Fernando Machado Lima Ferreira

SMERA: Simulador Multi-Estocástico de Redes de Atividades - Um método para estimar o tempo de execução de processos baseados na UML

Rio de Janeiro

2007

Fernando Machado Lima Ferreira

SMERA: Simulador Multi-Estocástico de Redes de Atividades - Um método para estimar o tempo de execução de processos baseados na UML

Orientador:
Prof. Ph.D. Eber Assis Schmitz

Co-orientador:
Prof. D.Sc. Fabio Protti

MESTRADO EM INFORMÁTICA
INSTITUTO DE MATEMÁTICA / NÚCLEO DE COMPUTAÇÃO ELETRÔNICA
CENTRO DE CIÊNCIAS DA MATEMÁTICA E DA NATUREZA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Rio de Janeiro

2007

FERREIRA, F. M. L.

SMERA: Simulador Multi-Estocástico de Redes de Atividades - Um método para estimar o tempo de execução de processos baseados na UML na UML, [Rio de Janeiro], 2007.

82p., 29,7 cm (IM/NCE/UFRJ, MSc., Informática, 2007)

Dissertação (Mestrado) - Universidade Federal do Rio de Janeiro, IM/NCE

1. UML. 2. Simulação. 3. RCPS. I. Título. II. Série

Dissertação submetida ao corpo docente do Instituto de Matemática (IM) / Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do grau de Mestre em Informática.

Aprovada por:

Prof. Ph.D. Eber Assis Schmitz - UFRJ - Orientador

Prof. D.Sc. Fabio Protti - UFRJ - Co-orientador

Prof. Ph.D. Juarez Alencar - UFRJ

Prof. Ph.D. Felipe Maia Galvão França - UFRJ

Prof. D.Sc. Paulo Eustáquio Duarte Pinto - UERJ

Dedicatória

Dedico esta dissertação a minha família,
aos meus amigos professores e orientadores,
a Deus por estar sempre presente,
e a minha esposa por todo apoio e dedicação.

Agradecimentos

A minha esposa Daniela por me dar apoio em todos os momentos, mesmo quando trabalhar nesta dissertação significava não poder lhe dar atenção e me ausentar na participação de nossas tarefas diárias.

Aos professores Eber Schmitz, Fábio Protti e Juarez Alencar por me guiarem no desenvolvimento deste trabalho. A orientação, experiência e dedicação que vocês me forneceram foi fundamental para a conclusão e qualidade deste trabalho.

Ao Núcleo de Computação Eletrônica por fomentar esta pesquisa, e por ter me abrigado durante quatro ótimos anos, nos quais trabalhei como analista de sistemas com uma equipe de amigos, dos quais tenho certeza que manterei para o resto de minha vida.

Aos meus pais por me incentivarem a sempre estudar, nunca permitindo que desistisse de lutar diante das desavenças da vida. Muito obrigado por toda a educação, carinho e amizade.

A Deus pela vida maravilhosa... por ser carioca, e brasileiro... e por isso não desisto NUNCA!!!

”Prometo, que no exercício profissional
como bacharel em Ciência da Computação,
serei fiel aos preceitos da honra e da ciência,
promovendo o uso e o desenvolvimento da Informática
em benefício do cidadão e da sociedade.”
SBC

Resumo

O aperfeiçoamento de processos requer alguma definição formal do processo e suas medidas associadas de desempenho. O processo pode ser então analisado, diagnosticado e hipóteses são feitas de onde deveriam ser feitas mudanças. Subseqüentemente, o novo processo pode ser concebido, testado e conferido se as melhorias desejadas foram alcançadas de fato. Isto é uma operação cara e que consome tempo. Simulação é uma ferramenta barata e poderosa para o diagnóstico e teste de várias alternativas de melhoria antes do teste de campo delas. Esta dissertação apresenta um método para obter a distribuição de probabilidade do tempo de execução de uma grande variedade de processos que estão descritos nos diagramas de atividades da UML. Tal método é baseado na simulação de Monte Carlo, que permite a identificação de fatores que fortemente influenciam o tempo de execução de processo, enquanto favorece mudanças que aumentam eficiência de processo, com impacto considerável no desenvolvimento de táticas e estratégias empresariais.

Abstract

The process improvement requires some formal process definition and its associated measures of performance. The process can then be analyzed, diagnosed and hypotheses made to where changes should be made. Subsequently, the new process can be conceived, tested and checked whether the desired improvements have actually been achieved. This is a time consuming and costly operation. Simulation is a powerful low-cost tool for the diagnosis and test of several alternatives of improvement prior to their field test. This paper presents an UML-based method to obtain the probability distribution of the execution time of a large variety of processes described in the activities diagrams. Such method, which is based upon Monte Carlo simulation, allows for the identification of factors that most strongly influence the process execution time, favoring changes that increase process efficiency with considerable impact upon the deployment of business tactics and strategies.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AOA	Atividade nos Arcos (*Activity On Arc*)
AON	Atividades nos Nós (*Activity On Node*)
CAAN	Redes de Atividades de Alternativa Controlada (*Controlled Alternative Activity Networks*)
DA	Diagrama de Atividade
DABF	Diagrama de Atividade Bem Formado
ES	Engenharia de Software
FDP	Função de Distribuição de Probabilidade
GERT	Técnica de Revisão e Avaliação Gráfica (*Graphic Evaluation and Review Technique*)
JVM	Máquina Virtual Java (*Java Virtual Machine*)
OMG	*Object Management Group*
PF	Pontos de Função
PU	Processo Unificado
RAD	Rede de Atividades Determinista
RAND	Rede de Atividades Não Determinista
RANDRE	Redes de Atividades Não-Deterministas com Recursos Escassos
RCPSP	Problemas de Escalonamento de Projetos com Recursos Escassos (*Resource Constrained Project Scheduling Problems*)
RUP	*Rational Unified Process*
UML	Linguagem de Modelagem Unificada (*Unified Modeling Language*)
VA	Variável Aleatória
XML	*Extensible Markup Language*
XP	Programação Extrema (*eXtreme Programming*)

Sumário

1 Introdução

A competição acirrada juntamente com a crescente exigência dos clientes estão levando as empresas a aperfeiçoarem continuamente seus processos para a conquista de novos mercados. Estes processos podem ser definidos como um conjunto estruturado de atividades ordenadas no tempo e espaço que produzem um produto ou um serviço, com início e fim, com entradas e saídas bem definidas (HAMMER; CHAMPY, 1994). Neste contexto, as empresas que possuem processos mais eficientes conseguem oferecer o melhor produto ou serviço no menor tempo.

Concomitantemente com o crescimento da concorrência entre as empresas, a Linguagem Unificada de Modelagem (UML) se tornou largamente utilizada não apenas para a especificação de softwares, estruturas, comportamento e arquiteturas, mas também para modelar processos de negócio e representação de estrutura organizacional (OMG, 2006b).

O crescimento da utilização da UML para modelagem de processos de negócio motivou a criação de grupos como o *Business Modeling and Integration Domain Task Force* da *Object Management Group* (OMG) que possui a missão de desenvolver especificações de modelos integrados para auxiliar a gerência das empresas, que irão promover a integração e a colaboração de pessoas, sistemas, processos e informação através das empresas (OMG, 2006a).

Estes modelos de processos descritos na UML foram criados para atender vários objetivos: compreensão dos mecanismos chaves do processo existente; criação de sistemas de informação para auxiliar o negócio; servir de base no processo de aperfeiçoamento do negócio; exibir a estrutura de um processo inovado; experimentar um novo conceito e identificar oportunidades (ERIKSSON; PENKER, 2000).

Mesmo utilizando estes diagramas da UML para visualizar a proposta de aperfeiçoamento dos processos, qualquer alteração nestes acarreta riscos substanciais visto a dificuldade de prever os resultados das mudanças antes que estas sejam colocadas em prática.

Isto deve-se ao fato de que as atividades dos processos são estocásticas, ou seja, o tempo de execução de cada atividade é variável. Isso implica que as métricas do processo serão variáveis aleatórias, e a duração do processo será descrita por uma Função de Distribuição de Probabil-

idade (FDP). Além disso, os fluxos de execução dos processos também são estocásticos, uma vez que os elementos de decisão fazem com que existam diferentes caminhos do início até o final do processo.

Por isso, é evidente que a taxa de falha nos projetos de aperfeiçoamento de processos são altas, e desta forma, é razoável alegar que a simulação dos modelos pode oferecer um grande potencial para análise do comportamento e dos resultados (HERBERT, 1979) e assim, reduzir os riscos associados aos mesmos.

Uma forma de simular estes modelos é encará-los como um problema de escalonamento de projetos limitados por recursos (Resource Constrained Project Scheduling Problems - RCPSP) que continua a ser uma área ativa de pesquisa, atraindo nos últimos anos, um interesse crescente dos pesquisadores na procura por uma melhor solução (VALLS; BALLEST; QUINTANILLA, 2005).

Porém, a simulação destes modelos descritos na UML pode apresentar resultados errôneos, ou até mesmo não ser passível de simulação devido a inconsistências no modelo. Isso deve-se ao fato de que embora a UML seja um padrão largamente reconhecido e utilizado, ela é criticada por permitir imprecisão semântica, que induz uma interpretação subjetiva, e, por dificuldades na modelagem formal de processos (ESHUIS; WIERINGA, 2001).

Conforme os fatos apresentados, pode-se destacar:

1. O aperfeiçoamento de processos é arriscado.
2. A UML é um padrão para modelagem de processos.
3. A UML permite a formação de modelos ambíguos.
4. A simulação é uma boa ferramenta de análise.
5. Encontrar a duração de um processo pode ser um RCPSP.

Estes fatos nos levam a definição da proposta:

1.1 Proposta e Escopo

Neste trabalho propõe-se um método para simulação de processos descritos nos Diagramas de Atividades (DA) da UML para se obter a FDP da duração do processo. Para tal, faz-sem necessárias as seguintes etapas:

- Definir Regras de Formação dos DA da UML de forma a remover a ambigüidade para o algoritmo de simulação.
- Transformar os Diagramas de Atividades em uma estrutura de grafo.
- Elaborar um algoritmo de simulação.
- Coletar e disponibilizar o resultado da simulação.

Pata tal, este trabalho está organizado da seguinte forma:

1.2 A estrutura

O Capítulo 2 apresenta uma revisão dos principais trabalhos e pesquisas desenvolvidos para a obtenção de uma estimativa do tempo de execução de um processo, ressaltando suas características e pontos fracos.

O Capítulo 3 apresenta as regras de formação da UML e o algoritmo de simulação descritos na proposta.

O Capítulo 4 apresenta um estudo de caso com um exemplo numérico de forma a deixar claro o funcionamento do método.

O Capítulo 5 apresenta as conclusões, a discussão dos resultados e os pontos que ficaram em aberto para serem desenvolvidos em trabalhos futuros.

2 *Técnicas de Simulação*

Conforme apresentado, uma vasta quantidade de pesquisas em escalonamento de projetos vêm sendo desenvolvidas, embora uma grande maioria assuma que existam informações completas sobre o problema do escalonamento para que este possa ser resolvido, e também assumem que o processo é estático e determinista. Entretanto, no mundo real, as atividades do projeto são sujeitas a considerável incerteza (HERROELEN; LEUS, 2005).

Estas pesquisas pode ser divididas em dois grupos segundo a classificação de suas redes de atividades: deterministas, onde todos os caminhos são percorridos; e não-deterministas, onde existe uma probabilidade de execução associada a cada trecho ou percurso. Exemplos de redes de atividades deterministas são o Programa de Avaliação e Revisão Técnica (Program evaluation and review technique - PERT) e o Método do Caminho Crítico (Critical Path Method - CPM) (WIEST; LEVY, 1969). Como exemplo de redes de atividades não deterministas tem-se a Avaliação Gráfica e Técnica de Revisão (Graphic Evaluation and Review Technique - GERT) (PRITSKER, 1966b), as Redes de Atividades de Alternativa Controlada (Controlled Alternative Activity Networks - CAAN) (GOLENKO-GINZBURG, 1998), e as Redes de Atividades Não-Deterministas com Recursos Escassos (RANDRE) (KLING, 2005).

A introdução da restrição de recursos para a execução das atividades nas redes de atividades deterministas transforma o problema do escalonamento em uma classe de problemas conhecida como RCPSP (DEMEULEMEESTER, 1999). A função objetivo mais pesquisada é a determinação do menor tempo de execução, que se mostrou ser um problema NP Completo (BLAZEWICZ; LENSTRA; KAN, 1983; LENSTRA; RINNOOY, 1979), e por isso é necessário a utilização de heurísticas para a resolução do problema (HARTMANN; KOLISCH, 2000; KOLISCH; HARTMANN, 2005); embora recentemente existam estudos que comprovam a resolução do problema por métodos exatos (ZAMANI, 2001).

Uma sub-classe do RCPSP que oferece uma visão mais realista do problema é conhecida como RCPSP Estocástico (YANG; GEUNES; O'BRIEN, 2001), onde a duração de cada atividade é dada por uma Variável Aleatória (VA) que segue alguma FDP (DEMEULEMEESTER;

HERROELEN, 2002). Esta é uma importante especialização pois permite reproduzir as incertezas inerentes ao processo, mas leva também a uma maior complexidade de análise (YANG; GEUNES; O'BRIEN, 2001).

Desta forma, os modelos que permitem expressar as incertezas do processo real são aqueles que permitem expressar a variação do tempo de execução de cada atividade e o fluxo da rede de atividades não-determinista. Assim, é apresentada uma revisão dos principais métodos mencionados na literatura que abordam estes pontos:

2.1 GERT

A Técnica de Revisão e Avaliação Gráfica (Graphical Evaluation and Review Technique - GERT) foi desenvolvida por Alan A. B. Pritsker para atender as necessidades do projeto espacial Apollo da Administração Nacional de Aeronáutica e Espaço (National Aeronautics and Space Administration - NASA) em 1966 (PRITSKER, 1966a).

GERT é uma técnica para o estudo de redes estocásticas, que combina a teoria de redes, teoria de probabilidades e simulação que resulta em uma ferramenta eficiente para análise de sistemas (MOORE; CLAYTON, 1976).

As redes GERT são caracterizadas por atividades nos arcos (Activity on Arch - AOA) e por nós que possuem algumas características especiais. Os arcos são descritos por dois parâmetros: A probabilidade do arco ser percorrido e o tempo necessário para atravessar o arco. O tempo associado a um arco (atividade) pode ser constante ou uma variável aleatória; neste caso, quando o arco é percorrido, seleciona-se um tempo fixo para o arco.

Os nós da rede GERT estão divididos em duas partes: entrada e saída (Figura 2.1). Na entrada (lado esquerdo) existem três tipos de nós:

(1) Ou-Exclusivo: A realização de qualquer arco em direção ao nó causa a realização do nó; entretanto, um e apenas um dos arcos em direção a este nó pode ser realizado em um dado instante de tempo.

(2) Ou-Inclusivo: A realização de qualquer arco em direção ao nó causa a realização do nó. O tempo da realização é o menor tempo de conclusão das atividades em direção ao nó Ou-Exclusivo.

(3) E: O nó será realizado apenas se todos os arcos em direção ao nó são realizados. O tempo da realização é desta forma o maior tempo de finalização das atividades em direção

ao nó E.

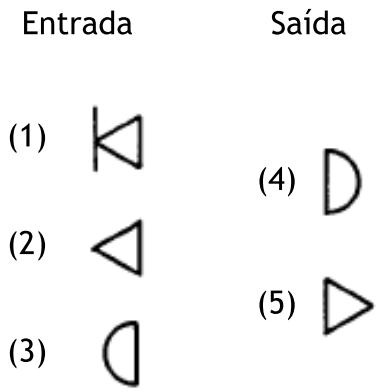

Figura 2.1: Tipos de nós na rede GERT

Na saída (lado direto) existem dois tipos de nós:

(4) Determinístico: Todos os arcos originados do nó são ativados se o nó é realizado.

(5) Probabilístico: Exatamente um arco originado do nó é ativado se o nó é realizado.

Desta forma, pode-se combinar os tipos de entrada com os de saída, que dão origem aos seis tipos de nós das redes GERT, conforme demonstrado na Figura 2.2.

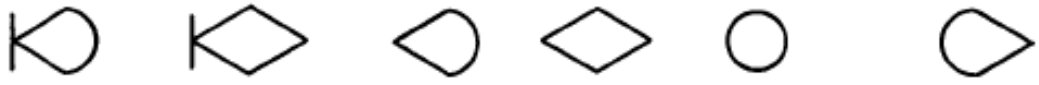

Figura 2.2: Conjuntos de nós possíveis em GERT

Um grande poder de representação deste modelo é a possibilidade de informar o comportamento das atividades em cada retorno, pois cada nó informa o número de atividades predecessoras que devem ser terminadas para que a dada atividade se inicie. No quadrante superior esquerdo é informado o número de atividades que devem ser terminadas para que a atividade se inicie na primeira execução, enquanto o quadrante inferior esquerdo indica o número para as demais execuções da atividade (Figura 2.3).

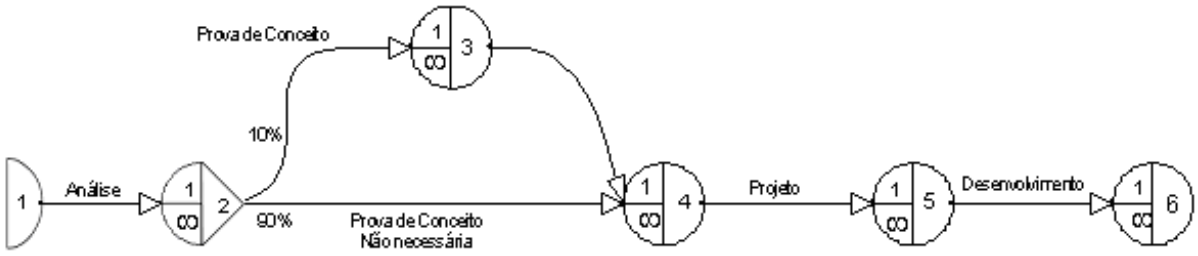

Figura 2.3: Exemplo de rede GERT. Fonte KLING (2005)

2.1.1 Limitações do modelo GERT

Uma limitação do modelo GERT é o fato de não incorporar o conceito de recurso, o que torna o modelo otimista, uma vez que os recursos, por serem escassos, aumentam o tempo total de execução do projeto. Isso pode ser percebido com mais clareza se considerarmos uma situação bem simples:

Suponha que temos apenas um programador (que é o nosso recurso), e temos duas atividades, a primeira é a implementação de testes funcionais, e a segunda, a implementação dos casos de uso do sistema. Suponha agora, que cada uma dessas atividades consome um programador por uma semana. Como só temos um programador, não podemos executar as duas atividades simultaneamente. Assim a duração total para executar estas duas tarefas é de duas semanas, mesmo que estas estejam em paralelo no modelo.

Se aplicássemos estas duas atividades em paralelo ao modelo GERT, o tempo necessário para suas execuções seria de uma semana, o que contradiz o resultado verdadeiro que é de duas semanas. Desta forma estaremos sempre sendo otimistas, fornecendo uma duração menor do que a duração real necessária para a execução de todas as atividades.

Uma outra limitação é a impossibilidade de informar o comportamento dos nós para cada execução da atividade. Neste modelo só é possível informar quantidade de atividades predecessoras que devem ter terminado para que atividade seja executada em sua primeira execução e nas seguintes. Desta forma, obriga a toda execução posterior a atender a mesma condição (informada no quadrante inferior esquerdo do nó), e não é possível informar quantas passagens (retornos) são esperados para aquela atividade.

2.1.2 Características do modelo GERT

Conforme apresentado, pode-se classificar as redes GERT da seguinte maneira:

1. Atividades nos arcos (AOA).

2. Permite transições não deterministas.
3. Permite formação de grafos cíclicos.
4. Impossibilita informar comportamento diferenciado a cada retorno.
5. Não implementa restrição de recursos.
6. As durações das atividades são não deterministas.
7. Notação proprietária.

2.2 CAAN

A Rede de Atividades de Alternativa Controlada (Controlled Alternative Activity Networks - CAAN) consiste em um método para a resolução do RCPSP que é baseada em simulação e utiliza um sub-conjunto das redes GERT.

Este método possui nós de decisão determinísticos e nós alternativos de bifurcação probabilísticos. O modelo possuiu uma lista de recursos renováveis que são limitados por uma quantidade fixa durante a duração do projeto. Cada atividade no projeto requer uma quantidade fixa de diversos tipos de recursos. A duração da atividade é uma variável aleatória com uma dada função de densidade. O problema é minimizar a duração esperada do projeto por determinação do tempo de início de cada atividade (GOLENKO-GINZBURG; GONIK; LASLO, 2003).

O modelo CAAN é caracterizado por ser finito, conexo, orientado, com atividades nos arcos (AOA), com as seguintes propriedades:

- A rede G(N, A) possui uma fonte e não menos que 2 sumidouros.
- A lista de nós da rede G(N, A) inclui quatro tipos de nós:
  1. (x): com o receptor lógico "E"
  2. (α): com o receptor lógico "E"e com o emissor "Ou-Exclusivo"
  3. (β): com o receptor "Ou-Exclusivo"e com o emissor "Seguir Obrigatório"
  4. (γ): com o receptor "Ou-Exclusivo"e com o emissor "Ou-Exclusivo"
- A lista de nós alternativos (tipos 2 e 4) são sub-divididos em dois sub-grupos:
  1. $\overline{\overline{N}} \subset N$: nós alternativos $\{\overline{\overline{\alpha}}\}$ com arcos estocásticos.
  2. $\overline{N} \subset N$: nós (decisão) alternativos determinísticos $\overline{\alpha}$.

2.2.1 Limitações do modelo CAAN

Visto que o modelo CAAN é estruturado a partir de um sub-grupo das redes GERT, e por ser caracterizado por um grafo G(N, A) finito, é inviabilizada a representação de retornos, o que impossibilita a representação de re-trabalhos de atividades nos modelos. Por exemplo, a representação de um modelo onde se executa uma atividade de correção de erros, e em seguida se verifica se todos os testes passaram; caso negativo, volta-se a executar a atividade de correção de erros.

Outra limitação do modelo CAAN é a impossibilidade de representação de redes de atividades de alternativa generalizada não-divisível, que corresponde a representação de qualquer grafo orientado, acíclico com cruzamentos, como por exemplo o da Figura 3.3.

2.2.2 Características do modelo CAAN

Conforme apresentado, pode-se classificar as redes CAAN da seguinte maneira:

1. Atividades nos arcos (AOA).
2. Permite transições não deterministas.
3. Implementa restrição por recursos.
4. Não permite a formação de grafos cíclicos.
5. As durações das atividades são não deterministas.
6. Sem notação gráfica.

2.3 RANDRE

O modelo Redes de Atividades Não Deterministas com Recursos Escassos (RANDRE) surgiu através de uma modificação no modelo de rede atividades utilizado para descrever as redes no RCPSP Estocástico. As redes de atividades foram alteradas para suportar uma construção comum nos diagramas de atividades da UML conhecida como decisão. (KLING, 2005).

O modelo RANDRE tem como objetivo fornecer um esquema prático para simulação de redes de atividades não deterministas descritas nos diagramas de Atividades (DA) da UML, seguindo os padrões de formação chamados de RANDREs básicas (Figura 2.4 (KLING, 2005)).

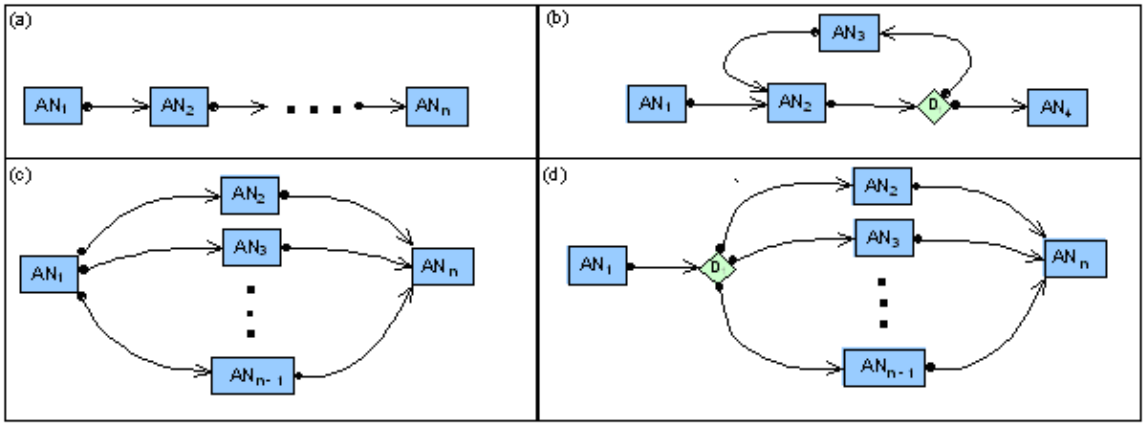

Figura 2.4: RANDREs básicas: (a) Série, (b) retorno, (c) paralelo, (d) decisão

Para que o DA seja passível de simulação, é necessário que ele possa ser decomposto em RANDREs básicas que são: série (ou trivial), retorno (ou loop), paralelo e decisão. Desta forma, o algoritmo de simulação transforma o DA em uma estrutura de árvore. As folhas dessa árvore são as atividades, e os nós internos são as RANDREs básicas. A Figura 2.5 ((KLING, 2005) mostra a transformação do DA para uma RANDRE.

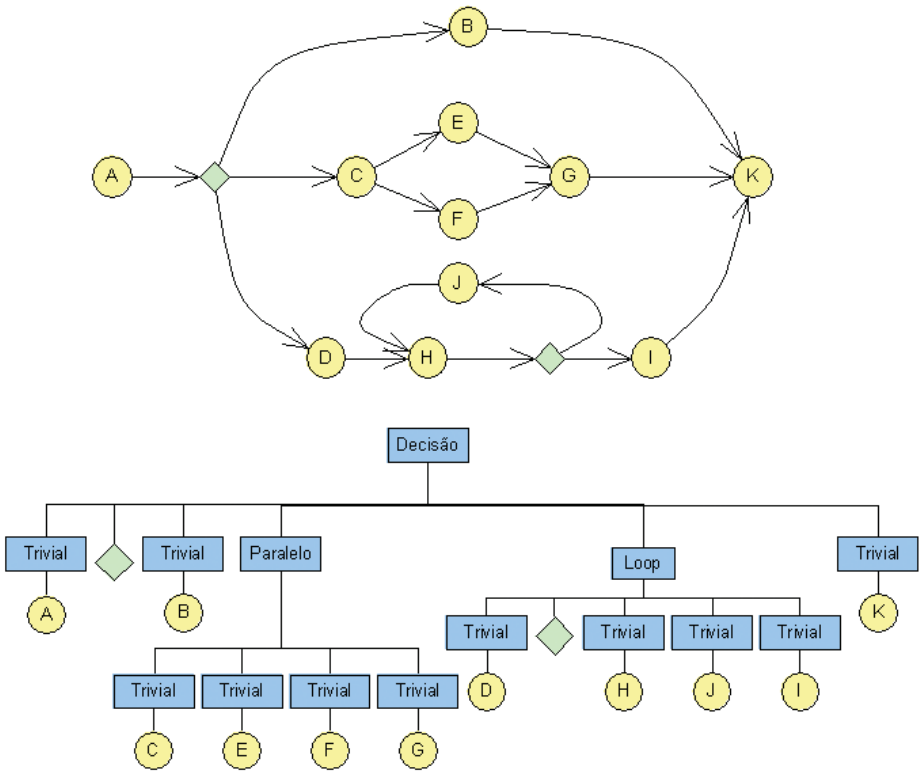

Figura 2.5: Transformação em Árvore

Uma vez que se tenha transformado o DA, faz-se necessário cadastrar as informações complementares para se executar a simulação. Estas informações dizem respeito a:

- Recursos Disponíveis: Conjunto de recursos com nome e quantidade que podem ser utilizados para a execução das atividades.
- Recursos Necessários: Conjunto de recursos que são necessários para que uma determinada atividade possa ser executada. Devem ser informados para cada atividade.
- Duração: Cada atividade deve possuir uma variável aleatória com a função de distribuição de probabilidade com um dos seguintes tipos: Constante, Bernoulli, Triangular ou Exponencial.
- Probabilidade de Desvio: Para cada nó de decisão (ou desvio) deve-se informar a probabilidade de cada arco que sai do elemento de decisão. Pode-se cadastrar uma probabilidade para cada execução do nó de decisão, como por exemplo: para a primeira execução 50%, para a segunda execução 100%.

Uma vez que todas as informações tenham sido cadastradas, pode-se executar o algoritmo de simulação que consiste na repetição da execução de dois algoritmos principais: Algoritmo para seleção de um cenário (seleção do cenário a partir da hierarquia) e Algoritmo para escalonamento de um cenário (com recursos ou sem recursos).

2.3.1 Limitações do modelo RANDRE

Embora o modelo RANDRE tenha introduzido um grande poder de expressão no modelo RCPS Estocástico, sua estrutura interna de organização do grafo utiliza uma árvore (Figura 2.5), o que restringe a formação de alguns modelos nos diagramas de atividades, como por exemplo o caso patológico (KLING, 2005) para RANDRE demonstrado na Figura 2.6, e a formação de grafos cruzados, como o demonstrado na Figura 3.3 (KLING, 2005).

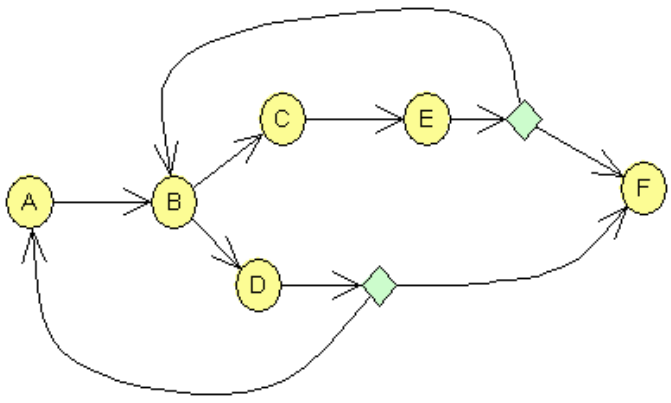

Figura 2.6: RANDRE Patológica

Infelizmente, essas restrições quanto à utilização de retornos sobrepostos e grafos cruzados se mostraram demasiadamente proibitivas. Note que utilizando esta formação em árvore, os diagramas como os das Figuras 2.6 e 3.3 não possuem representação.

No Capitulo 4 apresenta-se o estudo de caso realizado para verificação da cobertura do algoritmo proposto. Neste estudo verificou-se que todos os processos apresentavam as características de laços sobrepostos e cruzamentos, e por isso não puderam ser simulados utilizando-se o método RANDRE, devido a decomposição em RANDREs básicas.

2.3.2 Características do modelo RANDRE

Conforme apresentado, pode-se classificar o método RANDRE da seguinte maneira:

1. Atividades nos nós (AON)
2. Transições não deterministas
3. Grafo cíclico
4. Implementa restrição de recursos
5. Durações não deterministas
6. Representação gráfica (DA da UML)
7. Não permite retornos sobrepostos
8. Não permite grafos com cruzamentos

3 O método SMERA

Conforme visto anteriormente, os atuais métodos propostos na literatura possuem restrições que inviabilizam a representação de processos complexos (com retornos, cruzamentos, restrição por recursos...). A proposta deste trabalho é prover um método de simulação de processos descritos na UML de forma a possibilitar a estimativa do tempo de execução dos mesmos. Este algoritmo deve permitir a formação de diagramas de atividades que utilizem laços sobrepostos e cruzamentos, de forma a viabilizar a modelagem de processos complexos reais.

Para isso será necessário formular algumas regras de formação nos modelos de processos dos Diagramas de Atividade da UML, devido à possibilidade de interpretação subjetiva e à ambigüidade mencionados anteriormente. Além disso, será necessário se cadastrar informações complementares aos diagramas para que eles possam ser simulados; estas informações dizem respeito a: lista de probabilidades dos desvios, duração das atividades e recursos necessários.

Como a entrada para o algoritmo de simulação são os diagramas da UML, é necessário cadastrar estas informações complementares. Optou-se por trabalhar com a ferramenta RAPDIS (Rules And Processes for the Development of Information Systems) por ser uma ferramenta desenvolvida pelo nosso grupo de pesquisa (GETI, 2007), e desta forma, pode se incluir estas informações nos diagramas.

3.1 O projeto RADPIS

O RAPDIS é um ambiente de Arquitetura guiada por Modelo (Model Driven Architecture - MDA) para o desenvolvimento de Sistemas de Informação. Através dele é possível construir modelos com alto nível de abstração e transformá-los automaticamente em modelos com baixa abstração, facilitando e tornando mais rápido o processo de desenvolvimento (MORGADO et al., 2006).

As principais funcionalidades deste ambiente são:

- Modelagem de Negócios - definição dos termos, objetivos, recursos, processos e regras;
- Modelagem de Sistemas de Informação - casos de uso, classes, seqüência e estados;
- Transformação do Modelo de Negócio no Modelo de Sistema de Informação - geração automática dos diagramas de casos de uso e do diagrama de classes de domínio;
- Transformação do Modelo de Sistema de Informação no Modelo para uma Plataforma Específica - geração automática de código (Java e Delphi).

Esta ferramenta permite o cadastro de recursos disponíveis, bem como as suas quantidades, que serão fixos durante toda a execução da simulação. Além disso, todos os recursos serão considerados renováveis, isto é, não serão consumidos durante sua utilização, de forma que ao final de sua utilização, eles poderão ser utilizados por outras atividades.

Nos diagramas de atividades, os recursos necessários para a execução de uma atividade são expressos através do elemento Insumo/Produto da UML com o fluxo pontilhado em direção a atividade.

Para se cadastrar as informações referentes às atividades, deve se utilizar a tela de definições da atividade. Na seção de simulação deve-se informar o número de recursos necessários para a execução da atividade, bem como a estimativa da duração da atividade, conforme demonstrado na Figura 3.1.

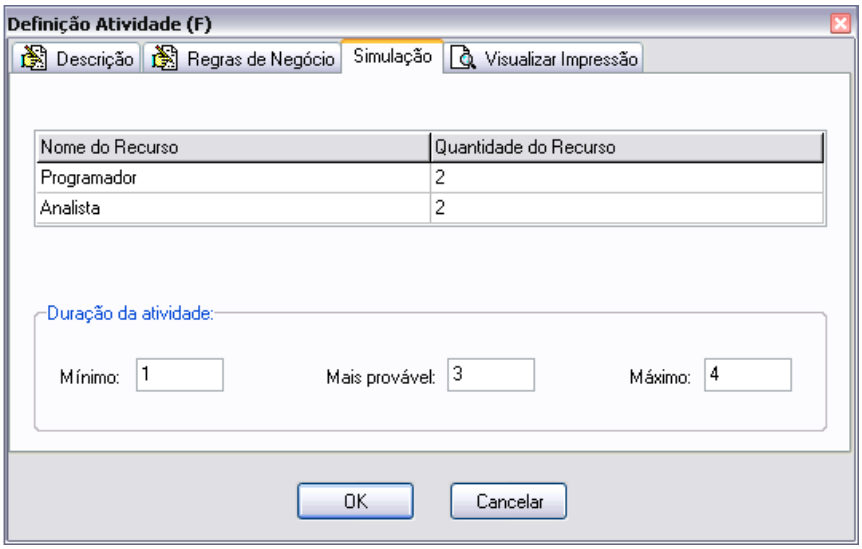

Figura 3.1: Cadastro de recursos necessários e duração

A duração das atividades são expressas por uma distribuição triangular de probabilidade[1], visto que a exatidão da estimativa pode ser aumentada considerando o total de risco da esti-

[1] A distribuição triangular e o algoritmo de amostragem estão descritos na seção de anexo

mativa original. As estimativas de três pontos se baseiam na determinação de três tipos de estimativas (PMI, 2004):

- **Mais provável:** A duração da atividade quando fornecidos os recursos com mais probabilidade de serem atribuídos, sua produtividade, as expectativas realistas de disponibilidade para a atividade do cronograma, as dependências de outros participantes e as interrupções.
- **Otimista:** A duração da atividade se baseia em um cenário para o melhor caso do que está descrito na estimativa mais provável.
- **Pessimista:** A duração de atividade se baseia em um cenário para o pior caso do que está descrito na estimativa mais provável.

O mesmo ocorre para os desvios (elemento de decisão da UML). Para cada desvio, deve ser informado a probabilidade de seleção de cada fluxo que possui origem no desvio. Permite-se que possa ser informada uma probabilidade para cada execução do desvio, desta forma, permitindo o cadastro de uma lista de probabilidades para cada fluxo, conforme demonstrado na Figura 3.2. No caso de não existir uma probabilidade associada ao número de execução do desvio, utiliza-se a última probabilidade cadastrada.

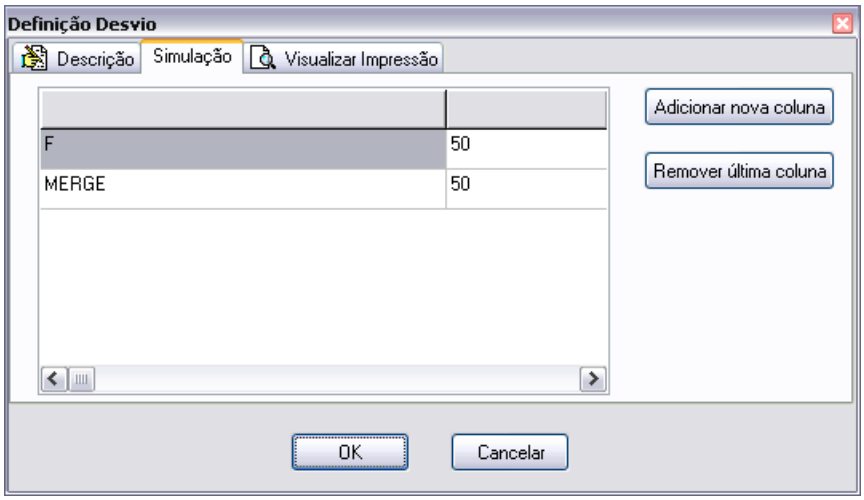

Figura 3.2: Cadastro de probabilidade de fluxos

Esses diagramas com todas as informações preenchidas devem seguir algumas regras de formação que serão apresentadas na próxima seção.

3.2 Regras de Formação da UML

Conforme mencionado anteriormente, a UML possibilita a formação de modelos ambíguos que podem inviabilizar a simulação dos mesmos. Desta forma, torna-se necessário a utilização das seguintes regras:

- **RF1** - Delimitadores de Fluxo Únicos
- **RF2** - Balanceamento do E
- **RF3** - Balanceamento do OU
- **RF4** - Respeito ao Sincronismo

3.2.1 RF1 - Delimitadores de Fluxo Únicos

Esta regra obriga que cada diagrama de atividade dos processos seja delimitado pelos elementos de início e fim de fluxo da UML. Cada Diagrama de Atividade (DA) deve conter apenas um elemento de início de fluxo, e apenas um elemento de fim de fluxo. Caso o modelo apresente mais de um elemento terminador de fluxo, deve-se fazer com que todos os elementos finais sejam transformados em atividades (com duração constante igual a zero caso necessário) e apontar para um novo e único elemento de fim de fluxo.

Esta regra é requisito necessário para o funcionamento algoritmo do RCPS Paralelo Padrão descrito por (OLAGUIBEL; GOERLICH, 1989), que será apresentado adiante, e que será utilizado para se obter o tempo mínimo de execução de um cenário determinista.

3.2.2 RF2 - Balanceamento do E

De forma a explicitar de forma clara a representação do ”E”, ou seja, uma atividade dispara mais de uma atividade, faz-se necessário a utilização do objeto de barra de sincronismo da UML. Desta forma, cada atividade possui apenas um fluxo de saída, e vários fluxos de saída estão representados apenas nas barras de sincronismo.

A Figura 3.3 mostra um diagrama de atividades com atividades que disparam mais de uma atividade. Aplicando a RF2 neste diagrama ele passa a ser conhecido como Diagrama de Atividade Bem Formado (DABF), que está representado na Figura 3.4.

Além disso, todo início de sincronismo deve possuir um elemento correspondente de fim de sincronismo, como tem sido adotado desde a UML 1.4, que adotou a restrição de que todo

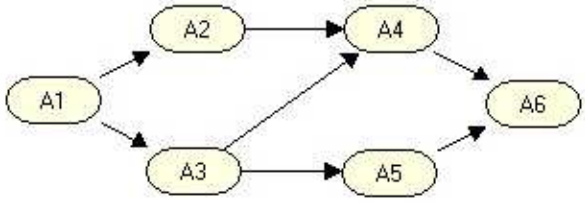

Figura 3.3: Diagrama sem regras de formação

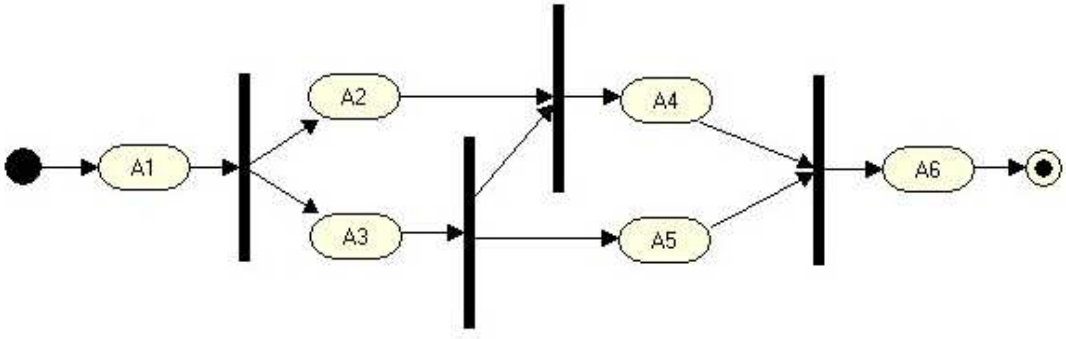

Figura 3.4: Diagrama bem formado

diagrama de atividade deve possuir bifurcações (início de sincronismo) e junções (fim de sincronismo) balanceados (ESSHUIS, 2002). Ressalta-se que as bifurcações e as junções não necessitam ser bem alinhadas, o que viabiliza a formação de redes cruzadas.

3.2.3 RF3 - Balanceamento do OU

O mesmo conceito do balanceamento pode ser aplicado aos elementos de decisão (ou desvio). Todo elemento de decisão deve possuir um elemento correspondente de união, de forma a explicitar de forma clara a lista de precedência de cada atividade.

Na Figura 3.5 é apresentado um diagrama que não apresenta a RF3. Tomando-se como referência a atividade A6, não é possível definir a lista de precedência desta atividade, uma vez que nesta atividade chegam três fluxos distintos, embora apenas dois destes fluxos sejam percorridos.

Desta forma, torna-se necessário utilizar um elemento para união de fluxos oriundos de um desvio. Optou-se por utilizar o próprio elemento de decisão da UML, que será distinguido de acordo com o número de fluxos que entram e que saem do elemento. Um elemento de decisão que possua um único fluxo de entrada e vários de saída será tratado como um desvio; o que recebe vários fluxos de entrada e possui apenas um de saída será tratado como uma união.

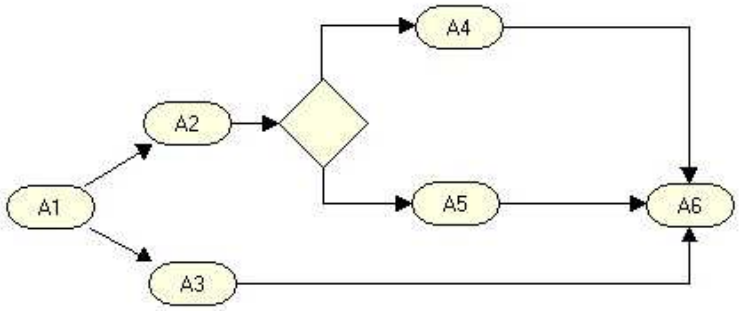

Figura 3.5: Precedência Ambígua

Aplicando-se a RF3 na Figura 3.5 (juntamente com as regras RF1 e RF2) obtém-se o Diagrama de Atividade Bem Formado (DABF) que é apresentado na Figura 3.6.

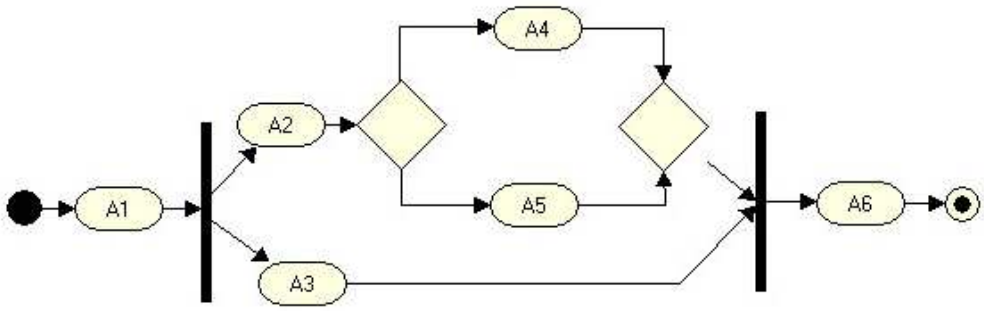

Figura 3.6: Precedência clara

3.2.4 RF4 - Respeito ao Sincronismo

Esta regra de formação diz respeito à correta utilização das barras de sincronismo juntamente com os elementos de desvio. É proibido a utilização dos desvios para sobrepor um elemento de fim de sincronismo (junção). A Figura 3.7 demonstra o uso incorreto do desvio, que viola esta regra de formação.

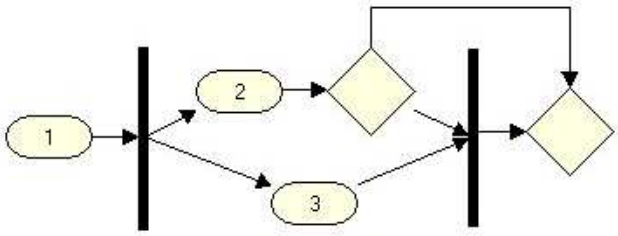

Figura 3.7: Perda de Sincronismo

Observa-se que caso o caminho do desvio até a união seja selecionado, o caminho do desvio até a junção não será percorrido; nesta caso, ao visitar-se o elemento de junção, este ficará

aguardando que todos os fluxos, que o possuam como destino, sejam percorridos, o que neste caso nunca ocorrerá. Nesta situação, ocorrerá um *starvation* onde o nó de junção ficará esperando para sempre um fluxo que nunca ocorrerá.

Desta forma, esta regra de formação visa sanar os dois tipos de conflito estrutural conhecidos como *deadlock* e perda de sincronismo, conforme observado por (MACEDO, 2003).

3.3 Mapeamento da UML para Grafo

Conforme observado anteriormente, faz-se necessário a utilização das quatro regras de formação dos diagramas de atividades para a simulação processos. A próxima etapa é a de transformar os diagramas de atividades em um grafo G orientado, que pode ser descrito com a notação de atividades nos nós (Activity On Node - AON) que possui uma lista de nós, e arcos que definem a lista de precedência de execução entre os nós.

Defini-se $G(B,E,LN,LL)$ onde B corresponde ao nó inicial, E corresponde ao nó final, LN corresponde a lista de nós, e LL corresponde a lista de arcos(precedência) entre os nós.

O algoritmo de simulação proposto trabalha apenas com um sub-conjunto dos elementos dos Diagramas de Atividades da UML, e para estes elementos, é apresentado a seguir os seus respectivos mapeamentos para o grafo G.

3.3.1 Delimitadores de Fluxos

Todo fluxo de trabalho deve possuir delimitadores de pontos de início e fim bem definidos. Nos diagramas de atividades, estes delimitadores são conhecidos pelos elementos de Início de fluxo e Fim de fluxo. Estes elementos serão mapeados para B e E, além de serem adicionados a LN.

3.3.2 Atividade & Processo

As atividades e os processos de uma organização representam as tarefas realizadas para atender a um determinado objetivo. Na UML, o mesmo símbolo é utilizado para representar os processos e as atividades. Desta forma ambos serão tratados como atividades. Toda atividade deverá ter apenas uma única entrada e uma única saída (MACEDO, 2003).

Cada atividade de um Diagrama de atividade será mapeada para um nó em LN com o tipo de nó T_N igual a $Activity$. Cada nó deste tipo deve possuir uma distribuição de probabilidade do

tempo de execução (D_N) e uma lista de recursos (com suas respectivas quantidades) necessários (R_N).

3.3.3 Transição

As transições representam o encadeamento dos elementos para a realização de um fluxo de trabalho. Cada transição será mapeada para um elemento de *link* (ou arco) e será adicionada em *LL*.

Cada *link* possui um nó de origem (S_L) e um nó de destino (S_D), além de uma lista de probabilidades de execução (P_L). Desta forma é possível associar a um *link* a probabilidade deste arco ser percorrido para cada passagem.

3.3.4 Desvios

Os desvios são nós de decisão onde o fluxo de execução do processo pode seguir em um dentre vários percursos disjuntos possíveis. Este elemento, conforme apresentado nas regras de formação, será utilizado para dois propósitos diferentes. Quanto um desvio possuir apenas uma entrada e várias saídas, ele assumirá o papel de um nó de decisão, onde será sorteado um link (de acordo com sua probabilidade) entre os vários possíveis; e neste caso ele será mapeado para um nó do tipo *Split* que será adicionado em L_N.

Quando um desvio possuir vários fluxos de entrada e apenas um fluxo de saída, ele assumirá o o papel de um nó de junção, e será mapeado para um nó do tipo *Join*, também sendo adicionado em L_N.

3.3.5 Barras de Sincronismo

As barras de sincronismo possibilitam a representação de fluxo de trabalho paralelo (concorrentes), e são utilizadas para representar duas ações distintas: a bifurcação e a sincronização (MACEDO, 2003). A bifurcação representa a multiplicação de uma transição, viabilizando a utilização da mesma transição em vários pontos; neste caso será mapeado para um nó do tipo *Fork*, sendo adicionado em L_N. A sincronização, que representa o sincronismo de várias transições independentes (AND implícito)que dá origem a uma única transição; e neste caso, será mapeado para um nó do tipo *Join*, também sendo adicionado em L_N.

3.3.6 Objetos

Em uma modelagem de processo podemos representar os objetos que interagem com a atividade durante sua execução. Estes objetos representam os insumos necessários para a sua realização (recursos), os produtos gerados (saídas). Os insumos e produtos poderão ser informativos, materiais, recursos humanos, sistêmicos ou de qualquer outro tipo (MACEDO, 2003). Os recursos serão mapeados para os recursos necessários de cada atividade, conforme mencionado anteriormente.

3.4 Algoritmo de Simulação

De posse do grafo *G*, que é uma rede de atividades não determinista, o próximo passo é aplicar o algoritmo de simulação para obter a distribuição de probabilidade do tempo de execução do processo.

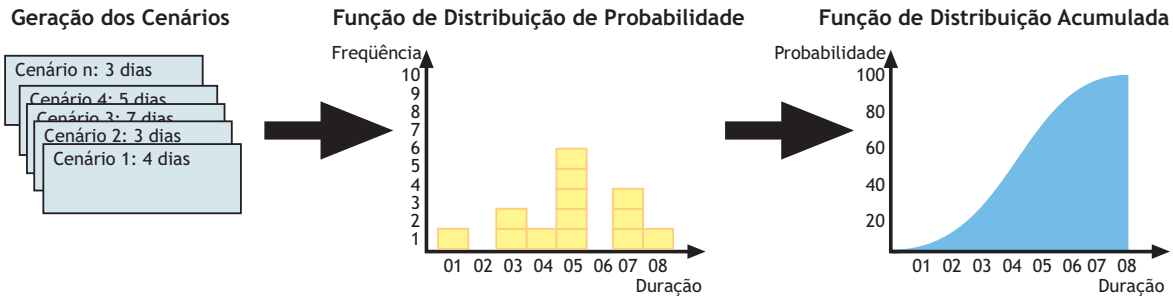

Figura 3.8: Coleta de Amostras

O algoritmo de simulação baseia-se no método de simulação de Monte Carlo (RUBINSTEIN, 1981) onde se executa a rede diversas vezes para se obter amostras do tempo de execução do processo, de forma a obter uma função de distribuição de probabilidade acumulada aproximada do tempo de execução (Figura 3.8). O algoritmo é dividido em três partes:

- Algoritmo de Desdobramento - O objetivo deste algoritmo é obter uma amostra da rede gerando uma instância determinista (DAN). Esta instância contém apenas atividades e transições deterministas.
- Algoritmo RCPS paralelo Padrão - Responsável por escalonar as atividades de acordo com a heurística fornecida. Determina o tempo de início de execução de cada atividade, bem como o tempo total de execução.
- Coletor Estatístico - Responsável por recolher as informações obtidas na execução da rede e disponibilizá-las em um relatório que possibilite a análise dos resultados.

O pseudo código do algoritmo é demonstrado na Tabela 3.1.

```
Simular ( G : Rede De Atividade, NumeroDeAmostras : Inteiro, H : Heurística )
  Faça de i = 1 até NumeroDeAmostras
    Executar Algoritmo Desdobramento em G
    Executar Algoritmo RCPS Paralelo Padrão em DAN
    Coletar Dados Resultado de DAN
  Próximo
```

Tabela 3.1: Algoritmo de Simulação

Em seguida, apresenta-se as três etapas do algoritmo de simulação.

3.4.1 O Algoritmo de Desdobramento

Conforme mencionado anteriormente, o Algoritmo de desdobramento é responsável por obter amostras deterministas (*DAN*) da rede de atividades não determinista (*G*). A Tabela 3.2 apresenta a nomenclatura utilizada para a apresentação do pseudo-código do algoritmo de desdobramento.

Nomenclatura	
N	Nó
T_N	Tipo do Nó
NT_N	Número de execuções do Nó
D_N	Distribuição de probabilidade da duração do Nó
L	Arco
S_L	Origem do arco
D_L	Destino do Arco
P_L	Array de probabilidades do arco
$G(B,E,LN,LL)$	Rede de Atividades
B	Nó inicial
E	Nó Final
LN	Lista de Nós
LL	Lista de Arcos
SU	Lista de nós sucessores
LF	Lista de Arcos para corrigir o destino
NV	Lista de Nós para visitar
AN	Nó atual para visita
CD	Tempo de execução constante

Tabela 3.2: Definições do Algoritmo de Desdobramento

A primeira etapa do algoritmo de desdobramento é inicialização, onde é criada uma nova

instância determinista de rede. Nesta inicialização, são criados os nós inicial e final, de acordo com os nós inicial e final da rede não determinista, e coloca-se o nó inicial da rede não determinista na lista de nós a serem visitados (Tabela 3.3).

$NDAN \leftarrow Gn(Bn, En, LNn, LLn)$, rede a expandir
$DAN \leftarrow$ nova $Gd(Bd, Ed, LNd, LLd)$
$Bd \leftarrow Bn$
$Ed \leftarrow En$
$LNd \leftarrow \emptyset$
$LLd \leftarrow \emptyset$
$NV \leftarrow \emptyset$
$NV \leftarrow Bn$

Tabela 3.3: Inicialização do Algoritmo de Desdobramento

Após a etapa de inicialização, é executado o laço principal do algoritmo, que percorre o grafo G até que não existam mais nós a serem visitados. Este algoritmo é baseado em uma busca em largura, onde cada nó visitado coloca seus filhos no final de uma lista de nós a serem visitados. No laço principal, inicialmente se atribui ao nó atual o primeiro nó na lista de nós a serem visitados, e depois remove-se este nó da lista. Em seguida, a lista de sucessores recebe todos os nós destino dos arcos que possuem como origem o nó atual. O próximo passo é preencher a lista de arcos a serem corrigidos, que são aqueles que possuem como destino o nó atual. A próxima etapa, é executar de acordo com o tipo do nó atual, uma ação diferenciada, que será apresentada adiante. Por fim, a lista de nós a serem visitados é acrescida da lista de nós sucessores. O pseudo-código deste algoritmo é apresentado na Tabela 3.4.

Enquanto $NV \neq$ Vazio
$AN \leftarrow$ primeiro elemento de NV
$NV \leftarrow NV$ - primeiro elemento
$SU \leftarrow Dn$ de Ln — $Sn = AN$
$LF \leftarrow Ld$ — $Dd = AN$
Executar Algoritmo do T_N of AN
$NV \leftarrow NV + SU$
Próximo
retorne DAN

Tabela 3.4: Execução do Algoritmo de Desdobramento

Caso o tipo do nó atual seja *Start*, varre-se a lista de sucessores, criando arcos do nó inicial da rede determinista (DAN) para cada nó da lista de sucessores (Tabela 3.5).

Quando o nó atual for do tipo *End*, seleciona-se cada arco na lista de arcos de DAN que

Caso T_N = Start
  Para cada N em SU faça
  $LLd \leftarrow LLd + L(Bd,N)$
  Próximo

Tabela 3.5: Caso do Tipo Start

possui como destino o nó atual, alterando-o para o nó final da DAN, conforme apresentado na Tabela 3.6.

Caso T_N = End
  Para cada L em LF faça
  $D_L \leftarrow Ed$
  Próximo

Tabela 3.6: Caso do Tipo Fim

No caso em que o nó atual seja uma atividade, amostra-se um valor de sua distribuição do tempo de execução e se incrementa o número de execuções do nó atual. É criado um novo nó com tipo *Activity* com todas as características do nó atual, porém com o tempo de execução constante, com o valor amostrado. Em seguida, varre-se a lista de nós sucessores, criando arcos do nó atual para cada sucessor, e adicionando na lista de arcos da rede determinista. Da mesma forma, para cada arco na lista de arcos a serem corrigidos (que possuíam o destino o antigo nó atual), atualiza-se o destino para o novo nó atual. Por fim, adiciona-se o novo nó atual à lista de nós da rede determinista, conforme demonstrado na Tabela 3.7.

Caso T_N = Activity
  $CD \leftarrow$ obter amostra da D_N de AN
  NT_N de $AN \leftarrow TT_N + 1$
  $AN \leftarrow$ novo nó N
  T_N de $AN \leftarrow$ Activity
  D_N de $AN \leftarrow$ CD
  Para cada N em SU faça
  $LLd \leftarrow LLd + L(AN,N)$
  Próximo
  Para cada L em LF faça
  $D_L \leftarrow AN$
  Próximo
  $LNd \leftarrow LNd + AN$

Tabela 3.7: Caso do Tipo Atividade

Quando o nó atual for do tipo *Fork*, para cada arco que possui o nó atual como destino (arcos a serem corrigidos), varre-se a lista de sucessores, criando-se novos arcos com o nó de origem do arco a ser corrigido, e como destino o nó sucessor. Em seguida remove-se o antigo

arco da lista de arcos a serem corrigidos. Isto faz com que sejam criados arcos de todos os predecessores do nó atual para todos os sucessores, conforme Tabela 3.8.

```
Caso T_N = Fork
  Para cada L em LF faça
    Para cada N em SU faça
      LLd ← LLd + L(S_L,N)
    Próximo
    LLd ← LLd - L
  Próximo
```

Tabela 3.8: Caso do Tipo Fork

No caso em que o nó atual é do tipo *Join* se verifica se todos os predecessores já foram visitados. Caso afirmativo, é executado o mesmo procedimento do nó *Fork*, ou seja, criam-se arcos dos predecessores para todos os sucessores do nó atual. Caso negativo, o nó ainda não pode ser visitado, e por isso, faz-se com que a lista de sucessores contenha apenas o nó atual, para que ele seja adicionado novamente na lista de nós a serem visitados. O pseudo-código é apresentado na Tabela 3.9.

```
Caso T_N = Join
  se todos os predecessores foram visitados então
    Para cada L em LF faça
      Para cada N em SU faça
        L_Ld ← L_Ld + L(S_L,N)
      Próximo
      LLd ← LLd - L
    Próximo
  Caso contrário
    SU ← AN
```

Tabela 3.9: Caso do Tipo Join

Quando o nó atual for do tipo *Split* sorteia-se um arco dentre todos os arcos que possuem origem no nó atual e possuam destino nos nós da lista de nós sucessores. Esse sorteio é feito de acordo com o número de vezes que o nó *Split* foi visitado e as probabilidades associadas a cada arco; o algoritmo de escolha aleatória encontra-se na seção de anexo. Uma vez selecionado o arco, incrementa-se o número de execuções do nó Split, e em seguida o destino do arco selecionado é atribuído ao nó atual. Em seguida, para cada arco na lista de arcos a serem corrigidos, altera-se o destino para o nó atual. Por fim, a lista de sucessores é substituída por apenas o nó atual, conforme demonstrado na Tabela 3.10.

Por fim, quando o nó for do tipo *Merge* sebe-se que apenas um arco dentre todos os que possuem o nó atual como destino foi selecionado, e neste caso, cria-se um novo arco com a

Caso T_N = Split
$L \leftarrow Ln$ aleatório — $S_L n = AN$, $D_L n \subset SU$
NT_N de $AN \leftarrow NT_N + 1$
$AN \leftarrow D_L$ de L Para cada L em LF faça
$D_L \leftarrow AN$
Próximo
$SU \leftarrow AN$

Tabela 3.10: Caso do Tipo Split

mesma origem do arco selecionado, e o sucessor do nó atual como nó destino [2]. O pseudo-código é apresentado na Tabela 3.11.

Caso T_N = Merge
Para cada L em LF faça
Para cada N em SU faça
$LLd \leftarrow LLd + L(S_L, N)$
Próximo
$LLd \leftarrow LLd$ - L
Próximo

Tabela 3.11: caso do Tipo Merge

De forma a explicitar o funcionamento do Algoritmo de Desdobramento, é apresentado a seguir um exemplo da execução do algoritmo na rede de atividades não determinista (RAND) apresentada na Figura 3.9 [3]. Nesta rede o array de probabilidade do arco $\overline{SM}$ é $\{100\%, 0\%\}$, e o array de probabilidade do arco $\overline{SEnd}$ é $\{0\%, 100\%\}$.

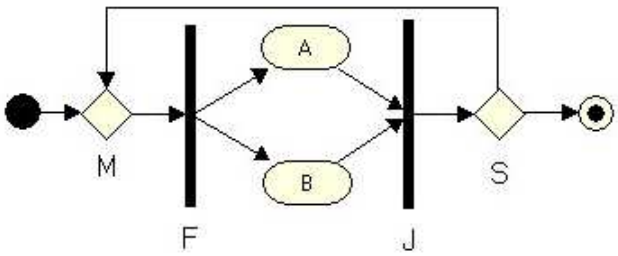

Figura 3.9: Rede não determinista bem formada

Na inicialização do algoritmo cria-se uma rede de atividades determinista (RAD) vazia, onde são criados e adicionados os nós inicial e o nó final (Figura 3.10). A fila de nós a serem visitados (NV) recebe o nó inicial (Start) da RAND.

O nó atual (AN) recebe o nó Start de NV. A lista NV fica então vazia. A lista de Links

[2] Como o arco e sucessor estão dentro de listas, varre-se as duas listas, mesmo contendo apenas um elemento.
[3] A sessão de anexo contém um conjunto de demonstrações da aplicação do algoritmo de desdobramento.

Figura 3.10: Inicialização

a serem corrigidos (LF) não recebe nenhum link, visto que não existem nós predecessores ao elemento Start. A lista de sucessores (SU) recebe o nó Merge. A visita ao nó Start cria um link do nó Start da RAD para o nó Merge da RAND (Figura 3.11). Adiciona-se o nó Merge em NV.

Figura 3.11: Visita ao nó inicial

Em seguida, AN recebe o nó Merge, deixando NV novamente vazia. A lista LF recebe o link $\overline{StartMerge}$. A lista SU recebe o nó Fork. A visita ao nó merge altera o nó destino do link $\overline{StartMerge}$ para o Link $\overline{StartFork}$ (Figura 3.12). Adiciona-se o nó Fork em NV.

Figura 3.12: Visita ao nó Merge

O AN recebe o nó Fork e, novamente, NV fica vazia. LF recebe o link $\overline{StartFork}$. A lista SU recebe as atividades A e B. A visita ao nó fork remove o link $\overline{StartFork}$ e cria dois novos links $\overline{StartA}$ e $\overline{StartB}$ (Figura 3.13). A lista NV recebe as atividades A e B.

O AN recebe a atividade A, deixando NV apenas com a atividade B. A lista LF recebe o link $\overline{ForkA}$. A lista SU recebe o nó Join. Ao visitar a atividade A, amostra-se um valor do seu tempo de execução e cria-se uma atividade A1 com tempo de execução constante com o valor

Figura 3.13: Visita ao nó fork

da duração da amostra obtida. O destino do link $\overline{ForkA}$ é alterado para $\overline{ForkA1}$, e cria-se um novo link $\overline{A1Join}$ (Figura 3.14). A lista NV recebe o nó Join.

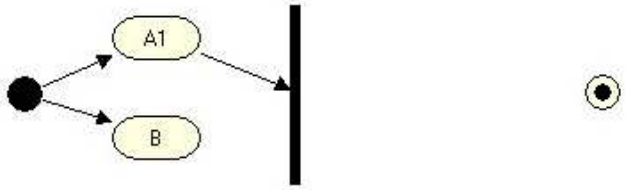

Figura 3.14: Visita a atividade A

Em seguida, AN recebe a atividade B, e NV passa a ter apenas o nó Join. A lista LF recebe o link $\overline{ForkB}$. A lista SU recebe o nó Join. Ao visitar a atividade B, amostra-se um valor de seu tempo de execução e cria-se uma atividade B1 com o tempo de execução constante com valor da duração amostra obtida. O destino do link $\overline{ForkB}$ é alterado para $\overline{ForkB1}$, e cria-se um novo link $\overline{B1Join}$ (Figura 3.15). A lista NV não é alterada, continuando apenas com o nó Join.

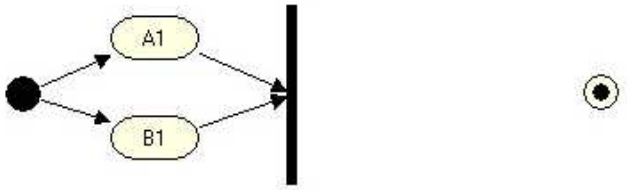

Figura 3.15: Visita a atividade B

O AN recebe o nó Join, deixando a lista NV novamente vazia. A lista LF recebe os links $\overline{A1Join}$ e $\overline{B1Join}$. A lista SU recebe o no Split. Ao visitar o nó Join, é verificado se todos os seus sucessores já foram visitados, neste caso A e B que já foram visitados, e desta forma alter-se o destino dos links em LF obtendo-se os links $\overline{A1Split}$ e $\overline{B1Split}$ (Figura 3.16). A lista NV recebe o nó Split.

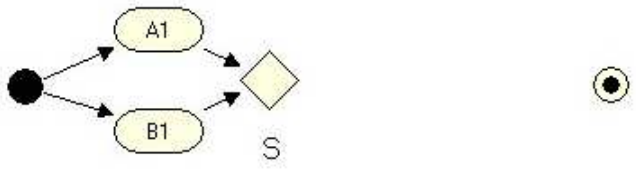

Figura 3.16: Visita ao nó Join

O AN recebe em seguida o nó Split, o que deixa a lista NV vazia. A lista LF recebe os links $\overline{A1Split}$ e $\overline{B1Split}$. A lista SU recebe os nós Merge e End. Ao visitar o nó split sorteia-se um link de acordo com a probabilidade dos arcos $\overline{SplitMerge}$ e $\overline{SplitEnd}$, que são 100% e 0% respectivamente para a primeira execução; e desta forma, é selecionado o arco $\overline{SplitMerge}$ (Figura 3.17). A lista SU é atualizada para conter apenas o nó Merge. O destino dos links em LF são atualizados para $\overline{A1Merge}$ e $\overline{B1Merge}$. A lista NV recebe o nó Merge.

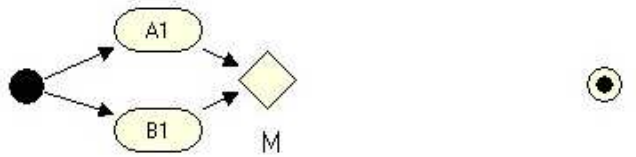

Figura 3.17: Visita ao nó Split

Em seguida, AN recebe o nó Merge. A lista LF recebe os links $\overline{A1Merge}$ e $\overline{B1Merge}$. A lista SU recebe o nó Fork. Ao visitar o nó Merge, substitui-se os destinos dos links em LF para o nó em SU, obtendo-se os links $\overline{AFork}$ e $\overline{BFork}$ (Figura 3.18). A lista NV recebe o nó Fork.

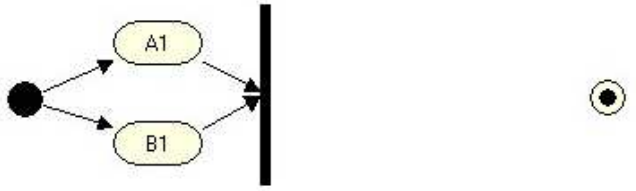

Figura 3.18: Segunda visita ao nó Merge

Ao visitar o nó Fork pela segunda vez (Figura 3.19), ocorre o mesmo procedimento da primeira execução, descrita na Figura 3.13. Desta vez são criados os links $\overline{A1A}$, $\overline{A1B}$, $\overline{B1A}$ e $\overline{B1B}$.

Na visita a atividade A pela segunda vez, ocorrem os mesmos passos de sua primeira visita,

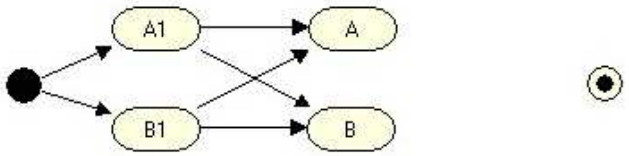

Figura 3.19: Segunda visita ao nó fork

e desta forma, amostra-se um outro valor de sua distribuição e cria-se uma atividade B2 com duração constante igual ao valor amostrado (Figura 3.20).

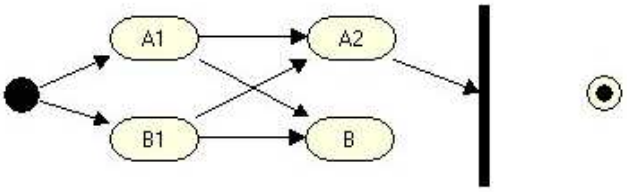

Figura 3.20: Segunda visita a atividade A

O mesmo ocorre com a visita a atividade B, criando uma nova atividade B2 com duração constante amostrada da atividade B (Figura 3.21).

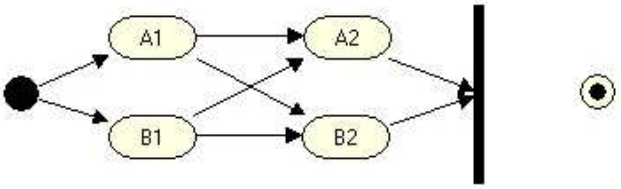

Figura 3.21: Segunda visita a atividade B

Da mesma forma, a segunda visita ao nó Join verifica que todos os predecessores já foram visitados, e, desta forma, altera-se o destino dos links para $\overline{A2Split}$ e $\overline{B2Split}$ (Figura 3.22).

Na segunda visita ao nó Split, os mesmos procedimentos são efetuados, porém as probabilidades de desvio dos links para a segunda execução são 0% e 100% para os links $\overline{SplitMerge}$ e $\overline{SplitEnd}$ respectivamente. Desta forma o sucessor selecionado será o nó End (Figura 3.23).

Por fim, visita-se o nó End, que substitui o nó End da rede não determinista para o nó End criado na inicialização do algoritmo. Esta visita deixa a lista NV vazia, fazendo com que o algoritmo retorne a rede determinista criada.

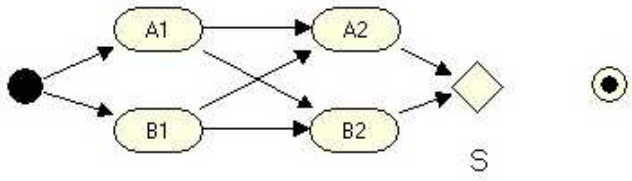

Figura 3.22: Segunda visita ao nó Join

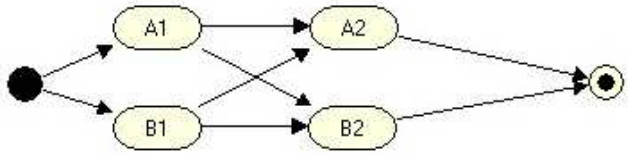

Figura 3.23: Segunda visita ao nó Split

3.4.2 O Algoritmo do RCPS

Uma vez que se tenha obtido uma instância de rede de atividades determinista (DAN), onde só existam arcos determinísticos e atividades com durações constantes, o próximo passo é escalonar as atividades, respeitando suas listas de precedência, de forma a determinar o tempo mais cedo de início de cada atividade.

Para tal, utilizar-se-a um algoritmo baseado no RCPS paralelo padrão, para solução heurística, que consiste na execução dos seguintes passos (OLAGUIBEL; GOERLICH, 1989):

- Inicialização: Atribui ao tempo atual (t) o valor zero; esvazia a lista de recursos em uso (RU); esvazia a lista de atividades em execução (EX); esvazia a lista de atividades terminadas (DN); atribui zero para o tempo de início da atividade inicial; e coloca a atividade inicial em DN.

- Construção das Elegíveis (ELG): Cria uma lista de atividades candidatas (CA) com as atividades que possuem todos os seus predecessores em DN.

- Construção das Disponíveis (AVL): Remove-se todas as atividades em CA que não possuem todos os recursos necessários para sua execução disponíveis.

- Construção do Escalonamento: Ordena-se todas as atividades em CD de acordo com a heurística (h) fornecida. Em seguida, coloca-se as atividades em execução uma a uma, até que não existam recursos disponíveis(RA) suficientes para executar uma atividade.

- Determinação do novo tempo: Seleciona-se o tempo mais cedo de término dentre todas as atividades em execução, removendo todas as atividades em EX que possuem término igual ao tempo determinado, colocando-as em DN; liberam-se todos os recursos que estavam sendo utilizados por estas atividades.

- Parar: Quando a lista de atividades terminadas (DN) contiver a atividade final, todas as atividades foram escalonadas; o algoritmo terminou.

Para o escalonamento das atividades, é necessário a utilização de heurísticas por se tratar de um problema NP-completo, conforme apresentado anteriormente. Para o algoritmo do RCPS paralelo padrão são apresentadas quatorze heurísticas na literatura (OLAGUIBEL; GOERLICH, 1989). Destas heurísticas, seis foram incorporadas ao método SMERA, que são:

- *First In First Out* (FIFO): Ordena as atividades de acordo com sua ordem de chegada, as primeira a entrarem na fila serão as primeira a saírem;

- *Least Immediate Successors* (LIS): Ordena as atividades com menor número de sucessores imediatos em primeiro lugar;

- *Most Immediate Successors* (MIS): Ao contrário de LIS, ordena as atividades com maior número de sucessores imediatos na frente da fila;

- *Longest Processing Time* (LPT): Coloca as atividades com maior tempo de execução no início da fila;

- *Shortest Processing Time* (SPT): Oposto de LPT, colocar no início da fila as atividades com o menor tempo de processamento;

- *Random* (RND): Escolhe do forma aleatória qual atividade terá maior prioridade que as demais.

O pseudo-código é apresentado na Tabela 3.12:

3.4.3 O coletor estatístico

O coletor estatístico é responsável por disponibilizar as informações da simulação. Ele recebe como entrada a rede de atividades determinista (DAN) com as atividades escalonadas, e fornece um documento do Excel, no mesmo diretório onde se encontra o diagrama da UML, com os seguintes relatórios: Tempo de Execução, Tempo de Espera em Fila e Arcos da Rede.

```
t    ← 0
RU ← ∅
EX ← ∅
DN ← ∅
Enquanto End ∋ DN faça
    CA ← a ∈ A|∀pred de a ⊂ DN
    Se CA ≠ ∅ então
        CA ← CA − a ∈ A|∃R_k de a > R_k de RA
        Se CA ≠ ∅ então
            Ordenar CA de acordo com h
            Para cada a ∈ CA faça
                se ∀R_k de a ≤ R_k de RA
                    EX ← a
                    t_a ← t
                    RU ← RU + R_a
                senão
                    Fim Para
            Próximo
        Fim Se
    Fim Se
    t ← min(t_a + d_a|a ∈ EX)∀A
    EX ← EX − a|t_a + d_a = t
    DN ← DN + a|t_a + d_a = t
    RU ← RU − R_a|t_a + d_a = t
    RA ← RA + R_a|t_a + d_a = t
Próximo
```

Tabela 3.12: Algoritmo RCPS Paralelo Padrão

O primeiro relatório, Tempo de Execução, exibe na primeira coluna o tempo mínimo necessário para a execução do cenário. As demais colunas apresentam o tempo de execução de cada atividade. Cada linha do relatório corresponde a um cenário amostrado, com exceção da primeira linha, que apresenta o nome de cada atividade pertencente a rede.

O segundo relatório, Tempo de Espera em Fila, apresenta o tempo que cada atividade ficou aguardando antes de conseguir entrar em execução, ou seja, o tempo de quando ela entrou em EX - tempo de quando ela entrou pela primeira vez em CA. Considera-se desta forma, o tempo de espera em fila, o tempo transcorrido do momento em que todos os predecessores de uma atividade terminam sua execução, até o momento em que a atividade inicia sua execução. Na primeira coluna apresenta-se o tempo total de espera em fila, que representa o somatório total dos tempos de espera das atividades; e nas demais colunas, apresenta-se o tempo de espera em fila de cada atividade. da mesma forma que o primeiro relatório, a primeira linha apresenta o nome das atividades, e as demais os dados dos cenários.

O terceiro relatório, Arcos da Rede, expõe todos os arcos na forma de: nome do nó de origem - nome do nó destino; se iniciando no nó inicial indo em direção ao nó final. Este relatório viabiliza construir a rede determinista que fora construída, de forma a verificar os caminhos que foram tomados durante a simulação da rede não determinista e com isso fazer um levantamento estatístico da probabilidade de ocorrência de cada caminho. Neste relatório, cada coluna corresponde a um arco, e cada linha corresponde a um cenário amostrado.

3.5 Determinando o número de amostras

O número de cenários necessários a serem gerados, depende de dois parâmetros estatísticos: o intervalo de confiança (α) necessário tal que a diferença entre a distribuição amostrada e a real (desconhecida) esteja dentro de um valor máximo aceito de erro (D).

O número mínimo de cenários pode ser determinado pelo teorema de Kolmogorov-Smirnov (KS), onde seja $Sn(x)$ a função de distribuição acumulada com n amostras de uma variável aleatória X, $F(x)$ a sua função de distribuição acumulada real, e D a diferença máxima entre $Sn(x)$ e $F(x)$; pode se expressar D como (FELLER, 1948):

$$D = max(|F(X) - Sn|)$$

Seja $Dn,\alpha (0 \leq Dn,\alpha \geq 1)$ a diferença máxima entre $Sn(x)$ e $F(x)$ com intervalo de confiança igual a α; a probabilidade de obter um valor D que é menor ou igual a Dn,α é dado por:

$$P(D \leq Dn,\alpha) = 1 - \alpha$$

$$P(max(|F(X) - Sn|) \leq Dn,\alpha) = 1 - \alpha$$

$$P(Sn(X) - Dn,\alpha \leq F(X) \leq Sn(X) + Dn,\alpha) = 1 - \alpha \ \ \forall x$$

De acordo como o teorema de KS, a relação entre as variáveis n,α e Dn,α, para $n > 50$, tende a um valor limitante, que é demonstrado na Tabela 3.13, que mostra os valores de Dn,α para o intervalo de confiança de 90%, 95% e 99%.

$(1-\alpha)$	**0.1**	**0.05**	**0.01**
$D_{n,\alpha}$	$1.07/\sqrt{n}$	$1.36/\sqrt{n}$	$1.63/\sqrt{n}$

Tabela 3.13: Número mínimo de amostras necessárias

A Tabela 3.13 mostra que para se obter um erro máximo Dn,α de 2% com 95% de confiança, são necessários serem amostrados pelo menos 4649 amostras.

3.6 O Simulador de Processos

Como prova do funcionamento do método, desenvolveu-se a ferramenta *Process Simulator*, que implementa o método SMERA. Esta ferramenta foi projetada para trabalhar de forma integrada com o projeto RAPDIS (MORGADO et al., 2006), possibilitando sua utilização dentro do ambiente de modelagem de processos.

Optou-se pela utilização da linguagem JAVA (SUN, 2006) para a implementação da ferramenta por sua ampla utilização no mercado, e também por sua integração com ferramentas de testes, que visam garantir a aderência da implementação ao método. A ferramenta pode ser baixada gratuitamente no site do GETI (GETI, 2007).

O manual de utilização da ferramenta, bem como a documentação dos casos de teste se encontram na seção de Anexo.

4 Estudo de Caso

De forma a verificar a cobertura dos atuais métodos de simulação de processos, fez-se um estudo de levantamento da modelagem de processos nas disciplinas de Fundamentos da Engenharia de Software (período 2005-2 e 2006-2) e Modelagem de Sistemas de Informação II (período 2006-1) do curso de Bacharelado em Ciência da Computação da Universidade Federal do Rio de Janeiro (UFRJ), e no curso de Engenharia de Software (período 2006-2) do curso de Mestrado em Informática da UFRJ.

Nestas disciplinas os alunos deveriam desenvolver o modelo de processo de uma fábrica de software que possuísse todas as atividades necessárias para obtenção de CMM nível 2 (CHRISSIS; KONRAD; SHRUM, 2004). De posse destes modelos, verificou-se que os atuais métodos de simulação não poderiam ser utilizados devido a duas características presentes em todos os modelos: laços sobrepostos e cruzamentos.

Conforme apresentado anteriormente, o método proposto viabiliza a formação de cruzamentos, e para ilustrar o funcionamento do método com laços sobrepostos, vamos estimar o tempo de execução de um processo baseado no caso da RANDRE patológica (Figura 2.6) que não era passível de simulação, e fora obtido na análise dos modelos apresentados. Aplicando as regras de formação apresentadas, o novo diagrama do processo é apresentado na Figura 4.1.

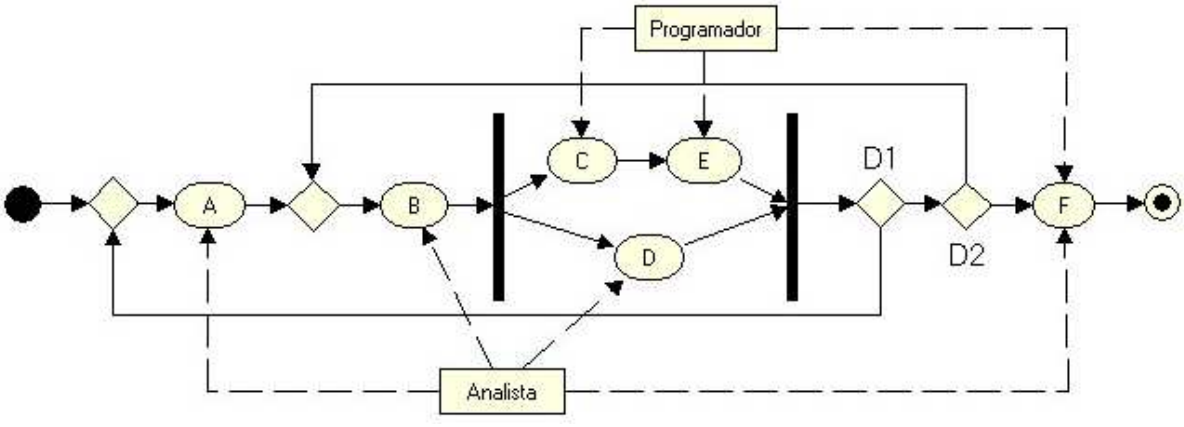

Figura 4.1: Diagrama da RANDRE Patológica bem formado

Ressalta-se, que este modelo de processo bem formado continua não sendo passível de simulação pelos métodos existentes (GERT, CAAN e RANDRE). Em seguida apresenta-se a descrição das atividades deste modelo.

4.1 O modelo da Fábrica de Software

O processo descrito na Figura 4.1 representa o processo de desenvolvimento de software, que vem sendo o assunto de um grande número de estudos, principalmente após a publicação de (ROYCE, 1970), e também por causa do modelo de aperfeiçoamento do processo de desenvolvimento de software que foi definido no CMMI, que prove um guia para o aperfeiçoamento contínuo do processo de desenvolvimento (CHRISSIS; KONRAD; SHRUM, 2004).

Mais recentemente, duas abordagens distintas se destacam: o Processo Unificado (PU), adotado pela OMG, que define uma extensiva lista de atividades, como por exemplo o Processo Unificado da Rational (Rational Unified Process - RUP); e as metodologias ágeis que definem um processo simplificado, como por exemplo XP e SCRUM (AMBLER, 2002). Este processo (Figura 4.1) é baseado no modelo cascata descrito em (PRESSMAN, 2004), contendo apenas as atividades técnicas.

O Processo se inicia com o levantamento de requisitos, seguido pela atividade de modelagem do sistema. Em seguida, duas atividades são disparadas em paralelo, a atividade de implementação do código e a atividade de SQA (Garantia de Qualidade de Software). Ao término da atividade de implementação se inicia a atividade de testes. Quando encerrada a atividade de testes, pode ser necessária alguma alteração na modelagem do sistema. Da mesma forma, caso a equipe de SQA verifique a existência de problemas no levantamento de requisitos, o fluxo pode ser reiniciado com a atividade de levantamentos de requisitos. Quando ambos os fluxos alcançam a barra de junção, a atividade de Integração será executada. A Tabela 4.1 mostra a relação dos nomes para os códigos das atividades.

Código	**Atividade**
A	Levantamento de Requisitos
B	Modelagem
C	Implementação
D	Gerenciamento SQA
E	Teste
F	Integração

Tabela 4.1: Nomes das Atividades

4.1.1 Definindo a Duração das Atividades

A duração de cada atividade em uma fábrica de software varia de acordo com o tamanho do projeto a ser desenvolvido. Por este motivo, surgiram as medidas de tamanho de software, como por exemplo, os Pontos de Função (PF) que medem o tamanho funcional do projeto a ser desenvolvido. A partir de 2002 os PF passaram a condição de padrão internacional, através da norma ISO/IEC 20926 (BFPUG, 2007).

Segundo (JONES, 1995) um projeto de desenvolvimento de software de tamanho médio está compreendido de 100 até 1000 Pontos de Função. Escolhendo um valor próximo a média, será utilizado neste estudo de caso um projeto com 500 PF.

Segundo (JONES, 1996) a média de horas trabalhadas por pontos de função em um projeto de desenvolvimento de software pode ser distribuída segundo a Tabela 4.2.

Código	**Atividade**	**HT/PF**
1	Requisitos	0,75
2	Protótipo	0,9
3	Projeto	1,5
4	Codificação	2,5
5	Teste(integração)	0,9
6	Teste(aceitação)	0,4
7	Implantação	0,4
8	Gerenciamento do Projeto	1,5

Tabela 4.2: Horas de Trabalho por Pontos de Função

Mapeando as atividades da Tabela 4.2 para as atividades descritas na Tabela 4.1 e multiplicando pelo tamanho do projeto (500 PF), será obtido o número de horas de trabalho necessárias para cada atividade, conforme Tabela 4.3.

Atividade	**Código**	**HT/PF**	**Total HT**
A	1	0,75	500
	8	0,25	
B	2	0,90	1325
	3	1,50	
	8	0,25	
C	4	2,50	1250
D	8	1,00	500
E	5	0,90	650
	6	0,40	
F	7	0,40	200

Tabela 4.3: Total de Horas de Trabalho

Assumindo que os recursos disponíveis para a execução deste processo são dois Analistas e dois programadores, pode-se alocá-los nas atividades e dividir o número de horas de trabalho necessárias para cada atividade pelo número de recursos alocados, obtendo-se o tempo de execução mais provável de cada atividade. Considerando uma variação de 20% na duração das atividades, obtém-se a distribuição triangular de probabilidade da duração de cada atividade conforme mostra a Tabela 4.4.

Atividade	Recursos	Mínimo	Provável	Máximo
A	2 Analistas	200	250	300
B	2 Analistas	530	662,5	795
C	2 Programadores	500	625	750
D	2 Analistas	200	250	300
E	2 Programadores	260	325	390
F	2 Analistas 2 Programadores	40	50	60

Tabela 4.4: Distribuição de Probabilidade das Atividades

4.1.2 Definindo a probabilidade dos desvios

O desvio $D1$ corresponde à verificação se todos os requisitos do sistema foram captados de forma correta, caso exista a necessidade de correção da especificação dos requisitos, é necessário a re-execução da atividade de levantamento de Requisitos. Para esta simulação, assume-se que a probabilidade de ocorrer um erro de especificação é de 50%, e que depois não ocorrem mais estes tipos de erro. As probabilidades do desvio $D1$ são apresentados na Tabela 4.5.

Execução	Retorno	Seguir
Primeira	50 %	50 %
Segunda	0 %	100 %

Tabela 4.5: D1: Probabilidades de Desvio

O desvio $D2$ corresponde a erro de modelagem e prototipagem, que obrigam a re-execução da atividade de modelagem. Para esta simulação, assume-se que a probabilidade de ocorrer um erro na primeira execução da atividade de modelagem é de 50%, e que ainda possam existir erros que não foram tratados, ou que foram ocasionados pela alteração do modelo, de forma que existe ainda a possibilidade de serem encontrados erros, que tornará necessário uma nova re-execução da atividade de modelagem, que neste caso possui probabilidade de 30%. As probabilidades do desvio $D2$ são apresentadas na Tabela 4.6.

Execução	Retorno	Seguir
Primeira	50 %	50 %
Segunda	30 %	70 %
Terceira	0 %	100 %

Tabela 4.6: D2: Probabilidades de Desvio

4.2 Simulação e Resultados

Utilizando-se o *Process Simulator* para a simulação deste modelo, optou-se pela utilização da heurística FCFS, por ser uma prática comum em fábricas de Software, onde as demandas são atendidas por ordem de chegada. Para o parâmetro de número de amostras, utilizou-se o valor de 4649, que conforme apresentado, é o valor mínimo para se obter uma função de distribuição acumulada com no máximo 2% de erro e com 95% de confiança.

Utilizando-se o relatório de tempo de execução gerado pelo Coletor estatístico, foram gerados os gráficos de distribuição de probabilidade (barras) e distribuição acumulada (linha) para os valores totais obtidos em cada cenário. Os gráficos são apresentados na Figura 4.2, onde estão representados nos eixos das abscissas o tempo de execução, e no eixo das ordenadas o número de ocorrências (lado esquerdo) e a probabilidade acumulada (lado direito).

De posse deste gráfico, cabe ao gerente do projeto decidir qual o risco desejado. Um corte comumente utilizado é o corte 80/20, onde 80% dos cenários estão cobertos na estimativa, que neste caso corresponde ao valor de 5300 horas.

Considerando um dia de trabalho com oito horas de expediente, basta dividir o número de horas encontrado (5300) e dividi-lo por oito, obtendo-se o número de dias necessários para a execução do processo, que neste caso será aproximadamente 663 dias.

Além do relatório de tempo de execução, destaca-se, neste caso, o relatório de arcos, onde estão descritas as redes formadas em cada cenário. Agrupando os caminhos idênticos, e totalizando-os, obtém-se a probabilidade de ocorrência de cada cenário, conforme apresentado na Tabela 4.7.

D1	D2	Cenários	Total
1 Falha	2 Falhas	369	7,38%
1 Falha	1 Falha	865	17,30%
1 Falha	0 Falhas	1220	24,40%
0 Falhas	2 Falhas	399	7,98%
0 Falhas	1 Falha	872	17,44%
0 Falhas	0 Falhas	1275	25,50%

Tabela 4.7: Cenários Gerados e Ocorrência

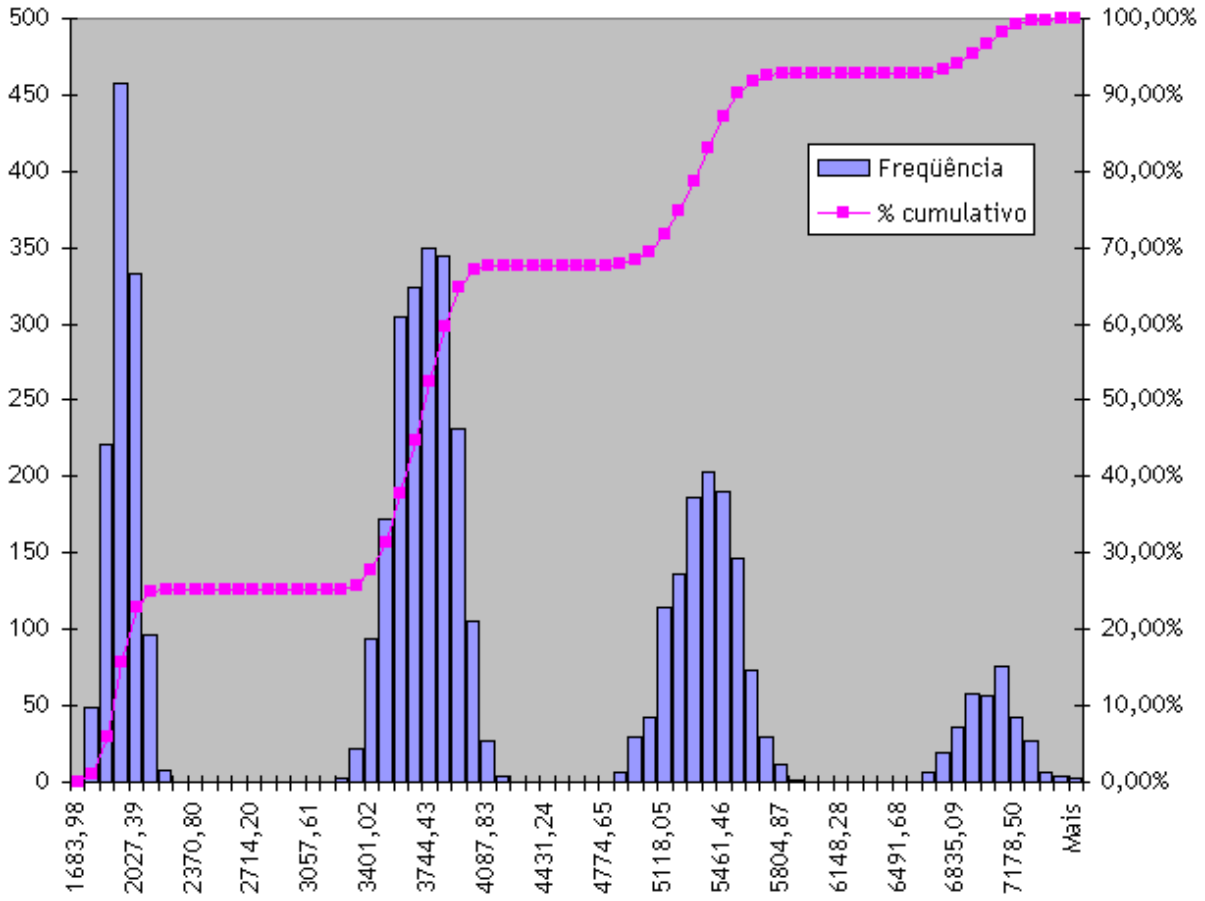

Figura 4.2: Distribuição de Probabilidade

Conforme se esperava, foram obtidos seis cenários distintos, onde suas probabilidades de ocorrência estão condizentes com as probabilidades informadas nos desvio. Por exemplo, a probabilidade de não ocorrerem falhas na etapa de requisitos (desvio $D1$) é de 50%, e de não ocorrem falhas na modelagem (desvio $D2$) é de 50%, logo a probabilidade não ocorrerem falhas é de 25%. Este resultado esperado está de acordo com o resultado encontrado que foi de 25,5%, visto que o número de amostras gerados eram suficientes para garantir um erro de 2% com 95% de confiança.

5 *Conclusões*

Ao longo dos anos, a UML se tornou uma linguagem padrão para a modelagem e especificação de processos, de forma que grande parte das empresas construíram a modelagem de seus processos nos diagramas de atividades da UML. Além disso, a competição entre as empresas forçou-as a aprimorarem seus processos, e a simulação se mostrou ser uma ferramenta poderosa para análise comportamental e de resultados.

Infelizmente, os métodos de simulação de processos, até hoje desenvolvidos, não permitem uma construção abrangente, visto que são impostas restrições de cruzamentos e laços sobrepostos, inviabilizando a construção de modelos complexos.

Para a construção de um novo método de simulação de processos baseados na UML fez-se necessário a determinação de regras de formação dos modelos descritos nos diagramas de atividades. De posse dos diagramas bem formados, fez-se a transformação para uma estrutura de grafo, e desenvolveu-se um novo algoritmo de simulação que consiste na repetição de um algoritmo de desdobramento, um algoritmo de escalonamento e um coletor estatístico.

Para demonstrar o correto funcionamento do método, desenvolveu-se uma ferramenta, que fora utilizada no estudo de caso para estimar o tempo de execução de um processo, que não é passível de simulação pelos demais métodos existentes, até o momento.

Nas seções seguintes, são apresentadas a discussão sobre o método apresentado, as contribuições deste trabalho, e os trabalhos futuros que são necessários para cobrir alguns pontos que ficaram em aberto.

5.1 Discussão

Abaixo respondem-se algumas das perguntas fundamentais relativas à utilização do método, e de técnicas descritas nesta dissertação sobre a estimativa do tempo de execução de processos baseados na UML.

5.1.1 Quais são as vantagens de se utilizar um método baseado na UML para se estimar o tempo de execução de um processo?

Com o objetivo de aprimoramento da eficiência e a redução de custos, na última década, a industria e o mundo acadêmico empenharam-se com grande esforço para o desenvolvimento de uma linguagem padrão para a modelagem de processos. Neste respeito, a UML apresenta uma das soluções mais atraentes para permitir que os analistas de processo e desenvolvedores de software possam falar o mesmo idioma. Não surpreendentemente, a UML tornou-se um padrão não só para sistemas de informação na indústria de software, como para toda a modelagem de processos.

Além disso, há um número considerável de livros, artigos, programas de treinamento e material de aprendizagem de distância que ajudam o novato e os peritos a dominarem qualquer aspecto da UML. Sem mencionar a grande variedade de ferramentas que fazem a utilização da UML mais rápida e menos propensa a erros.

Desta forma, o desenvolvimento de um método baseado na UML, é construído sob uma linguagem de modelagem e de especificação extensamente utilizada onde os processos já estão mapeados e descritos através dos diagramas de atividades. Assim sendo, não é necessária a construção de um novo modelo de processo para que possa fazer uso do poder da simulação.

5.1.2 Como o método auxilia na estimativa do tempo de projetos?

O gerenciamento do tempo de um projeto inclui os processos necessários para a conclusão do projeto no prazo, que são (PMI, 2004):

- **Definição da atividade -** identificação das atividades específicas do cronograma que precisam ser realizadas para produzir as várias entregas do projeto.
- **Seqüenciamento de atividades -** identificação e documentação das dependências entre as atividades do cronograma.
- **Estimativa de recursos da atividade -** estimativa do tipo e das quantidades de recursos necessários para realizar cada atividade do cronograma.
- **Estimativa de duração da atividade -** estimativa do número de períodos de trabalho que serão necessários para terminar as atividades individuais do cronograma.
- **Desenvolvimento do cronograma -** análise dos recursos necessários, restrições do cronograma, durações e seqüências de atividades para criar o cronograma do projeto.

- **Controle do cronograma -** controle das mudanças no cronograma do projeto.

Destas etapas descritas, a definição de cada atividade, bem como o seqüenciamento e precedências entre elas estão descritas dentro dos diagramas de atividades da UML. A estimativa de recursos da atividade e a estimativa de duração da atividade foram introduzidos como parte das informações complementares nos diagramas de atividades através do projeto RAPDIS (MORGADO et al., 2006). O desenvolvimento do cronograma é gerado pelo algoritmo de simulação que gera amostras com a duração do processo em cada cenário amostrado, bem como a duração de cada atividade. A última etapa, a de controle de cronograma, é de responsabilidade do gerente do processo, que deve acompanhar sua execução ao longo do tempo transcorrido, e avaliar o resultado das simulações obtidas.

Desta forma, todas as etapas do gerenciamento do tempo de projetos foram mapeadas pelo método proposto, com exceção do controle de cronograma que é uma tarefa do gerente do processo.

5.1.3 Como o método proposto nesta dissertação favorece o aperfeiçoamento do tempo de execução de um processo?

O método é construído em um modelo matemático que mais realisticamente reflete o ambiente complexo e dinâmico no qual o processo acontece, onde tempo de execução de atividade, disponibilidade de recursos e a execução de algumas atividades é incerta. Em tal um ambiente os tempos de execução calculados de processos são funções de distribuição de probabilidade, refletindo todas as incertezas que afligem o processo. Além disso, a simulação é uma ferramenta barata e poderosa para processo ”e se”de análise.

Fazendo com que o desenvolvedor modele, analise e mude processos de um modo mais fácil e realístico, o método proposto nesta dissertação favorece a redução contínua de tempo gasto no processo de melhoria e de redução do tempo de execução de processos.

5.1.4 Como escolher entre várias alternativas de aprimoramento?

O método proposto nesta dissertação amostra o universo de todos os possíveis cenários que descrevem o comportamento de um processo, onde cada cenário é composto de um elemento particular de uma função de distribuição de probabilidade que descreve o comportamento das atividades e a duração total do processo que são figurados com o apoio de uma política de execução. Registrando-se um número grande de cenários, pode ser construída uma função de

distribuição de probabilidade que descreve o tempo de execução de processo.

Em geral, porque cada alternativa de melhoria de processo concebida tem que ser simulada e cada uma rende uma função de distribuição de probabilidade diferente, é provável que os analistas do processo terminem em uma situação onde eles têm que escolher a uma entre várias possíveis funções de distribuição de probabilidade, onde cada um tem sua própria média e desvio padrão.

Embora este problema esteja no reino da teoria de decisão, se a pessoa estabelece uma variação máxima da média que é aceitável, como por exemplo um risco máximo que ela está disposta a correr para que os processos terminem de fato dentro de um determinado intervalo, a escolha entre as alternativas será aquela que tem a menor média.

5.1.5 Quais são as implicações deste método nas táticas e estratégias das companhias?

Se uma companhia obtiver processos eficientes e que rendam algumas economias significantes no custo, pode-se utilizar esta economia para abrir novos mercados que não puderam ser alcançados devido a restrição de custo, ou ainda ampliar o seu *market share*. Alternativamente, a tal companhia pode passar alguma parte desta economia, que fora obtida como resultado da melhoria do processo, aos seus clientes, reduzindo o custo de seus produtos ou serviços, e desta forma, se tornando mais competitiva.

Assim sendo, o presente método apresentado nesta dissertação favorece a concepção e o desenvolvimento de muitas táticas empresariais e estratégias que não poderiam ser feitas caso contrário.

5.2 Contribuição

Nesta dissertação desenvolveu-se um método para a obtenção da função de distribuição de probabilidade acumulada do tempo de execução de um processo descrito na UML. Este método viabilizou a simulação de redes de atividades complexas que possuíam algumas características, como laços sobrepostos e cruzamentos, que não possibilitavam a utilização dos algoritmos de simulação de processos existentes até então.

Deste trabalho pode se destacar como principais contribuições:

- O algoritmo do simulação que devido ao novo algoritmo de desdobramento desenvolvido

possibilitou a simulação de processos com as características já apresentadas;

- As regras de formação que removem a ambigüidade dos diagramas de atividades;
- O coletor estatístico que disponibiliza os resultados de uma simulação em um arquivo do Excel, de forma que os resultados possam ser interpretados e analisados de forma simples;
- Uma ferramenta que viabiliza a simulação dos processos descritos nos diagramas de atividades da UML, de fácil utilização, e que trabalha de forma integrada com uma ferramenta de modelagem.

5.3 Trabalhos Futuros

Como pontos de extensão para esta dissertação pode se destacar:

- Elaborar um estudo de outras heurísticas importantes para serem utilizadas e incorporar ao Simulador de Processos.
- Identificar dados adicionais sobre a simulação do processo que são importantes para análise do processo e adicioná-las ao Coletor Estatístico.
- Analisar os modelos de Custo utilizados em gerenciamento de projetos e incorporá-los ao modelo de simulação, viabilizando a utilização de outras funções objetivo.
- Aplicar o método SMERA a um processo real, utilizando-o para aprimorar o processo, e comparar os resultados encontrados pela simulação com os obtidos no processo real.
- Estudar uma maneira de simular múltiplas requisições de execuções do processo, de forma a permitir a análise do comportamento do projeto em situações de estresse onde os recursos se tornam ainda mais escassos.
- Desenvolver um algoritmo de validação dos diagramas de atividades da UML de forma a identificar o respeito as regras de formação, apresentando os pontos dos diagramas onde às regras são violadas.

Referências Bibliográficas

AMBLER, S. *Agile Modeling*. USA: John Willey & Sons, 2002.

BFPUG. *Brazilian Function Point Users Group*. Disponível em : http://www.bfpug.com.br/, 2007. Acesso em: 10 de Outubro de 2007.

BLAZEWICZ, J.; LENSTRA, J. K.; KAN, A. H. G. R. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, v. 5, p. 11–24, 1983.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. *CMMI Guidelines for Process Integration and Product Improvement*. 1. ed. USA: Addison Wesly, 2004. ISBN 0321154967.

CONTRIBUTORS, E. et al. *The Eclipse Project*. Disponível em: http://www.eclipse.org, 2007. Acesso em: 27 de Fevereiro de 2007.

DEMEULEMEESTER, E. e. a. *A classification scheme for project scheduling*. Dordrecht: Kluwer, 1999.

DEMEULEMEESTER, E. L.; HERROELEN, W. S. *Stochastic project scheduling*. 1. ed. USA: Springer, 2002. 535-588 p. ISBN 1402070519.

ERIKSSON, H. E.; PENKER, M. *Business Modeling With UML: Business Patterns at Work*. 1. ed. USA: Wiley, 2000. ISBN 0471295515.

ESHUIS, H.; WIERINGA, R. J. *A Formal Semantics for UML Activity Diagrams - Formalising Workflow Models*. Enschede, The Netherlands, 2001. 40 p.

ESSHUIS, R. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. 240 p. Tese (Doutorado) — Centre for Telematics and Information Technology (CTIT), Department of Computer Science, Enschede, The Netherlands, 2002.

FELLER, W. On the kolmogorov-smirnov limit theorems for empirical distributions. *The Annals of Mathematical Statistics*, v. 19, n. 2, p. 177–189, Junho 1948.

GAMMA, E.; BECK, K. *JUnit*. Disponível em : http://junit.sf.net, 2006. Acesso em: 26 de Setembro de 2006.

GETI. *Gestão Estratégica de Tecnologia da Informação - GETI*. Disponível em: http://www.geti.dcc.ufrj.br, 2007. Acesso em: 01 de Março de 2007.

GOLENKO-GINZBURG, D. Controlled alternative activity networks in project management. *European Journal of Operational Research*, v. 7, p. 336–346, 1998.

GOLENKO-GINZBURG, D.; GONIK, A.; LASLO, Z. Resource constrained scheduling simulation model for alternative stochastic network projects. *Mathematics and Computers in Simulation*, v. 63, p. 105–117, 2003.

HAMMER, M.; CHAMPY, J. *Reengenharia: revolucionando a empresa*. 5. ed. Rio de Janeiro, RJ: Campus, 1994.

HARTMANN, S.; KOLISCH, R. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 127, p. 394–407, 2000.

HERBERT, J. E. Applications of simulation in project management. *Winter Simulation Conference*, 1979.

HERROELEN, W.; LEUS, R. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, v. 165, p. 289–306, 2005.

JONES, C. *Patterns of large software systems: failure and success*. 1. ed. Burlington, MA, USA: Software Productivity Res. Inc., 1995. ISBN 0018-9162.

JONES, C. *Applied software measurement: assuring productivity and quality*. 2. ed. Hightstown, NJ, USA: McGraw-Hill Inc, 1996. ISBN 0-07-032826-9.

KLING, D. V. *Um método para simulação de redes de atividades não-deterministas com recursos escassos*. Dissertação (Mestrado) — Instituto de Matemática, Núcleo de Computação Eletrônica, Mestrado em Informática, Rio de Janeiro, RJ, Fevereiro 2005.

KOLISCH, R.; HARTMANN, S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 2005.

LENSTRA, J.; RINNOOY, K. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 1979.

MACEDO, R. S. *Validação da sintexe e da semântica estatica de modelos de processo*. Dissertação (Mestrado) — Mestrado em Informática, Rio de Janeiro, RJ, 2003.

MOORE, L. J.; CLAYTON, E. R. *GERT Modeling and Simulation: Fundamentals and Applications*. Melbourne: Krieger Publishing Co., 1976. 236 p.

MORGADO, G. et al. Rapdis: Um processo mda para o desenvolvimento de sistemas de informação. *Simpósio Brasileiro de Sistemas de Informação*, Curitiba, SC, Novembro 2006.

OLAGUIBEL, R. A. V.; GOERLICH, J. M. T. *Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and Empirical Analysis*. Amsterdam, Netherlands: Elsevier Science Publishers B. V., 1989. 113-133 p. ISBN 0444873589.

OMG. *Business Modeling and Integration Domain Task Force*. Disponível em: http://bmi.omg.org/index.htm/, 2006. Acesso em: 23 de Março de 2006.

OMG. *UML Resource Page*. Disponível em: http://www.uml.org/, 2006. Acesso em: 23 Março de 2006.

PMI. *Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos (Guia PMBOK)*. 3. ed. Pensilvania, EUA: Project Management Institute, Inc, 2004. ISBN 1930699743.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 6. ed. USA: McGraw-Hill Science/Engineering/Math, 2004. ISBN 007301933X.

PRITSKER, A. A. B. *GERT: GRAPHICAL EVALUATION AND REVIEW TECHNIQUE*. 1700 Main St, Santa Monica, California, Abril 1966.

PRITSKER, A. A. B. Graphical evaluation and review technique. *Journal of Industrial Engineering*, May 1966.

ROYCE, W. W. Managing the developement of large software systems: Concepts and techiniques. *Proceedings WesCon*, Agosto 1970.

RUBINSTEIN, R. Y. *Simulation and the Monte Carlo Method*. [S.l.]: Wiley, 1981. ISBN 978-0-471-08917-9.

SUN. *JAVA Technology - The Source for Java Developers*. Disponível em: http://java.sun.com, 2006. Acesso em: 24 de Fevereiro de 2007.

VALLS, V.; BALLEST, F.; QUINTANILLA, S. Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, v. 165, p. 375–386, 2005.

WIEST, J. D.; LEVY, F. K. *A Management Guide to Pert/Cpm*. [S.l.]: Prentice Hall, 1969. ISBN 013548586X.

YANG, B.; GEUNES, J.; O'BRIEN, W. *Resource Constrained Project Scheduling: Past Work and New Directions*. Enschede, The Netherlands, April 2001.

ZAMANI, M. R. A high-performance exact method for the resource-constrained project scheduling problem. *Computers & Operations Research*, v. 28, p. 1387–1401, 2001.

Anexo

Manual do Simulador de Processos

O Simulador de Processos (*Process Simulator*) é uma aplicação desenvolvida na plataforma Java (SUN, 2006). Para executar o Simulador de Processos é necessário que se tenha a Máquina Virtual Java (Java Virtual Machine - JVM) versão 5 ou superior.

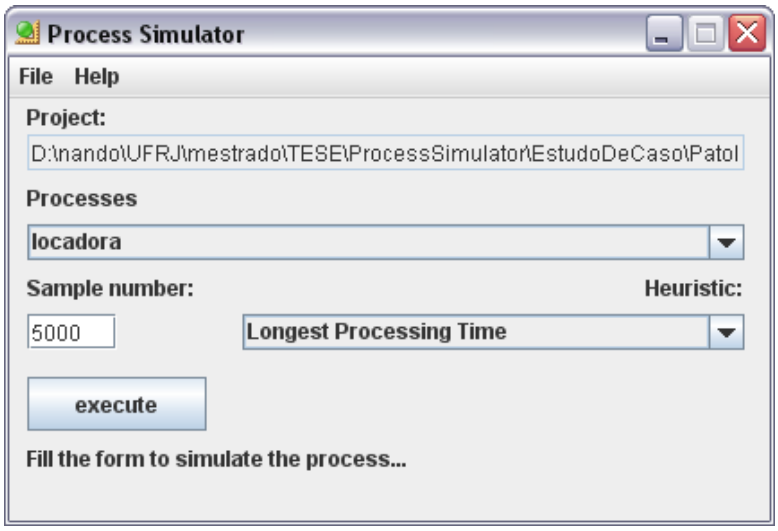

Figura 5.1: Interface Simulador de Processos

O Modelo do processo deve ser desenvolvido utilizando-se a ferramenta RAPDIS (MORGADO et al., 2006), que pode ser baixada gratuitamente no site do GETI (GETI, 2007). O modelo do processo deve seguir algumas regras de formação para garantir que o processo está sendo expresso de forma correta e coesa para o algoritmo de simulação. Essas regras são:

- **RF1** - Delimitadores de Fluxo Únicos
- **RF2** - Balanceamento do E
- **RF3** - Balanceamento do OU
- **RF4** - Respeito ao Sincronismo

O Simulador de Processos pode ser executado diretamente do RAPDIS utilizando-se a opção do menu ”Simular Processo”, ou pode ser executado em separado rodando o atalho sp.bat, ou ainda executando o arquivo ”ProcessSimulator.jar”. Caso o Simulador de Processos seja invocado pelo RAPDIS, o campo Projeto (*Project*) será automaticamente preenchido com o projeto aberto no RAPDIS, caso contrário será necessário utilizar o menu Arquivo (*File*) e selecionar a opção Abrir Projeto (*Open Project*), e selecionar o arquivo de projeto do RAPDIS.

Para simular o processo, deve-se escolher o diagrama desejado na lista de Processos (*Process*), informar o número de amostras (*Sample Number*), selecionar a heurística desejada (*Heuristic*) e executar a simulação pressionando-se o botão executar (*Execute*).

Após a confirmação do término da simulação, o resultado estará disponível em um arquivo XLS (Excel) com o mesmo nome do diagrama (processos) simulado no diretório onde se encontrado o arquivo do projeto do RAPDIS.

JUnit e Casos de Teste

A ferramenta foi desenvolvida utilizando-se o framework de testes JUnit que é um framework de testes de regressão escritos por Erich Gamma and Kent Beck. Ele é usado por desenvolvedores que implementam os casos de testes em Java.

JUnit é um software de Código Aberto (*Open Source Software*), que é distribuído sobre a licença Pública Comum (*Common Public License*) versão 1.0 e esta hospedado por *SourceForge* (GAMMA; BECK, 2006).

Os casos de testes desenvolvidos são apresentados a seguir:

ActivityNetworkUnitTest

Conjunto de Testes unitários que visam verificar o correto funcionamento da classe *ActivityNetwork* (Rede de Atividades) do pacote `br.ufrj.dcc.geti.ps.an`, que representa todas as redes (deterministas e não-deterministas) no Simulador de Processos.

Os métodos de testes são:

verifyMethodGetPredecessors

Preenche um rede de atividades com quatro atividades (A, B, C, D) de cria arcos entre elas { (A,B), (A,C), (B,D), (C,D) } e verifica se: os predecessores de B é unicamente A, e os predecessores de C é unicamente A, e os predecessores de D são B e C.

verifyMethodGetSucessors

Cria a mesma rede de atividades e verifica se os sucessores de A são B e C, se os sucessores de B é unicamente D e se os sucessores de C é unicamente D.

HeuristicsUnitTest

Classe de teste responsável por verificar as heurísticas de escalonamento das atividades, que são utilizadas no algoritmo de RCPS Paralelo Padrão. Todas as heurísticas estão implementadas no pacote `br.ufrj.dcc.geti.ps.heuristics`.

Os testes implementados para as heurísticas foram:

verifiifySortLPT - *Longest Processing Time*

São colocadas três atividades (A1, A2, A3) com distribuição constante 3, 4 e 1 respectivamente. É verificado se o resultado da ordenação é: A2, A1, A3.

verifiifySortSPT - *Shortest Processing Time*

São colocadas as mesmas atividades com as mesmas durações e é verificado se o resultado obtido foi: A3, A1, A2.

verifySortFIFO - *First In First Out*

Coloca-se duas atividades A1 e A2 com tempo de início igual a 1 e 2 respectivamente. É verificado se o resultado da ordenação obtido é: A1, A2.

verifySortLIS - *Least Immediate Successors*

Novamente são colocadas duas atividades A1 e A2 com o número de sucessores igual a 1 e 2 respectivamente. O resultado esperado para a ordenação é: A1, A2.

verifySortMIS - *Most Immediate Successors*

É utilizado o mesmo cenário, e espera-se que o resultado seja o oposto: A2, A1.

Performance

Este teste visa verificar o desempenho do algoritmo de Simulação. A rede de atividades utilizada foi o da figura 4.1. Os teste foram executados pelo Eclipse (CONTRIBUTORS et al., 2007), e feitos com a seguinte configuração de equipamento:

O tempo gasto com a simulação foi de: 2344 milissegundos (ou 2,3 segundos) utilizando-se

Processador	Pentium IV 3.2 GHz
Memória	1256MB DDR 333 MHz
JVM	JDK 6 (1.6.0)

Tabela 5.1: Configuração do Equipamento

5000 amostras; 4641 milissegundos (ou 4,6 segundos); 7469 milissegundos (ou 7,5 segundos) utilizando-se 15000 amostras. De posse destes resultados pode se afirmar que o tempo de execução do algoritmo cresce linearmente com o crescimento do número de amostras.

RCPSParallelAlgorithmUnitTest

Esta classe de teste visa garantir o correto funcionamento da implementação do Algoritmo Paralelo Padrão do RCPS baseado no algoritmo descrito por (OLAGUIBEL; GOERLICH, 1989).

Os métodos de testes implementados foram:

verifyASingleActivity

Verifica se o algoritmo está calculando o tempo de execução de uma rede trivial com apenas a atividade inicial, uma atividade com duração constante 4 e a atividade final. O resultado esperado para a rede é 4.

verifyMethodHasResourcesAvailable

Testa o método que verifica a disponibilidade de recursos para a execução da atividade informada. Faz teste com um tipo de recurso, com dois tipos de recursos e com recursos insistentes.

verifyTwoParallelalActivities

Verifica se o algoritmo está trabalhando de forma adequada quando existem duas atividades em paralelo A e B em uma rede de atividades com durações 4 e 5 respectivamente. Neste caso não estão sendo levados em contas recursos para a execução das atividades, logo o resultado esperado e verificado é 5.

verifyTwoParallelalActivitiesResourceConstrained

Utiliza-se a mesma rede de atividades apresentada, porém, as atividades competem pelo mesmo recurso, e desta forma, uma atividades terá que esperar o término da outra para que possa entrar em execução. Sendo assim o resultado esperado é 9.

UnfoldingAlgorithmUnitTest

Esta classe de teste visa verificar o correto funcionamento do algoritmo de desdobramento, o conjunto de testes tem como entrada os grafos seguindo as regras de formação e a saída do algoritmo deve conter apenas redes de atividades com fluxo determinista, durações constantes das atividades, e conter apenas os elementos de início de fluxo, atividades e o elemento de final de fluxo.

singleActivityDANEqualsToNDAN

Teste básico que verifica se uma rede de atividades não determinista, que possui apenas uma atividade ladeada pelos elementos de início e fim de fluxo, ao ser desdobrada gera uma rede determinista que é exatamente igual a original.

twoParalellalActivities

Este cenário de teste visa verificar se o algoritmo de desdobramento está transformando de forma correta o elementos de fork e join para uma rede de atividades determinista contendo apenas atividades, com arcos correspondentes a rede original.

A rede não determinista é composta pelo elemento de início de fluxo, seguido pelo elemento de fork, que aponta para as atividades A e B, que por sua vez apontam para o elemento de join, que leva ao elemento de fim de fluxo.

Ao ser desdobrada, a rede resultante deve conter apenas os elementos de Início e Fim, e as atividades A e B, com os links {(I,A), (I,B), (A,F), (B,F)}.

classicalEberGraph

Testa a capacidade do algoritmo de transformar corretamente o grafo da figura 3.3 para o grafo da figura 3.4. Verificam-se os predecessores e os sucessores de cada atividade de forma a validar se os forks e joins cruzados foram mapeados de forma correta.

verifyRamdonLinkMethod

Verifica se o método `getRadomLink` do algoritmo de desdobramento que escolhe de forma aleatória um dos caminhos que saem de um nó está implementado de forma correta.

São criados três arcos {(A,B), (A,C), (A,D)} e cada um com o array de probabilidades diferentes e para cada array verifica-se o sucessor selecionada para a primeira e para a segunda execução.

startSplitAOrBMergeEndAlwaysChooseA

Verifica o correto funcionamento do elemento de split onde sempre uma atividade é escolhida. A rede é formada por: Elemento de início de fluxo, seguido pelo elemento de split que aponta para as atividades A e B, que por sua vez apontam para o elemento de merge, que aponta para o elemento de fim de fluxo.

A probabilidade da transição para A ser escolhida é de 100%, e desta forma, a rede determinista gerada será sempre: Início → A → Fim

startSplitAOrBMergeEndWithEqualProbability

Este cenário de teste utiliza a mesma rede anterior, porém a probabilidade de escolha entre as atividades é sempre de 50%. São amostrados 5000 cenários, e é verificado se a porcentagem de ocorrência das atividades A e B não diferem de mais de 2% (de erro), segundo o teorema de KS (FELLER, 1948).

simpleLoop

Verifica se os retornos para re-execução das atividades estão sendo tratados de forma correta, ou seja, é verificado se o array de probabilidades de desvio estão sendo interpretados de forma correta e se a amostragem da duração das atividades estão independentes.

A rede não determinista se inicia pelo elemento de início de fluxo, seguido por um elemento de merge, que segue para a única atividade. A atividade aponta para o elemento de split que possui o array de probabilidades de desvio {(100%,0%),(0%,100%)} para o merge e para o elemento de fim de fluxo respectivamente.

Desta forma, a rede de atividades determinista obtida após a execução do algoritmo de desdobramento é: Início → A → A → Fim.

verififyMethodHasAllActivitiesWaiting

Este cenário de teste visa validar se o método que verifica se todos os predecessores do join foram alcançados na busca. Este método é responsável por garantir que o sincronismo será respeitado pelo algoritmo de desdobramento. São criadas duas listas, uma referente aos links a serem corrigidos, ou seja, uma lista que contém todos os links que possuem como destino o nó atual; e uma outra lista que links que contém todos os links da rede não determinista. São feitos dois testes, um onde todos os predecessores foram visitados e outro onde duas atividades ainda não foram visitados. Desta forma, no primeiro caso de teste, espera-se que o método retorne verdadeiro, e no segundo caso de teste, falso.

testTwoParallelalFowsWithOneFlowWithMuchMoreActivities

Neste cenário de teste é verificado se o respeito ao paralelismo entre os diversos fluxos

entre os elementos de fork e join está funcionando de forma correta. A rede é formada por dois caminhos simultâneos entre os elementos de fork e join, onde um caminho possui apenas uma atividade enquanto o outro possui quatro atividades. Desta forma, uma caminho alcança o elemento de join, que fica aguardando ser alcançado pelo outro caminho.

ActivityUnitTest

Conjunto de teste que visam verificar o correto funcionamento da classe Activity, que se encontra no pacote de modelo.

verifyMethodGetSample

Este teste visa garantir que a amostragem do tempo de execução da atividade está sendo amostrado de forma correta. É criado uma lista contendo três distribuições constantes com os valores de um, dois e três. Incrementa-se o número de vezes que a atividade foi executada, e verifica-se se para a primeira execução o valor amostrado corresponde a um, para a segunda dois, e para a terceira 3.

Este teste visa garantir que as futuras implementações do RAPDIS que suportem o cadastro de várias distribuições de probabilidade para cada atividade possa ser incorporado ao método SMERA.

Principais Algoritmos

Distribuição Triangular

A distribuição Triangular é uma estimativa de três pontos onde a estimativa mais provável possui quatro vezes mais peso que as outras duas. A Média desta distribuição é dada por:

$$\overline{X} = \frac{minimo + (4 \times provavel) + maximo}{6}$$

E possui Desvio Padrão (α):

$A = minimo^2 + provavel^2 + maximo^2$

$B = minimo \times provavel + minimo \times maximo + provavel \times maximo$

$C = minimo \times provavel \times maximo$

$$\alpha = \sqrt{\frac{A-B-C}{18}}$$

O algoritmo de amostragem da distribuição triangular é dado de acordo com um valor sorteado aleatoriamente entre zero e um. Verifica-se se o valor sorteado pertence a primeira metade ou a segunda, e então amostra-se o valor.

$$\mu 0 \leftarrow valor\ aleatorio\ entre\ [0,1]$$
$$se \quad \mu 0 < \frac{provavel - minimo}{maximo - minimo}$$

então retorne:

$$minimo + \sqrt{\mu 0 \times (provavel - minimo) \times (maximo - minimo)}$$

senão retorne

$$maximo - \sqrt{(1-\mu 0) \times (maximo - minimo) \times (maximo - provavel)}$$

Escolha aleatória do Arco

Algoritmo responsável por escolher de forma aleatória o arco dentre o conjunto de arcos que saem de um no do tipo split, sendo escolhido o arco de acordo com a probabilidade associada a cada arco para a execução informada. O algoritmo recebe como parâmetros uma lista de arcos e o número de vezes que o nó split foi visitado.

Escolha Aleatória do Arco (*Links*, *NumeroDeExecucoes*)

Array $probabilidade \leftarrow \{\}$

Para cara L em *Links* faça

$probabilidade \leftarrow probabilidade + L_p$ do *NumeroDeExecucoes*

Fim Para

De i = 1 até o tamanho de $probabilidade$ faça

$probabilidade_i \leftarrow probabilidade_i + probabilidade_{i-1}$

Fim De

$\mu 0 \leftarrow$ valor aleatório entre $[0,1]$

int pos = 0

Enquanto $\mu 0 > probabilidade_{pos}$ faça

pos++;

Fim Enquanto

retorne L de $Links_{pos}$

Teste de cobertura do algoritmo de Desdobramento

Todas as redes que seguem as regras de formação apresentadas neste trabalho, podem ser compostas por quatro estruturas básicas que são: formação em série, formação em paralelo, desvio e retorno. Essas estruturas são apresentadas a seguir, juntamente com o resultado da aplicação do algoritmo de desdobramento nestas estruturas; por fim apresenta-se o resultado do algoritmo de desdobramento para um teste de composição.

Série

Estruturas deste tipo são caracterizadas por possuir elementos em cadeia, ou seja, se forma seqüencial. As figuras 5.2, 5.4 e 5.6 demonstram este tipo de formação, e as figuras 5.3, 5.5 e 5.7 suas respectivas redes obtidas com a execução do algoritmo de desdobramento. É importante ressaltar que cada atividade nestes exemplos poderiam ser substituídos por outras estruturas, e neste caso estas estruturas estariam na formação de série.

Figura 5.2: Estrutura em série com apenas uma atividade.

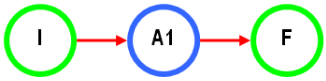

Figura 5.3: Desdobramento da rede com uma atividade.

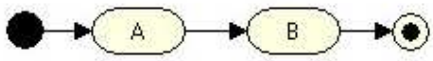

Figura 5.4: Duas atividades em série.

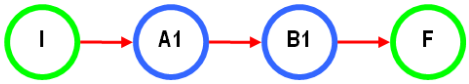

Figura 5.5: Desdobramento de duas atividades em série.

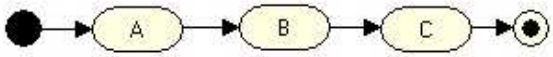

Figura 5.6: Três atividades em série.

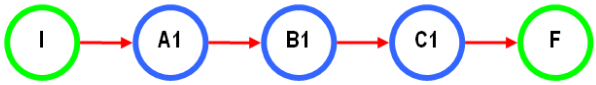

Figura 5.7: Desdobramento de três atividade em série.

Paralelo

As formações em paralelo são caracterizadas pela utilização das barras de sincronismo. As figuras 5.8, 5.10 e 5.12 apresentam formações deste tipo, e as figuras 5.9, 5.11 e 5.13 suas respectivas redes obtidas com a aplicação do algoritmo de desdobramento.

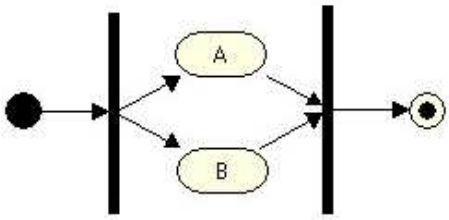

Figura 5.8: Duas atividades em paralelo.

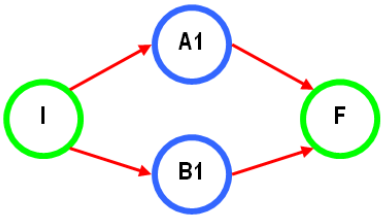

Figura 5.9: Desdobramento de duas atividades em paralelo.

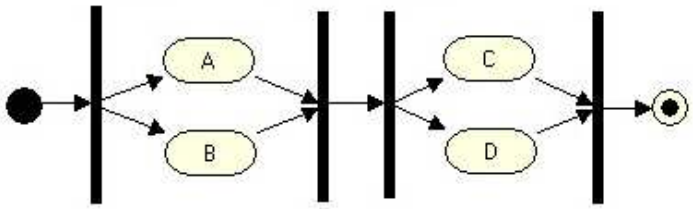

Figura 5.10: Dois paralelismos em série.

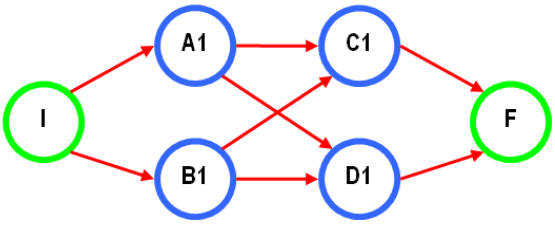

Figura 5.11: Desdobramento de dois paralelismos em série.

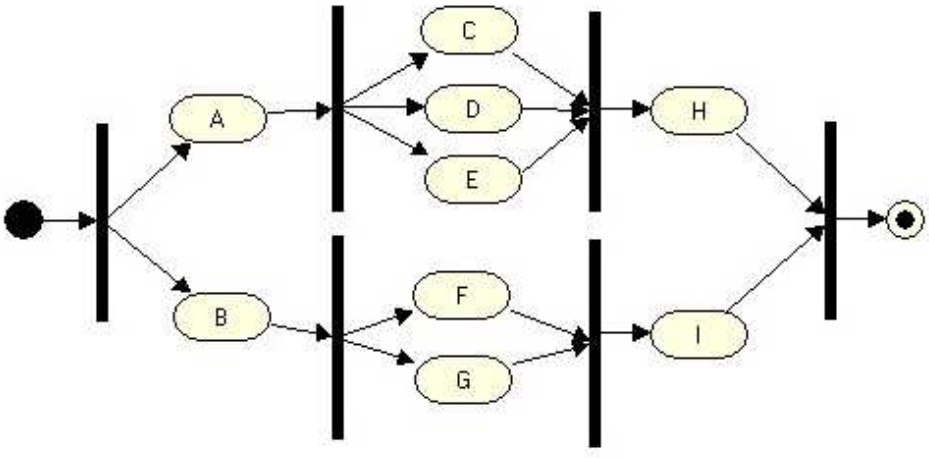

Figura 5.12: Dois paralelismos aninhados.

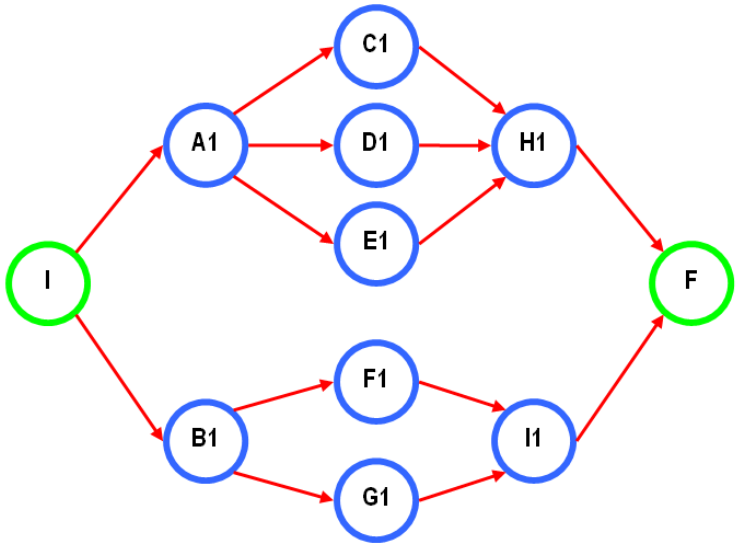

Figura 5.13: Desdobramento de dois paralelismos aninhados.

Desvio

As formações de desvio são caracterizadas pela utilização do elemento de decisão. As figuras 5.14 e 5.16 apresentam formações deste tipo, e as figuras 5.15 e 5.17 apresentam todos os grafos obtidos com a aplicação do algoritmo de desdobramento, respectivamente.

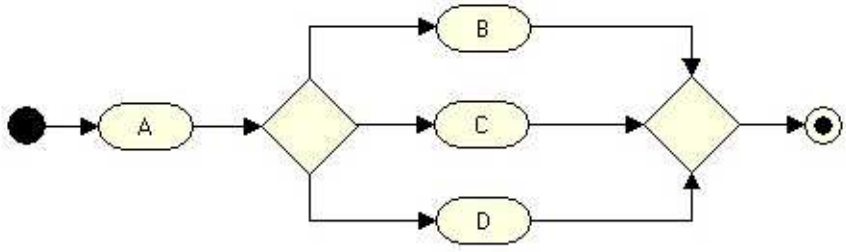

Figura 5.14: Atividade em série seguida de um desvio.

Retorno

A formação de retorno é uma especialização da formação desvio, onde o nó de junção encontra-se antes do nó de desvio, quando a rede é varrida do nó inicial para o final. Desta forma, é necessário definir as probabilidades de desvio de forma correta para que não exista a possibilidade do algoritmo de desdobramento ficar em loop infinito. Assim sendo vamos definir as probabilidade de retorno como {50%, 0%}, e as probabilidades de seguir em frente (para o nó final) de {50%, 100%}. As figuras 5.18, 5.20 e 5.22 apresentam formações deste tipo, e as

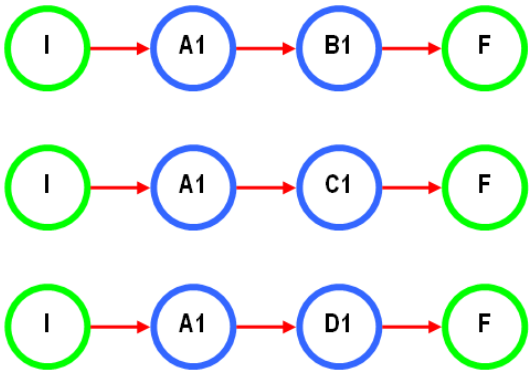

Figura 5.15: Desdobramento da atividade em série seguida de um desvio.

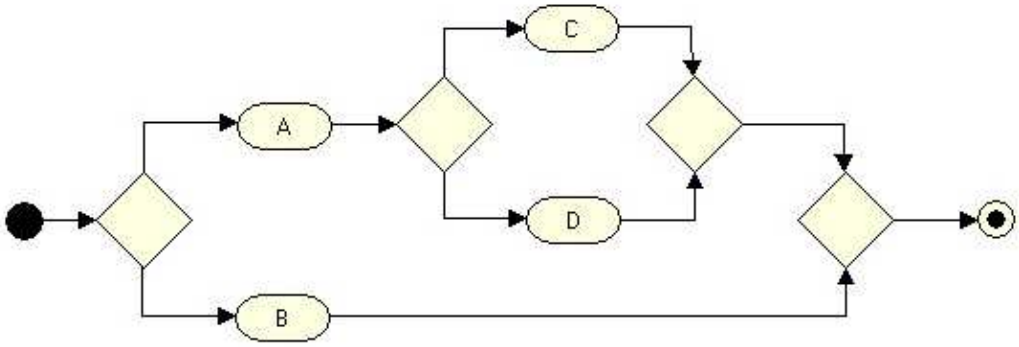

Figura 5.16: Desvios aninhados.

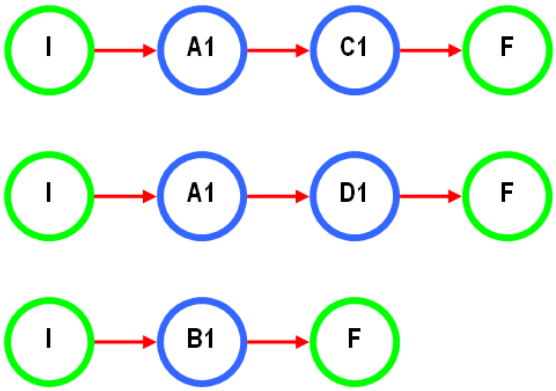

Figura 5.17: Desdobramento de desvios aninhados.

figuras 5.19, 5.21 e 5.23 os seus respectivos grafos obtidos com a execução do algoritmo de desdobramento.

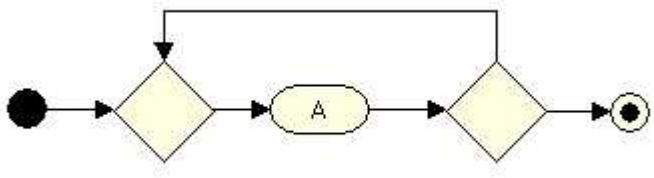

Figura 5.18: Retorno com apenas uma atividade.

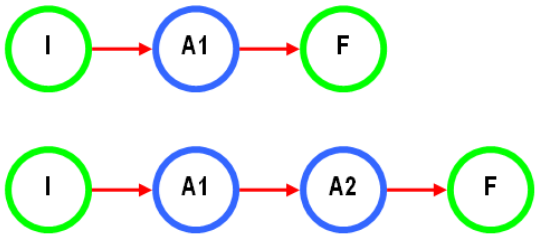

Figura 5.19: Desdobramento de retorno com apenas uma atividade.

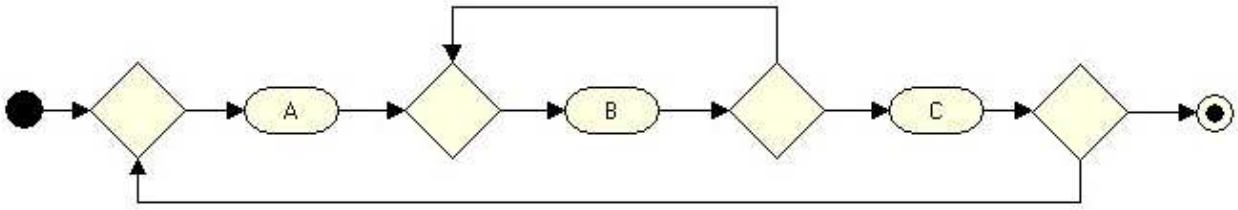

Figura 5.20: Retornos aninhados.

Composição

Desta forma, pode se utilizar estes quatro tipos de formações básicas para construir qualquer tipo de rede, desde que respeite as regras de formação. A figura 5.24 apresenta um exemplo de rede complexa que é formada pelos quatro tipos de formações apresentadas. A figura 5.25 apresenta todos as grafos obtidos com a aplicação do algoritmo de desdobramento nesta rede.

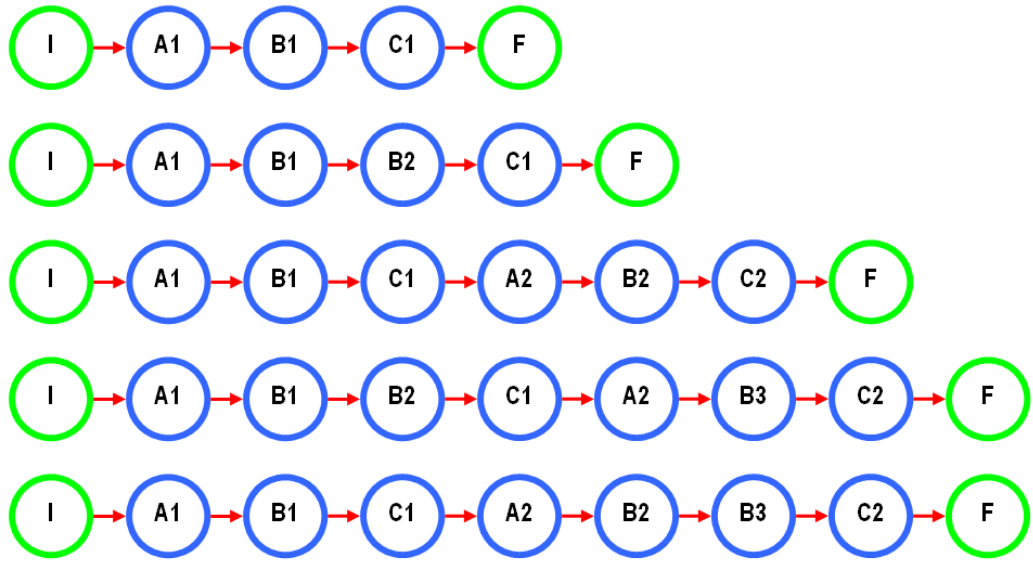

Figura 5.21: Desdobramento de retornos aninhados.

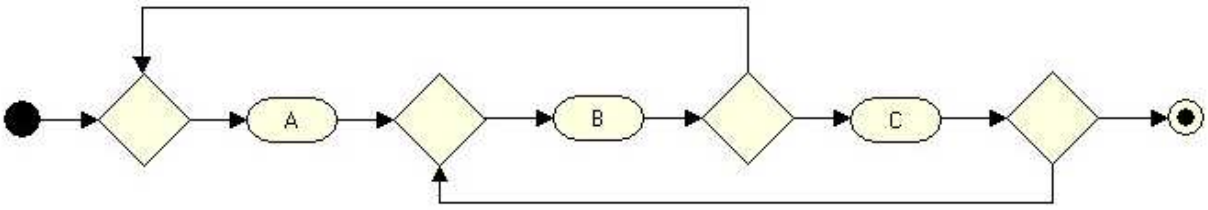

Figura 5.22: Retornos sobrepostos.

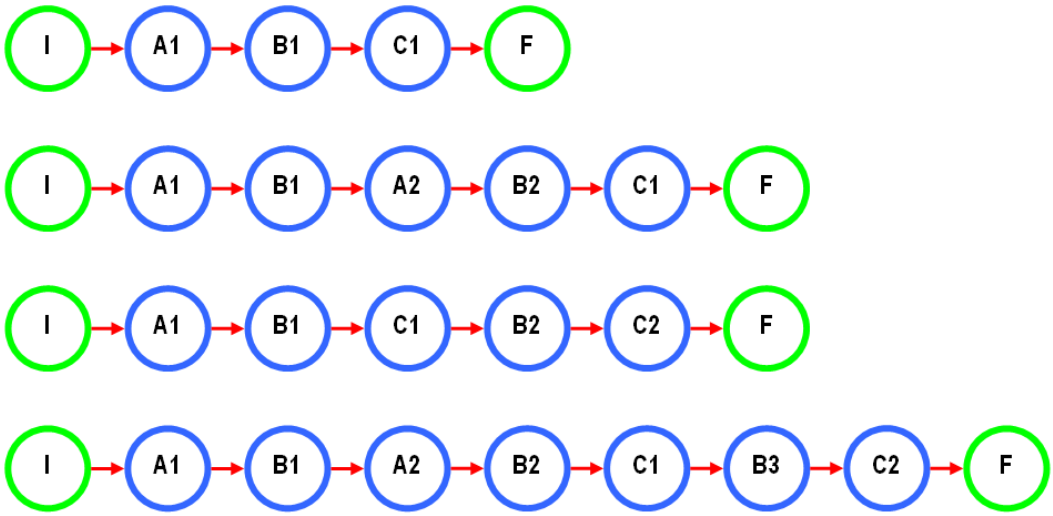

Figura 5.23: Desdobramento de retornos sobrepostos.

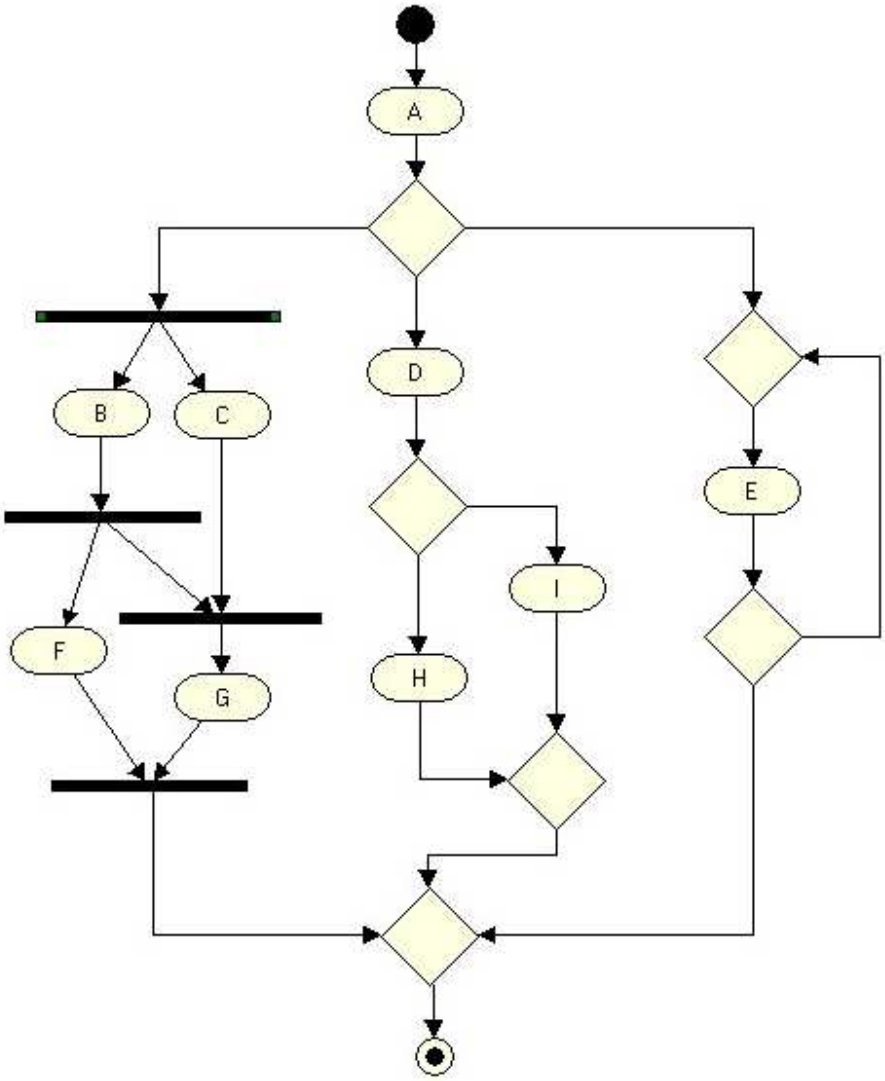

Figura 5.24: Composição com as quatro formações.

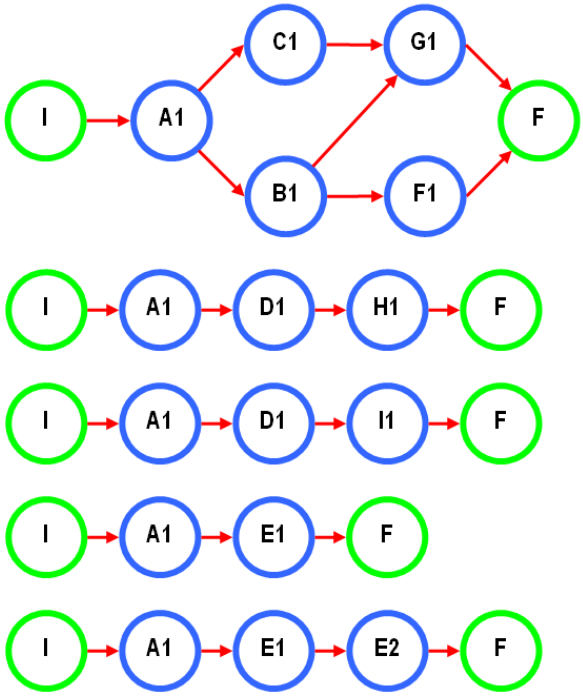

Figura 5.25: Desdobramento da composição com as quatro formações.