padrões estruturais** – em linguagens de programação é natural que uma estrutura de bloco tenha seus pontos de entrada e saída especificados de forma clara. Entretanto, em *workflows* é comum blocos possuírem paralelismos e múltiplos pontos de entrada ou saída. Os padrões estruturais representam essa flexibilidade requerida das linguagens de modelagem de *workflows*;
4. **Padrões de múltiplas instâncias** – os padrões desta categoria auxiliam na modelagem de casos específicos de instâncias de *workflows*, nos quais partes do processo precisam ser instanciadas múltiplas vezes;
5. **Padrões baseados em estados** – estendem a expressividade de linguagens de *workflow* por meio da representação de estados, pois sistemas de *workflow* típicos são completamente direcionados à modelagem das atividades e eventos;
6. **Padrões de cancelamento** – modelam o cancelamento de atividades condicionado à ocorrência de um determinado evento.

Como os padrões de controle de fluxo são referenciados e amplamente utilizados neste trabalho, a Seção 3.2 deste texto apresenta a definição de todos eles. Na Seção 3.2, os principais padrões são também acompanhados por exemplos didáticos e descrições gráficas.

2.7 Conclusão

Neste capítulo foram apresentados conceitos relacionados à teoria de processos. A descrição dos principais aspectos das redes de Petri, álgebra de processos, plano de navegação e padrões de *workflows* norteiam o restante deste trabalho. A opção pela álgebra

de processos para a criação e implementação de uma extensão da linguagem SQL (*Structured Query Language*) será detalhada nos capítulos 3 e 4.

Capítulo 3

Navigation Plan Definition Language (NPDL)

3.1 Definição da NPDL

A SQL (*Structured Query Language*) é a linguagem de consulta implementada pela maioria dos sistemas gerenciadores de banco de dados relacionais. O modelo relacional não possui estruturas apropriadas para a representação de processos, requerendo, para isso, uma estrutura de dados adicional. Assim sendo, a linguagem definição padrão da linguagem SQL não oferece comandos para a fácil manipulação desta estrutura.

Com o objetivo de viabilizar o controle de processos de negócio para um modelo relacional de dados foi definida a *Navigation Plan Definition Language* (NPDL). Essa linguagem é uma extensão da linguagem SQL e se baseia no conceito de plano de navegação da arquitetura *RiverFish* e em operadores de álgebra de processos.

Sistemas de *workflow* tradicionais são dirigidos a eventos [1]; neste caso, não é desejável que todos os estados possíveis dos sistemas sejam explicitamente modelados, uma vez que o enfoque está na ordenação das ações dentro dos processos que os caracterizam. Sob esta perspectiva, as redes de Petri não seriam o formalismo mais indicado para o tratamento do problema, o que justifica a escolha da álgebra de processos como arcabouço operacional. Além disso, conforme discutido na Seção 2.4 deste texto, a álgebra de processos é um formalismo bastante apropriado para manipulação computacional.

Assim como na álgebra de processos, processos em NPDL são definidos por expressões algébricas. Em NDPL a expressão algébrica de um processo é construída a partir do conjunto A de ações atômicas, de operadores NPDL e do conjunto P , sendo P

o conjunto de todos os processos definidos. A NPDL contém os operadores mais comuns da álgebra de processos e também define operadores adicionais que modelam comportamentos frequentes em processos de *workflow*. A necessidade desses operadores adicionais será justificada na Seção 3.2.

Uma ação atômica NPDL é equivalente a uma transação que respeita as propriedades ACID (atomicidade, consistência, independência e durabilidade) em um banco de dados: quando executada, ela é realizada na íntegra ou é totalmente desfeita em caso de problemas na execução [22]. Assim como geralmente ocorre com as transações, não há nenhuma forma de comunicação direta entre as ações executadas de forma concorrente: as ações só se comunicam por meio de compartilhamento de dados.

Como exemplo, os comandos NPDL seguintes especificam um processo chamado “ProcessoCalculo” (identificado por **P2**) que realiza a soma ou a multiplicação convencional de dois números um número indeterminado de vezes.

```
CREATE ACTION A1 'LerPrimeiroValor';
CREATE ACTION A2 'LerSegundoValor';
CREATE ACTION A3 'CalcularAdicao';
CREATE ACTION A4 'CalcularMultiplicacao';
CREATE ACTION A5 'MostrarResultado';
CREATE PROCESS P1 'AuxProcessoCalculo';
CREATE PROCESS P2 'ProcessoCalculo';
SET P1 = (A1 || A2).( A3 + A4 ).A5;
SET P2 = P1. P2 + P1;
```

A NPDL inicialmente foi criada com apenas 3 operadores de encadeamento de ações: o operador de **composição seqüencial**, o operador de **composição alternativa** e o operador de **composição paralela**. Com estes operadores, é possível atender com facilidade à categoria dos padrões básicos de controle de fluxo. Com o objetivo de atender a todas as categorias de padrões de controle de fluxo descritos em [4], a NPDL foi enriquecida com outras propriedades e operadores. A NPDL contém os seguintes operadores:

- a) **Composição Seqüencial** (operador “.”): o termo de processo $A . B$ significa que inicialmente somente o termo A está habilitado para execução. O termo B será habilitado para execução somente após o fim da execução do termo A ;

- b) **Composição Alternativa** (operador “+”): o termo de processo $A + B$ significa que inicialmente ambos termos A e B estarão habilitados para execução;
- c) **Composição Paralela** (operador “||”): o termo de processo $A // B$ significa que os termos A e B podem ser executados paralelamente, dividindo a linha de execução em duas⁸;
- d) **Composição Paralela Entrelaçada** (operador “|*”): o termo de processo $A |* B$ significa que os termos A e B podem ser executados em qualquer ordem (i.e., $A.B + B.A$), mas não paralelamente;
- e) **Composição Multi-Convergente** (operador “&”): o termo de processo $A \& B$ significa que o termo B será habilitado para execução após o término da execução de cada linha de execução do termo A . Por exemplo, o processo $P = (X // Y) \& Z$ indica que Z será habilitado duas vezes: uma após o término da execução de X e outra após o término da execução de Y , mas no processo $P = (X + Y) \& Z$, Z será habilitado apenas uma vez, após a execução da atividade escolhida na composição alternativa (X ou Y);
- f) **Composição Discriminatória** (operador “^”): o termo de processo $A ^ B$ significa que o termo B será habilitado para execução após o primeiro término de uma linha de execução do termo A . Por exemplo, o processo $P = (X // Y) ^ Z$ indica que Z será habilitado para execução somente uma vez; se a atividade Y terminar primeiro, então Z será habilitado depois da execução de Y . Neste caso, quando X terminar, Z não será habilitado novamente – esse comportamento diferencia o operador “^” do operador “&”;
- g) **Repetição Ilimitada** (operador “?*”): o termo de processo $A ?*$ significa que o termo A pode ser executado, de forma paralela, uma ou mais vezes;
- h) **Repetição Numericamente Limitada** (operador “?n”, sendo n um número inteiro positivo e não nulo): o termo de processo $A ?5$ significa que o termo A deve ser executado, de forma paralela, 5 vezes;
- i) **Repetição Limitada por Função** (operador “?f”, sendo f uma função atômica que retorna um número inteiro positivo e não nulo): o termo de

⁸ Uma **linha de execução** pode ser vista como um subprocesso. Quando um sistema é composto por mais de uma linha de execução, essas linhas podem ser executadas de forma independente, como no caso de um paralelismo.

processo $A?f_l$ significa que o termo A deve ser executado, de forma paralela, o número de vezes retornado pela função f_l em tempo de execução;

- j) **Execução Condicional** (operador “ $\%r$ ”, sendo r é uma regra, ou seja, uma função atômica que retorna um valor *booleano* – verdadeiro ou falso): o termo de processo $\%r_l A$ significa que o termo A será habilitado para execução se o valor de retorno da regra r_l (avaliada em tempo de execução) for **verdadeiro**. Sendo assim, uma regra na NPDL pode ser entendida como uma **condição** para a execução de uma atividade;
- k) **Execução Condicional Negativa** (operador “ $\%!r$ ”, sendo r é uma regra, ou seja, uma função atômica que retorna um valor *booleano* – verdadeiro ou falso): o termo de processo $\%!r_l A$ significa que o termo A será habilitado para execução se o valor de retorno da regra r_l (avaliada em tempo de execução) for **falso**.

Os comportamentos dos operadores de execução condicional (j) e (k) não são representados em álgebra de processos. Os operadores (d), (e), (f), (g), (h) e (i) também não existem na extensão de álgebra de processos ACP, entretanto, seus comportamentos podem ser representados com a combinação de outros operadores de álgebra de processos. A justificativa da criação de cada um dos novos operadores NPDL será feita na Seção 3.2. A descrição completa da sintaxe NPDL encontra-se no Apêndice A. A Tabela 3.1 mostra as precedências dos operadores da NPDL.

Tabela 3.1: Precedências dos operadores da NPDL

	Alta ← ————— Precedência —————→ Baixa						
Operador	$\%r$ $\%!r$	$?*$ $?n$ $?f$	$\cdot$	$\parallel$	$ *$	$\wedge$ $\&$	$+$

3.2 Especificação dos Padrões de Controle de Fluxo em NPD

A seguir são apresentados exemplos da especificação dos padrões de controle de fluxo usando comandos NPD. Também são mostradas as expressões correspondentes em álgebra de processos, com o objetivo de identificar as limitações da álgebra de processos na especificação desses padrões e justificar a criação dos operadores adicionais da NPD. O Apêndice C apresenta um quadro síntese dessa especificação.

3.2.1 Padrões Básicos de Controle de Fluxo

1) Seqüência

Definição: Ocorre quando uma atividade em um processo de *workflow* é habilitada após o término de uma outra atividade⁹ no mesmo processo.

Exemplo: A atividade **B** só é habilitada para execução após o término da execução de **A**.

Figura 3.1: Seqüência (Fonte: [42])

Em álgebra de processos:

$$P = A . B$$

Em NPD:

$$\text{SET } P = A . B$$

2) Divisão Paralela

Definição: É um ponto no processo de *workflow* no qual uma linha de execução única se divide em múltiplas linhas de execução que podem ser realizadas paralelamente, possibilitando que as atividades sejam executadas simultaneamente ou em qualquer ordem.

Exemplo: A atividade **A** é executada, habilitando a execução de **B** e **C** em paralelo (ou seja, em qualquer ordem ou simultaneamente).

⁹ Uma atividade pode ser compreendida como uma ação atômica ou como um subprocesso.

Figura 3.2: Divisão Paralela (Fonte: [42])

Em álgebra de processos:

$$P = A . (B \parallel C)$$

Caso a linha de execução seja dividida em mais de duas linhas, o mapeamento para álgebra de processos deve combinar mais de um operador binário “ $\parallel$ ”, como na expressão:

$$P = A . (B \parallel (C \parallel (D \parallel E))) \quad \text{ou} \quad P = A . (B \parallel C \parallel D \parallel E)$$

O uso dessa expressão é garantido pelas propriedades de comutatividade e associatividade do operador $\parallel$ [23]:

$$\begin{aligned} X \parallel Y &= Y \parallel X \\ (X \parallel Y) \parallel Z &= X \parallel (Y \parallel Z) \end{aligned}$$

Em NPD L:

$$\text{SET } P = A . (B \mid C)$$

O operador de composição paralela “ $\parallel$ ” da NPD L representa o operador de entrelaçamento “ $\parallel$ ” da álgebra de processos com a ressalva de que na NPD L não há o conceito de comunicação da ACP. A função de comunicação foi introduzida na ACP para manter válidos os axiomas da extensão após a introdução do operador de entrelaçamento. Mas o comportamento da comunicação está embutido no significado do operador de entrelaçamento. Além disso, não há nenhum padrão de controle de fluxo de *workflow* que requeira explicitamente sincronização no disparo de ações iniciais de linhas de execução paralelas. Sendo assim, eliminou-se a possibilidade de comunicação entre ações na NPD L definindo que $\gamma(a, b) = \delta$ para $\forall a, b \in A$, sendo A o conjunto das ações atômicas da NPD L.

3) Sincronização

Definição: É um ponto no processo de *workflow* no qual múltiplas atividades paralelas convergem para uma única linha de execução. Esse ponto tem como objetivo a sincronização das atividades.

Exemplo: A atividade *C* é habilitada somente depois que *A* e *B* tiverem sido executadas.

Figura 3.3: Sincronização (Fonte: [42])

Em álgebra de processos:

$$P = (A \parallel B) . C$$

Caso a sincronização envolva mais de duas linhas de execução, novamente é preciso combinar mais de um operador binário “ $\parallel$ ” para representá-la, como em:

$$P = (A \parallel (B \parallel (C \parallel D))) . E \text{ ou } P = (A \parallel B \parallel C \parallel D) . E$$

Em NPD L:

$$\text{SET } P = (A \mid B) . C$$

4) Escolha Exclusiva

Definição: É um ponto no processo de *workflow* no qual um ramo é escolhido dentre os possíveis ramos de atividades. Essa escolha baseia-se em uma decisão ou em um dado de controle do *workflow*.

Exemplo: As atividades *B* e *C* são habilitadas após a execução de *A*, mas somente **uma** delas será escolhida **implicitamente** para execução.

Figura 3.4: Escolha Exclusiva (Fonte: [42])

Em álgebra de processos:

$$P = A . (B + C)$$

Caso a escolha envolva mais de duas opções de linha de execução, o mapeamento para álgebra de processos deve combinar mais de um operador binário “+”, como na expressão:

$$P = A . (B + (C + (D + E)))$$

O uso dessa expressão é garantido pelas propriedades de comutatividade e associatividade do operador “+” [23]:

$$X + Y = Y + X$$

$$(X + Y) + Z = X + (Y + Z)$$

Em NPDL:

Como uma expressão em álgebra de processos especifica **todas as possibilidades** de execução de um processo, ela não provê mecanismos para a especificação de decisões tomadas em tempo de execução. O operador binário “+” (composição alternativa) da álgebra de processos indica que os dois termos compostos estão habilitados para a execução, entretanto somente um deles será realmente executado. Neste caso, ele pode tanto representar as características do padrão **escolha exclusiva**, quanto as características do padrão **escolha postergada**.

Para representar a **escolha exclusiva** na NPDL, utiliza-se o operador “+” e o operador adicional “%*r*”, sendo *r* uma regra. Uma regra em NPDL é uma função atômica cujo valor de retorno deve ser *booleano*, ou seja, verdadeiro ou falso.

O operador “+” da NPDL mapeia o comportamento da **escolha postergada**, enquanto que “%*r*” habilita a atividade associada a ele quando a avaliação da regra (feita em tempo de execução do processo) retorna o valor **verdadeiro**. A NPDL possui ainda o operador “%!*r*”, que apresenta o comportamento complementar ao “%*r*”, habilitando a atividade associada a ele quando a regra retorna o valor **falso**.

A especificação NPDL do exemplo dado é:

$$\text{SET } P = A . (\%r \ B + \%!r \ C)$$

sendo *r* a regra associada à **escolha exclusiva**

Essa expressão NPDL define que a habilitação das atividades *B* e *C* está condicionada à avaliação, em tempo de execução, da regra *r*. Se, após a execução de *A*,

a execução de r retornar o valor verdadeiro, somente a atividade B será habilitada para execução. Se o valor de retorno de r for falso, somente a atividade C será habilitada.

5) Junção Simples:

Definição: É um ponto no processo de *workflow* no qual dois ou mais ramos alternativos de atividades se unem sem sincronização.

Exemplo: Inicialmente, as atividades A e B estão habilitadas; após a execução de uma delas, C é habilitada.

Figura 3.5: Junção Simples (Fonte: [42])

Em álgebra de processos:

$$P = (A + B) . C$$

Conforme ilustra o exemplo a seguir, é preciso combinar mais de um operador binário “+” para representar casos de junção que envolvam mais de duas linhas de execução.

$$P = (A + (B + (C + D))) . E$$

ou

$$P = (A + B + C + D) . E$$

Em NPD L:

$$\text{SET } P = (\%r A + \%!r B) . C$$

sendo **r** a regra associada à **escolha exclusiva**

3.2.2 Padrões Avançados de Ramificação e Sincronização

6) Escolha Múltipla

Definição: É um ponto no processo de *workflow* no qual um ou mais ramos de atividades são escolhidos. A escolha baseia-se em uma decisão ou dados de controle do *workflow*.

Exemplo: Inicialmente, uma das três opções estará habilitada - a **divisão paralela de B e C** ($B \parallel C$) ou somente **B** ou somente **C**.

Figura 3.6: Escolha Múltipla (Fonte: [42])

Em álgebra de processos:

Não existe uma especificação direta do padrão **escolha múltipla** em álgebra de processos. O comportamento deste padrão pode ser representado por meio da combinação de outros padrões. Para se obter uma expressão algébrica da **escolha múltipla**, os seguintes passos devem ser seguidos:

1) Obtenha os termos algébricos correspondentes às opções da **escolha múltipla**. No caso do exemplo da Figura 3.6, tem-se uma **escolha múltipla** com duas opções de termos:

$$B \quad C$$

2) Gere todos os termos possíveis que indiquem a execução em paralelo dos termos levantados no passo 1. No caso do exemplo, os termos paralelos possíveis são:

$$B \parallel C \quad C \parallel B$$

Pela comutatividade do operador $\parallel$, tem-se que $B \parallel C = C \parallel B$, então somente um destes termos precisa ser considerado nos passos seguintes.

3) A expressão final é obtida por meio da composição alternativa dos termos resultantes dos passos 1 e 2, usando o operador “+”:

$$A.(B \parallel C + B + C)$$

Em uma **escolha múltipla** com mais de duas opções, o mapeamento segue as mesmas regras descritas acima. Exemplo de mapeamento de **escolha múltipla** com 3 opções (B , C e D):

- 1) Termos correspondentes às opções da **escolha múltipla**:

$$B \quad C \quad D$$

- 2) Termos paralelos possíveis:

$$B \parallel C \parallel D \quad B \parallel C \quad B \parallel D \quad C \parallel D$$

- 3) Expressão final:

$$A.(B \parallel C \parallel D + B \parallel C + B \parallel D + C \parallel D + B + C + D)$$

Em NPDL:

Na expressão NPDL, é preciso indicar as **escolhas exclusivas**:

$$\text{SET } P = A.(\%r_1 (B \mid C) + \%!r_1 (\%r_2 B + \%!r_2 C))$$

sendo r_1 e r_2 as regras associadas às **escolhas exclusivas**

7) Junção Sincronizada

Definição: É um ponto no processo de *workflow* no qual múltiplos caminhos convergem para uma única linha de execução. Se mais de um caminho foi escolhido, então é feita a sincronização das linhas ativas de execução. Se somente um caminho foi escolhido, então o ramo alternativo de atividades converge sem sincronismo.

Exemplo: Se na **escolha múltipla** a execução tiver sido encaminhada para a **divisão paralela** de **B** e **C** ($B \parallel C$), então a **junção sincronizada** aguarda a finalização das ações **B** e **C** para habilitar **D**. Caso a **escolha múltipla** dispare somente a execução de **B**, então **D** será habilitado após a execução de **B**. Caso a **escolha múltipla** dispare somente a execução de **C**, então **D** será habilitado após a execução de **C**.

Figura 3.7: Junção Sincronizada (Fonte: [42])

Em álgebra de processos:

$$P = A.(B \parallel C + B + C).D$$

ou

$$P' = B \parallel C + B + C$$

$$P = A.P'.D$$

em que P' corresponde à especificação da **escolha múltipla** em álgebra de processos.

Caso a **escolha múltipla** envolva mais de duas opções de linha de execução, o termo entre parênteses é obtido por meio dos passos descritos anteriormente para a especificação da **escolha múltipla**. Exemplo de **junção sincronizada** com **escolha múltipla** de 3 opções:

$$P' = B \parallel C \parallel D + B \parallel C + B \parallel D + C \parallel D + B + C + D$$

$$P = A.P'.F$$

Em NPDL:

$$\text{SET } P = A.(\%r_1 (B \mid C) + \%!r_1 (\%r_2 B + \%!r_2 C)).D$$

ou

$$\text{SET } P_1 = \%r_1 (B \mid C) + \%!r_1 (\%r_2 B + \%!r_2 C)$$

$$\text{SET } P = A . P_1 . D$$

sendo P_1 o mapeamento da **escolha múltipla** e

r_1 e r_2 as regras associadas às **escolhas exclusivas**

8) Junção Múltipla

Definição: É um ponto no processo de *workflow* para o qual dois ou mais ramos de atividades convergem sem sincronismo. Se mais de um ramo foi ativado, de forma concorrente, então a atividade posterior à junção é executada a cada vez que é encerrada uma execução de um dos ramos de entrada.

Exemplo: Se na **escolha múltipla** a execução tiver sido encaminhada para a **divisão paralela** de B e C ($B \parallel C$), então a **junção múltipla** executa D , a cada vez que uma linha de execução da **divisão paralela** é finalizada. Neste exemplo, após o término da execução de B e após o término da execução de C . Caso a **escolha múltipla** dispare somente a execução de B , então D será habilitado após a execução de B . Caso a **escolha múltipla** dispare somente a execução de C , então D será habilitada após a execução de C .

Figura 3.8: Junção Múltipla (Fonte: [42])

Em álgebra de processos:

O termo a ser executado na junção deve ser *adicionado ao final* dos termos resultantes do mapeamento da **escolha múltipla**, seguindo os passos descritos abaixo:

1) Tome os termos mapeados da **escolha múltipla**. No exemplo da Figura 3.8, a representação da **escolha múltipla** em álgebra de processos fica:

$$B \parallel C + B + C$$

A cada termo da expressão resultante composto por meio do operador “+”, daremos o nome *fator*. No exemplo, temos 3 fatores: $B \parallel C$, B e C .

2) Para cada fator da expressão resultante do passo 1 que envolva o operador de paralelismo “ $\parallel$ ” (no caso do exemplo, somente o fator $B \parallel C$), compor sequencialmente os operandos do “ $\parallel$ ” e a expressão executada na **junção múltipla**:

$$(B.D) \parallel (C.D)$$

3) Para os fatores restantes (no caso do exemplo, B e C), compor sequencialmente cada fator com a expressão executada na **junção múltipla**:

$$B.D \quad C.D$$

4) O resultado final do mapeamento é a composição alternativa dos fatores resultantes dos passos 2 e 3, usando o operador “+”:

$$A.((B.D) \parallel (C.D) + B.D + C.D)$$

No caso da **escolha múltipla** envolver mais de duas opções de linhas de execução, a especificação da **junção múltipla** em álgebra de processos deve seguir os mesmos passos descritos anteriormente. Exemplo: **Escolha múltipla** de B , C e D , convergindo para uma **junção múltipla** com a atividade E :

1) Termos mapeados da **escolha múltipla**:

$$B \parallel C \parallel D + B \parallel C + B \parallel D + C \parallel D + B + C + D$$

2) Transformação dos fatores que envolvem “ $\parallel$ ”:

$$(B.E) \parallel (C.E) \parallel (D.E) \quad (B.E) \parallel (C.E) \quad (B.E) \parallel (D.E) \quad (C.E) \parallel (D.E)$$

3) Transformação dos fatores restantes:

$$B.E \quad C.E \quad D.E$$

4) Resultado final:

$$A.((B.E) \parallel (C.E) \parallel (D.E) + (B.E) \parallel (C.E) + (B.E) \parallel (D.E) + (C.E) \parallel (D.E) + B.E + C.E + D.E)$$

Em NPD:

Para facilitar o uso do padrão **junção múltipla** na NPD, foi criado o operador “&”, que faz de modo automatizado os passos de mapeamento para álgebra de processos.

$$\text{SET } P = A.(\%r_1 (B \parallel C) + \%!r_1 (\%r_2 B + \%!r_2 C)) \& D$$

ou

$$\text{SET } P_1 = \%r_1 (B \parallel C) + \%!r_1 (\%r_2 B + \%!r_2 C)$$

$$\text{SET } P = A . P_1 \& D$$

sendo P_1 o mapeamento da **escolha múltipla** e

r_1 e r_2 as regras associadas às **escolhas exclusivas**

9) Discriminador

Definição: É um ponto no processo de *workflow* que espera pelo término da execução de um de seus ramos de entrada antes de habilitar a execução da atividade subsequente. A partir desse momento, ele espera o término da execução dos demais ramos de entrada e os “ignora”. Desta forma, diferentemente da Junção Múltipla, o número de ramos ativos de entrada do discriminador não determina o número de vezes que a atividade subsequente será executada.

Exemplo: Se na **escolha múltipla** a execução tiver sido encaminhada para a **divisão paralela** de **B** e **C** ($B \parallel C$), então o **discriminador** habilita a execução de **D** após o término da linha de execução que for encerrada primeiro. Se **B** terminar primeiro, então **D** será habilitada para execução; após **C** terminar, diferentemente do que ocorre na **junção múltipla**, **D** não será habilitada novamente. Caso a **escolha múltipla** dispare somente a execução de **B**, então **D** será habilitada após a execução de **B**. Caso a **escolha múltipla** dispare somente a execução de **C**, então **D** será habilitada após a execução de **C**.

Figura 3.9: Discriminador (Fonte: [42])

Em álgebra de processos:

A atividade a ser executada no **discriminador** deve ser *permutada* nos termos resultantes do mapeamento da **escolha múltipla**, seguindo os passos descritos na sequência:

1) Tome os termos mapeados da **escolha múltipla**. No exemplo da Figura 3.9, a representação da **escolha múltipla** em álgebra de processos fica:

$$B \parallel C + B + C$$

A cada termo da expressão resultante composto por meio do operador “+”, daremos o nome *fator*. No exemplo, temos 3 fatores: $B \parallel C$, B e C .

2) Para cada fator da expressão resultante do passo 1 que envolva o operador de paralelismo “ $\parallel$ ” (no caso do exemplo, somente o fator $B \parallel C$), gerar todos os fatores possíveis resultantes da composição sequencial de um dos operandos do “ $\parallel$ ” com a expressão executada no **discriminador**. Além disso, gerar também um fator que seja o resultado da composição sequencial do próprio fator de origem com a expressão executada no **discriminador**:

$$(B.D) \parallel C \quad B \parallel (C.D) \quad (B \parallel C).D$$

3) Para os fatores restantes (no caso do exemplo, B e C), compor sequencialmente cada fator com a expressão executada no **discriminador**:

$$B.D \quad C.D$$

4) O resultado final do mapeamento é a composição alternativa dos fatores resultantes dos passos 2 e 3, usando o operador “+”:

$$A.((B.D) \parallel C + B \parallel (C.D) + (B \parallel C).D + B.D + C.D)$$

No caso da **escolha múltipla** envolver mais de duas opções de linhas de execução, a especificação do **discriminador** para álgebra de processos deve seguir os mesmos passos descritos anteriormente. Exemplo: **escolha múltipla** de B , C e D , convergindo para um **discriminador** com a atividade E :

- 1) Termos mapeados da **escolha múltipla**:

$$B \parallel C \parallel D + B \parallel C + B \parallel D + C \parallel D + B + C + D$$

- 2) Transformação dos fatores que envolvem $\parallel$:

$$(B.E) \parallel C \parallel D \quad B \parallel (C.E) \parallel D \quad B \parallel C \parallel (D.E) \quad (B \parallel C \parallel D).E$$

$$(B.E) \parallel C \quad B \parallel (C.E) \quad (B \parallel C).E$$

$$(B.E) \parallel D \quad B \parallel (D.E) \quad (B \parallel D).E$$

$$(C.E) \parallel D \quad C \parallel (D.E) \quad (C \parallel D).E$$

- 3) Transformação dos fatores restantes:

$$B.E \quad C.E \quad D.E$$

- 4) Resultado final:

$$\begin{aligned} & A.((B.E) \parallel C \parallel D + B \parallel (C.E) \parallel D + B \parallel C \parallel (D.E) + (B \parallel C \parallel D).E + \\ & (B.E) \parallel C + B \parallel (C.E) + (B \parallel C).E + \\ & (B.E) \parallel D + B \parallel (D.E) + (B \parallel D).E + \\ & (C.E) \parallel D + C \parallel (D.E) + (C \parallel D).E + \\ & B.E + C.E + D.E) \end{aligned}$$

Em NPD

Para facilitar o uso do padrão **discriminador** na NPD, foi criado o operador “ $\wedge$ ”, que faz de modo automatizado a permutação descrita na especificação em álgebra de processos.

$$\text{SET } P = A.(\%r_1 (B \mid C) + \%!r_1 (\%r_2 B + \%!r_2 C)) \wedge D$$

ou

$$\text{SET } P_1 = \%r_1 (B \mid C) + \%!r_1 (\%r_2 B + \%!r_2 C)$$

$$\text{SET } P = A . P_1 \wedge D$$

sendo P_1 o mapeamento da **escolha múltipla** e

r_1 e r_2 as regras associadas às **escolhas exclusivas**

3.2.3 Padrões Estruturais

10) Ciclo Arbitrário

Definição: É um ponto em um processo de *workflow* no qual uma ou mais atividades podem ser feitas repetidamente.

Exemplo:

Figura 3.10: Ciclo Arbitrário (Fonte: [6])

Em álgebra de processos:

É possível representar comportamento ilimitado com o uso de equações recursivas em álgebra de processos. O ciclo arbitrário do exemplo da Figura 3.10 pode ser representado em álgebra de processos como:

$$\begin{aligned} P_1 &= A.P_2 + B.P_3 \\ P_2 &= C.P_3 \\ P_3 &= D.(E + F.(G + P_2)) \end{aligned}$$

Em NPD L:

Assim como na álgebra de processos, em NPD L, os processos também podem ser definidos em termos de outros processos:

$$\begin{aligned} \text{SET } P_3 &= D.(\%r_1 E + \%!r_1 (F.(\%r_2 G + \%!r_2 P_2))) \\ \text{SET } P_2 &= C.P_3 \\ \text{SET } P_1 &= \%r_3 (A.P_2) + \%!r_3 (B.P_3) \end{aligned}$$

sendo r_1, r_2 e r_3 as regras associadas às escolhas exclusivas

11) Terminação Implícita

Definição: Um dado subprocesso pode ser encerrado quando não há nada mais a ser feito, ou seja, quando não há mais atividades ativas no *workflow*.

Em álgebra de processos:

Na ACP, o estado final de um processo é representado implicitamente.

É importante considerar a diferença entre um processo que terminou com sucesso e um processo que ficou infinitamente bloqueado. Esse último caso é denominado *deadlock*. Embora não exista um termo específico para representar o estado

final de um processo na ACP, existe a constante especial “ δ ” que representa o *deadlock*.

Em NPDL:

A NPDL, neste caso, se comporta de forma semelhante à álgebra de processos. Não existe um termo em NPDL que indique estado final de processo. O símbolo associado ao *deadlock* em NPDL é o “#”. Um exemplo de uso do “#” é apresentado na especificação do padrão 19 - **Atividade Cancelável**.

3.2.4 Padrões de Múltiplas Instâncias

12) Múltiplas Instâncias Sem Sincronização

Definição: Este padrão representa um ponto no *workflow* no qual múltiplas instâncias de uma atividade podem ser criadas, isto é, há uma facilidade para gerar novas linhas de execução. Cada uma das linhas de execução gerada é independente das demais. Além disso, não há a necessidade de sincronizar estas linhas.

Em álgebra de processos:

Em álgebra de processos, múltiplas instâncias podem ser obtidas por meio da definição de subprocessos recursivos. Dado o processo exemplo $P = A$, que permite a execução da atividade A , para definir um processo P' no qual a atividade A possa ter múltiplas instâncias (ou seja, possa ser executada diversas vezes paralelamente), é preciso definir P' como $P' = (P' \parallel A) + A$.

Para modelar um processo mais complexo, que possa executar múltiplas instâncias de A e que execute outras atividades (sem se preocupar com a sincronização destas instâncias), basta compor o P' com outras atividades usando o operador de paralelismo $\parallel$, como em $P_1 = P' \parallel (B.C)$.

Em NPDL:

Para facilitar a representação dos padrões de múltiplas instâncias, criou-se na NPDL o operador unário “ $?*$ ”. Este operador reproduz o comportamento que define a criação de subprocessos recursivos para a representação de múltiplas instâncias.

SET P1 = A?* || B.C

13) Múltiplas Instâncias Com Conhecimento Prévio em Tempo de Projeto

Definição: Ocorre quando, em uma instância de processo, uma atividade é habilitada múltiplas vezes, sendo que o número de instâncias a serem executadas da atividade em questão é conhecido em tempo de projeto. Após o término da execução de todas as instâncias, uma outra atividade do *workflow* é habilitada para execução.

Em álgebra de processos:

Dado o processo exemplo $P = A$, que permite a execução da atividade A , para definir um processo P' no qual a atividade A possa ter um determinado número de instâncias, por exemplo, 3, poderíamos definir P' como $P' = A \parallel A \parallel A$.

Diferentemente do padrão 12, este padrão exige que todas as instâncias sejam finalizadas antes de habilitar a execução da próxima atividade do *workflow*. Neste caso, para que as atividades sejam sincronizadas, basta compor a atividade “repetitiva” com a atividade seguinte por meio do operador de composição seqüencial, como foi feito na especificação do padrão **sincronização**. Por exemplo, para modelar um processo que execute as 3 instâncias de A e, após o término da execução destas instâncias, execute outras atividades, basta compor sequencialmente P' com outras atividades usando o operador “•”, como em $P_1 = P'.B.C$.

Em NPD L:

A NPD L possui uma variação do operador “?” que permite indicar o número de vezes que o termo deve ser instanciado e executado. O número associado ao “?” deve ser um número inteiro positivo e não nulo.

Por exemplo, o comando NPD L

SET P = A ?5 . B + C

define o processo P que deve ou executar a atividade C ou executar 5 instâncias da atividade A e, após finalizada a execução das instâncias, habilitar a execução de B .

14) Múltiplas Instâncias Com Conhecimento Prévio em Tempo de Execução

Definição: Ocorre quando, em uma instância de processo, uma atividade é habilitada múltiplas vezes, sendo que o número de instâncias a serem executadas da atividade em questão é conhecido em tempo de execução. Após o término da execução de todas as instâncias, uma outra atividade do *workflow* é habilitada para execução.

Em álgebra de processos:

Como já discutido anteriormente, a álgebra de processos mapeia todas as possibilidades de execução de um processo. Por meio dela é possível modelar um processo com comportamento repetitivo ilimitado ou com um número de repetições definidos em tempo de projeto, como é o caso dos dois padrões de múltiplas instâncias anteriores. Entretanto, não conseguimos limitar com álgebra de processos o número de instâncias a serem criadas se esse número não for conhecido em tempo de projeto, pois esta é uma preocupação associada à instanciação e execução do processo.

Em NPDL:

A NPDL possui uma segunda variação do operador “?” que permite a associação de uma função para determinar em tempo de execução quantas instâncias da atividade serão criadas. A função que pode ser associada ao “?” é uma função atômica cujo valor de retorno é um inteiro positivo e não nulo. A função será avaliada em tempo de execução do processo.

$$\text{SET } P1 = A \text{ ?}f \text{ . } B \text{ . } C$$

sendo f uma função que retorna um número inteiro e maior que zero

15) Múltiplas Instâncias Sem Conhecimento Prévio em Tempo de Execução

Definição: Ocorre quando, em uma instância de processo, uma atividade é habilitada múltiplas vezes. O número de instâncias a serem executadas da atividade em questão não é conhecido nem em tempo de projeto, nem em tempo de execução, ou seja, ele não é conhecido antes do momento de criação das instâncias da atividade. Após o término da execução de todas as instâncias criadas, uma outra atividade subsequente do *workflow* é habilitada para execução.

Em álgebra de processos:

Este padrão tem sua especificação em álgebra de processos bastante semelhante à do padrão 12. A diferença é a necessidade de sincronização das instâncias criadas antes da habilitação da próxima atividade do *workflow*. Para efetuar tal sincronização, utiliza-se o operador de composição seqüencial “•”. Por exemplo, um processo que possa executar múltiplas instâncias de A e que só execute B após a sincronização de todas as instâncias de A criadas pode ser definido como:

$$P' = (P' \parallel A) + A$$

$$P_1 = P'.B$$

Em NPD:

SET P1 = A?* . B

3.2.5 Padrões Baseados em Estados

16) Escolha Postergada

Definição: É um ponto no processo de *workflow* no qual um entre vários ramos de atividades é escolhido. Diferentemente da **Escolha Exclusiva**, na **Postergada** a escolha não é feita de modo explícito, ou seja, não se baseia em dados ou em uma decisão. As diferentes alternativas são oferecidas ao ambiente, que poderá executar apenas uma delas.

Exemplo: Após a execução de **A**, **B** e **C** são habilitadas, mas somente **uma** delas será escolhida **explicitamente** para execução.

Figura 3.11: Escolha Postergada (Fonte: [42])

Em álgebra de processos:

$$P = A . (B + C)$$

Como já discutido no mapeamento do padrão de **escolha exclusiva**, o operador “+” da álgebra de processos representa o comportamento exato da **escolha postergada**.

Em NPD:

SET P = A.(B+C)

17) Roteamento Paralelo Entrelaçado

Definição: Ocorre quando um conjunto de atividades é executado em qualquer ordem arbitrária, ou seja, cada atividade do conjunto é executada uma vez, sendo que a

ordem é decidida em tempo de execução e duas atividades nunca são executadas ao mesmo tempo.

Exemplo: A atividade **A** é executada, habilitando a execução de **B** seguida pela execução de **C**, ou a execução de **C** seguida pela execução de **B**. Após o término da execução de **B** e **C** (ou **C** e **B**), a atividade **D** é habilitada para execução.

Figura 3.12: Roteamento Paralelo Entrelaçado (Fonte: [42])

Em álgebra de processos:

$$P = A . (B \parallel C) . D$$

Caso as linhas de atividades possíveis se dividam em mais de duas, o mapeamento para álgebra de processos deve combinar mais de um operador binário “ $\parallel$ ”. Exemplo: $P = A . (B \parallel (C \parallel (D \parallel E))) . F$

Utilizou-se para este mapeamento o operador “ $\parallel$ ” da álgebra de processos, assim como no padrão **divisão paralela**. Na álgebra de processos, tomando duas ações atômicas a e b , se dissermos que um processo P é tal que $P = a \parallel b$ é equivalente dizer que $P = ab + ba$ (considerando que $\gamma(a, b) = \delta$). A álgebra de processos não restringe a forma como a implementação do processo P deve ser feita (não obriga que as ações a e b sejam realmente executadas concorrentemente), ela apenas garante que ao final da execução de P , as ações a e b foram executadas. Como esta é uma preocupação de implementação, definimos na NPDL um novo operador binário, o “ $|*$ ”, que implementa o comportamento do **roteamento paralelo entrelaçado**, enquanto o operador “ $\parallel$ ” da NPDL representa o paralelismo real (**divisão paralela**).

Em NPDL:

$$\text{SET } P = A . (B \mid * C) . D$$

Internamente, o serviço de execução de *workflows* definidos em NDPL traduz a expressão

$$P = A.(B \mid * C).D$$

para a expressão equivalente

$$P = A.(B.C + C.B).D$$

Tal quais os operadores “||”, “+” e “•”, o operador “|*” é associativo, o que possibilita a representação de um **roteamento paralelo entrelaçado** entre mais de duas atividades, i.e.:

$$(A \mid * B) \mid * C = A \mid * (B \mid * C) = A \mid * B \mid * C$$

A associatividade do operador “|*” tem um efeito colateral que pode ser notado quando um ou mais termos operandos do “|*” precisam ser tratados como atividades atômicas. Por exemplo, se a intenção real de uma definição de processo como

$$(A \mid * B) \mid * C$$

é garantir que A , B e C sejam executados em qualquer ordem (mas não paralelamente) e que C nunca seja executado entre a execução de A e B (ou B e A), a forma correta de definir o processo em NPDL é expandindo o termo $(A \mid * B)$, tornando a definição desta forma:

$$(A.B + B.A) \mid * C$$

18) Marco

Definição: Ocorre quando a habilitação de uma atividade é dependente de um estado específico da instância de um processo. Ou seja, a atividade só é habilitada se um certo marco tiver sido atingido e ainda não tiver expirado.

Exemplo: Considere três atividades A , B e C . A atividade A somente será habilitada depois que a atividade B tiver sido executada, mas se C não tiver sido executada ainda, ou seja, A não estará habilitada antes da execução de B e nem depois da execução de C . A rede de Petri da Figura 3.13 ilustra o padrão. O estado entre B e C é modelado pelo lugar M . Este lugar é o **marco** para A . Note que a atividade A não remove a ficha de M , ela somente testa a presença da ficha.

Figura 3.13: Marco

Em álgebra de processos:

No caso de **marcos** que envolvam atividades de uma só linha de execução, a especificação em álgebra de processos é direta, exigindo apenas a definição de um subprocesso recursivo que represente a possibilidade da execução da ação do **marco** repetidas vezes (até que a condição do **marco** não tenha expirado ainda). Por exemplo, o processo P ilustrado na Figura 3.13 é representado em álgebra de processos por:

$$\begin{aligned} P_1 &= A.P_1 + C \\ P &= B.P_1 \end{aligned}$$

Quando o padrão **marco** envolve atividades de mais de uma linha de execução, não há uma regra bem definida para a especificação, cada caso tem que ser analisado separadamente. Entretanto, assim como no mapeamento dos **ciclos arbitrários**, subprocessos e equações recursivas devem ser utilizados para a representação em álgebra de processos.

Figura 3.14: Marco envolvendo duas linhas paralelas (Baseado em [42])

O processo da Figura 3.14 é um exemplo de **marco** que envolve duas linhas que são executadas em paralelo. Um possível mapeamento desse processo para álgebra de processos é:

$$\begin{aligned} P_1 &= F.P_1 + F.C.(D \parallel G) \\ P &= A.(B \parallel E).P_1.H + A.(B.C.D \parallel E.G).H \end{aligned}$$

É importante ressaltar que no processo da Figura 3.14 é possível a ocorrência de um *deadlock*. Se a atividade C for executada logo após a execução de B , sem possibilitar que F seja executada, a segunda linha de execução da **divisão paralela** nunca vai ser encerrada. Como é necessário o sincronismo das linhas da divisão para que a atividade H possa ser habilitada, se F não for executada o processo entrará em *deadlock*. Esta possibilidade do *deadlock* foi representada na expressão em álgebra de processos pelo termo $A.(B.C.D \parallel E.G).H$.

Em NPDL:

A representação em NPDL deste padrão consiste na tradução da especificação em álgebra de processos para comandos de definição de processos, sendo que nesta definição podem ser utilizados todos os operadores NPDL definidos anteriormente. Exemplo:

$$\begin{aligned} \text{SET } P1 &= F.P1 + F.C.(D \mid G) \\ \text{SET } P &= A.(B \mid E).P1.H + A.(B.C.D \mid E.\#.G).H \end{aligned}$$
3.2.6 Padrões de Cancelamento**19) Atividade Cancelável**

Definição: Ocorre quando uma atividade pode ser desabilitada durante a execução da instância do processo. Quando a atividade é desabilitada, é necessário remover a linha que controla a execução da atividade em questão.

Em álgebra de processos:

É possível representar o comportamento do padrão **atividade cancelável** em álgebra de processos por meio da substituição, na expressão do processo, da atividade que permite cancelamento por um termo envolvendo o símbolo de *deadlock* “ δ ”. Esta atividade pode ser representada alternativamente composta com “ δ ”, por meio do operador “+”. Por exemplo, se no processo $P = A.B.(C + D)$, B é uma atividade que pode ser cancelada, então o mapeamento correto do processo deve ser: $P = A.(B + \delta).(C + D)$

Em NPDL:

O símbolo “#” na NPDL representa o “ δ ” da álgebra de processos.

$$\text{SET } P = A.(B + \#).(C + D)$$
20) Caso Cancelável

Definição: Ocorre quando a instância do *workflow* pode ser removida completamente.

Em álgebra de processos:

Este padrão está inteiramente associado ao *workflow* em seu tempo de execução e por isso não recebe representação na álgebra de processos.

Em NPDL:

Toda instância de processo definido em NPDL pode ser cancelada em qualquer instante de sua execução, sendo esta, uma responsabilidade do serviço de execução.

3.3 Conclusão: Usando NPDL para especificar um exemplo de *workflow*

Como conclusão do Capítulo 3, ilustrar-se-á uma especificação completa em NPDL, retomando o sistema de tratamento de reclamações utilizado na Seção 2.2 como exemplo de modelagem em rdP. A Figura 3.15 mostra novamente o modelo do sistema em rdP, desta vez, adicionando letras maiúsculas como rótulos para as transições.

Figura 3.15: Sistema de processamento de reclamações (Fonte: [6])

O primeiro passo no sistema é o registro da reclamação, e como consequência um questionário é enviado ao reclamante. A reclamação é avaliada ao mesmo tempo em que o questionário é enviado. Se o reclamante não retornar o questionário preenchido dentro de 2 semanas o prazo de retorno é considerado expirado e o resultado do questionário será descartado. Após a avaliação, uma reclamação pode ser processada ou não. A reclamação somente é processada após o processamento do questionário ou após o prazo de retorno ter expirado. Após estas atividades, a reclamação é arquivada.

Esse caso ilustra, além de um ciclo, o uso de vários padrões de controle de fluxo: **divisão paralela**, após a atividade *A* (registro), que habilita a execução paralela de *B* (envio do questionário) e *F* (avaliação); **escolha postergada**, entre as atividades *C* (processamento do questionário) e *D* (expiração do prazo); **escolha exclusiva** entre as atividades *G* (processamento requerido) e *H* (omissão) e entre *L* (OK – processamento correto) e *K* (NOK – processamento incorreto); **marco** para a atividade *I* (processamento da reclamação); e, finalmente, **sincronização** para a execução da atividade *E* (arquivamento).

Conforme discutido anteriormente, é possível especificar esses padrões em álgebra de processos e em NPDL. Em álgebra de processos, o sistema de tratamento de reclamações é definido pelo processo *P*:

$$P_1 = I.J.(K.P_1 + L)$$

$$P = A.((B.(C + D) \parallel F.G).P_1 + (B.(C + D) \parallel F.H)).E$$

É importante frisar que em álgebra de processos não é possível diferenciar uma **escolha exclusiva** de uma **escolha postergada**. Como resultado, a NPDL permite a definição de um processo *P* que representa com mais precisão o comportamento do sistema de tratamento de reclamações:

$$\text{SET } P_1 = I.J.(\%!r_1 K.P_1 + \%r_1 L)$$

$$\text{SET } P = A.((B.(C + D) \parallel F . \%r_2 G).P_1 + (B.(C + D) \parallel F . \%!r_2 H)).E$$

sendo *r*₁ uma regra que verifica alguns atributos manipulados pela atividade *J* (verificação do processamento) e *r*₂ uma regra que verifica alguns atributos manipulados pela atividade *F* (avaliação).

O objetivo deste exemplo é mostrar como a NPDL pode simplificar uma modelagem e ainda expressar aspectos do sistema que a álgebra de processos sozinha

não está habilitada a representar. Esta maior capacidade de representação se deve aos operadores adicionados à NPDL para atender todos os padrões de controle de fluxo.

Capítulo 4

Implementação da NPDL

A NPDL, apresentada no capítulo 3, é uma linguagem para auxiliar a criação e manutenção da representação de processos em um banco de dados relacional, implementada como uma extensão da linguagem SQL.

O interpretador da NPDL é parte da ferramenta *NavigationPlanTool*. A *NavigationPlanTool* foi desenvolvida para atender às características especificadas pela WfMC (*Workflow Management Coalition*) para um sistema de *workflow* [28]. Essas características estão ilustradas na Figura 4.1.

Em um sistema de *workflow* é possível identificar três interfaces: (1) projeto e definição dos processos; (2) instanciação e controle dos processos; e (3) interação com usuários e aplicativos. Embora padrões venham sendo definidos para cada uma destas interfaces, esses padrões estão mais voltados para a sintaxe que para a semântica dos mecanismos de controle de fluxo [6]. Os arcabouços formais para especificação de *workflows*, como as redes de Petri e as álgebras de processos, fazem-se necessários para eliminar a ausência de padronização semântica.

A NPDL atende à interface 1 de um sistema de *workflow*. Para as demais interfaces, a *NavigationPlanTool* também provê mecanismos para o controle da instanciação de processos, disponibilizando serviços para a manutenção de instâncias de processos e serviços para o monitoramento da execução do plano de navegação. Em outubro de 2005, a *NavigationPlanTool* foi apresentada na Sessão de Demos do XX Simpósio Brasileiro de Banco de Dados [15].

Figura 4.1: Características de um sistema de *workflow* (Fonte: [28])

4.1 A *NavigationPlanTool*

A *NavigationPlanTool* foi implementada na forma de uma biblioteca de funções. Esta biblioteca provê mecanismos para a manutenção de ações e processos em bancos de dados relacionais e para o controle de instanciação e execução desses processos. A biblioteca disponibiliza operações como criação/remoção de instâncias e serviços para o monitoramento da execução do plano de navegação. Esses serviços são também responsáveis pela manutenção de *logs* de execução de planos de navegação no banco de dados e pela recuperação de execuções que tenham sido interrompidas antes de serem finalizadas.

A linguagem de programação escolhida para o desenvolvimento da *NavigationPlanTool* é a Java (*Java 2 Platform Standard Edition - J2SE 5.0*) [29], por ser portátil e por contar com a JDBC API - *Java DataBase Connectivity Application Programming Interface*. A JDBC habilita programas Java a executarem comandos em SQL e interagirem com qualquer banco de dados compatível com a definição padrão da linguagem SQL, tornando assim a NPDLP uma extensão independente de sistema gerenciador de banco de dados.

Por ser desenvolvida na forma de biblioteca de funções, a *NavigationPlanTool* pode ser integrada de forma fácil a outras aplicações Java.

A *NavigationPlanTool* é composta por três serviços: o **Interpretador NPDL**, o **Serviço de Instanciação de Processos** e o **Serviço Monitor de Execução de Instâncias de Processos**. Nas seções seguintes, cada um desses serviços será detalhado.

Outras informações sobre o desenvolvimento, estudos de caso ou instruções para o uso da *NavigationPlanTool* podem ser obtidos no documento [14].

4.2 O Interpretador da Linguagem NPDL

A função do **Interpretador NPDL** é receber um comando de entrada e efetuar as análises léxica, sintática e semântica. Caso o comando seja identificado como um comando NPDL válido, ele é executado sobre as relações criadas pelo próprio interpretador em um banco de dados relacional, para o armazenamento dos dados dos processos, ações e instâncias.

O interpretador é uma implementação da interface *java.sql.Connection* de Java, que recebeu o nome **NPDLConnection**. A interface *java.sql.Connection* representa uma conexão com um banco de dados específico. A partir de um objeto de uma classe de conexão é possível executar consultas SQL sobre o banco de dados.

Criar uma conexão NPDL com um banco de dados significa preparar este banco para ser usado no controle de processos de negócio via *NavigationPlanTool*. Quando uma aplicação instancia um objeto da classe **NPDLConnection**, a rotina de iniciação do objeto verifica a existência das relações utilizadas pela *NavigationPlanTool* no banco de dados alvo da conexão. Caso essas relações ainda não existam no banco, elas são criadas pela rotina de iniciação.

Uma conexão do tipo **NPDLConnection** permite que tanto comandos NPDL quanto comandos SQL sejam submetidos ao banco de dados. Ao receber um comando, uma conexão NPDL o verifica e, caso o identifique como um comando NPDL válido, executa as rotinas que traduzem o comando NPDL para operações em SQL (padrão da linguagem). Esses comandos SQL são então executados sobre as relações associadas à *NavigationPlanTool*. Quando o comando submetido para a conexão NPDL não é reconhecido como um comando NPDL válido, ele é repassado diretamente ao banco de dados.

4.2.1 Estruturas relacionais de dados criadas pelo Interpretador NPDL

As estruturas de dados criadas em banco de dados relacional baseiam-se nas estruturas sugeridas pela arquitetura *RiverFish*, descrita na Seção 2.5 deste texto. Para representar processos de negócio e controlar sua instanciação e execução por meio da abordagem proposta por esta dissertação, são necessárias as estruturas descritas no diagrama da Figura 4.2. A notação utilizada no diagrama é a especificada em [39] para Modelos Entidade-Relacionamento Estendidos.

Em NPDL, o plano de navegação de um processo é definido por uma expressão algébrica que pode envolver ações, operadores, regras, funções, números e outros processos. O relacionamento **REL_PLANO_NAVEGAÇÃO** entre as entidades **PROCESSO** e **PASSO** representa esta definição. A especialização da entidade **PASSO** é total e disjunta, ou seja, todo elemento dessa entidade obrigatoriamente possui um dos seguintes tipos: processo, ação, operador, regra (que é utilizada associada aos operadores “%” e “%!”, função (que é utilizada associada ao operador “?”) e número (que é utilizado associado ao operador “?”).

Figura 4.2: Diagrama Entidade-Relacionamento

Antes de ser armazenada em banco de dados por meio do relacionamento **REL_PLANO_NAVEGAÇÃO**, a expressão algébrica de processo é convertida em uma expressão equivalente na notação prefixa. A expressão prefixa é mais apropriada para o armazenamento em banco de dados e para manipulação nos algoritmos que avaliam a validade da expressão. O atributo **ordem_passo** do relacionamento **REL_PLANO_NAVEGAÇÃO** é utilizado para armazenar a posição do passo dentro da expressão algébrica. Esta informação é necessária para a posterior recuperação da expressão do processo.

A entidade **INSTÂNCIA_PROCESSO** e o relacionamento **REL_LOG_INSTANCIA** estão associados à instanciação e ao controle de execução de processos e por isso são utilizadas pelos outros serviços da *NavigationPlanTool*.

Um elemento da entidade **INSTÂNCIA_PROCESSO** está sempre associado a um elemento da entidade **PROCESSO**, por meio do relacionamento **REL_INSTANC_PROCESSO**. Uma instância representa uma requisição a um processo, por isso possui como atributos o usuário que efetuou a requisição e a sua data. O relacionamento **REL_LOG_INSTÂNCIA** representa os dados relativos ao estado da execução dos passos que compõem o plano de navegação de uma instância de processo. Cada elemento de **REL_LOG_INSTÂNCIA** representa a execução de um passo em uma determinada instância de processo. O atributo **estado_execução** do relacionamento **REL_LOG_INSTÂNCIA** pode assumir um dos valores descritos na Tabela 4.1.

Os tipos de passos que são realmente “executáveis” são **AÇÃO**, **REGRA** E **FUNÇÃO**. Por este motivo, somente estas três entidades da especialização de **PASSO** possuem o atributo **chamada**, que representa uma instrução para a execução do passo.

O Apêndice B deste texto traz exemplos de como os comandos NPD L são traduzidos para comandos SQL e depois aplicados sobre o modelo físico de dados que representa o diagrama da Figura 4.2.

Tabela 4.1: Possíveis estados para passos

Valor	Representando um passo...
I	Iniciado, mas ainda não finalizado
F	Finalizado
C	Cancelado

4.3 O Serviço de Instanciação de Processos

O **Serviço de Instanciação de Processos** disponibiliza funções para a criação de instâncias de processos. Uma instância representa uma requisição de um determinado processo. Toda instância possui uma referência para o processo ao qual está associada, e pode também armazenar informações sobre o usuário que a requisitou e a sua data de criação.

Quando a criação de uma instância é solicitada à *NavigationPlanTool* por uma aplicação, o serviço de instanciação retorna à aplicação um identificador da instância. Por meio desse identificador, a aplicação pode obter os demais dados da instância e interagir com o serviço que controla a execução do plano de navegação da instância.

4.4 O Serviço Monitor de Execução de Instâncias de Processos

O serviço **Monitor de Execução de Instâncias de Processos** é responsável por ligar uma instância de processo aos seus dados de execução (plano de navegação). Neste serviço, estão agrupadas as funções que iniciam e monitoram a execução do plano de navegação de uma instância de processo.

Para controlar a execução do plano de navegação, são utilizados elementos do relacionamento **REL_LOG_INSTÂNCIA** (descrito na Figura 4.2) e estruturas de dados em memória.

Quando a execução de uma instância é iniciada, o plano de navegação associado a ela é obtido no relacionamento **REL_PLANO_NAVEGAÇÃO**. A partir da expressão algébrica prefixa que representa o plano de navegação, é construída uma **árvore de expressão** da instância. As árvores de expressão são utilizadas pelo serviço monitor para determinar a ordem de execução dos passos de um plano de navegação. Neste trabalho, denominou-se de **árvore de navegação** a árvore de expressão de uma instância de processo.

Após a criação da árvore de navegação, o monitor realiza a recuperação do estado atual da execução da instância, por meio de consultas aos dados mantidos pelo relacionamento **REL_LOG_INSTÂNCIA**. Esta operação só é realizada para as instâncias que tiveram o seu processo de execução iniciado, mas ainda não finalizado.

Cada passo iniciado, finalizado ou cancelado do plano de navegação de uma instância resulta na inserção ou atualização de um registro no *log*. Instâncias que não iniciaram ao menos um passo de seu plano de navegação não possuem informações gravadas no *log*.

A execução de instâncias distintas de um mesmo processo pode resultar em *logs* diferentes. Isso ocorre porque o plano de navegação mapeia todas as possibilidades de execução de um processo, mas são os dados da instância que determinarão o comportamento final de cada nova execução desse processo.

Muitas linguagens para a modelagem de *workflows* e seus mecanismos de controle de execução afirmam que utilizam formalismos como rdPs ou álgebras de processos como base para o seu funcionamento. Aalst, em [1], critica o fato de poucos explicitarem a forma como isso é feito. Segundo o autor, é importante que o formalismo e sua forma de implementação sejam claramente explicitados para melhorar a qualidade e a aplicabilidade da linguagem.

Atendendo a crítica feita por Aalst, a próxima seção deste texto detalha a implementação do **Serviço Monitor de Execução de Instâncias de Processos** da *NavigationPlanTool*, com o objetivo de mostrar como as propriedades da álgebra de processos são utilizadas, neste trabalho, para facilitar o controle e execução de processos.

4.4.1 A navegação pelo plano

A “navegação” pelo plano de uma instância de um processo é quem determina a ordem de execução dos passos desse processo. Os algoritmos do serviço monitor utilizam a semântica dos operadores da NPDL e regras de transição da álgebra de processos para percorrer uma árvore de navegação.

Um nó de uma árvore de navegação pode representar um dentre três tipos de elementos: um operador NPDL, uma ação ou um processo. Uma árvore de navegação é uma árvore binária completa, ou seja, todo nó pertencente a ela é “folha”, ou possui dois “filhos”. Os nós internos da árvore de expressão representam sempre operadores binários da NPDL, enquanto que os nós folhas representam ações ou processos.

Um processo P , cuja definição em NPDL é $P = ((a+b) \cdot c) \parallel ((a+c) \cdot d)$, sendo a , b , c e d ações previamente definidas, tem sua árvore de navegação ilustrada na Figura 4.3.

Figura 4.3: Árvore de navegação do processo $P = ((a+b) \cdot c) \parallel ((a+c) \cdot d)$

Como os nós das árvores de navegação representam também os possíveis passos de execução, um outro atributo importante de um nó é o seu estado atual. Todos os nós de uma árvore de navegação têm sempre um estado definido. Um nó, em um dado momento, pode estar em um dos seguintes estados:

1. Não iniciado
2. Iniciado
3. Finalizado
4. Cancelado

O serviço monitor de execução oferece para as aplicações que utilizam a *NavigationPlanTool* as seguintes funcionalidades¹⁰:

- a) **Obtenção dos possíveis passos para a execução** – esta função retorna para a aplicação os nomes dos passos que a instância pode executar, considerando o seu estado atual. Um passo na NPDL pode ser uma ação, uma regra ou uma função que retorna um número inteiro;
- b) **Indicação do início da execução de um passo** – esta função recebe da aplicação o nome de um passo. Caso ele seja um passo atual válido, então é criado um registro na relação de *log* indicando que o passo foi iniciado. A cada passo iniciado está associado um identificador que permite localizar no *log* o registro referente a esse passo. O identificador do passo

¹⁰ Cabe ressaltar que todas as funcionalidades são aplicadas sobre uma determinada instância de processo.

iniciado é retornado para a aplicação. Os estados dos nós da árvore de navegação são atualizados para refletir o início da execução do novo passo;

- c) **Indicação do fim da execução de um passo** - esta função recebe da aplicação um identificador de um passo já iniciado. Caso ele seja um identificador válido, o registro que representa o passo na relação de *log* é atualizado, indicando que a sua execução foi finalizada. Os estados dos nós da árvore de navegação são atualizados para refletir o fim da execução do passo;
- d) **Indicação do cancelamento da execução de um passo** - esta função recebe da aplicação um identificador de um passo já iniciado. Caso ele seja um identificador válido, o registro que representa o passo na relação de *log* é atualizado, indicando que a sua execução foi cancelada. Os estados dos nós da árvore de navegação são atualizados para refletir o cancelamento da execução do passo.

É importante destacar que as ocorrências de (b), (c) ou (d) modificam o estado da instância. Quando a funcionalidade (a) é requerida pela aplicação, o serviço monitor percorre a árvore para obter os passos válidos.

Se a execução de instância acaba de ser iniciada, todos os nós de sua árvore de navegação possuem estado atual “Não iniciado”. Se a execução da instância já tiver sido iniciada em alguma execução anterior do serviço monitor, haverá registros relativos à instância no *log* e eles deverão ser “carregados” para a árvore de navegação. Isso é feito por meio de uma atualização nos estados dos nós correspondentes aos passos registrados como iniciados, finalizados ou cancelados no *log*.

O percurso e a obtenção de passos válidos podem ser descritos de forma simplificada pelo algoritmo do procedimento recursivo **obtenhaPróximosPassos**, descrito na Tabela 4.2.

O monitor de execução começa a percorrer a árvore de navegação a partir de seu nó raiz. Portanto, a primeira chamada de **obtenhaPróximosPassos** recebe no parâmetro **nóAtual** o nó raiz e no parâmetro **passosVálidos** um conjunto vazio. Ao final da execução do procedimento, o conjunto **passosVálidos** contém os passos atualmente habilitados para a execução da instância.

Tabela 4.2: Procedimento obtenhaPróximosPassos(nóAtual, passosVálidos)

Linha	Instrução
1	Se (nóAtual é nulo)
2	Saia do Procedimento
3	Se (estado do nóAtual é "Finalizado") ou (estado do nóAtual é "Cancelado")
4	Execute obtenhaPróximosPassos(pai de nóAtual, passosVálidos)
5	Se (nóAtual possui regra associada a ele)
6	Se (estado da regra é "Não iniciada")
7	Inclua a regra no conjunto passosVálidos
8	Saia do procedimento
9	Se (estado da regra é "Iniciada") ou (estado da regra é "cancelada")
10	Saia do procedimento
11	Se (estado da regra é "Executada") e (seu valor de retorno é "Falso") e (a regra é de uma Execução Condicional)
12	Saia do procedimento
13	Se (estado da regra é "Executada") e (seu valor de retorno é "Verdadeiro") e (a regra é de uma Execução Condicional Negativa)
14	Saia do procedimento
15	Se nóAtual possui função associada a ele
16	Se (estado da função é "Não iniciada")
17	Inclua a função no conjunto passosVálidos
18	Saia do procedimento
19	Se (estado da função é "Iniciada") ou (estado da função é "Cancelada")
20	Saia do procedimento
21	Se (nóAtual é repetitivo)
22	Execute tratarRepeticao(nóAtual)
23	Senão Se (nóAtual é uma Ação)
24	Se (estado do nóAtual é "Não iniciado")
25	Inclua a ação no conjunto passosVálidos
26	Senão Se (nóAtual é um Operador)
27	Caso nóAtual seja uma Composição Seqüencial
28	Se (estado de nóAtual é "Não iniciado")
29	Execute obtenhaPróximosPassos(filho esquerdo de nóAtual, passosVálidos)
30	Senão Se (estado de filho esquerdo de nóAtual ="Iniciado")
31	Execute obtenhaPróximosPassos(filho esquerdo de

```

nóAtual, passosVálidos )

32      Senão
33          Execute obtenhaPróximosPassos( filho direito de nóAtual,
                                           passosVálidos )

34      Caso nóAtual seja uma Composição Alternativa
35          Se ( estado de nóAtual é "Não iniciado" )
36              Execute obtenhaPróximosPassos( filho esquerdo de
                                                nóAtual, passosVálidos )
37              Execute obtenhaPróximosPassos( filho direito de nóAtual,
                                                passosVálidos )

38      Senão Se (estado de filho esquerdo de nóAtual="Iniciado")
39          Execute obtenhaPróximosPassos( filho esquerdo de
                                                nóAtual, passosVálidos )

40      Senão
41          Execute obtenhaPróximosPassos( filho direito de nóAtual,
                                           passosVálidos )

42      Caso nóAtual seja uma Composição Paralela
43          Criar conjunto vazio passosVálidosEsquerda
44          Criar conjunto vazio passosVálidosDireita
45          Execute obtenhaPróximosPassos( filho esquerdo de nóAtual,
                                           passosVálidosEsquerda )
46          Execute obtenhaPróximosPassos( filho direito de nóAtual,
                                           passosVálidosDireita )

47          Criar conjunto vazio passosVálidosParalelos
48          Para cada passoEsquerda pertencente a
                                           passosVálidosEsquerda
49              Para cada passoDireita pertencente a
                                           passosVálidosDireita
50                  Crie o passoParalelo, que representa a execução de
                    passoEsquerda paralelamente a passoDireita
51                  Incluir passoParalelo no conjunto passosVálidos
52                  Incluir todos os elementos do conjunto
                    passosVálidosEsquerda no conjunto passosVálidos
53                  Incluir todos os elementos do conjunto
                    passosVálidosDireita no conjunto passosVálidos
54      Caso nóAtual seja uma Composição Paralela Entrelaçada
55          Execute tratarComposiçãoParalelaEntrelaçada( nóAtual,
                                                         passosVálidos
                                                         )

56      Caso nóAtual seja uma Composição Multi-Convergente
57          Execute tratarComposiçãoMultiConvergente( nóAtual,

```

```

passosVálidos )
58      Caso nóAtual seja uma Composição Discriminatória
59          Execute tratarComposiçãoDiscriminatória( nóAtual,
passosVálidos )
60      Senão Se ( nóAtual é um Processo )
61          Se ( processo do nóAtual é o mesmo processo que deu origem à
        árvore de navegação )
62              Execute tratarRecursão( nóAtual, passosVálidos )
63      Senão
64          Crie uma árvore de navegação do processo associado ao
        nóAtual
65          Conecte a nova árvore a árvore antiga, substituindo o
        nóAtual pela raiz da nova árvore
66          Considere como nóAtual a raiz da árvore que foi criada
67          Execute obtenhaPróximosPassos( nóAtual, passosVálidos )
Fim

```

Os operadores de **execução condicional**, **execução condicional negativa**, **repetição ilimitada**, **repetição numericamente limitada** e **repetição limitada por função** não são tratados como nós na árvore de navegação. Eles são tratados como atributos de um nó da árvore por influenciarem a execução de todo o ramo iniciado pelo nó ao qual estão associados.

Quando uma ação ou trecho de processo é delimitado por algum dos operadores de execução condicional, a regra que condiciona a execução é atribuída ao nó que representa a ação delimitada ou ao nó que é o início do ramo com o trecho de processo delimitado. A linha 5 do algoritmo **obtenhaPróximosPassos** inicia o tratamento de nós cuja execução esteja associada a avaliação de uma regra.

O mesmo ocorre com as ações repetitivas ou trechos de processos repetitivos: o nó que representa a ação ou início do trecho na árvore é rotulado como repetitivo, e caso esta repetição seja limitada por um número ou uma função, este número ou função é associado ao nó.

Sempre que um nó possuir uma regra ou função associada, é necessário que esta regra ou função seja executada antes que o nó possa ser “visitado” pelo algoritmo. A tarefa de execução da regra ou função, assim como no caso das ações atômicas, é de responsabilidade da aplicação que usa a *NavigationPlanTool*. Por isso, as regras e funções também são retornadas pelo procedimento **obtenhaPróximosPassos** à aplicação como passos atuais válidos.

São considerados operadores básicos em NPDL os operadores **composição sequencial**, **composição alternativa** e **composição paralela**. Os operadores de **composição paralela entrelaçada**, **composição multi-convergente**, **composição discriminatória** e todos os de **repetição**, conforme discutido na Seção 3.2 deste texto, são operadores que podem ter seu comportamento expresso por meio da combinação dos operadores básicos. Por este motivo, eles não são explicitamente tratados pelo procedimento **obtenhaPróximosPassos** (linhas 21, 54, 56 e 58 do algoritmo), mas sim por procedimentos de mapeamento que os eliminam da árvore de navegação. Esses mapeamentos são os descritos para os padrões **17 - Roteamento Paralelo Entrelaçado**, **8 - Junção Múltipla**, **9 - Discriminador** e todos os de **Múltiplas Instâncias**, na Seção 3.2. O mapeamento substitui estes operadores por ramos de nós que utilizam somente os operadores básicos e que expressem o mesmo comportamento.

Processos em NPDL que envolvam em sua definição outros processos têm seu tratamento explicitado a partir da linha 64 do algoritmo do procedimento **obtenhaPróximosPassos**. A árvore de navegação para o processo referenciado é criada e “conectada” a árvore de navegação da instância, substituindo o nó que representava o processo. Após a substituição do nó de processo pela árvore que o representa, o procedimento de obtenção dos passos válidos é aplicado sobre o nó raiz da árvore do processo referenciado.

Processos recursivos requerem um tratamento diferenciado. Quando o algoritmo de percurso se depara com um nó que representa o próprio processo em execução, ele chama uma outra rotina (linha 61 do procedimento **obtenhaPróximosPassos**), responsável por realizar a “expansão” da recursão. Essa rotina faz uma cópia da árvore de navegação e a “conecta” à árvore que lhe deu origem, substituindo o nó que representava a recursão. A rotina garante ainda que ocorra apenas um nível de expansão por nó recursivo em cada chamada externa¹¹ do procedimento **obtenhaPróximosPassos**. Essa restrição é necessária para garantir que o procedimento não entre em um ciclo infinito, expandindo indefinidamente os nós recursivos.

Para que o procedimento **obtenhaPróximosPassos** funcione de maneira apropriada, toda mudança de estado de um nó folha da árvore de navegação requer uma

¹¹ Neste contexto, entende-se por **chamada externa** uma chamada que tenha sido originada por uma aplicação que utilize a *NavigationPlanTool*.

atualização do estado dos seus nós ancestrais¹². Para possibilitar esta operação, todo nó mantém um apontador para o seu nó “pai”, além dos apontadores para os nós “filho esquerdo” e “filho direito”. O algoritmo do procedimento recursivo **atualizaEstadoÁrvore**, listado na Tabela 4.3, descreve como a atualização dos nós da árvore de navegação é feita.

Tabela 4.3: Procedimento **atualizaEstadoÁrvore**(nó)

Linha	Instrução
1	Se (nó é nulo)
2	Saia do Procedimento
3	Se (nó é um Operador)
4	Caso nó seja Composição Seqüencial
5	Se (estado do filho esquerdo do nó é “Cancelado”) ou (estado do filho direito do nó é “Cancelado”)
6	Faça estado do nó ser “Cancelado”
7	Senão Se (estado do filho esquerdo do nó é “Finalizado”) e (estado do filho direito do nó é “Finalizado”)
8	Faça estado do nó ser “Finalizado”
9	Senão Se (estado do filho esquerdo do nó é “Iniciado”) ou (estado do filho direito do nó é “Iniciado”) ou (estado do filho esquerdo do nó é “Finalizado”) ou (estado do filho direito do nó é “Finalizado”)
10	Faça estado do nó ser “Iniciado”
11	Senão
12	Faça estado do nó ser “Não iniciado”
13	Caso nó seja Composição Alternativa
14	Se (estado do filho esquerdo do nó é “Cancelado”) e (estado do filho direito do nó é “Cancelado”)
15	Faça estado do nó ser “Cancelado”
16	Senão Se (estado do filho esquerdo do nó é “Finalizado”) ou (estado do filho direito do nó é “Finalizado”)
17	Faça estado do nó ser “Finalizado”
18	Senão Se (estado do filho esquerdo do nó é “Iniciado”) ou (estado do filho direito do nó é “Iniciado”)
19	Faça estado do nó ser “Iniciado”
20	Senão
21	Faça estado do nó ser “Não iniciado”

¹² Em uma estrutura de dados de árvore, um nó *A* é dito **ancestral** de algum outro nó *N*, se *A* fizer parte do conjunto de nós pertencentes ao caminho entre a raiz da árvore e *N*.

```

22      Caso nó seja Composição Paralela
23          Se ( estado do filho esquerdo do nó é "Cancelado" )
24              ou( estado do filho direito do nó é "Cancelado" )
25              Faça estado do nó ser "Cancelado"
26      Senão Se ( estado do filho esquerdo do nó é "Finalizado" )
27              e ( estado do filho direito do nó é "Finalizado" )
28              Faça estado do nó ser "Finalizado"
29      Senão Se ( estado do filho esquerdo do nó é "Iniciado" )
30              ou ( estado do filho direito do nó é "Iniciado" )
31              ou ( estado do filho esquerdo do nó é "Finalizado" )
32              ou ( estado do filho direito do nó é "Finalizado" )
33              Faça estado do nó ser "Iniciado"
34      Senão
35          Faça estado do nó ser "Não iniciado"
36      atualizaEstadoÁrvore( pai do nó )
Fim

```

O início e a finalização de regras ou funções implicam somente na atualização do estado associado à própria regra ou função, não requerendo, portanto, a atualização de outros nós da árvore.

No procedimento **atualizaEstadoÁrvore** somente são analisados os nós de operadores básicos. O **atualizaEstadoÁrvore** é executado sempre que as funcionalidades (b), (c) e (d) do serviço monitor de execução são aplicadas sobre ações atômicas. Mas estas funcionalidades só podem ser aplicadas sobre ações válidas, retornadas pelo procedimento **obtenhaPróximosPassos**. Sendo assim, os nós ancestrais de um nó de ação passado como parâmetro para **atualizaEstadoÁrvore** sempre representarão operadores básicos, uma vez que a execução do procedimento **obtenhaPróximosPassos** já “eliminou” os nós ancestrais que representavam os demais operadores.

A lógica que define como são feitos o percurso na árvore de navegação e a atualização dos estados dos nós de operadores está fundamentada nas regras de transição para os operadores da álgebra de processos, definidas na Seção 2.3 deste texto.

É possível que um determinado passo válido possua mais de um nó representado na árvore de navegação em um dado momento. Por exemplo, uma instância do processo $P = (a.b) + (a.c)$, sendo a , b e c ações previamente definidas, tem como árvore de navegação a ilustrada na Figura 4.4.

Ao iniciar a execução da instância representada pela árvore da Figura 4.4, apenas um passo está habilitado – a ação **a**. No entanto, dois nós de ramos diferentes da árvore de navegação habilitaram esta ação – os nós denominados de **x** e **y** na Figura 4.4. Quando a aplicação indicar que ação **a** foi iniciada e, posteriormente, finalizada, as ações **b** e **c** devem ser habilitadas para execução. Somente após o início da execução de uma das duas ações é possível saber se a ação **a**, executada anteriormente, era a representada pelo nó **x** ou a representada pelo nó **y**.

Figura 4.4: Árvore de navegação do processo $P = (a.b) + (a.c)$

Para tratar de forma adequada a situação descrita, o serviço monitor efetua a chamada do procedimento **atualizaEstadoÁrvore** para cada um dos nós válidos atuais que representam a ação alvo da mudança de estado. Isso garante que nenhum ramo de execução possível seja desconsiderado pelo procedimento **obtenhaPróximosPassos**.

Quando os diferentes nós válidos atuais que representam a ação a ser iniciada forem descendentes de um mesmo nó de paralelismo, como na árvore da Figura 4.5(a), é necessário um tratamento diferenciado.

A Figura 4.5(a) mostra os estados atribuídos aos nós no início da execução de uma instância do processo $P = ((a+b).c) \parallel ((a+c).d)$. A sigla “N” representa o estado “Não iniciado”, “I” representa “Iniciado” e “F”, “Finalizado”.

Inicialmente, há quatro nós ativos que na figura estão destacados por um círculo tracejado. De acordo com a expressão do processo, a ação **a** pode ser executada duas vezes. A Figura 4.5(b), mostra como ficaria os estados dos nós da após o início e fim da execução da ação **a** como passo inicial (caso fosse aplicado o procedimento **atualizaEstadoÁrvore** sobre os dois nós válidos que a representam). É possível

verificar que a árvore da Figura 4.5(b) não expressa o estado em que o sistema deveria se encontrar, pois não permite uma segunda execução da ação *a*.

Figura 4.5: Estados dos nós na árvore de navegação do processo

$$P = ((a+b) \cdot c) \parallel ((a+c) \cdot d)$$

O tratamento de casos de ações representadas por mais de um descendente de um mesmo nó de paralelismo exige que sejam feitas replicações do ramo de paralelismo. A Figura 4.6 mostra a replicação aplicada sobre a árvore da Figura 4.5(a).

Figura 4.6: Exemplo de replicação de ramo em uma árvore de navegação

As replicações dos ramos são unidas por operadores de composição alternativa e conectadas à árvore de navegação em substituição ao nó de paralelismo fonte do conflito. O procedimento **atualizaEstadoÁrvore** é então aplicado somente uma vez em cada ramo replicado, sendo que em cada ramo, ele é aplicado sobre um nó diferente. Por exemplo, no ramo x da Figura 4.6, **atualizaEstadoÁrvore** foi aplicado sobre o nó representante de a número 1, enquanto que no ramo y , foi aplicado sobre o nó número 2. Desta forma, a árvore de navegação após a execução da ação a continua representando o comportamento definido pelo processo $P = ((a + b) . c) \parallel ((a + c) . d)$.

O serviço monitor de execução considera que a execução de uma instância foi finalizada com sucesso quando o estado do nó raiz de sua árvore de navegação estiver assinalado como “Finalizado”.

4.5 Conclusão: Exemplo de execução de plano de navegação

Como conclusão deste capítulo, esta seção retomará o exemplo do processo de *Aquisição de Item de Acervo da Biblioteca do IME – USP*, apresentado na Seção 2.3 e representado pela Figura 2.5 deste texto. O objetivo é ilustrar, por meio de uma simulação de execução, o funcionamento dos algoritmos de obtenção de passos válidos e de atualização de estado de uma árvore de navegação.

Considere que o conjunto de regras e o conjunto de ações pertencentes ao sistema de Aquisição são definidos, respectivamente, como $R = \{r_1\}$ e $A = \{a_1, a_2, a_3, a_4, a_5\}$, sendo que:

- r_1 verifica se os dados do pedido estão completos e se há verba disponível
- a_1 prioriza itens do pedido
- a_2 cota itens do pedido
- a_3 libera pedido para compra
- a_4 registra recebimento da compra
- a_5 registra problema com a compra

A Aquisição pode ser especificada em NPDL pelo processo P tal que:

$$\begin{aligned} \text{SET } P_1 &= a_1 \mid * a_2 + (a_1 \mid * a_2) . P_1 \\ \text{SET } P &= \%r_1 P_1 . a_3 . (a_4 + a_5? * . a_4) \end{aligned}$$

Estas expressões diferem da especificação em álgebra de processos apresentada na Seção 2.3 pelos seguintes aspectos:

1. O passo de verificação do pedido e a indicação de recebimento de verba são tratados como uma regra na NPDL, representando de forma melhor as características do processo de Aquisição. A execução das atividades envolvidas na Aquisição só estará habilitada se o pedido for válido e se houver verba disponível.
2. Na especificação em NPDL, não foi representada a possibilidade de cancelamento do pedido em cada passo do processo de Aquisição. Conforme descrito na especificação do padrão “Caso Cancelável”, na Seção 3.2.6, esta característica está associada ao processo em seu tempo de execução. Na *NavigationPlanTool*, a execução de uma instância de processo pode ser interrompida quando for desejado.
3. As ações de priorização e cotação de pedidos (a_1 e a_2) foram compostas por meio do operador “|*”, em substituição ao operador “.” utilizado na expressão de processo da Seção 2.3. Esta substituição foi feita com o objetivo de exemplificar o mapeamento do operador “|*” em operadores básicos da NPDL. Este mapeamento acontecerá durante a simulação da execução. O termo $a_1 / * a_2$ pode ser repetido um número irrestrito de vezes antes da liberação do pedido para a compra (ação a_3), por isso o termo pertence ao processo recursivo P_1 .
4. Após a liberação do pedido (ação a_5), é possível que haja a ocorrência de problemas na compra. Esses problemas são registrados pelo processo de Aquisição por meio da ação a_5 . Neste caso, utilizou-se o operador de repetição “?*” associado à ação a_5 , para representar a possibilidade de ocorrência de mais de um problema na compra.

A árvore de navegação inicial de uma instância do processo de Aquisição está representada na Figura 4.7(a). À esquerda superior de cada nó há uma letra indicando um dos seguintes estados: “N” – para “Não iniciado”, “I” – para “Iniciado” e “F” – para “Finalizado”. É importante observar que, associada ao nó que representa o processo P_1 na árvore, aparece a regra r_1 . Assim como os nós da árvore, a regra também possui uma indicação de seu estado. Ligado ao nó da ação a_5 , está o operador “?*”, indicando que a ação pode ter sua execução repetida.

Figura 4.7: Estado inicial da árvore de navegação para o processo Aquisição

No início da execução, o procedimento **obtenhaPróximosPassos** é aplicado sobre o nó raiz da árvore, destacado pela linha tracejada na Figura 4.7(a). Durante o percurso na árvore, o procedimento se depara com a regra r_1 e a adiciona ao conjunto de passos atuais válidos (de acordo com a linha 6 do algoritmo de percurso). Como a execução dos demais passos depende da execução do passo condicionado a r_1 , nenhum outro passo pode ser executado inicialmente.

Considerando agora que a regra r_1 seja executada e seu valor de retorno seja “Verdadeiro”, uma nova aplicação do procedimento **obtenhaPróximosPassos** sobre o nó raiz da árvore parará no nó x da Figura 4.7(b). O nó x representa o processo P_1 e, de acordo com a linha 64 do algoritmo de percurso, ele precisa ser substituído por sua árvore de navegação. A Figura 4.8 mostra a árvore após a substituição. A continuação do percurso na árvore detecta a presença do operador “|*” em dois lugares distintos (nós x e y da Figura 4.8). Como especifica a linha 55 do algoritmo de percurso, este operador requer um tratamento especial.

Figura 4.8: Árvore de navegação após a substituição de P_1

Em NPD, um termo da forma $a*b$ é equivalente a $a.b + b.a$. A Figura 4.9 mostra a árvore de navegação após o mapeamento dos operadores “|*”. Após este mapeamento, o procedimento **obtenhaPróximosPassos** está apto a identificar os passos atuais válidos de execução. Como mostra os nós em destaque na Figura 4.9, existem dois passos habilitados para execução na árvore: as ações a_1 e a_2 .

Figura 4.9: Árvore de navegação após o mapeamento dos operadores “|*”

Considerando que a ação a_2 seja executada, tem-se a aplicação do procedimento **atualizaEstadoÁrvore** sobre os nós x e z da Figura 4.9, resultando na árvore da Figura 4.10. Também sobre os nós x e z , o procedimento **obtenhaPróximosPassos** é executado, identificando como único passo atual válido a ação a_1 , representada pelos nós x e y da Figura 4.10.

Figura 4.10: Árvore de navegação após a execução da ação a_2

Após a execução de a_1 , **atualizaEstadoÁrvore** é aplicado sobre os nós x e y da Figura 4.10, resultando na árvore da Figura 4.11. A execução do procedimento **obtenhaPróximosPassos** sobre estes mesmos nós se deparará com o nó y da Figura 4.11, que novamente representa o processo P_1 . O nó é substituído pela árvore de navegação do processo P_1 , resultando na árvore da Figura 4.12. Na árvore da Figura 4.12, os nós x e y representam operadores “ $|*$ ”, que são mapeados em operadores básicos, resultando na árvore da Figura 4.13.

Figura 4.11: Árvore de navegação após a execução da ação a_1

Figura 4.12: Árvore de navegação após a substituição de P_1

Figura 4.13: Árvore de navegação após a substituição dos operadores “|*”

Figura 4.14: Árvore de navegação após a execução da ação a_3

Figura 4.16: Árvore de navegação final do processo de Aquisição

Capítulo 5

Conclusão

Este capítulo apresenta um resumo desta dissertação, suas contribuições e limitações, bem como a indicação de possíveis trabalhos futuros.

5.1 Resumo

A abordagem tradicional para o tratamento de *workflows* é implementá-los como parte da aplicação computacional. As principais desvantagens desta abordagem é que ela torna o gerenciamento do processo rígido e, muitas vezes, não possibilita o reaproveitamento de passos já definidos em outros processos.

O surgimento do paradigma SOC, apresentado em 2003, reforçou a necessidade de separação entre a construção dos processos de negócio e a implementação das atividades que os compõem. As tecnologias de gerenciamento de processos de negócio que foram desenvolvidas para atender a este requisito impulsionaram a criação de diversas linguagens e ferramentas para definição e controle de processos, principalmente baseadas em serviços *web*.

Entre tantas linguagens para a definição de processos de negócio, destacou-se a ausência de um padrão formal como base de representação, capaz de expressar de forma não ambígua a semântica associada às construções existentes nestas linguagens. Os formalismos apontados como candidatos naturais a este papel são as redes de Petri e as álgebras de processos. Trabalhos como [2], [6] e [37] descrevem o uso destes formalismos na especificação de processos de negócio.

Aalst, Hofstede, Kiepuszewski e Barros, em [4], identificaram padrões para o controle de fluxos em *workflows*. Um dos objetivos dos autores ao definir os padrões foi o de estabelecer parâmetros para a comparação entre as linguagens de definição de processos de negócio, já que esta comparação não pode ser realizada em termos de expressividade.

Esta dissertação de mestrado apresenta uma linguagem implementada como uma extensão da SQL que habilita um gerenciador de banco de dados relacional à criação e manutenção de processos de negócio. Esta linguagem, denominada NPDL, utiliza operadores da álgebra de processos e outros operadores adicionais para representar o comportamento dos padrões de controle de fluxo.

O interpretador da NPDL é parte de uma biblioteca de funções, a *NavigationPlanTool*, desenvolvida para apoiar também as fases de instanciação e execução de processos de negócio.

5.2 Contribuições

A escolha da álgebra de processos como arcabouço formal para a NPDL, devido à sua forma algébrica e puramente textual, possibilitou a criação de uma representação de processos mais apropriada para a manipulação computacional e para o armazenamento em uma estrutura relacional de dados. Além disso, a característica composicional da álgebra de processos, sem reduzir o seu potencial de análise, habilita a composição de grandes processos a partir de processos menores. Isso constitui um aspecto importante para processos de negócio, nos quais é comum a ocorrência de reutilização de definições.

A maioria das extensões de álgebra de processos não apóia a modelagem de aspectos associados ao processo em seu tempo de execução. Isso compromete o uso de álgebra de processos na especificação dos padrões de controle de fluxo **Múltiplas Instâncias com Conhecimento Prévio em Tempo de Execução**, **Escolha Exclusiva**, e consequentemente **Escolha Múltipla**, **Junção Sincronizada**, **Junção Múltipla** e **Discriminador**, conforme discutido na Seção 3.2 deste texto.

Em NDPL é possível tratar uma escolha exclusiva por meio dos operadores “%” e “%!”, que habilitam ou desabilitam a execução de uma atividade de acordo com uma

condição, ou seja, uma função atômica *booleana* avaliada em tempo de execução do processo. O padrão **Múltiplas Instâncias com Conhecimento Prévio em Tempo de Execução** é representado na NPDL pelo operador de repetição “ $?f$ ”, sendo f uma função inteira, também avaliada em tempo de execução do processo.

Com o objetivo de facilitar a tarefa de definição de processos, foram introduzidos na NPDL os operadores de composição múltipla “ $\&$ ”, composição discriminatória “ $\wedge$ ”, composição paralela entrelaçada “ $|*$ ” e repetição “ $?$ ” que representam, respectivamente, os padrões **Junção Múltipla**, **Discriminador**, **Roteamento Paralelo Entrelaçado** e todos os de **Múltiplas Instâncias**. Estes operadores não pertencem a ACP, extensão da álgebra de processos utilizada neste trabalho. Entretanto, eles podem ser representados pela combinação dos operadores básicos.

A Seção 3.3 deste texto mostrou um exemplo de especificação completa em NPDL de um sistema de pequeno porte. Por meio desse exemplo é possível verificar que a NPDL pode ser aplicada de modo satisfatório na definição de processos de negócio, superando algumas limitações e dificuldades da álgebra de processos aplicada a este fim.

A utilização das árvores de navegação, construídas a partir das expressões algébricas que definem processos em NPDL, facilitou o controle de execução de instâncias feito pela *NavigationPlanTool*. Em uma árvore de navegação, os nós folhas representam os passos que integram a definição do processo, enquanto os nós internos representam os operadores que determinam a forma de encadeamento desses passos. As regras de transição para os operadores da ACP são utilizadas no algoritmo que percorre a árvore para definir os passos atualmente habilitados para execução e no algoritmo de atualização do estado da instância de processo após o início ou término da execução de um passo. Processos recursivos e casos especiais de paralelismo são tratados por meio de replicações de ramos da árvore, conforme descreve a Seção 4.4.1 deste texto.

O armazenamento dos dados de definição dos processos e dos dados de execução das instâncias em um banco de dados relacional torna a *NavigationPlanTool* uma ferramenta escalável, pois permite a recuperação eficiente do estado de execução das instâncias.

A *NavigationPlanTool* está implementada na forma de uma biblioteca de funções. Por isso, na abordagem proposta por este trabalho, não há gargalos no gerenciamento dos processos de negócio porque não há um mecanismo centralizador.

Além disso, as definições de processos podem ser compartilhadas entre diferentes aplicações que utilizam a *NavigationPlanTool*, por meio do acesso a um banco de dados comum a essas aplicações. Esta característica é muito importante em ambientes cooperativos, além de possibilitar que o aspecto composicional das expressões algébricas de processos seja bastante explorado.

A *NavigationPlanTool* foi aceita para a Sessão de Demos do XX Simpósio Brasileiro de Banco de Dados, que ocorreu em Outubro de 2005 [15].

5.3 Limitações

Em álgebra de processos e em NPDL, os estados não são representados explicitamente. Esta característica impossibilita a criação de uma regra genérica para a especificação do padrão **Marco** (Seção 3.2.5 deste texto), requerendo, para cada caso de uso, um mapeamento particular.

Embora atualmente a NPDL seja capaz de representar todos os padrões de controle de fluxo (como demonstrado na Seção 3.2), ela tem uma deficiência que pode comprometer o seu uso em alguns tipos de processos de negócio: a incapacidade de representar explicitamente, nas definições de processos, aspectos relacionados a dados, conforme detalha o item 5.4.1 na próxima seção deste texto.

5.4 Trabalhos Futuros

Finalizando a conclusão deste trabalho, quatro possíveis futuros trabalhos são apresentados. As sugestões 5.4.1 e 5.4.2, pois já estão sendo desenvolvidas pelo grupo de pesquisa em banco de dados do IME-USP.

5.4.1 Representação em NPDL dos dados envolvidos nos processos

A versão atual da NPDL não provê mecanismos para a especificação dos dados de entrada e saída dos processos definidos. Deste modo, a troca de dados entre os

processos ocorre segundo a abordagem conhecida como *blackboard*. Nesta abordagem, os processos compartilham uma área de memória (que pode, inclusive, ser o próprio banco de dados utilizado pela ferramenta *NavigationPlanTool*). Os processos obtêm desta memória compartilhada o valor dos seus parâmetros de entrada no início de suas execuções e nela depositam os seus valores de saída no final de suas execuções.

Essa abordagem para o tratamento de dados é bastante simplista. Assim como ocorreu com o controle de fluxo, Russell, Hofstede, Edmond e Aalst definiram em [38] padrões de dados para *workflows*. Existe uma série de conceitos relacionados à representação e à utilização de dados em sistemas de *workflows*. Esses conceitos especificam a forma como os dados podem ser empregados dentro dos processos e caracterizam a interação de elementos de dados com as outras construções presentes no *workflow* ou no ambiente em que ele é executado.

Uma outra possível abordagem é a utilização das propriedades da μ CRL (*micro Commom Representation Language*), que é uma extensão minimal da ACP com tipos de dados abstratos equacionais. Esta extensão foi definida por Groote e Reniers em [26]. Em μ CRL, variáveis de processos e ações podem ser parametrizadas com dados.

Agregar à NPD L os padrões de dados ou propriedades da μ CRL a habilitaria para a representação de processos mais completa com a inclusão de dados.

5.4.2 Associação da NPD L a ferramentas gráficas de modelagem

Expressões típicas de processos em NPD L são bastante simples. Entretanto, a identificação dos padrões de controle nos processos e a sua tradução para expressões em NPD L nem sempre são tarefas triviais, requerendo do projetista experiência nesse tipo de modelagem. Aliar o uso da NPD L a uma ferramenta gráfica para a definição de processos auxiliaria a etapa de projeto, uma vez que descrições gráficas de processos são mais intuitivas que as textuais. Estudos como [13] mostram que é possível combinar os benefícios das redes de Petri e da álgebra de processos em um modelo unificado.

5.4.3 Desenvolvimento de mecanismos de análise de processos para as expressões em NPD

A subárea de análise de processos é o atual enfoque da pesquisa no Gerenciamento de Processos de Negócio. A popularidade de termos como *Process Mining* (mineração de processos) e BPI – *Business Process Intelligence* (Análise de Processos de Negócio) são resultados dessa tendência. Enquanto o BPI prioriza a análise, predição e otimização de processos [18], as técnicas de mineração de processos se baseiam em *logs* de execução para identificar relações de dependência entre as atividades que constituem os processos. A pesquisa de Agrawal, Gunopulos e Leymann [7] é a principal referência para os trabalhos sobre mineração de processos. Aalst e Weijters, em [3], realizaram uma comparação entre as diferentes técnicas existentes.

Os *logs* de execução das instâncias controladas pela *NavigationPlanTool* são “pedaços” das expressões algébricas que definem seus processos de origem. Análises envolvendo tanto dados do *log*, quanto dados das definições dos processos, explorando as propriedades algébricas das expressões, podem resultar em informações que auxiliem na reestruturação do processo.

5.4.4 “Reconfiguração” de expressões de processos

A “reconfiguração” envolve a reescrita das expressões algébricas que representam os processos, com o objetivo de simplificá-las ou de atingir representações “normalizadas”, ou seja, livres de pontos inatingíveis ou redundantes na definição.

Uma abordagem inovadora para o tratamento deste problema foi proposta por Pankratius e Stucky em [36]. Os autores definiram uma notação baseada nos operadores da álgebra relacional para a composição de **módulos de workflows**. Módulos de *workflows* podem ser definidos como conjuntos de atividades inter-relacionadas. Os autores também criaram uma definição para normalização de *workflows* inspirada no conceito de dependências funcionais e formas normais de banco de dados.

Referências Bibliográficas¹³

- [1] AALST, W. M. P van der. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype. **BPTrends**, v. 3, n. 5, p. 1-11, mai. 2005.
- [2] AALST, W. M. P. van der; HOFSTEDE, A. H. M. ter. YAWL: Yet Another Workflow Language. **Information Systems**, v. 30, n. 4, p. 245-275. 2005.
- [3] AALST, W. M. P van der; WEIJTERS, T. Process mining: a research agenda. **Computers in Industry**, v. 53, n. 1, p. 231-244. 2004.
- [4] AALST, W. M. P. van der; HOFSTEDE, A. H. M. ter; KIEPUSZEWSKI, B.; BARROS, A. P. Workflow Patterns. **Distributed and Parallel Databases**, v. 14, n. 1, p. 5-51, jul. 2003.
- [5] AALST, W. M. P. van der; HOFSTEDE, A. H. M. ter; WESKE, M. Business Process Management: A Survey. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, 1., 2003, Eindhoven. **Proceedings...** Berlin: Springer, 2003. p. 1-12.
- [6] AALST, W. M. P van der. The application of petri nets to workflow management. **The Journal of Circuits, Systems and Computers**, v. 8, n. 1, p. 1-53. 1998.

¹³ De acordo com:

Associação Brasileira de Normas Técnicas. **NBR 6023**: informação e documentação: referências: elaboração. Rio de Janeiro, 2002.

-
- [7] AGRAWAL, R.; GUNOPULOS, D.; LEYMAN, F. Mining process models from workflow logs. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, 6., 1998, Valencia. **Proceedings...** Berlin: Springer, 1998. p. 469-483.
- [8] ANDREWS, T. et al. **Business Process Execution Language for Web Services**, Version 1.1. Disponível em: <<ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf>>. Acesso em: 17 jul 2006.
- [9] BAETEN, J. C. M.; BASTEN, T. Partial-Order Process Algebra (and its Relation to Petri-Nets). In: BERGSTRÄ, J. A.; PONSE, A.; SMOLKA, S. A. **Handbook of process algebra**. Amsterdã: Elsevier Science Inc., 2001. p. 769-872.
- [10] BASTEN, T. **In Terms of Net: System Design with Petri Nets and Process Algebra**. 1998. 237 f. Tese (Doutorado) – Institute for Programming Research and Algorithmics, Eindhoven University of Technology, Eindhoven, 1998.
- [11] BERGSTRÄ, J. A.; PONSE, A.; SMOLKA, S. A. **Handbook of process algebra**. Amsterdã: Elsevier Science Inc., 2001. 1356 p.
- [12] BERGSTRÄ, J. A.; KLOP, J. W. Process Algebra for synchronous communication, **Information and Control**, v. 60, n. 1-3, p. 109-137. 2001.
- [13] BEST, E.; DEVILLERS, R.; KOUTNY, K. A unified model for nets and process algebras. In: BERGSTRÄ, J. A.; PONSE, A.; SMOLKA, S.A. **Handbook of process algebra**. Amsterdã: Elsevier Science Inc., 2001. p. 873-944.
- [14] BRAGHETTO, K. R.; TAKAI, O. K.; FERREIRA, J. E.; PU, C. **Controlling Processes in Relational Data Model with NPD and NavigationPlanTool**. São Paulo: IME-USP. (Relatório técnico) Disponível em: <<http://www.ime.usp.br/~jef/kelly>>. Acesso em: 17 jul 2006.

[15] BRAGHETTO, K. R.; TAKAI, O. K.; FERREIRA, J. E.; PU, C. NavigationPlanTool: Uma Ferramenta para o Controle de Processos no Modelo de Dados Relacional. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS (SESSÃO DE DEMOS), 20., 2005, Uberlândia. Disponível em: <<http://www.sbbd-sbes2005.ufu.br/arquivos/NavPlanTool.pdf>>. Acesso em: 17 jul 2006.

[16] **BUSINESS Process Management Initiative**. Disponível em: <<http://www.bpmi.org>>. Acesso em: 17 jul 2006.

[17] **BUSINESS Process Modeling Notation**. Disponível em: <<http://www.bpmn.org>>. Acesso em: 17 jul 2006.

[18] CASATI, F.; DAYAL, U.; GRIGORI, D.; SHAN, M. C. Improving Business Process Quality through Exception – Understanding, Prediction, and Prevention. In: INTERNATIONAL CONFERENCE IN VERY LARGE DATA BASES, 27., 2001, Roma. **Proceedings...** Roma: Morgan Kaufmann, 2001. p. 159-168.

[19] **EXTENSIBLE Markup Language (XML) 1.0**. Disponível em: <<http://www.w3.org/TR/2004/REC-xml-20040204>>. Acesso em: 17 jul 2006.

[20] FERREIRA, J. E.; TAKAI, O. K.; PU, C. Integration of Business Processes with Autonomous Information Systems: A Case Study in Government Services. In: INTERNATIONAL IEEE CONFERENCE ON E-COMMERCE TECHNOLOGY, 7., 2005, Monique. **Proceedings...** Munique: IEE Conference Proceedings, 2005. p. 471-478.

[21] FERREIRA, J. E.; TAKAI, O. K.; PU, C. Integration of Collaborative Information System in Internet Applications using RiverFish Architecture. Aceito para INTERNATIONAL CONFERENCE ON COLLABORATIVE COMPUTING: NETWORKING, APPLICATIONS AND WORK SHARING, 1., 2005, San Jose.

-
- [22] FERREIRA, J.E.; FINGER, M. Controle de Concorrência e Distribuição de Dados: Teoria Clássica, suas Limitações e Extensões Modernas. In: ESCOLA DE COMPUTAÇÃO, 12., 2000, São Paulo. **Livro-texto...** São Paulo: Instituto de Matemática e Estatística, USP, 2000. p. 188.
- [23] FOKKINK, W. J. **Introduction to Process Algebra**: Texts in Theoretical Computer Science. Berlin: Springer-Verlag New York, Inc., 2000. 163 p.
- [24] GEORGAKOPOULOS, D.; PAPAOGLOU, M. P. Service Oriented Computing. **Communications of the ACM**, v. 26, n. 10, p. 25-28, out. 2003.
- [25] GLABBEEK, R. J. van. The linear-time – branching time spectrum I. The semantics of concrete, sequential processes. In: BERGSTR, J. A., PONSE, A., SMOLKA, S. A. **Handbook of process algebra**. Amsterdã: Elsevier Science Inc., 2001. p. 3-99.
- [26] GROOTE, J. F.; RENIERS, M. A. Algebraic process verification. In: BERGSTR, J. A., PONSE, A., SMOLKA, S. A. **Handbook of process algebra**. Amsterdã: Elsevier Science Inc., 2001. p. 1151-1208.
- [27] HOARE, C. A. R. Communicating sequential processes. **Communications of the ACM**, v. 21, n. 8, p. 666-677, ago.1978.
- [28] HOLLINSWORTH, D. **The Workflow Reference Model**. Winchester: Workflow Management Coalition, 1995. 55 p. (Relatório técnico, WPMC-TC00-1003)
- [29] **JAVA 2 Platform Standard Edition - J2SE 5.0**. Disponível em: <<http://java.sun.com/j2se/1.5.0>>. Acesso em: 17 jul 2006.
- [30] LEYMAN, F.; ROLLER, D.; SCHIMIDT, M. T. Web services and business process management. **IBM Systems Journal**, v. 41, n. 2, p. 198-211. 2002.

-
- [31] LEYMANN, F. **Web Services Flow Language (WSFL)**, Version 1.0. Disponível em: <<http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>>. Acesso em: 17 jul 2006.
- [32] MACIEL, P. R. M.; LINS, R. D.; CUNHA, P. R. F. Introdução às Redes de Petri e Aplicações. In: ESCOLA DE COMPUTAÇÃO, 10., 1996, Campinas. **Livro-texto...** Campinas: Instituto de Computação, UNICAMP, 1996. p. 197.
- [33] MILNER, R. **A Calculus of Communicating Systems**. Secaucus: Springer-Verlag New York, Inc., 1982. 260 p.
- [34] MURATA, T. Petri Nets: Properties, Analysis and Applications. **Proceedings of The IEEE**, v. 77, n. 4, p. 541-580, abr. 1989.
- [35] PADUA, S. I. D.; SILVA, A. R. Y.; PORTO, A. J. V.; INAMASU, R. I. The potencial of Petri nets in modeling and analysis of workflow. **Gestão & Produção**, v. 11, n. 1, p.109-119, jan./abr. 2004.
- [36] PANKRATIUS, V.; STUCKY, W. A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In: ASIA-PACIFIC CONFERENCE ON CONCEPTUAL MODELING, 2., 2005, Newcastle. **Proceedings...** Newcastle: Australian Computer Society Inc., 2005. p. 79-88.
- [37] PUHLMANN, F.; WESKE, M. Using the Pi-Calculus for Formalizing Workflow Patterns. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, 3., 2005, Nancy. **Proceedings...** Berlin: Springer. 2005. p. 153-168.
- [38] RUSSELL, N.; HOFSTEDE, A. H. M. ter; EDMOND, D.; AALST, W. M. P. van der. **Workflow data patterns**. Brisbane: Queensland University of Technology, 2004. 75 p. (Relatório técnico, FIT-TR-2004-01)

[39] TEOREY, T.; LIGHTSTONE, S.; NADEAU, T. **Database Modeling and Design: Logical Design**. 4 ed. San Francisco: Morgan Kaufmann Publishers, 2006. 296 p.

[40] THATTE, S. **XLANG Web Services for Business Process Design**. Disponível em: <http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm>. Acesso em: 17 jul 2006.

[41] **WORKFLOW Management Coalition, Terminology & Glossary**. Winchester: Workflow Management Coalition, 1999. 65 p. (Relatório técnico, WFMC-TC-1011) Disponível em: <http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf>. Acesso em: 17 jul 2006.

[42] **WORKFLOW Patterns**. Disponível em: <<http://www.workflowpatterns.com>>. Acesso em: 17 jul 2006.

Apêndice A

A Sintaxe da *Navigation Plan Definition Language*

Considere que a gramática da linguagem NPDL é definida por (N, T, P, S) , sendo N o **conjunto de símbolos de variáveis** (ou símbolos não-terminais); T o **conjunto dos símbolos terminais**; P o **conjunto das regras de produção da gramática** e S o **símbolo inicial** de N .

A NPDL obedece às seguintes propriedades:

1) $S \in N$

2) $N = \{ \text{action-description, add-action-execution-call, add-function-execution-call, add-process-service-description, add-rule-execution-call, choice-operator-sign, command, condition, deadlock-symbol, delete-action, delete-function, delete-process, delete-rule, discriminator-operator-sign, execution-call, factor, function-description, function-limited-repetition, interleaved-parallel-operator-sign, left-parentheses, limited-repetition, list-actions, list-functions, list-navigation-plan, list-processes, list-rules, multi-merge-sign, new-action, new-function, new-process, new-rule, parallel-operator-sign, process, process-description, process-expression, right-parentheses, S, service-description, sequential-operator-sign, term, rule-description, unlimited-repetition} \}$

3) $T = \{ \text{ACTION, ACTIONS, ADD, CALL, CREATE, DESCRIPTION, DROP, EXECUTION, FROM, FUNCTION, FUNCTIONS, NAVIGATION, PLAN, PROCESS, PROCESSES, RULE, RULES, SELECT, SERVICE, SET, \n, (,), =, ., +, ^, ||, |*, \#, *, ?, \&, \%, \%!} \}$

4) O conjunto P é gerado a partir das regras de produção da Tabela A.1, descritas no formato BNF (*Backus-Naur Form*).

Tabela A.1: Regras de produção da NPD

```

<S> ::= (<command>? \n )+
<command> ::= <new-action>
    | <new-function>
    | <new-process>
    | <new-rule>
    | <add-action-execution-call>
    | <add-function-execution-call>
    | <add-process-service-description>
    | <add-rule-execution-call>
    | <delete-action>
    | <delete-function>
    | <delete-process>
    | <delete-rule>
    | <process-expression>
    | <list-actions>
    | <list-functions>
    | <list-rules>
    | <list-processes>
    | <list-navigation-plan>
<new-process> ::= CREATE PROCESS <process-description>
    [ <service-description> ]
<new-action> ::= CREATE ACTION <action-description>
    [ <execution-call> ]
<new-function> ::= CREATE FUNCTION <function-description>
    [ <execution-call> ]
<new-rule> ::= CREATE RULE <rule-description>
    [ <execution-call> ]
<add-process-service-description> ::= ADD <process-description>
    SERVICE DESCRIPTION <service-description>
<add-action-execution-call> ::= ADD <action-description>
    EXECUTION CALL <execution-call>
<add-function-execution-call> ::= ADD <function-description>
    EXECUTION CALL <execution-call>
<add-rule-execution-call> ::= ADD <rule-description>
    EXECUTION CALL <execution-call>
<process-expression> ::= SET <process-description> = <process>
<delete-process> ::= DROP PROCESS <process-description>
<delete-action> ::= DROP ACTION <action-description>
<delete-function> ::= DROP FUNCTION <function-description>
<delete-rule> ::= DROP RULE <rule-description>
<list-processes> ::= SELECT PROCESSES
<list-actions> ::= SELECT ACTIONS

```

```

<list-functions> ::= SELECT FUNCTIONS
<list-rules> ::= SELECT RULES
<list-navigation-plan> ::= SELECT NAVIGATION PLAN FROM PROCESS
    <process-description>
<left-parentheses> ::= (
<right-parentheses> ::= )
<choice-operator-sign> ::= +
<sequential-operator-sign> ::= .
<parallel-operator-sign> ::= ||
<interleaved-parallel-operator-sign> ::= |*
<multi-merge-operator-sign> ::= &
<discriminator-operator-sign> ::= ^
<deadlock-symbol> ::= #
<unlimited-repetition> := ?*
<limited-repetition> := ? <unsigned_integer>14
<function-limited-repetition> := ? <function>
<condition> := % <rule>
<not-condition> := %! <rule>
<process> ::= <term>
    | <process> <choice-operator-sign> <term>
<term> ::= <subterm>
    | <term> <interleaved-parallel-operator-sign> <subterm>
<subterm> ::= <prefactor>
    | <subterm> <parallel-operator-sign> <prefactor>
<prefactor> ::= <factor>
    | <prefactor> <sequential-operator-sign> <factor>
<factor> ::= <subfactor>
    | <factor> <multi-merge-operator-sign> <subfactor>
    | <factor> <discriminator-operator-sign> <subfactor>
<subfactor> ::= <action>
    | <process-description>
    | <process>
    | <deadlock-symbol>
    | <left-parentheses> <process> <right-parentheses>
    | <subfactor><unlimited-repetition>
    | <subfactor><limited-repetition>
    | <subfactor><function-limited-repetition>
    | <condition><subfactor>
    | <not-condition><subfactor>
<action> ::= <action-description>

```

¹⁴ <unsigned_integer> está definido na especificação BNF do ANSI SQL92.

```
<rule> ::= <rule-description>
<function> ::= <function-description>
<action-description> ::= <identifier>15
<rule-description> ::= <identifier>
<process-description> ::= <identifier>
<service-description> ::= <character_string_literal>16
<execution-call> ::= <character_string_literal>
```

¹⁵ <identifier> está definido na especificação BNF do ANSI SQL92.

¹⁶ <character_string_literal> está definido na especificação BNF do ANSI SQL92.

Apêndice B

A Estrutura Relacional de Dados da *NavigationPlanTool*

Quando uma conexão para a execução de comandos NPDL é estabelecida pela primeira vez com um determinado banco de dados, o serviço interpretador da NPDL cria no banco as relações utilizadas pela *NavigationPlanTool* para a manutenção de processos, instâncias e seus dados de execução. A ferramenta utiliza cinco relações; a estrutura destas relações e exemplos dos dados armazenados são mostrados nas Tabelas de B.2 a B.8.

Os comandos SQL executados pelo serviço interpretador NPDL sobre o banco de dados para a criação das relações são os listados na Tabela B.1.

Tabela B.1: Comandos SQL para a criação das relações

```
/* Criação da tabela de Processos */
CREATE TABLE np_process(
    process_id DECIMAL(15) NOT NULL,
    process_description VARCHAR NOT NULL,
    service_description VARCHAR NULL,
    CONSTRAINT pk_np_process PRIMARY KEY (process_id));
CREATE UNIQUE INDEX np_process_unique_index
    ON np_process(process_description);
GRANT ALL PRIVILEGES ON np_process TO PUBLIC;

/* Criação da tabela de Passos */
CREATE TABLE np_step (
    step_id DECIMAL(15) NOT NULL,
    step_description VARCHAR NOT NULL,
    step_type CHAR NOT NULL,
    execution_call VARCHAR NULL,
```

```

        CONSTRAINT pk_np_step PRIMARY KEY (step_id));
CREATE UNIQUE INDEX np_step_unique_index
    ON np_step (step_description);
GRANT ALL PRIVILEGES ON np_step TO PUBLIC;

/* Criação da tabela de Plano de Navegação */
CREATE TABLE np_navigation_plan (
    navigation_plan_id DECIMAL(15) NOT NULL,
    process_id DECIMAL(15) NOT NULL,
    component_type CHAR(1) NOT NULL,
    component_id DECIMAL(15) NULL,
    CONSTRAINT pk_np_navigation_plan PRIMARY KEY
        (navigation_plan_id),
    CONSTRAINT fk_np_navigation_plan FOREIGN KEY
        (process_id) REFERENCES np_process (process_id));
CREATE UNIQUE INDEX np_navigation_plan_unique_index ON
    np_navigation_plan (navigation_plan_id);
GRANT ALL PRIVILEGES ON np_navigation_plan TO PUBLIC;

/* Criação da tabela de Instâncias */
CREATE TABLE np_requested_process (
    instance_id DECIMAL(15) NOT NULL,
    process_id DECIMAL(15) NOT NULL,
    user_name VARCHAR NOT NULL,
    request_date DATE NULL,
    request_time TIME NULL,
    CONSTRAINT pk_np_requested_process PRIMARY KEY (instance_id),
    CONSTRAINT fk_np_requested_process FOREIGN KEY
        (process_id) REFERENCES np_process (process_id));
CREATE UNIQUE INDEX np_requested_process_unique_index ON
    np_requested_process (instance_id);
GRANT ALL PRIVILEGES ON np_requested_process TO PUBLIC;

/* Criação da tabela de Log de execução de instâncias */
CREATE TABLE np_instance_log (
    instance_log_id DECIMAL(15) NOT NULL,
    instance_id DECIMAL(15) NOT NULL,
    component_id DECIMAL(15) NOT NULL,
    returned_value VARCHAR NULL,
    execution_status CHAR(1) NOT NULL,
    initial_date DATE NOT NULL,
    initial_time TIME NOT NULL,
    final_date DATE NULL,

```

```

    final_time TIME NULL,
    CONSTRAINT pk_np_instance_log PRIMARY KEY (instance_log_id),
    CONSTRAINT fk_np_instance_log FOREIGN KEY (instance_id)
        REFERENCES np_requested_process (instance_id),
    CONSTRAINT fk2_np_instance_log FOREIGN KEY (component_id)
        REFERENCES np_step (step_id));
CREATE UNIQUE INDEX np_instance_log_unique_index ON
    np_instance_log (instance_log_id);
GRANT ALL PRIVILEGES ON np_instance_log TO PUBLIC;

/* Insere na tabela de Passos os operadores existentes na NPD L */
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (1, 'O', '.');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (2, 'O', '+');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (3, 'O', '||');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (4, 'O', '|*');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (5, 'O', '%');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (6, 'O', '%!');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (7, 'O', '&');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (8, 'O', '^');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (9, 'O', '?*');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (10, 'O', '?F');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (11, 'O', '?N');
INSERT INTO np_step (step_id, step_type, step_description)
    VALUES (12, 'O', '#');

```

Os comandos da Tabela B.1 utilizam apenas a sintaxe do SQL padrão, para que possam ser aplicados sobre bancos mantidos por qualquer SGBDR.

As relações criadas pela *NavigationPlanTool* possuem em seu nome o prefixo “np_”, para diferenciá-las das demais relações que o banco de dados acessado possa conter.

A relação **np_process**, representada na Tabela B.2, armazena o nome de um processo (coluna **process_description**) e uma descrição da sua funcionalidade (coluna **service_description**). Dados de ações, como nome (coluna **step_description**) e instrução para execução (coluna **execution_call**), são mantidos na relação **np_step**, representada na Tabela B.3.

A relação **np_step** também mantém dados dos operadores da NPDL, regras (utilizadas nos operadores “%” e “%!”), funções (utilizadas no operador “?”) e números (também utilizados no operador “?”). Regras e funções, assim como as ações, possuem valor para a coluna **execution_call**; operadores e números possuirão sempre NULL como valor para a coluna.

Um registro da relação **np_step** pode assumir em sua coluna **step_type** um dos valores indicados na Tabela B.4.

Tabela B.2: Relação **np_process**

process_id	process_description	service_description
1	PPagamento	Controla o processo de pagamento...
2	PValidacao	Valida os dados do cliente...
3	PEstoque	Controla estoque...

Tabela B.3: Relação **np_step**

step_id	step_description	step_type	execution_call
1	.	O	NULL
2	+	O	NULL
3		O	NULL
4	*	O	NULL
5	%	O	NULL
6	%!	O	NULL
7	&	O	NULL
...	...	...	...
20	A1	A	EnviaForm.exe
21	A2	A	ProcessaForm.exe
22	R1	R	PossuiCredito.exe
23	F1	F	NumDeltens.exe
24	5	N	NULL
25	A3	A	CancelaRequisicao.exe

Tabela B.4: Tipos possíveis de passos

Valor	Representando um(a)...
A	Ação
R	Regra (função que retorna valor lógico)

F	Função que retorna um inteiro
N	Número inteiro
O	Operador

De acordo com a estrutura definida para as relações **np_process** e **np_step**, a tradução dos comandos NPDL

```
CREATE PROCESS ProcExemplo;
ADD ProcExemplo SERVICE_DESCRIPTION 'Processo exemplo...';
CREATE ACTION AcaoExemplo;
ADD AcaoExemplo EXECUTION_CALL 'AcaoExemplo.exe';
CREATE RULE RegraExemplo;
ADD RegraExemplo EXECUTION_CALL 'RegraExemplo.exe';
```

para SQL é:

```
INSERT INTO np_process (process_id, process_description)
VALUES (4, 'ProcExemplo');
UPDATE np_process
SET service_description = 'Processo exemplo...'
WHERE process_id = 4;
INSERT INTO np_step (step_id, step_description, step_type)
VALUES (26, 'AcaoExemplo', 'A');
UPDATE np_step SET execution_call = 'AcaoExemplo.exe'
WHERE step_id = 26;
INSERT INTO np_step (step_id, step_description, step_type)
VALUES (27, 'RegraExemplo', 'R');
UPDATE np_step SET execution_call = 'RegraExemplo.exe'
WHERE step_id = 27;
```

A relação **np_navigation_plan**, da Tabela B.5, armazena os dados do plano de navegação dos processos. Em NPDL, o plano de navegação de um processo é definido por uma expressão algébrica que pode envolver ações, operadores, regras, funções, números e outros processos previamente definidos. A relação **np_navigation_plan** mantém os relacionamentos entre processos e passos por meio das colunas **process_id** e **component_id**. Como o plano de navegação de um processo pode ser recursivo ou envolver outros processos, a coluna **component_id** pode referenciar registros tanto da relação **np_step** quanto da relação **np_process**. A coluna **component_type** indica qual

relação o valor do campo **component_id** referencia. Um registro de **np_navigation_plan** pode assumir em sua coluna **component_type** um dos valores indicados na Tabela B.6.

Tabela B.5: Relação **np_navigation_plan**

navigation_plan_id	process_id	component_id	component_type
1	1	1	S
2	1	2	S
3	1	20	S
4	1	6	S
5	1	22	S
6	1	21	S
7	1	25	S
8	3	1	S
9	3	1	P
10	3	19	S

Tabela B.6: Tipos possíveis de componentes

Valor	Para component_id referenciando relação...
S	np_step (Tabela 2)
P	np_process (Tabela 1)

Na relação **np_navigation_plan**, as expressões algébricas dos processos ficam armazenadas em notação prefixa.

A tradução do comando NPDNL de definição de plano de navegação do processo “ProcExemplo”

```
SET ProcTest = A2 . (%R1 AcaoExemplo + %!R1 PEstoque);
```

resulta nos seguintes comandos SQL (considerando como dados já cadastrados os relacionados nas Tabelas B.2, B.3 e B.5):

```
/* A expressão do processo em notação prefixa é:
   ProcTest = . A2 + %R1 AcaoExemplo %!R1 PEstoque
*/
INSERT INTO np_navigation_plan (navigation_plan_id,
                                process_id,component_id, component_type)
VALUES (11, 4, 1, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
                                process_id,component_id, component_type)
VALUES (12, 4, 21, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
```

```

        process_id,component_id, component_type)
VALUES (13, 4, 2, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
        process_id,component_id, component_type)
VALUES (14, 4, 5, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
        process_id,component_id, component_type)
VALUES (15, 4, 22, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
        process_id,component_id, component_type)
VALUES (16, 4, 26, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
        process_id,component_id, component_type)
VALUES (17, 4, 6, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
        process_id,component_id, component_type)
VALUES (16, 4, 22, 'S');
INSERT INTO np_navigation_plan (navigation_plan_id,
        process_id,component_id, component_type)
VALUES (17, 4, 3, 'P');

```

A relação **np_requested_process**, ilustrada pela Tabela B.7, representa as instâncias de processos. Cada instância está associada ao seu processo (coluna **process_id**), ao usuário que a solicitou (coluna **user_name**) e à data da requisição (colunas **request_date** e **request_time**).

Tabela B.7: Relação **np_required_process**

instance_id	processo_id	request_date	request_time	user_name
1	3	12/05/2005	22:45:23	user_6
2	3	06/09/2005	17:38:22	user_2
3	1	18/12/2005	09:05:58	user_9

Tabela B.8: Relação **np_instance_log**

instance_log_id	instance_id	component_id	execution_status	initial_date	initial_time	final_date	final_time
1	1	20	F	18/12/2005	10:03:51	18/12/2005	10:03:54
2	1	22	F	18/12/2005	10:05:37	18/12/2005	10:05:38
3	3	15	F	20/12/2005	18:29:17	20/12/2005	18:29:18
4	3	32	F	20/12/2005	18:31:05	20/12/2005	18:31:09
5	3	17	I	21/12/2005	09:46:07		

Cada registro da relação **np_instance_log**, como mostra a Tabela B.8, representa a execução de um passo em uma determinada instância de processo. A coluna **instance_id** indica a instância à qual a execução está associada, **component_id** indica o passo executado e **execution_status** mantém o estado da execução (iniciada, finalizada ou cancelada) do passo. Além disso, um registro do *log* também possui informações sobre a data e hora de início e término da execução e o seu valor de retorno (utilizado no caso da execução de regras ou funções).

Um registro da relação **np_instance_log** pode assumir como valor para a coluna **execution_status** um dos valores descritos na Tabela B.9.

Tabela B.9: Possíveis estados para passos

Valor	Representando um passo...
I	Iniciado, mas ainda não finalizado
F	Finalizado
C	Cancelado

É importante ressaltar que a estrutura relacional de dados utilizada pela *NavigationPlanTool* não precisa ser mantida em um banco de dados de acesso exclusivo à ferramenta. A prática mais comum é manter esta estrutura no banco que armazena os dados da aplicação que a utiliza ou em um banco de dados acessado por várias aplicações, com o objetivo de compartilhar as definições de processos.

Apêndice C

Síntese dos Padrões de Controle de Fluxo em NPD

As Figuras C.1 e C.2 sintetizam a especificação dos padrões de controle de fluxo em NPD. Quando possível, as figuras trazem uma representação gráfica dos padrões.

Figura C.1: Padrões de controle de fluxo em NPD (Parte 1)

Não há uma representação gráfica para os padrões que envolvem múltiplas instâncias. O padrão 12 – **Múltiplas Instâncias sem Sincronização** define que “dentro do contexto de uma instância de *workflow*, múltiplas instâncias de uma atividade podem ser criadas, gerando linhas de execução independentes. Essas linhas não precisam ser sincronizadas”. Em NPD, o operador unário “?” permite a criação de múltiplas instâncias de uma atividade. Por exemplo, no processo P_1 definido como

$$\text{SET } P_1 = A? * \parallel B.C$$

tem-se que a atividade A pode ser executada uma ou mais vezes, de forma paralela.

O padrão 13 – **Múltiplas Instâncias com Conhecimento Prévio em Tempo de Projeto** especifica que “para uma instâncias de processo, uma atividade é instanciada um número específico de vezes e este número é conhecido em tempo de projeto do processo. Após o término da execução de todas as instâncias da atividade, alguma outra atividade precisa ser iniciada”. NPD possui uma segunda versão do operador “?” que permite especificar o número de instâncias da atividade que serão criadas. Por exemplo, um processo P_2 que deve executar 3 instâncias da atividade A pode ser definido em NPD como:

$$\text{SET } P_2 = A ? 3 . B + C$$

O padrão 14 – **Múltiplas Instâncias com Conhecimento Prévio em Tempo de Execução** define que “para uma instância de processo, uma atividade é instanciada múltiplas vezes e o número de instâncias pode variar, mas é conhecido em algum estágio durante a execução do processo, antes que as instâncias da atividade tenham que ser criadas. Após o término da execução de todas as instâncias da atividade, alguma outra atividade precisa ser iniciada”. NPD possui uma terceira versão do operador “?”, que permite associar a ele uma função que é avaliada em tempo de execução e que retorna um número inteiro positivo e não nulo, indicando o número de instâncias da atividade que deverão ser criadas. Exemplo:

$$\text{SET } P_4 = A ? f_1 . B + C$$

em que f_1 é uma função que retorna um número inteiro maior que zero.

O último padrão de múltiplas instâncias é o 15 - **Múltiplas Instâncias sem Conhecimento Prévio em Tempo de Execução**. Este padrão é semelhante ao padrão 12, diferenciando-se do primeiro simplesmente por requerer a sincronização das instâncias

antes de habilitar a execução da atividade seguinte. Para representá-lo, utiliza-se o operador de repetição ilimitada “?” combinado com o operador de composição sequencial “•”. Por exemplo: quando um processo deve executar múltiplas instâncias da atividade A e então ele só pode executar B depois que a execução das instâncias de A tiver sido finalizada, a expressão NPDL correspondente é:

$$\text{SET } P_5 = A?^* \cdot B$$

Figura C.2: Padrões de controle de fluxo em NPDL (Parte 2)