UM MÉTODO PARA SIMULAÇÃO DE REDES DE ATIVIDADES NÃO-DETERMINISTAS COM RECURSOS ESCASSOS

Daniel Veiga Kling

IM – NCE – UFRJ – Mestrado em Informática

Éber Assis Schmitz
Ph.D., Imperial College – 1980

Fábio Protti
D.Sc., UFRJ – 1998

Rio de Janeiro
2005

UM MÉTODO PARA SIMULAÇÃO DE REDES DE ATIVIDADES NÃO-DETERMINISTAS COM RECURSOS ESCASSOS

Daniel Veiga Kling

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

Aprovada por:

_____
Prof. Éber Assis Schmitz, Ph.D

_____
Prof. Fábio Protti, D.Sc.

_____
Prof. Antônio Juarez Alencar, D.Phil.

_____
Prof. Renato Florido Cameira, D.Sc.

Rio de Janeiro – RJ
Fevereiro de 2005

Ficha Catalográfica

KLING, DANIEL VEIGA.

Uma método para simulação de redes de atividades não-deterministas com recursos escassos, [Rio de Janeiro], 2005.

ix, 92p., A8p., B11p., 29,7 cm (IM/NCE/UFRJ, MSc., Informática, 2005)

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

1. Escalonamento de projetos
2. RCPSP
3. Monte Carlo

I. IM/NCE/UFRJ II. Título (série)

Agradecimentos

Aos meus pais José Carlos e Mariângela que me deram o apoio necessário para poder chegar até aqui e por terem sempre me incentivado a nunca parar de estudar.

Ao meu irmão Diogo por ser esse amigo maravilhoso que ele sempre foi e acreditar mais em mim que até eu mesmo.

Aos meus avôs, avós, tios, tias, primos e primas por formar essa família espetacular sem a qual eu com certeza não seria o mesmo.

Ao prof. Éber Schmitz por ter sido sempre um grande orientador, acreditando e me incentivando a buscar horizontes cada vez mais altos.

Ao meu co-orientador Fábio Protti por ter sido fundamental para a conclusão desta dissertação. Sem os seus conselhos provavelmente este trabalho não teria a metade da qualidade que tem hoje.

Aos professores Juarez Alencar e Renato Cameira por terem aceitado fazer parte desta banca.

Aos meus amigos que estiveram ao meu lado em todos esses anos de estudos e que de alguma forma tiveram influência na minha formação pessoal e profissional.

A todas as outras pessoas que de uma forma ou de outra contribuíram para a realização deste trabalho.

Resumo

UM MÉTODO PARA SIMULAÇÃO DE REDES DE ATIVIDADES NÃO-DETERMINISTAS COM RECURSOS ESCASSOS

Daniel Veiga Kling

Orientador: Prof. Éber Assis Schmitz

Departamento: Informática

As redes de atividades tem sido amplamente utilizadas para representar as tarefas que devem ser realizadas para se chegar a um resultado final um projeto. Esta forma de representação permite criar um modelo do mundo real, que pode ser estudado e analisado antes que o projeto seja executado na prática.

Por se tratar de um modelo preditivo, as redes de atividades devem resolver o problema da imprevisibilidade dos eventos do mundo real. Não é possível prever com exatidão a duração das atividades ou os resultados de eventos condicionais. Por isso o modelo de atividades deve permitir a representação não-determinista das atividades, nas quais estes atributos são expressos através de funções de probabilidade. No mundo real os recursos disponíveis para executar as atividades também são restritos, por isso estes devem ser explicitados na rede de atividades.

Este trabalho apresenta um método para simulação de redes de atividades não-deterministas com recursos escassos. Um novo modelo para representação de atividades chamado RANDRE é apresentado. Um método para mapeamento entre os diagramas de atividades da UML e as redes RANDRE é apresentado, de modo que os projetos descritos daquela maneira podem ser simulados utilizando este método.

São apresentados ainda algoritmos que permitem a simulação estocástica destas redes. Uma ferramenta chamada SimProcess foi implementada, permitindo colocar em prática todos os conceitos apresentados. O texto mostra ainda um estudo de caso da análise do processo de uma fábrica de software utilizando esta ferramenta.

Abstract

A METHOD FOR SIMULATION OF NON-DETERMINISTIC ACTIVITY NETWORKS WITH CONSTRAINED RESOURCES

Daniel Veiga Kling

Advisor: Prof. Éber Assis Schmitz

Department: Computer Science

The activity networks have been widely used to represent tasks that must be accomplished in order to get to the final result in a project. This kind of representation generates a model of the real world that can be studied and analyzed before the project is actually run.

For being a predictive model, the activity networks must address the fact that the real world´s events are unforeseeable. There is no way to predict exactly the duration of the activities and the results of conditional events. For this reason the activity model must allow the non-deterministic representation of the activities, in which this attributes are expressed through probability functions. In the real world the resources available to execute these activities are limited, so they must also be included in the activity network.

This work presents a method for simulation of non-deterministic activity networks with constrained resource. A new model for representing this kind of network called RANDRE is presented. A method for mapping activity diagrams from the UML to RANDREs is also presented. This way the networks represented in that form can be simulated using the RANDRE method.

This work presents also some algorithms that can be used in statistical simulations of these network. A software called SimProcess was implemented to put into practice all the concepts presented here. The text contains also a study case showing the analysis of a software factory using the RANDRE method and the SimProcess tool.

Índice

Índice de Figuras

Índice de Tabelas

1 - Introdução

1.1 - Apresentação

O escalonamento de atividades de um projeto é um problema muito estudado, tendo seus primeiros trabalhos datados da década de 60. O escalonamento de tarefas era tratado, no entanto, de maneira determinista, ou seja, os métodos assumiam que tudo que estava para acontecer em um projeto era conhecido antes dele ser executado, o que é invariavelmente falso.

Existe uma frase muito utilizada na área de engenharia de software que diz que não existem projetos atrasados, o que existe são planos incorretos. Na maior parte das vezes, os planos são incorretos porque não levam em consideração os riscos inerentes à tarefa de estimativa de prazos. Por este motivo, existe uma grande demanda por métodos que permitam obter análises de prazos com maior precisão, levando em consideração estas incertezas.

1.2 - Motivação

A motivação para a realização deste trabalho surgiu da inexistência de um modelo quantitativo de rede de atividades capaz de representar transições não deterministas e escassez de recursos em um único modelo. Conforme veremos mais a frente, nenhum dos modelos criados até hoje possuem estas duas características ao mesmo tempo. A representação destas informações no modelo de atividades é importante porque tem um impacto direto sobre os resultados das análises de tempo de execução obtidos através da simulação destas redes.

1.3 - Objetivo

O objetivo deste trabalho é apresentar um procedimento para simulação estocástica de redes de atividades com durações e estrutura não-deterministas e recursos escassos. O procedimento se baseia na geração de cenários de execução de uma rede de atividades e no escalonamento destes cenários. Estes passos são efetuados de maneira repetitiva de modo a avaliar a maior quantidade possível de cenários de determinado projeto. Ao final, dados estatísticos são gerados, permitindo analisar o comportamento de determinado projeto, principalmente em termos da duração de suas atividades e do projeto como um todo.

1.4 - Contribuições do Trabalho

Este trabalho traz como contribuições:

1. Um modelo para representação de redes de atividades não deterministas com recursos escassos, chamada aqui de RANDRE.
2. Um algoritmo para geração de cenários de execução a partir de uma RANDRE.
3. Um algoritmo para simulação de RANDREs, com um cenário por vez.
4. Um algoritmo para simulação de RANDREs, com cenários concorrentes.
5. Uma ferramenta, chamada SimProcess, que implementa todos os conceitos apresentados aqui.

1.5 – Plano do Trabalho

Este trabalho está estruturado em sete capítulos. O capítulo 2 apresenta uma visão geral do problema de representação de redes de atividades, mostrando os principais modelos existentes e os problemas inerentes à área. O capítulo 3 apresenta o modelo RANDRE, um novo meio para representar redes de atividades com características diferentes dos modelos pré-existentes. No capítulo 4 são mostrados os algoritmos utilizados para simulação de RANDREs, e será apresentado o SimProcess, programa criado para permitir a simulação destas redes. O capítulo 5 traz um estudo de caso de uma fábrica de software fictícia na qual o SimProcess é utilizado para analisar a rede de atividades para geração de softwares. O capítulo 6 finaliza o trabalho, mostrando algumas discussões sobre o método, conclusões finais e sugere algumas propostas para trabalhos futuros com o objetivo de dar continuidade a este projeto.

2 - Modelos de atividade

As redes de atividades são constituídas por tarefas e eventos conectados em uma certa ordem lógica para formar um modelo de determinado sistema de interesse. Elas têm tido um papel cada vez mais importante na análise de sistemas nas últimas décadas. Uma razão importante para a grande aceitação deste tipo de modelo é a relativa facilidade com que muitos tipos diferentes de sistemas e processos podem ser modelados nesta forma.

Um modelo de rede de atividades define as regras de formação das redes de atividades que podem ser construídas à partir dele [5]. As redes de atividades são compostas por informações sobre as atividades e sobre o seu modo de seqüenciamento, permitindo identificar como as tarefas podem ser encadeadas. A maior parte dos modelos de atividades possui ainda um método de representação gráfica, permitindo que as redes de atividades possam ser representadas em diagramas, facilitando a sua visualização e entendimento. Abaixo estão algumas formas de representação de redes de atividades:

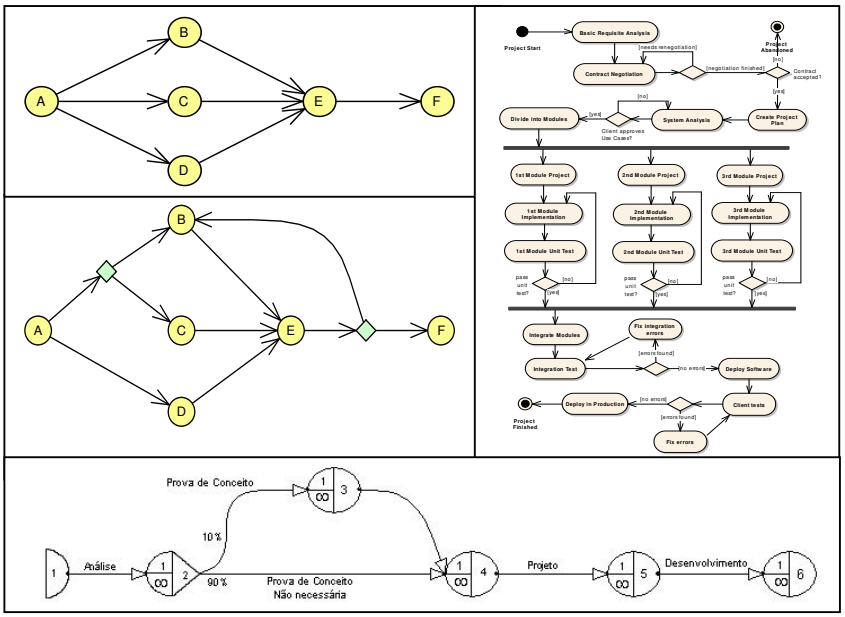

Figura 1: Formas de representação de redes de atividades

Na figura 1 podemos ver alguns exemplos de redes de atividades que serão estudadas mais adiante neste texto. Conforme podemos ver acima, os diversos tipos de modelos de atividades diferem muito nas suas representações gráficas. No entanto, este não é o único ponto divergente entre eles. Os diversos modelos diferem também quanto às informações que contém. Podemos enumerar alguns conjuntos de características destes modelos conforme as informações do mundo real que eles representam ou não. Estas características podem ser utilizadas para auxiliar na avaliação do grau de complexidade de cada modelo, fornecendo parâmetros para que possamos compará-los.

2.1 - Características de modelos de atividades

A maior parte dos modelos possui algum modo de representação gráfica que tem como objetivo facilitar o entendimento da relação entre as inúmeras atividades representadas. Estas representações gráficas podem ser muito simples ou extremamente complexas. A maior parte delas se baseia na utilização de grafos direcionados para representar as atividades e o modo de encadeamento entre elas, ou seja, a ordem em que devem ser executadas. Este grafo pode possuir inúmeras propriedades diferentes nos diversos modelos, podendo possuir nós especiais, arcos especiais, ou outros apetrechos gráficos. Algumas das características apresentadas abaixo estão relacionadas à topologia do grafo e à semântica dada a cada um dos elementos contidos nele.

Além das informações contidas em seus grafos, os modelos podem possuir diversas outras informações anexas. Estas informações são as mais diversas, e vão desde a quantidade de recursos necessários para executar cada atividade a informações sobre o modo como um modelo de atividades deve ser instanciado.

Serão mostradas aqui apenas algumas das principais características que podem ser observadas nos modelos de interesse para este trabalho.

2.1.1 - Atividades nos arcos e atividades nos nós

A maior parte dos modelos utiliza grafos para a representação gráfica das redes de atividades. Os grafos são constituídos por nós e arcos interligados. Ao representar uma rede de atividades através de grafos os modelos podem utilizar para representar as atividades tanto os arcos quanto os nós. Portanto, o primeiro modo de classificar os modelos é verificar se eles representam as atividades em seus diagramas através de nós ou arcos.

A mais comum das formas de representação é conhecida como “Activity On Node” (Atividade No Nó) ou, abreviadamente AON [1]. Esta forma dá mais ênfase à representação das atividades do que aos estados entre as execuções das atividades. As arestas são utilizadas para explicitar o fluxo de execução das atividades, indicando a ordem de precedência entre elas. Uma aresta ligando uma atividade X a uma atividade Y indica, normalmente, que Y só pode iniciar após o término de X. Algumas variações desta regra podem existir de acordo com as construções existentes nos diagramas.

A segunda forma de representação é conhecida como “Activity On Arc” (Atividade No Arco) ou AOA [1]. Esta forma privilegia a representação dos estados entre a execução das tarefas. Estes estados são representados como nós do grafo. As atividades servem como transição entre estes estados, sendo representadas por arestas. Em alguns diagramas que seguem este estilo, os vértices do grafo possuem atributos especiais utilizados para indicar, por

exemplo, quando as atividades ligadas a ele podem ser disparadas, e quais devem ser disparadas.

Um exemplo dos dois tipos de representação pode ser visto na Figura 2:

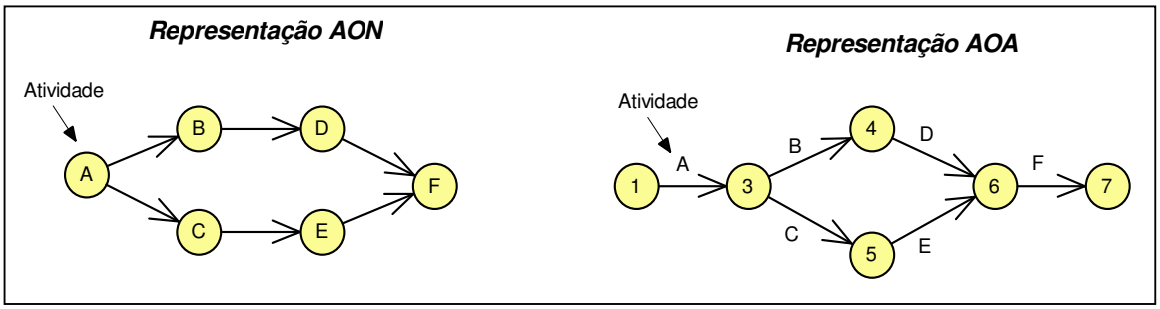

Figura 2: Formas de representação AON e AOA

2.1.2 - Representação de transições não deterministas

Outro modo de se classificar os modelos de atividades é verificar se eles possuem a capacidade de representar possíveis alterações no fluxo de execução das atividades enquanto é executado. Em situações da vida real é extremamente difícil prever com exatidão que tarefas precisarão ser executadas de modo a se obter determinado resultado final. Normalmente, no entanto, é possível enumerar um conjunto finito de cenários que podem ocorrer no momento de execução das atividades. Por exemplo, é difícil dizer com exatidão se amanhã irá chover ou não, mas é possível dizer que existem dois cenários possíveis para chuva amanhã: ou choverá, ou não choverá. A capacidade de representação de transições não deterministas indica se um modelo pode representar situações deste tipo, ou se as ignora, e deixa a tarefa de fazer essas previsões a cargo do projetista.

Chamamos aqui de transição a passagem do fluxo de execução que ocorre no momento do término de determinada atividade. No momento que ocorre uma transição outras atividades são disparadas, e desta maneira a rede de atividades é executada. Uma transição determinista é aquela onde o conjunto de tarefas executadas após a transição não depende de nenhum evento aleatório. De maneira mais formal, após o término da atividade A um conjunto C de n atividades é sempre disparado. O conjunto de atividades disparado é, portanto, bem definido e constante. Já na transição não determinista, as atividades disparadas são selecionadas aleatoriamente. Isso significa que após o término da atividade A um determinado número m ($m \leq n$) de atividades são selecionadas aleatoriamente do conjunto C para ser disparadas.

A maior parte dos modelos permitem apenas a representação de transições deterministas, como o CPM, PERT, RCPS. Isto significa que, nestes modelos, após o término de uma atividade é sempre possível dizer quais atividades deverão ser executadas em seguida. Isto facilita muito o trabalho de análise e simulação. Como não existe a possibilidade de variação na rede, as regras de precedência de atividades são bem definidas e fixas. No

entanto, esta classe de modelos tende a ser otimista, representando apenas as situações mais prováveis e ignorando possíveis casos de exceção.

Quando um modelo permite a definição de transições não deterministas ele aumenta a quantidade de informações do mundo real que podem ser representadas. No entanto, este maior poder de representação vem associado a um grande aumento de complexidade nas análises. As transições não deterministas ocorrem quando, após o término de uma ou mais atividades, existe mais de um caminho possível a ser escolhido. Ao selecionar um destes caminhos os outros passam a não existir naquela execução do modelo de atividades.

Como exemplo de transição não determinista, vamos supor que estivéssemos fazendo um plano para um piquenique no parque. Para atingir este objetivo é necessário colocar no plano algumas atividades, como comprar os alimentos a serem consumidos, prepará-los, preparar o carro, arrumar as crianças, e colocar o pé na estrada. No entanto, consultando a previsão do tempo é possível descobrir que existe uma chance de 60% de chuva para o dia do piquenique. É importante incluir essa possibilidade no plano, porque não desejamos sermos pegos de surpresa, sem possuir um plano B. O plano deve dizer então que se chover o itinerário será mudado para um passeio no shopping. Esta é uma transição não-determinista, porque neste ponto do plano é necessário colocar duas atividades sendo que apenas uma será executada. A existência de transições não deterministas torna o modelo complicado porque passa a ser difícil identificar as regras de precedência. Por exemplo, se existisse a atividade voltar para casa ao final do passeio, qual atividade deve ser terminada antes que esta possa começar: ir ao shopping ou fazer o piquenique?

2.1.3 - Grafo acíclico e cíclico

A existência de transições não deterministas possibilita a existência de algumas construções que não eram possíveis usando apenas transições deterministas. A principal delas é a construção de ciclos, ou seja, atividades que podem ser repetidas enquanto determinada condição seja verdadeira.

As atividades repetitivas são muito comuns no mundo real. Muitas vezes é necessário executar várias vezes uma ou várias tarefas até que o objetivo final seja alcançado. Como exemplo, vamos supor a atividade de costurar uma calça nova. Você tira as medidas da pessoa e costura a calça. Ao final você prova e verifica se a roupa ficou boa. Se não ficou você conserta os defeitos e experimenta de novo. Isso é feito até que a calça fique boa, gerando um ciclo de atividades. Para representar essas atividades nos modelos é necessário que o grafo de relacionamento entre atividades seja cíclico. Modelos que suportam apenas transições deterministas não conseguem representar ciclos, porque se o fizessem não haveria meio de sair do ciclo. Um modelo que permite transições não-deterministas pode ser tanto cíclico ou acíclico.

Ao permitir a existência de ciclos os modelos ganham mais um nível de representação do mundo real, mas também ganham mais um nível de complexidade de análise. Vimos que

em um modelo com transições não deterministas o fato de uma atividade poder existir ou não traz complicações às regras de precedência. Em um modelo com ciclos as atividades podem, além de existir ou não, ser executadas mais de uma vez. Isso gera mais complicações nas regras de precedência, porque a relação de precedência da atividade com a anterior passa a depender de ela ser a primeira, segunda, terceira ou enésima execução da atividade. Voltando ao exemplo da costura, a primeira execução da atividade "Costurar calça" deve ser executada após o término de "Tirar medidas". No entanto, a segunda execução de "Costurar calça" deve ser executada após a primeira execução de "Provar roupa" e assim por diante. Simular ciclos é uma tarefa complexa, sendo poucos os modelos que permitem a representação deles, como, por exemplo, o GERT.

2.1.4 - Recursos infinitos e recursos restritos

Para ser realizada qualquer atividade precisa ser executada por alguém utilizando um determinado conjunto de ferramentas. Damos o nome de recurso aos instrumentos, sejam eles humanos ou não, necessários para executar uma atividade. Na maioria dos casos do mundo real, a quantidade de recursos disponíveis para executar determinada tarefa é limitada. No entanto, os diferentes modelos de atividades podem representar ou não essas limitações de recursos.

Se o modelo de atividades não representa a dependência de recursos dizemos que ele possui recursos infinitos. Alguns modelos não representam recursos. Um dos motivos para isso é que a verificação de disponibilidade de recursos pode ser custosa aumentando a complexidade do escalonamento. Quando um modelo possui restrições de recursos, e existem duas ou mais atividades a serem executadas, a ordem em que elas são escalonadas pode influenciar o tempo total de execução do projeto. A tarefa de encontrar o tempo ótimo para execução de uma rede de atividades torna-se computacionalmente custosa sendo um problema NP-completo como demonstrado em [2].

Normalmente nos modelos que representam as restrições de recursos, cada atividade possui uma lista de recursos que devem estar livres para que ela possa ser executada. Estes modelos definem a quantidade disponível de cada um destes recursos. Quando a disponibilidade de um recurso se esgota, a atividade que depende daquele recurso não pode ser escalonada até que o recurso seja liberado.

2.1.5 - Durações deterministas e aleatórias

Um dado muito importante associado às atividades é o tempo necessário para que ela seja terminada. O tempo para execução de uma tarefa pode ser determinista ou aleatório.

A maneira mais simples de se modelar a duração de uma atividade é atribuir um valor fixo a ela. Neste caso dizemos que a duração da tarefa é determinista. A utilização de durações deterministas facilita o escalonamento das atividades e permite que o valor exato do tempo

total para realização do projeto possa ser calculado (desde que existam apenas transições deterministas). No entanto, no mundo real é extremamente difícil prever com exatidão a duração de uma atividade antes que ela seja executada. Os resultados obtidos com modelos que implementam apenas durações deterministas tornam-se imprecisos, visto que é extremamente improvável que a duração real da atividade seja aquela prevista inicialmente.

Outra maneira de se definir a duração das atividades é utilizando variáveis aleatórias. É difícil prever o tempo exato necessário para executar uma atividade. No entanto, é relativamente fácil enumerar faixas de valores que esta duração pode assumir. Por exemplo, é difícil prever que uma pessoa levará 2 horas e 37 minutos para assistir a um filme, mas é razoavelmente simples dizer que esta pessoa levará de 2 a 3 horas para executar esta tarefa. Nos modelos de atividades, estas estimativas tomam a forma de variáveis aleatórias, que são funções que associam valores (tempo) a probabilidades de ocorrência. Uma discussão extensa sobre variáveis aleatórias pode ser encontrada em [3]. As distribuições estatísticas mais comumente utilizadas para representar a duração de uma atividade são as distribuições Triangular e Beta com três parâmetros. Ambas possuem três parâmetros: o menor valor, o mais provável e o maior valor. O valor verdadeiro da duração é sorteado entre esses três valores.

2.2 – Modelos de Atividades

Por serem representações do mundo real, os diversos modelos criados até hoje diferem muito quanto à quantidade de informação que pode ser representada. Eles possuem enfoques e níveis de complexidade diferentes. Conforme vimos na seção anterior, existe um conjunto de categorias nas quais estes modelos podem ser classificados, de modo que possamos ter uma medida do seu grau de complexidade.

Desde o surgimento do CPM/PERT no final dos anos 50, diversos modelos diferentes para representação de cadeias de atividades surgiram. Nesta seção, serão apresentados alguns dos modelos de maior utilização desenvolvidos de lá pra cá.

2.2.1 – CPM

CPM é a sigla para Critical Path Method (Método do Caminho Crítico). Segundo [4] o CPM foi desenvolvido no final dos anos 50 pela DuPont Company em conjunto com a divisão Univac da Remington Rand Corporation. O propósito inicial do projeto CPM era fornecer uma técnica para o controle da manutenção das indústrias químicas da DuPont. No entanto, devido à sua generalidade logo ele se tornou um modelo universal aplicado às mais diferentes áreas. Isso ocorreu principalmente porque a técnica do CPM fornecia os seguintes benefícios:

1. Fornece uma visão gráfica do projeto
2. Prevê o tempo necessário para terminar um projeto

3. Mostra as atividades que são críticas para manter o cronograma e quais não são.

O CPM modela as atividades e os eventos de um projeto em uma rede. As atividades são representadas como nós da rede, e eventos, que representam o início e fim das atividades, são representados como arcos ou linhas. A Figura 3 mostra um exemplo do diagrama de uma rede CPM:

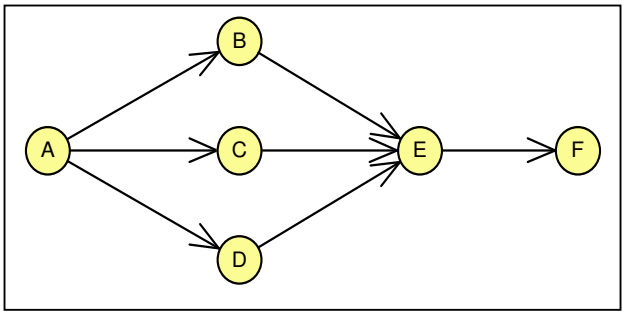

Figura 3: Exemplo de rede CPM

As setas na rede CPM indicam as relações de precedência entre atividades da rede. Considere a rede mostrada na Figura 3. A seta que liga a atividade A à atividade B indica que B só pode começar após o término de A. O mesmo acontece com as setas que ligam A a C e A a D. Da mesma forma, as setas que ligam B, C e D a E indicam que E só pode iniciar após o término das atividades B, C e D.

O método CPM assume que o tempo necessário para executar cada atividade é dado por uma estimativa de um único valor. O método não reconhece, portanto, a incerteza existente na obtenção desta estimativa, afinal, o tempo real para execução da atividade não é conhecido à priori. As redes CPM não possuem transições não deterministas e não representam os recursos necessários para executar cada atividade. Na classificação definida anteriormente as redes CPM são classificadas da seguinte maneira:

1. Atividades nos nós ou nos arcos
2. Sem transições não deterministas
3. Grafo acíclico
4. Sem restrição de recursos
5. Durações deterministas

Conforme o nome do método diz, um dos aspectos mais importantes do CPM é a determinação do caminho crítico. O caminho crítico da rede é a seqüência de atividades que constitui o caminho de maior duração da rede, determinando o menor tempo em que um projeto pode ser realizado. O significado do caminho crítico é que as atividades que estão nele não podem ser atrasadas sem que todo o projeto seja atrasado. Por causa do seu impacto no

projeto, a análise do caminho crítico é um importante aspecto no planejamento do projeto. No exemplo mostrado anteriormente existem três possibilidades para o caminho crítico: A – B – E – F; A – C – E – F; A – D – E – F. A identificação do caminho crítico entre as três possibilidades dependerá dos tempos associados a cada uma das atividades. O caminho crítico de um projeto pode ser determinado através de um algoritmo simples baseado no cálculo de duas variáveis associadas a cada atividade: o EST (Earliest Start Time – Menor tempo de início possível) e o LST (Latest Start Time – Maior tempo de início possível). Quando estas duas variáveis possuem o mesmo valor a atividade está no caminho crítico. O algoritmo mostrado em [5] permite calcular estes dois valores. O início de cada atividade pode ser escalonado entre estes dois valores, permitindo obter o cronograma do projeto.

O CPM foi desenvolvido para projetos complexos, mas rotineiros, com incerteza mínima dos tempos de término dos projetos [6]. Para projetos menos rotineiros, existem maiores incertezas quanto aos tempos de execução, e esta incerteza limita a utilidade do modelo determinista do CPM.

2.2.2 - PERT

Uma alternativa existente para o modelo CPM é conhecido como PERT, ou Program Evaluation and Review Technique (Técnica de Revisão e Avaliação de Programas). Ela foi desenvolvida em 1958 (simultaneamente ao desenvolvimento do CPM) para ajudar no planejamento do projeto do míssel Polaris da Marinha dos Estados Unidos. Estiveram envolvidos no desenvolvimento do PERT o Escritório de Projetos Especiais da Marinha e a Divisão de Sistemas de Mísseis da companhia de construção aérea Lockheed. O sucesso magnífico do projeto Polaris é altamente responsável pela ampla aceitação obtida pelo PERT junto ao governo e à área de negócios como uma ferramenta para planejamento e controle de projetos [4].

O modelo PERT é muito similar ao CPM. As redes de atividades são representadas da mesma forma, através de grafos com atividades nos nós, e possuem as mesmas regras de precedência. A diferença fundamental entre as duas técnicas é quanto ao modo como a estimativa do tempo para execução da atividade é fornecida. A técnica do PERT inclui um explícito reconhecimento da incerteza das estimativas de tempo permitindo que três estimativas sejam feitas para cada atividade. As três estimativas de tempo são a mais otimista (menor tempo para realizá-la), a mais provável, e a mais pessimista (maior tempo). O PERT assume que o tempo para a realização da atividade assume a forma de uma distribuição Beta Pert, com estes parâmetros.

Para calcular o tempo total para realização de um projeto, o PERT se baseia no cálculo do seu caminho crítico. O cálculo do caminho crítico é feito da mesma forma que no CPM, usando como tempo de realização das tarefas a média do tempo de realização de cada tarefa, que é dado por: (Otimista + 4 x Mais Provável + Pessimista) / 6. As atividades que estão no caminho são identificadas, e é possível dizer que o tempo para execução do projeto é a soma

dos tempos das atividades do caminho crítico. Como os tempos das atividades são dados por distribuições estatísticas, não é possível identificar um valor exato para o tempo total. No entanto, o teorema central do limite nos diz que a soma de variáveis aleatórias independentes tende a uma normal com média e variância calculados de acordo com a média e variância de cada uma das variáveis somadas. Por isso o resultado do PERT para a execução do PERT tende a uma distribuição Normal.

O PERT pode, então, ser classificado da seguinte forma segundo nossa classificação:

1. Atividades nos nós ou nos arcos
2. Sem transições não deterministas
3. Grafo acíclico
4. Sem restrição de recursos
5. Durações não deterministas

O modelo PERT pode fornecer informações sobre o tempo esperado para executar o projeto, a probabilidade de se completar o projeto antes, ou depois de determinada data, o caminho crítico que impacta a execução do projeto, as atividades que possuem folga e não estão no caminho crítico, e tempo de início e fim das atividades. No entanto, esta técnica possui algumas falhas que tornam perigosa a sua utilização no mundo real. Em 1º lugar o PERT assume uma distribuição Beta para a duração das atividades, o que pode não ser verdade em grande parte dos casos. Em 2º lugar, ainda que as durações sigam esta distribuição, o PERT assume que a distribuição do tempo total do projeto é o mesmo do caminho crítico. Como outros caminhos podem se tornar o caminho crítico se as suas atividades se atrasam, o PERT subestima consistentemente o tempo para a execução do projeto. Isto faz com que as estimativas obtidas utilizando PERT sejam muito otimistas, podendo causar resultados indesejáveis.

2.2.3 - Rede de atividades com restrição de recursos

A rede de atividades com restrições de recursos é o modelo de atividades utilizado para representar as redes de atividades utilizadas em um problema conhecido como RCPS. RCPSPs é a sigla para Resource Constrained Project Scheduling Problems (Problemas de Escalonamento de Projetos Limitados por Recursos). Através do texto esta categoria poderá ser referenciada também como RCPS, por questão de simplicidade. Os RCPSPs são uma superclasse de problemas relacionados ao escalonamento de atividades de um projeto quando existem recursos finitos para realizá-las. Eles envolvem, portanto, a atribuição de tarefas a um recurso, ou conjunto de recursos, com capacidade limitada de modo a atingir determinado objetivo.

Da mesma maneira que no CPM, as redes RCPS são formadas por um conjunto C de atividades em que cada atividade i tem uma duração fixa Duração(i), e um conjunto de atividades precedentes Prec(i). No entanto, diferentemente do CPM, a rede possui recursos

cuja quantidade disponível é representada por Disp(k), para um determinado recurso do tipo k. Cada atividade i consome Nec(i, k) do recurso k para ser executada. Quando uma atividade está sendo executada, os recursos necessários para sua execução ficam bloqueados até que ela seja terminada. Uma atividade não pode ser executada caso não estejam livres os recursos necessários para executá-la. O objetivo é encontrar um calendário que respeite simultaneamente as restrições de recursos e as relações de precedência minimizando a duração total do projeto. Por serem idênticas à do CPM, as relações de precedência podem ser expressas através de um diagrama de dependências igual ao utilizado para representar as atividades no CPM. Segundo a classificação que estamos utilizando o problema RCPS básico pode ser caracterizado como:

1. Atividades nos nós
2. Sem transições não deterministas
3. Grafo acíclico
4. Com restrição de recursos
5. Durações deterministas

Apesar de parecer ser um problema bastante simples, a inclusão das restrições de recursos aumenta a complexidade dos algoritmos de escalonamento. Segundo [2] o problema de encontrar a solução ótima para o tempo total de realização do projeto torna-se NP-Completo com a adição das restrições de recursos. São muito comuns, portanto, as soluções que utilizam heurísticas para tentar obter a melhor aproximação possível do tempo total. Em [7] pode ser encontrado um estudo comparativo das principais heurísticas utilizadas na solução do RCPS.

Conforme dito, o RCPS é uma superclasse de problemas que deu origem a muitas especializações. Inúmeras generalizações têm sido propostas de modo a incluir um ou outro detalhe. [8] divide os diferentes tipos de especializações do problema nas seguintes categorias:

1. Basic Single-Mode RCPSP (RCPSP de único modo)
2. Basic Multi-Mode RCPSP (RCPSP de múltiplo modo)
3. RCPSP with Non-regular objective functions (RCPSP com funções objetivo não regulares)
4. Stochastic RCPSP (RCPSP Estocástico)
5. Bin-packing-related RCPSP (RCPSP relacionados a empacotamento em recipiente)
6. Multi-resource-constrained Project scheduling problems – MRCPSP (Problemas de Escalonamento de Projetos Limitados por Múltiplos Recursos)

Dentre estas categorias, possui grande importância neste texto a categoria conhecida como RCPS Estocástico.

2.2.4 - Rede de atividades com restrições de recursos e durações aleatórias

A rede de atividades com restrições de recursos e durações aleatórias é o modelo utilizado para descrever as atividades utilizadas no problema RCPS Estocástico. O RCPS Estocástico é uma generalização dos problemas RCPS criada com o objetivo de adicionar ao problema básico a noção de incerteza quanto à duração das atividades.

No RCPS Estocástico o tempo de processamento de cada atividade é uma variável aleatória, que segue determinada distribuição de probabilidade. Isto ocorre da mesma maneira que vimos com o PERT. O RCPS Estocástico, no entanto, não impõe nenhuma limitação sobre qual a distribuição de probabilidade que a duração de uma atividade deve ter. Esta classe de problemas é muito mais realista que as vistas anteriormente, visto que no mundo real os recursos são finitos e o tempo para executar uma atividade é incerto. A função objetivo neste tipo de problema é diferente das comumente usadas nos problemas básicos RCPS. Não faz sentido falar em menor tempo para realização do projeto, porque apesar deste tempo poder existir, a sua probabilidade de ocorrência tende a ser muito pequena. O objetivo então é encontrar a distribuição de probabilidade do tempo total para a realização do projeto, ou seja, o mapeamento de cada valor possível na sua probabilidade de ocorrência.

Da mesma maneira que o RCPS determinista, o estocástico é um problema complexo, sendo também NP-completo. Ele pode ser classificado da seguinte maneira:

1. Atividades nos nós
2. Sem transições não deterministas
3. Grafo acíclico
4. Com restrição de recursos
5. Durações não deterministas

2.2.5 - GERT

GERT é a sigla para Graphical Evaluation and Review Technique (Técnica de Revisão e Avaliação Gráfica). Segundo [4], o GERT é uma combinação da teoria de redes, teoria de probabilidades e simulação que resulta em uma ferramenta eficiente para análise de sistemas. O GERT foi desenvolvido inicialmente por Alan B. Pritsker para o projeto espacial Apollo [9] [10]. A técnica GERT inclui a modelagem de projetos na forma de redes e a análise através de simulação. Isto resulta em resultados obtidos na forma de estatísticas operacionais (medidas do desempenho do sistema) em vez de uma simples solução ótima. O analista do sistema pode fazer experiências no modelo e observar a performance resultante com base nos resultados estatísticos obtidos.

As redes GERT são de certa maneira similares às redes CPM/PERT no sentido de que representam um conjunto de atividades conectadas em determinada ordem lógica. A diferença

principal para estas redes é que o GERT implementa transições não deterministas. Diferentemente dos modelos que vimos até agora, o GERT representa as atividades através dos arcos do grafo. Os nós do grafo são chamados de eventos, e eles possuem formas especiais que possuem determinados significados. Um nó de um diagrama GERT é dividido ao meio. O formato da parte direita do nó indica se a transição será determinista ou probabilística. Se ela for redonda todas as atividades na saída do nó serão executadas. Se ela for triangular apenas uma delas será selecionada. A parte esquerda do nó é utilizada para indicar o número de atividades que precisam ser encerradas para que o nó possa ser realizado. Cada nó possui um conjunto de arcos (atividades) que chegam, e um conjunto de arcos que saem dele. O número de atividades de chegada no nó que precisam ser terminadas para que as atividades de saída possam ser executadas é expressa na parte esquerda do nó. Esta parte esquerda é dividida ao meio e um número é colocado em cada metade. O número de cima indica o número de atividades que devem ser encerradas para a primeira realização do nó. Na de baixo se encontra o número de atividades a serem terminadas para que a segunda e subseqüentes realizações do nó sejam executadas. Na Figura 4 se encontra uma descrição desses nós:

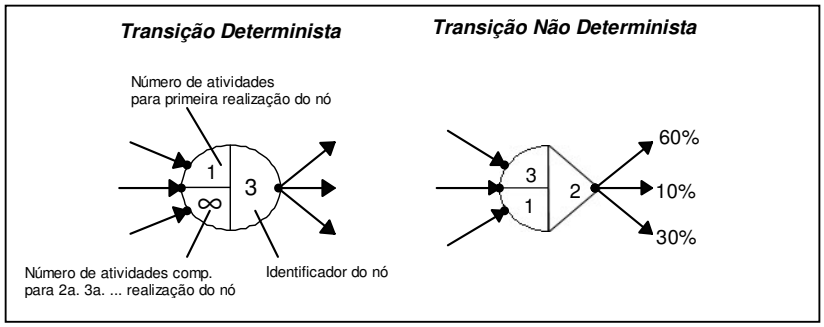

Figura 4: Nós de uma rede GERT

Os nós e atividades podem ser encadeados em uma determinada ordem lógica para representar uma rede de atividades. Esta rede pode possuir “loops”, ou seja, o grafo pode ser cíclico. Cada atividade no GERT possui uma distribuição de probabilidade associada que representa a sua duração. Diferentemente do PERT, a duração pode assumir diversas distribuições diferentes, não estando restrita apenas à Beta. O GERT, no entanto, não implementa a restrição de recursos para execução das atividades. O GERT pode ser classificado da seguinte maneira:

1. Atividades nos arcos
2. Permite transições não deterministas
3. Grafo cíclico
4. Sem restrição de recursos
5. Durações não deterministas

Um pequeno exemplo de uma rede GERT pode ser encontrado na Figura 5. Por questão de simplicidade a rede não possui “loops”, e possui apenas uma transição não determinista relacionada à necessidade de uma prova de conceito após a atividade de análise.

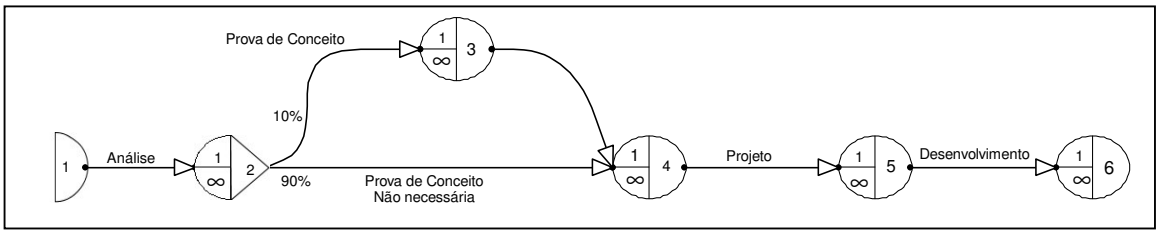

Figura 5: Exemplo de uma rede GERT

2.2.6 - Diagrama de atividades da UML

Diferentemente dos modelos analisados até aqui, o diagrama de atividades da UML não é um modelo quantitativo. Seu objetivo principal é fornecer um modelo que permita representar as redes de atividades de maneira clara, de modo que possa ser compreendida de maneira relativamente fácil.

A UML, ou Unified Modeling Language (Linguagem de Modelagem Unificada), é uma linguagem padrão para especificação, visualização, construção e documentação de artefatos de sistemas de software, assim como modelagem de negócios e outros sistemas não computacionais. Segundo [11] o desenvolvimento da UML começou no final de 1994 quando Grady Booch e Jim Rumbaugh da Rational Software Corporation começaram o trabalho de unificar os métodos Booch e OMT (Object Modeling Technique – Técnica de Modelagem de Objetos). No outono de 1995, Ivar Jacobson e sua compania Objectory se juntaram à Rational neste esforço de unificação. Os esforços de Booch, Rumbaugh e Jacobson resultaram no lançamento do documento UML 0.9 e 0.91 em Junho e Outubro de 1996. A UML possui um conjunto de diagramas que podem ser utilizados para representar aspectos específicos do desenvolvimento de projetos de sistemas. Os diagramas de casos de uso são utilizados para modelar requisitos do usuário, enquanto os diagramas de seqüência são usados para representar a maneira como os diferentes objetos do sistema interagem de modo a atender esses requisitos, e os diagramas de classe são utilizados para representar estes objetos estaticamente. Existem alguns outros diagramas além desses, mas o que possui importância para este estudo é chamado de diagrama de atividades.

Segundo [12] os diagramas de atividades são utilizados na modelagem de processos de negócios, para modelagem da lógica capturada por um caso de uso, ou para modelagem da lógica de regras de negócio. Os diagramas de atividades são representados por um grafo orientado cíclico, podendo conter nós de diferentes tipos. As arestas do grafo definem a ordem de execução entre as atividades e objetos representados no grafo. Existem diferentes tipos de

nós, cada um possuindo uma semântica específica. A seguir se encontra a descrição de alguns deles, e na Figura 6 a representação gráfica de cada um:

1. Nó inicial – O círculo negro que marca o início do diagrama. Todo diagrama possui apenas um estado inicial que é onde o fluxo se inicia.
2. Nó final – É representado por um círculo negro com outro círculo por fora. O diagrama pode possuir múltiplos nós finais, que são onde determinado fluxo de execução se encerra.
3. Atividade – Os retângulos com cantos arredondados representam as atividades. Uma atividade é a unidade onde alguma tarefa é executada.
4. Divisão – Uma barra negra (Horizontal ou Vertical) com uma seta entrando e múltiplas saindo. A divisão é um ponto do diagrama onde dois ou mais fluxos concorrentes são criados.
5. Junção – Uma barra negra (Horizontal ou Vertical) com múltiplas setas entrando e apenas uma saindo. A Junção é um ponto onde dois ou mais fluxos se juntam novamente em um único fluxo.
6. Decisão – Um losango com uma seta chegando e diversas saindo. As setas de saída normalmente incluem condições. Por isso, após uma decisão apenas um destes fluxos é selecionado.
7. Junção da Decisão – Um losango com múltiplas setas entrando e apenas uma saída. Marca o ponto onde os diversos fluxos que saem de uma decisão são unidos novamente.
8. Emissão de Evento / Recebimento de Evento – Um evento ocorre quando o fluxo de execução passa de uma parte do diagrama para outra. Quando o fluxo chega em um objeto de emissão de evento (representado por um pentágono) ele automaticamente passa para o ponto onde se encontra o objeto de recebimento de evento (representado por um pentágono obtuso) associado a ele.
9. Produto / Recurso – Representa um produto produzido pela execução de uma atividade, ou um recurso necessário para executá-la. É representado por um quadrado, e pode estar associado a uma atividade por setas pontilhadas (Fluxo de Objetos).
10. Raias – As raias definem partições do diagrama. São muito utilizadas para identificação de responsabilidades. Atividades localizadas na mesma raia utilizariam o mesmo tipo de recurso.

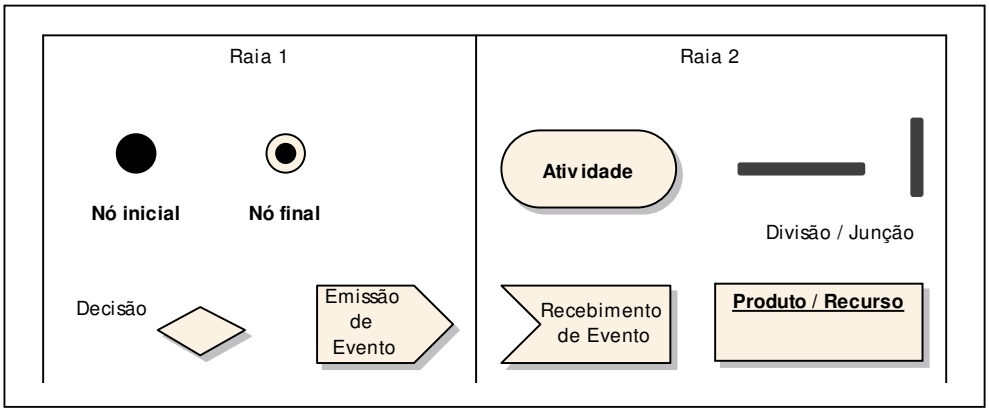

Figura 6: Elementos do diagrama de atividades

Na Figura 7 se encontra um exemplo de um diagrama de atividades que utiliza os objetos definidos acima para representar um processo simplificado de desenvolvimento de software.

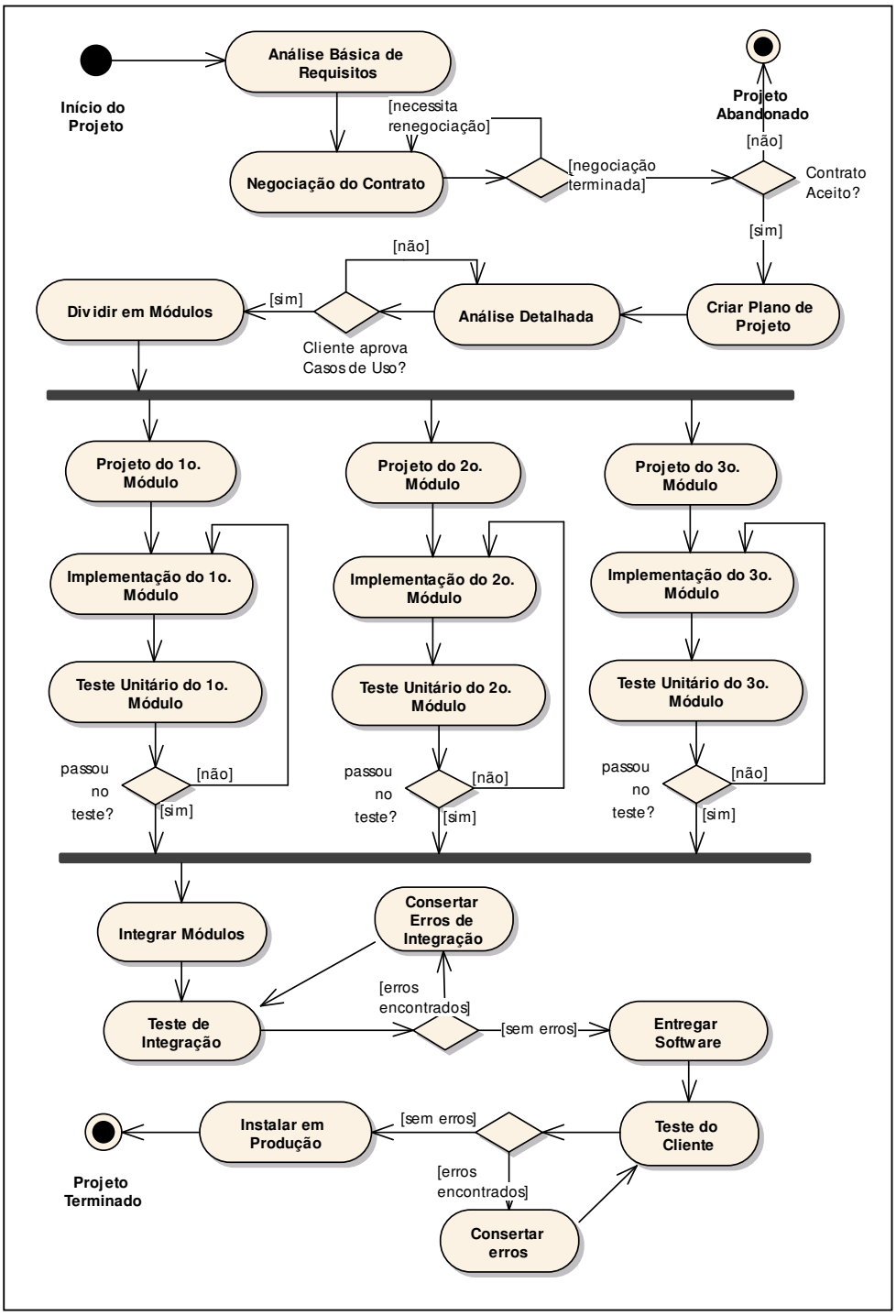

Figura 7: Exemplo de diagrama de atividades da UML

Os diagramas de atividades podem ser utilizados para modelar os mesmos tipos de redes de atividades que mostramos anteriormente. Conforme podemos ver, ele permite a representação de decisões, possuindo a habilidade portanto de representar transições não deterministas. Por não ser um modelo quantitativo, a duração de cada atividade não é representada no modelo. Os recursos necessários para executar cada atividade também não

são explicitados quantitativamente. Este modelo pode ser classificado da seguinte forma na nossa classificação:

1. Atividades nos nós
2. Com transições não deterministas
3. Grafo cíclico
4. Sem restrição de recursos
5. Sem durações

Apesar do seu poder de representação ser muito superior ao dos modelos anteriores falta aos diagramas de atividades uma natureza quantitativa.

2.3 - Algoritmos para cálculo do tempo de execução

Conforme vimos na seção anterior, a maior parte dos modelos para representação de atividades são quantitativos. Isso quer dizer que eles representam a duração das atividades através de números, permitindo assim que análises numéricas sejam feitas.

Ao se fazer uma análise quantitativa de uma determinada instância de uma rede de atividades, é necessário selecionar uma função objetivo para estudo. A função objetivo indica qual das variáveis disponíveis será analisada de modo a se obter dela um valor mínimo, máximo ou algum valor intermediário. Em um modelo existem diversas variáveis que podem ser utilizadas como função objetivo, como por exemplo o número de recursos livres durante a execução das atividades no RCPS, ou o menor número de atividades no caminho crítico no PERT, e etc. A função objetivo mais comum é conhecida como “minimum makespan” [18], ou seja, menor tempo para realização do projeto. O menor tempo para realização faz sentido em modelos deterministas como o CPM ou o RCPS. Em modelos não deterministas, como o PERT ou o GERT a função objetivo é a distribuição de probabilidade do tempo total para execução do projeto.

Existem diversas técnicas diferentes para calcular o tempo total do projeto. Cada técnica possui a sua complexidade computacional, e se adapta bem se aplicada a determinado tipo de modelo. Nesta seção serão mostradas algumas técnicas utilizadas para o cálculo do “minimum makespan” de redes de atividades similares à utilizada no RCPSP, visto que o modelo que será apresentado neste trabalho tem muitas características semelhantes a este modelo. Os algoritmos para solução do RCPS podem ser enquadrados em duas vertentes: algoritmos de solução exata e algoritmos de solução aproximada. Será analisada aqui também uma técnica para obtenção da distribuição do tempo total de projeto, muito utilizada para análise em modelos do tipo RCPS Estocástico.

2.3.1 - Algoritmos para solução exata do RCPSP

O objetivo destes algoritmos é obter o menor tempo possível para execução de determinada rede de atividades, respeitando as restrições de recurso contidas na rede. Conforme já observado este problema é NP-Completo, requerendo grande capacidade computacional para ser resolvido. Os métodos para solução exata têm sido mais freqüentemente usados para a geração de soluções "benchmark", ou seja, soluções utilizadas para verificação da qualidade das soluções aproximadas obtidas para o problema.

As estratégias que tem sido mais utilizadas para a minimização do "makespan" são do tipo: programação dinâmica [14], programação zero–um [15] e esquemas de enumeração explícita com branch and bound [16] [17] [18], sendo esta última a mais utilizada ultimamente. O Branch and bound é "uma técnica algorítmica para encontrar a solução ótima, utilizando a melhor solução encontrada para limitar a busca pela solução ótima. Se uma solução parcial não puder melhorar a melhor solução, ela é abandonada" [19].

Por serem muito custosas as técnicas de solução ótima não serão utilizadas neste trabalho. Utilizaremos com mais freqüência as técnicas para solução aproximada.

2.3.2 – Algoritmos para solução aproximada do RCPSP

Os algoritmos cujo objetivo são a produção de soluções próximas do resultado ótimo são chamados de métodos para solução aproximada. Estes métodos são também freqüentemente conhecidos apenas como métodos heurísticos.

Heurística é uma técnica que busca a resolução de problemas através da busca de soluções sub-ótimas, com um custo computacional razoável. Estas técnicas não são capazes de garantir a otimalidade da solução, nem indicar o quão próximo a solução encontrada se encontra do valor ótimo.

As heurísticas de para o RCPS se enquadram basicamente em quatro categorias distintas: "branch and bound" truncado, conceitos de arcos disjuntivos, abordagens meta-heurísticas e escalonamento baseado em regras de prioridade. O esquema "branch and bound" truncado é análogo ao esquema "branch and bound", mas em vez de rastrear toda a árvore conduz apenas uma exploração parcial, baseando-se em algum critério de decisão para limitar o espaço de busca do algoritmo [20]. O método dos arcos disjuntos aborda o problema dos escalonamento através da extensão das relações de precedência acrescentando arcos adicionais de modo a evitar que atividades sejam escalonadas ao mesmo tempo quando isso é proibido por escassez de recursos [20]. As estratégias de meta-heurísticas exploram o espaço viável de maneira global. Elas surgiram recentemente, tendo maior destaque o "Taboo Search" [21], "Simulated Annealing" [22] [23], algoritmos genéticos [24] e "Ant Colony Optimization" [25].

O último dos esquemas representa a classe mais importante, apesar de ser uma das mais antigas. Isso acontece, entre outras razões, devido à sua facilidade na elaboração, implementação e baixo custo computacional, em termos de tempo e memória. Este é o método utilizado na maioria das ferramentas comerciais de gerenciamento de projeto [26]. Este esquema é baseado em dois componentes básicos: uma regra de prioridade e um esquema de escalonamento. O objetivo do esquema de escalonamento é produzir um cronograma viável, à

medida que acrescenta as atividades em um escalonamento parcial. Quando existe mais de uma tarefa para ser escalonada em determinado momento, a regra de prioridade entra em jogo para fazer o desempate. Esta regra de prioridade tem impacto direto na qualidade da solução obtida. Esta forma de heurística para escalonamento do RCPS será utilizada neste trabalho para realizar o escalonamento das atividades.

2.3.3 - Simulação de Monte Carlo aplicável ao RCPSP Estocástico

No nível elementar, a simulação de Monte Carlo é um conceito surpreendentemente simples. O crédito pela invenção do método de Monte Carlo é dado a Stanislaw Ulam, um matemático polonês que trabalhou para John Von Neumann no Projeto Manhattan dos Estados Unidos durante a Segunda Guerra Mundial [27]. Ulam é conhecido por projetar a bomba de hidrogênio com Edward Teller em 1951. Ele inventou o método de Monte Carlo em 1946 quando ponderava sobre as possibilidades de ganhar no jogo de cartas chamado solitária. Como citado em [28], Ulam descreve o incidente desta maneira:

"Os primeiros pensamentos e tentativas que eu fiz para praticar [o método de Monte Carlo] foram sugeridos por uma questão que me ocorreu em 1946 quando eu estava convalescendo de uma doença e jogava solitária. A questão era: Quais são as chances de uma partida de solitária com 52 cartas ser completada com sucesso? Após algum tempo tentando estimar este valor por puro cálculo combinatório, eu me perguntei se um método mais prático do que o "pensamento abstrato" poderia ser jogar a solitária, digamos cem vezes, e simplesmente observar e contar o número de jogadas com sucesso. Já era possível vislumbrar o início da nova era de computadores rápidos, e eu imediatamente pensei em problemas de difusão de nêutrons e outras questões de física matemática, e com mais generalidade como transformar processos descritos por determinadas equações diferenciais em sucessões de operações randômicas equivalentes. Mais tarde [em 1946, eu] descrevi a idéia a John Von Neumann, e começamos a planejar cálculos reais".

O método de Monte Carlo, como é entendido nos dias de hoje, engloba qualquer técnica para amostragem estatística utilizada para obter soluções aproximadas de problemas quantitativos. Ele pode ser aplicado com bastante eficiência em problemas onde a solução analítica é extremamente difícil, ou impossível.

O método de Monte Carlo tem sido aplicado com grande sucesso para a resolução de problemas do tipo RCPS Estocástico [29]. Este tipo de problema possui as características ideais para ser resolvido através da simulação: é um problema intratável de maneira analítica e que possui como saída resultados estatísticos.

2.4 - Conclusões

Conforme vimos nesta seção, existem diversos tipos diferentes de modelos de atividades. Cada um possui as suas peculiaridades, representando um conjunto restrito de

características e incertezas existentes no mundo real. Existe espaço, no entanto, para a criação de um modelo que implemente as características mais complexas de cada um deles, gerando um modelo mais poderoso e completo. Nenhum dos modelos apresentados permite a representação de durações não deterministas, restrições de recursos e transições não deterministas em um mesmo modelo.

Nos últimos tempos temos visto um crescimento muito grande na utilização da UML em todos os aspectos da área de desenvolvimento de sistemas. A UML tem ganhado espaço pela sua universalidade, e ampla aceitação no mercado atual. Dentro deste crescimento, tem obtido um grande impulso a utilização de diagramas de atividades para a representação de modelos de processos, que nada mais são do que redes de atividades encadeadas logicamente. Os diagramas de atividades possuem um grande poder de expressão sendo mais claros que os modelos observados anteriormente. Os diagramas de atividades são, no entanto, modelos apenas descritivos, utilizados para expressar uma visão da realidade no papel. Eles não possuem um aspecto quantitativo como os outros modelos. É possível, no entanto, propor um modelo quantitativo que forneça aos diagramas da UML algumas características que estes não possuem. Este modelo adicionaria informações sobre estimativas de tempo e probabilidade aos objetos apresentados nos diagramas, permitindo que análises quantitativas sejam realizadas.

3 - O Modelo RANDRE

3.1 - Motivação

Conforme vimos no capítulo anterior, existem diversos modelos quantitativos de redes de atividades, cada uma delas com características bastante específicas. Nenhum deles, entretanto, implementa durações aleatórias, restrição de recursos e transições não deterministas em um único modelo.

O modelo PERT introduziu a noção de durações aleatórias ao permitir que a duração de uma atividade seja expressa através de uma distribuição estatística. Os modelos de atividades são comumente utilizados de maneira preditiva, ou seja, tem a função de permitir ao analista estudar o comportamento de um projeto ou processo antes que ele seja executado. Muitas vezes é impossível predizer a duração exata de uma atividade antes que ela seja executada. Por isso, para que um modelo seja útil, é necessário que ele permita que as durações das atividades sejam definidas através de aproximações no lugar de valores exatos. A maneira mais comum de se representar essas aproximações é através da utilização de distribuições estatísticas.

Um outro aspecto muito importante na execução de redes de atividade são os recursos utilizados para executar uma atividade. No mundo real as atividades são sempre executadas por alguém, ou alguma máquina, utilizando um determinado conjunto de ferramentas. A esse conjunto de materiais, humanos ou não, necessários para executar determinada atividade chamamos de recursos. O principal problema relacionado aos recursos é que no mundo real eles são escassos ou finitos. O fato de não existirem recursos suficientes para executar todas as atividades ao mesmo tempo faz com que as atividades atrasem. Cronogramas obtidos através de modelos que ignoram a dependência de recursos tendem a ser otimistas, resultando em tempos de término menores do que o esperado na realidade.

Além da incerteza quanto às durações, existe a incerteza quanto ao resultado obtido em determinados eventos relevantes no modelo de atividades. Ao se realizar um teste em um software, por exemplo, dois resultados podem ser obtidos: ou o programa passa no teste ou ele não passa. Se um programa não passa no teste ele precisa ser consertado, fazendo com que mais atividades sejam executadas e aumentando o tempo total de execução. Chamamos de transições não deterministas, ou decisões, estes pontos onde um entre n caminhos pode ser selecionado. Quando um modelo permite representar decisões ele tira do analista a tarefa de prever qual dos caminhos será escolhido, e faz com que os resultados finais obtidos reflitam a incerteza inerente à decisão.

Por todos os motivos apresentados acima, é importante a existência de um modelo que possua a capacidade de representar transições não deterministas, durações aleatórias e

recursos escassos. Por isso neste capítulo apresentaremos um novo modelo de atividades capaz de representar todas essas informações.

3.2 - Definição da RANDRE

O modelo de atividades que é o principal objeto de estudo desta dissertação será chamado aqui de RANDRE - Rede de Atividades Não Determinista com Recursos Escassos. Ele é um modelo de atividades semelhante aos vistos no capítulo anterior. Diferentemente dos outros modelos, no entanto, ele possui a capacidade de representar durações aleatórias, transições probabilísticas e recursos limitados.

A RANDRE surgiu através de uma modificação no modelo de atividades utilizado para descrever as redes utilizadas no RCPSP Estocástico. As redes de atividades foram alteradas para suportar uma construção comum nos diagramas de atividades da UML conhecida como decisão. Uma decisão é um elemento do diagrama que possui uma única entrada e múltiplas saídas, sendo que durante a sua execução apenas uma entre as suas saídas é selecionada. Na RANDRE funções discretas de probabilidade são associadas às saídas da decisão, sendo utilizadas para determinação da saída a ser escolhida.

A rede RANDRE pode ser descrita por um grafo G, direcionado podendo ser cíclico ou acíclico, com um conjunto de nós $V(G) = A \cup D$ onde $A = \{A_1, A_2, A_3, ... A_k\}$ é um conjunto de atividades e $D = \{D_1, D_2, D_3, ... D_j\}$ é um conjunto de decisões. O grafo é composto ainda por um conjunto A(G) contendo pares ordenados (i, j) onde i e $j \in V(G)$. Este conjunto contém as arestas do grafo, que são direcionadas, ou seja, $(i, j) \neq (j, i)$.

Cada atividade contida em A está associada a um conjunto $E(a) = \{E_1, E_2, ... E_k\}$ onde cada E_i é uma função de probabilidade que associa duração à sua probabilidade de ocorrência. Estas funções representam o tempo necessário para executar cada atividade. Cada uma destas distribuições está associada a uma instância de execução da atividade, por exemplo, a primeira distribuição está associada à primeira execução da atividade, a segunda à segunda execução e assim por diante. Se uma atividade for executada um número de vezes maior que a sua quantidade de distribuições, a última é a utilizada. Por exemplo, uma atividade que possui três distribuições de duração, quando executada pela quarta vez, tem a sua duração dada pela terceira distribuição. A função a ser utilizada, deve ser escolhida de acordo com o que se estima ser a duração da atividade na realidade.

O grafo G possui ainda um conjunto $R = \{(R_1, x_1), (R_2, x_2), ... (R_n, x_n)\}$ que representa o conjunto de recursos disponíveis para execução das atividades. No par ordenado (R_i, x_i), R_i é o tipo do recurso e x_i a quantidade de recursos disponíveis deste tipo. Cada atividade a possui um conjunto $R(a) = \{(R_1, y_1), (R_2, y_2), ... (R_n, y_n)\}$, onde R_i é o tipo do recurso e y_i a quantidade de recursos livres necessários para que a atividade a possa ser executada. Quando uma atividade a está para entrar em execução, devem existir y_i recursos do tipo R_i, para cada $i \in R(a)$, livres. Ao entrar em execução a atividade bloqueia estes recursos, decrementando o número de recursos disponíveis do tipo R_i de y_i unidades. Quando termina de executar, a

atividade libera novamente os recursos. A RANDRE permite a representação apenas de recursos renováveis, o que significa que os recursos podem ser utilizados novamente após liberados.

Graficamente, as redes RANDRE podem ser representadas da mesma maneira que as redes CPM, com círculos para representar as atividades, e utilizando losangos para representar as decisões. Abaixo se encontra um exemplo de uma pequena RANDRE:

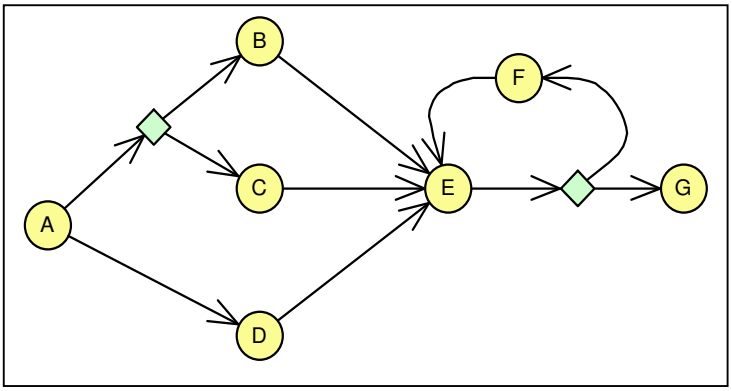

Figura 8: Exemplo de uma pequena RANDRE

Este modelo de rede é muito genérico, permitindo que sejam construídas redes muito complexas. Algumas dessas redes podem não fazer sentido, e outras podem tornar muito difícil a tarefa de identificar a relação de precedência entre as atividades. Um exemplo de rede patológica pode ser encontrado na Figura 9.

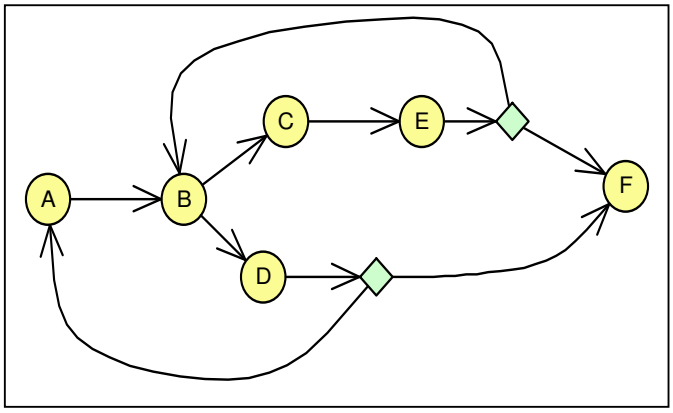

Figura 9: Exemplo de uma RANDRE patológica

É possível observar que é difícil identificar o nó inicial, visto que a rede não possui nós sem arestas incidentes. Além disso, podemos observar dois “loops” aninhados de maneira intercalada. Isto dificulta muito a identificação de qual tarefa deve ser executada em cada momento. Para evitar a ocorrência deste tipo de rede é necessário definir regras que permitam identificar o que é uma RANDRE bem formada. As redes RANDRE válidas são definidas indutivamente da seguinte forma:

1. Uma única atividade é uma RANDRE bem formada (RANDRE trivial)
2. Se AN_1, AN_2, AN_3, ..., AN_n são RANDREs válidas, e D_i é uma decisão, então:
    a. A RANDRE em série (Figura 10a) também é uma RANDRE válida.
    b. A RANDRE iterativa (Figura 10b) também é uma RANDRE válida.
    c. A RANDRE em paralelo (Figura 10c) também é uma RANDRE válida.
    d. A RANDRE decisão (Figura 10d) também é uma RANDRE válida.

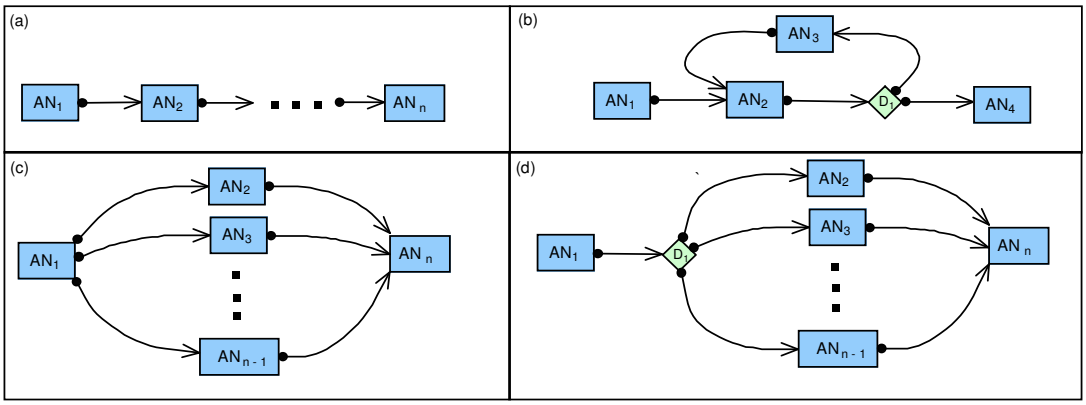

Figura 10: RANDREs básicas

Os blocos básicos acima podem ser aninhados da maneira necessária para construir RANDREs maiores e mais complexas. É importante salientar que todos os blocos acima começam por uma RANDRE, e terminam exatamente em uma RANDRE. Indutivamente sabemos, portanto, que cada RANDRE começa e termina em uma atividade, fazendo com que a unificação entre as RANDREs aconteça sempre unindo o nó inicial de uma ao nó final de outra. Estes blocos básicos podem parecer, à primeira vista, muito restritivos. Entretanto, estas restrições não são tão severas. Uma RANDRE "virtual" composta por atividades de duração nula pode sempre ser utilizada para completar a rede, sem que haja qualquer modificação no seu significado. Abaixo se encontra um exemplo da decomposição da RANDRE na Figura 8 utilizando os blocos elementares de construção. As atividades N1, N2, N3, e N4 representadas com círculo tracejado são atividades de duração nula utilizadas para completar o diagrama:

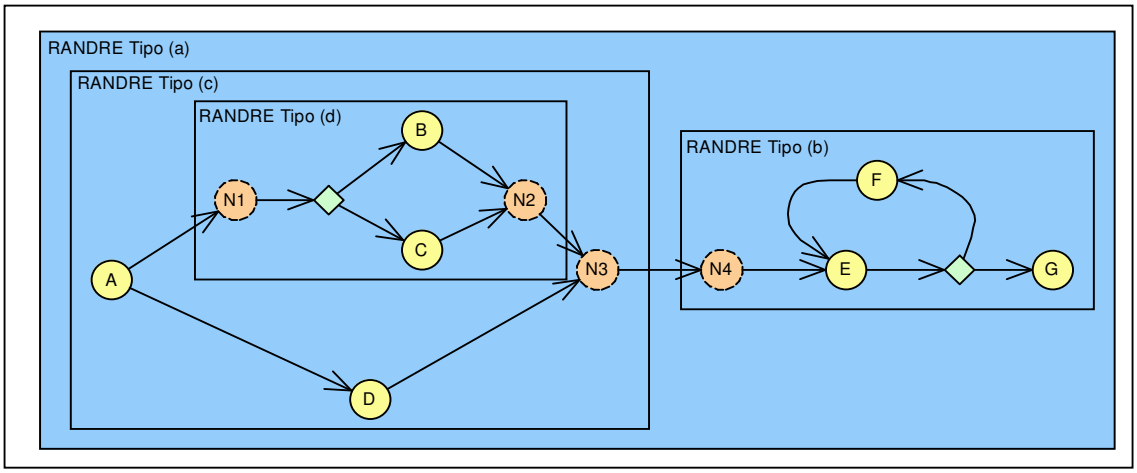

Figura 11: Decomposição de uma RANDRE em RANDREs básicas

3.3 – Relações de precedência

O conjunto de arestas de uma RANDRE é composto por arcos orientados que definem as relações de dependência temporal envolvendo as atividades e as decisões. Em uma rede CPM/PERT, a existência de uma aresta ligando uma atividade A a uma atividade B significa que A pode começar apenas após o término de B. Na RANDRE a relação de precedência é muito mais complexa do que isso.

Nas redes CPM/PERT, todas as atividades representadas no modelo são executadas exatamente 1 vez. Isso é diferente do que acontece na RANDRE, na qual uma atividade pode não ser executada ou pode ser executada mais de uma vez, no caso dos "loops". Por causa disso, pode existir uma seta ligando a atividade A à atividade B e, não necessariamente, B precisa esperar o término de A para começar. Para tentar estabelecer uma regra geral de precedência, vamos voltar aos blocos básicos de construção de RANDREs da Figura 10, e identificar as regras existentes para cada uma das sub-redes. Quando dizemos que a RANDRE AN_i só pode começar após a RANDRE AN_j, queremos dizer que a primeira atividade de AN_i só pode começar após a última atividade em AN_j. Abaixo estão as regras de precedência para as quatro construções básicas:

1. RANDRE em série (Figura 10a)
    a. AN_2 só pode iniciar após o término de AN_1.
    b. AN_3 só pode iniciar após o término de AN_2.
    c. ...
    d. AN_n só pode iniciar após o término de AN_{n-1}.
2. RANDRE iterativa (Figura 10b).
    a. Se é a primeira execução de AN_2:
        i. AN_2 só pode ser executada após a primeira execução de AN_1.

b. Senão
    i. A i-ésima execução de AN_2 só pode começar após a (i-1)-ésima execução de AN_3.
c. A i-ésima execução de AN_3 só é feita se a aresta D_1-AN_3 for sorteada na i-ésima passagem por D_1. Neste caso, a i-ésima execução de AN_3 só pode começar após a i-ésima execução de AN_2.
d. AN_4 é executada quando na i-ésima passagem por D_1 a aresta D_1-AN_4 for selecionada. Neste caso, a AN_4 só pode começar após a i-ésima execução de AN_2.

3. RANDRE em paralelo (Figura 10c)
    a. AN_2 só pode iniciar após o término de AN_1.
    b. AN_3 só pode iniciar após o término de AN_1.
    c. ...
    d. AN_{n-1} só pode iniciar após o término de AN_1.
    e. AN_n só pode iniciar após o término de AN_2, AN_3, AN_4, ... e AN_{n-1}.
4. RANDRE decisão (Figura 10d)
    a. AN_2 não é executada se não for selecionada no sorteio de D_1. Se for a selecionada só pode iniciar após o término de AN_1.
    b. AN_3 não é executada se não for selecionada no sorteio de D_1. Se for a selecionada só pode iniciar após o término de AN_1.
    c. ...
    d. AN_{n-1} não é executada se não for selecionada no sorteio de D_1. Se for a selecionada só pode iniciar após o término de AN_1.
    e. AN_n só pode iniciar após o término da AN_j, onde AN_j é a RANDRE sorteada em D_1.

3.4 – Equivalência RANDRE x Diagrama de atividades da UML

O modelo RANDRE, que está sendo proposto aqui, tem como objetivo fornecer um esquema prático para simulação de redes de atividades não deterministas. Ele tem como objetivo principal ser um meio de simulação para ferramentas existentes que são capazes de representar redes de atividades, de maneiras até melhores que a RANDRE.

O diagrama de atividades da UML é uma ferramenta poderosa para representação de cadeias de atividades. Esta forma de representar atividades é muito difundida, existindo diversas ferramentas de diagramação disponíveis. Os diagramas de atividades da UML corretamente construídos, ou seja, seguindo os padrões definidos aqui, podem ser mapeados em RANDREs válidas. Desta maneira, a técnica de simulação de RANDREs pode ser utilizada para simular redes de atividades da UML.

A sintaxe dos diagramas de atividades é muito genérica, permitindo que sejam produzidos diagramas que muitas vezes não fazem sentido. Por causa disso, é necessário limitar o escopo das construções possíveis, evitando que tais redes mal formadas possam ser produzidas. Para produzir um método de mapeamento de diagramas de atividade UML para RANDREs, vamos listar as construções básicas permitidas nos diagramas que poderão ser mapeados em RANDRE. Em seguida, será mostrado como elas podem ser transformadas em RANDREs válidas. Os blocos de construção da UML são construídos de maneira recursiva, da mesma maneira que fizemos nos blocos básicos da RANDRE. Os quadrados nos diagramas abaixo representam sub-diagramas UML válidos. Abaixo estão as principais construções da UML mapeada em RANDREs:

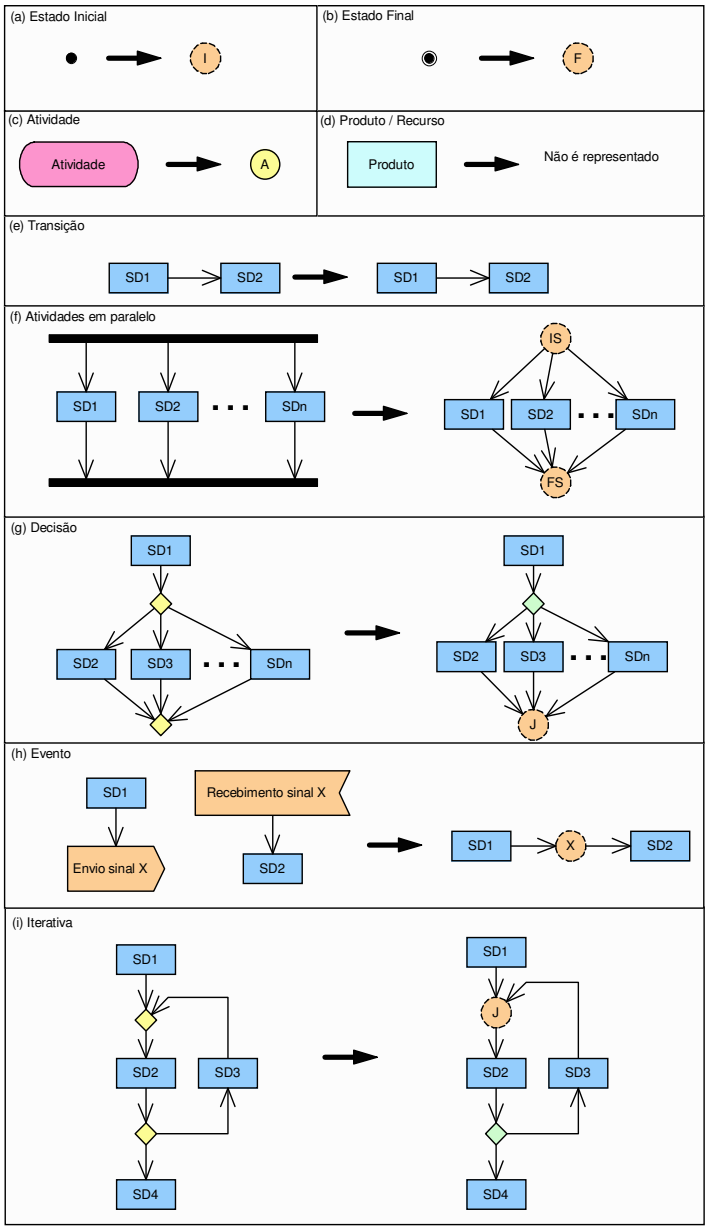

Figura 12: Mapeamento diagrama de atividades da UML x RANDRE

Na Figura 12 (a) e (b), estão representados os mapeamentos dos estados iniciais e finais do diagrama. Todos os dois podem ser mapeados diretamente para atividades com duração nula. O mapeamento de uma atividade é apresentado em (c) e pode ser feito diretamente, sem nenhuma alteração na atividade. Os produtos (d) são elementos especiais representados na UML que servem para identificar elementos produzidos por atividades, ou necessários para sua execução. Ele poderia ser utilizado para identificar os recursos necessários para executar uma atividade, mas por questão de simplicidade não o mapeamos para as redes RANDRE. Os produtos não tem influência na seqüência de execução de um diagrama de atividades, por isso não existe problema em não representá-los na RANDRE.

No diagrama (e) é mostrado o mapeamento de uma transição, que é uma construção básica da UML. Ela conecta dois sub-diagramas SD1 e SD2, que podem ser transformados em uma RANDRE em série ligando o mapeamento de SD1 a SD2. Em (f) é mostrada a construção que representa a execução de atividades em paralelo, que na UML é representada através de barras de sincronização. Na RANDRE cada barra é mapeada em uma atividade com duração nula, e a construção resultante se assemelha à RANDRE de atividades em paralelo. Em (g) está representada uma decisão, na qual o fluxo de execução se divide em n sub-diagramas. Na UML o nó de decisão é utilizado tanto para dividir os caminhos de execução quando para reuni-los no final. Quando utilizado para juntar dois caminhos, possuindo múltiplas entradas e apenas uma saída, ele é chamado de junção. A junção é mapeada diretamente em uma atividade com duração nula, enquanto a decisão é mapeada na decisão da RANDRE. A RANDRE resultante é semelhante à RANDRE decisão.

Os eventos são uma construção específica da UML para facilitar a representação gráfica dos diagramas. Quando o fluxo chega a um ponto de emissão de um evento ele é passado para o elemento de recebimento associado a ele. Por causa disso o par de elementos de emissão e recebimento de eventos pode ser mapeado diretamente para uma única atividade nula, como mostrado em (h). A construção iterativa, ou “loop”, representada em (i) pode utilizar as mesmas convenções de transformação mostrada em (g). A junção é mapeada em uma atividade nula, e a decisão em uma decisão equivalente.

Todas as RANDREs mostradas acima são válidas. As construções da UML podem ser convertidas, portanto, diretamente em redes RANDRE. As únicas informações que não podem ser obtidas à partir dos diagramas da UML são as distribuições de duração das atividades, as funções de probabilidade utilizadas para sortear os resultados das decisões, e os recursos necessários para executar cada atividade. Essas informações podem ser preenchidas posteriormente à conversão do diagrama, permitindo simular o diagrama de atividades da mesma maneira que simularíamos uma RANDRE.

4 - Algoritmos de Simulação

O poder das redes RANDRE está fortemente baseado no seu mecanismo de simulação, que permite a obtenção de dados estatísticos. Estes dados podem ser úteis para a avaliação do desempenho de redes de atividades. Este mecanismo não é simples, sendo composto por um grupo de algoritmos de simulação que atuam em conjunto para fornecer os resultados estatísticos desejados. Os mecanismos de simulação que serão apresentados aqui são baseados na geração de cenários de execução, escalonamento destes cenários e armazenamento dos resultados em histogramas.

Podemos dividir os algoritmos apresentados aqui em três categorias, conforme mostra a Figura 13.

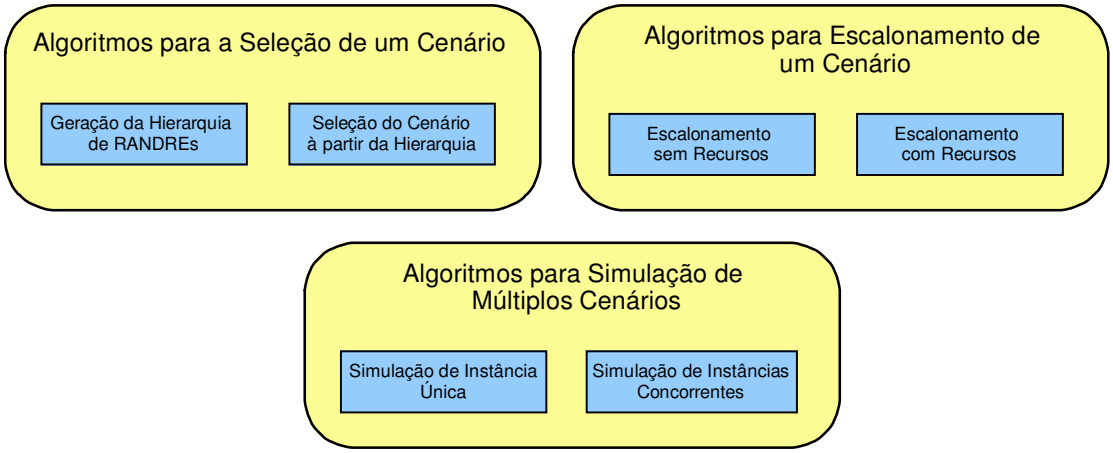

Figura 13: Algoritmos de simulação

A primeira é a categoria dos algoritmos que tem por objetivo obter um cenário de execução determinista a partir da rede não determinista original. Quando falarmos em cenário de execução estamos falando em um grafo originado pela seleção aleatória das saídas das decisões, e sorteio das durações do projeto segundo as suas distribuições de probabilidade. Neste algoritmo, uma RANDRE é "transformada" em uma rede de atividades que possui as mesmas características do RCPS, e representa um possível cenário de execução da RANDRE.

A segunda classe de algoritmos são aqueles utilizados para escalonar uma rede de atividades, gerando os seus tempos de início e fim. Após a execução dos algoritmos apresentados acima, uma rede RCPS é obtida, podendo ser escalonada utilizando um algoritmo que obtenha a solução para o RCPSP clássico. O algoritmo que mostraremos aqui utiliza heurísticas para selecionar qual atividade será executada, quando mais de uma pode ser selecionada, gerando soluções que dependem da heurística escolhida. As atividades podem ser escalonadas ainda, utilizando algoritmos semelhantes ao utilizado para solução do CPM. Como o CPM não possui dependências de recursos, os projetos escalonados utilizando este

algoritmo ignoram as dependências de recurso contidas na RANDRE. Ele é útil para comparações entre os resultados dos dois tipos de saída, permitindo identificar pontos onde a falta de recurso está atrasando o projeto.

O último tipo diz respeito aos algoritmos utilizados para executar a simulação de Monte Carlo. Os algoritmos apresentados acima geram como resultado o valor de tempo total de execução para um cenário de execução do projeto. Como a definição de cada cenário é aleatória, para obter uma melhor aproximação do tempo total de projeto é necessário executar múltiplos cenários. Serão apresentados aqui dois modos de se executar este cenário. O primeiro, que chamaremos de simulação de instância única, executa cada cenário da RANDRE em separado, um de cada vez. Ele consiste de uma execução repetitiva dos passos de seleção de um cenário, escalonamento de atividades, e anotação dos resultados. Ele é mais utilizado para a classe de redes de atividades que chamamos de projeto, ou seja, um conjunto de atividades que será executado uma única vez. Já o segundo tipo é mais aplicável à simulação de processos, que são conjuntos de atividades que podem ser executadas inúmeras vezes, de maneira repetitiva e concorrente. Um exemplo de um projeto seria o da construção de um prédio, afinal cada prédio é diferente de outro. Um exemplo de um processo poderia ser a montagem de carros em uma grande montadora de automóveis, nas quais a seqüência de atividades para construção de um carro é executada da mesma forma múltiplas vezes. O algoritmo para simulação de processos, que chamaremos de simulação de múltiplas instâncias concorrentes, permite que dois ou mais cenários diferentes de uma rede executem simultaneamente, competindo por recurso. Isto permite identificar possíveis gargalos de recurso, que podem ser um problema quando aquele processo for executado.

Neste capítulo mostraremos o funcionamento de cada um destes algoritmos, explicando qual é o seu objetivo, mostrando o seu código em uma linguagem estruturada. Tentaremos mostrar também que o algoritmo está certo, seja através da sua análise ou de exemplos de execução utilizando uma rede de atividades simples.

4.1 – Método para simulação da RANDRE

A principal característica da RANDRE é a sua capacidade de simular redes de atividades, gerando resultados estatísticos que podem ser utilizados para avaliar a sua performance. Para que a RANDRE possa ser simulada é necessário que seja definido um método que indique como essa simulação pode ser realizada, e quais dados podem ser obtidos à partir dela.

A função objetivo que desejamos obter da RANDRE é a distribuição de probabilidade do “minimum makespan”, ou seja, tempo mínimo para realização de um projeto. Conforme vimos, no entanto, em modelos que implementam restrições de recursos o problema de encontrar o menor tempo para realização do projeto é NP-completo. Como a RANDRE possui restrições de recursos, a obtenção do “minimum makespan” em uma única execução do projeto não é um problema simples de ser resolvido. Em modelos com recursos escassos a melhor

maneira de resolver este problema é a obtenção de aproximações do “minimum makespan” através da utilização de heurísticas. Conforme foi dito, as heurísticas são utilizadas para selecionar qual atividade deve ser colocada em execução quando mais de uma está disponível para ser executada, ou seja, todas as atividades com as quais ela possui relação de precedência já foram executadas. A escolha desta atividade tem influencia direta no tempo total obtido, porque dependendo dos recursos utilizados por esta tarefa, outras atividades podem ser impedidas de executar, aumentando o tempo para execução do projeto. A escolha da heurística tem influência direta, portanto, no quão próximo está o resultado obtido do tempo mínimo possível.

A RANDRE é um modelo probabilístico, possuindo durações e transições não deterministas. Esta classe de problema é muito complexa, inibindo tentativas de soluções analíticas. Isto acontece porque o problema combina uma generalização de processos Markovianos [31] (devido à adição das restrições de recursos) com uma generalização do RCPSP (devido à adição da incerteza quanto à duração da atividade e caminho de execução). Por esta razão, nós buscamos soluções através da utilização do método de Monte Carlo, na qual a simulação é utilizada para obtenção da distribuição de probabilidade do tempo total.

O método de simulação apresentado aqui é baseado na repetição da combinação de duas fases, quantas vezes forem necessárias. A primeira fase é responsável pela obtenção de uma instância determinista da rede não determinista, através da seleção de um de seus possíveis caminhos de execução. Aqui o termo “caminho” significa um possível cenário determinista, ou seja, uma rede representada por um grafo direcionado acíclico, onde toda atividade tem uma duração fixa. A instância da RANDRE possui, portanto, as mesmas características da rede associada ao RCPSP clássico. A segunda fase consiste do emprego de um procedimento heurístico para solução do RCPSP determinista. A execução repetitiva da “seleção do caminho”, combinada com o “escalonamento através de heurística” gera, a cada execução, um valor para o tempo total de execução do projeto. Esses valores são armazenados em histogramas que representam a distribuição do tempo para execução do projeto.

4.2 – Algoritmos para a seleção de um cenário

A primeira classe de algoritmos que estudada aqui tem por objetivo transformar uma rede de atividades não determinista em uma rede de atividades determinista. Através deles os valores da duração de cada atividade são sorteados, assim como a saída de cada uma das decisões. O objetivo principal é transformar uma RANDRE em uma rede similar à utilizada no problema RCPS, através da seleção de uma de suas possíveis instâncias de execução, conforme mostrado na Figura 14.

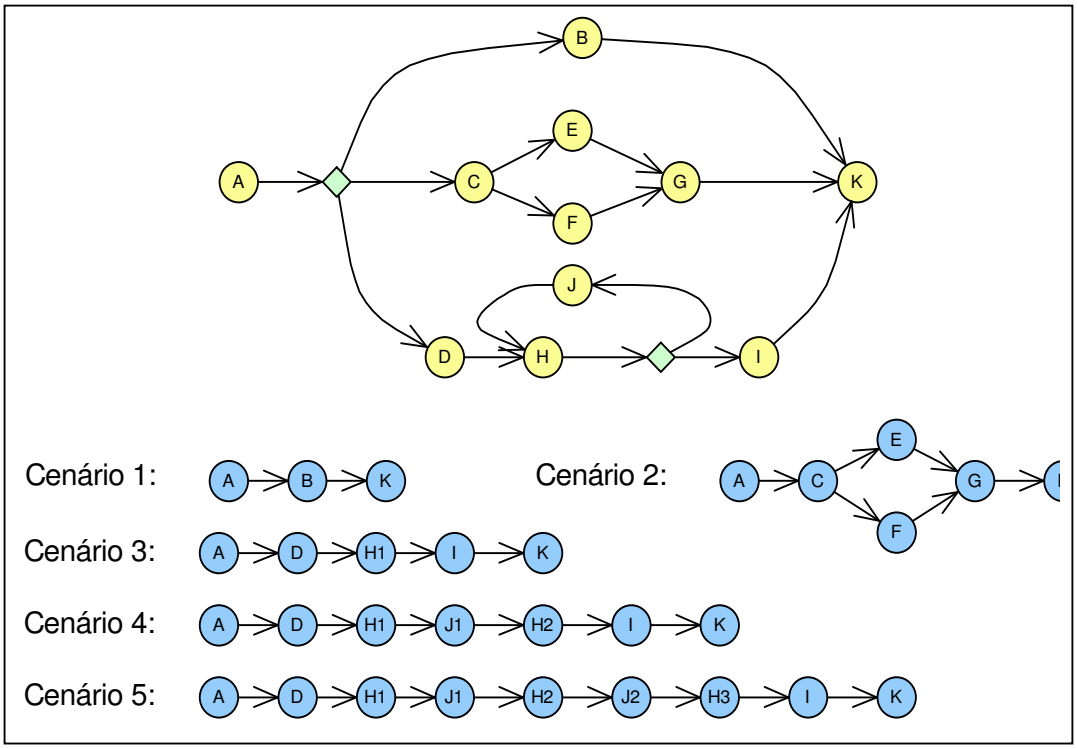

Figura 14: Exemplo de seleção de vários cenários em uma RANDRE

Podemos ver na Figura 14 um exemplo do que se espera dos algoritmos de seleção de um cenário. Na parte de cima se encontra uma RANDRE, com todas as suas decisões e "loops". Na parte de baixo, podemos ver alguns cenários que podem ser obtidos através da rede principal. Cada um destes cenários assume a forma de um grafo sem decisões e sem ciclos, semelhantes, portanto, aos grafos das redes CPM e RCPS.

Logo no primeiro cenário podemos ver uma característica interessante criada pela existência da decisão na RANDRE. Nem sempre uma atividade que se encontra na RANDRE é executada em um cenário. Por exemplo, a atividade E não aparece no Cenário 1, assim como a atividade D não aparece no Cenário 2.

Um outro dado importante é que a existência dos "loops" faz com que uma mesma atividade seja executada inúmeras vezes, e que em cada uma dessas execuções, ela possua um relacionamento de precedência diferenciado. Por exemplo, no Cenário 4 a primeira execução da atividade H depende do término da atividade D, enquanto que a segunda execução de H depende do término da primeira execução de J. Os "loops" criam ainda a possibilidade de existência de infinitos cenários diferentes para uma mesma RANDRE, visto que o número de vezes em que um loop é percorrido tem influência direta na geração do cenário. Vale lembrar que existem ainda as informações quanto às durações associadas a cada uma das atividades. Na RANDRE elas assumem a forma de variáveis aleatórias, enquanto nos cenários são valores sorteados a partir daquelas variáveis. Portanto, mesmo que dois cenários possuam o mesmo grafo de atividades, eles podem diferir quanto à duração das atividades.

Obter um cenário de execução a partir de uma RANDRE pode parecer uma tarefa simples. Entretanto, as aparências enganam e esse foi um dos problemas mais difíceis que tivemos durante a construção do presente trabalho. Para exemplificar a complexidade desta tarefa, vamos voltar à Figura 14 e observar com mais atenção as atividades K, G e H. Uma pergunta comum que se faz no escalonamento de atividades é: Quais tarefas devem terminar antes que a tarefa X possa executar? A resposta para esta pergunta é muito diferente para cada uma das atividades enumeradas acima. K só pode iniciar depois do término da atividade B, G ou I dependendo de qual o ramo selecionado após a atividade A. G só pode começar após o término de E e F. A primeira execução de H só pode começar após o término de D, e a i-ésima execução de H só começa após o fim da (i - 1)ésima execução de J (esta regra não funciona em loops aninhados, é apenas uma simplificação). As diferenças nas interpretações ocorrem por causa da topologia da rede, afinal todas as atividades que examinamos são basicamente iguais, ou seja, uma atividade com duas, ou três, entradas e uma, ou nenhuma, saída. Para gerar o cenário, portanto, é necessário estar ciente de que a topologia da rede deve ser levada em conta.

A primeira tentativa de um algoritmo de seleção de um cenário feita neste trabalho era baseada na visita dos nós do grafo em uma certa ordem, de modo que quando um nó fosse visitado, todos os anteriores a ele no grafo já tivessem tido a sua chance de executar. Por exemplo, no momento que a atividade K fosse ser colocada no grafo de resposta, as atividades B, G e I já deveriam ter tido a chance de executar, e K dependeria do término de todas as que tivessem sido executadas. Esta política funciona bem para a decisão, mas para os “loops” ela possui algumas limitações sérias. Quando a rede possui um “loop” essa ordem de visitação se torna difícil de definir, sendo necessário definir algumas regras para descarte de dependências. Por exemplo, a aresta que liga J a H deve ser ignorada na primeira iteração, enquanto que a que liga D a H deve ser descartada nas demais iterações. Este algoritmo foi implementado, mas acabou se tornando um pouco complicado. Ele funcionou com todos os casos de teste que utilizamos, e possuía um tempo de execução razoável. No entanto, devido à sua complexidade, a prova da sua validade se tornou extremamente complicada. O resultado era um algoritmo que parecia funcionar, mas que não pudemos provar sua validade. A decisão tomada foi tentar uma abordagem diferente.

O algoritmo para seleção de um cenário se baseia na decomposição da rede de atividades em uma hierarquia de RANDREs básicas expressa na forma de uma árvore, e na utilização desta árvore para montagem do cenário de execução. As folhas dessa árvore são as atividades, e os nós são representados pelas RANDREs elementares (Série, Paralelo, Decisão e Loop). Cada nó da árvore pode estar ligado a n RANDREs, sendo que as posições em que as RANDREs se encontram influem na semântica conferida a elas. Por exemplo, na RANDRE elementar Loop, a primeira RANDRE é aquela existente antes do loop, a segunda é aquela que o inicia, a terceira é o “passo de incrementação” do loop, e a última é o finalizador. Para selecionar um cenário dentro desta hierarquia, basta percorrê-la em profundidade, aplicando o escalonamento de forma recursiva em cada RANDRE básica. Para montar um cenário

utilizando este método são necessários dois algoritmos. O primeiro monta a hierarquia de RANDREs a partir da rede original, e o segundo obtém um cenário determinista desta hierarquia.

4.2.1 - Algoritmo para geração da hierarquia de RANDREs

A hierarquia de RANDREs assume a forma de uma árvore com uma única raiz, e inúmeras folhas. Cada folha é uma atividade, e os nós que compõem a árvore podem ser de cinco tipos distintos: Trivial, Serial, Paralelo, Decisão e Loop. Cada tipo de nó pode possuir um determinado conjunto de nós filhos. Na notação gráfica que utilizaremos aqui, a posição do nó filho indica qual das sub-redes ele representa na sua RANDRE correspondente, por exemplo, o primeiro nó ligado a uma sub-árvore do tipo decisão representa a AN_1 na RANDRE básica Decisão. Na Figura 15 podemos ver os cinco tipos diferentes de nós que podem compor a árvore:

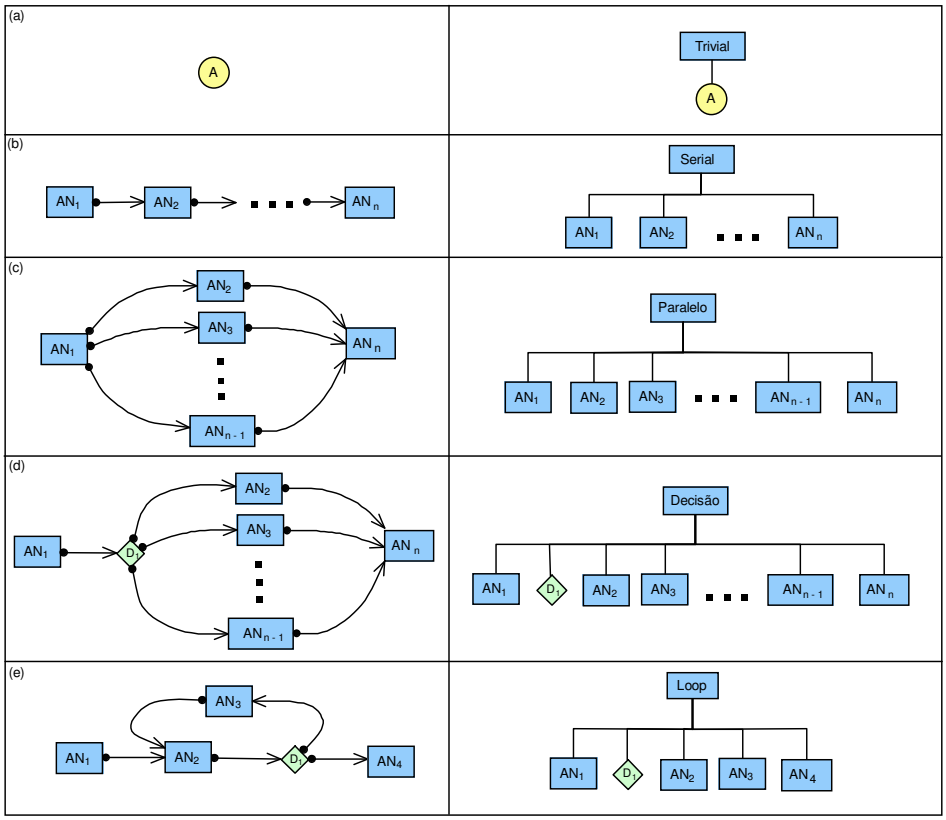

Figura 15: RANDREs básicas x Nós básicos da árvore

Cada um dos 5 tipos de RANDRE básicas é mapeado em uma estrutura específica em forma de árvore. O objetivo do algoritmo de construção da hierarquia é construir uma árvore

composta apenas de nós do tipo acima, à partir de uma RANDRE descrita através de um grafo que contém apenas atividades, decisões e ligações entre eles.

O algoritmo que mostraremos aqui é baseado na tentativa exaustiva de casar nós contidos no grafo com os padrões das RANDREs básicas. A cada iteração o algoritmo tenta encaixar cada nó do grafo como primeira RANDRE em uma das RANDREs básicas. Se o restante dos nós ligados a ele no grafo, segue o padrão de alguma das RANDREs básicas, os nós são agrupados em um único item da árvore. Este item tem o tipo da RANDRE básica com o qual foi possível casar o padrão, e possui como nós filhos as RANDREs e decisões que correspondem a cada posição do padrão. O algoritmo continua tentando casar os padrões até que nenhuma modificação seja realizada na rede. Neste momento, se restou apenas uma RANDRE, o algoritmo conseguiu montar a hierarquia com sucesso. Senão, é sinal que o algoritmo não conseguiu montar a hierarquia de RANDREs, sendo altamente provável que a rede não esteja formada corretamente. No quadro abaixo se encontra o algoritmo descrito em linguagem estruturada:

```
MontarHierarquiaRandre(G)
Transformar todas as atividades de G em nós do tipo Trivial;
faça
    para cada RANDRE R1 contida em G

        //Caso da RANDRE Serial:
        se R1 está ligada a uma única RANDRE R2 &
           R2 possui como única RANDRE antecessora R1
        então:
            Criar nova RANDRE do tipo Serial(R1, R2) e substituí-las
            no grafo;
            Retirar {R1, R2} de G;

        //Caso da RANDRE Paralelo:
        se R1 está ligada a n Randres P = {P1, P2, ... PN} &
           Cada RANDRE em P possui como antecessor apenas R1 &
           Cada RANDRE em P possui como sucessor apenas R2 &
           R2 possui como antecessores apenas as RANDREs contidas em P
        então:
            Criar nova RANDRE do tipo Paralelo(R1, P1, P2 .. PN, R2)
            e substituí-las no grafo;
            Retirar {R1, P1, P2 .. PN, R2} de G;

        //Caso da RANDRE Decisão:
        se R1 está ligada apenas a uma decisão D1 &
           D1 está ligada a n Randres P = {P1, P2, ... PN} &
           Cada RANDRE em P possui como antecessor apenas D1 &
           Cada RANDRE em P possui como sucessor apenas R2 &
           R2 possui como antecessores apenas as RANDREs contidas em P
        então:
            Criar nova RANDRE do tipo Decisão(R1, D1, P1, P2 .. PN, R2)
            e substituí-las no grafo;
            Retirar {R1, D1, P1, P2 .. PN, R2} de G;

        //Caso da RANDRE Loop:
        se R1 está ligada apenas a uma RANDRE R2 &
           R2 possui como antecessores apenas 2 RANDREs R1 e R3 &
           R2 está ligada apenas a uma decisão D1 &
           D1 está ligada apenas a duas Randres R3 e R4
           R3 e R4 possuem como antecessor apenas D1
           R3 está ligado apenas a R2
        então:
            Criar nova RANDRE do tipo Loop(R1, D1, R2, R3, R4)
            e substituí-las no grafo;
            Retirar {R1, D1, R2, R3, R4} de G;
    fim para

enquanto ocorrerem modificações na rede

se sobrou apenas uma RANDRE no grafo retorna SUCESSO
senão retorna FALHA.

Fim MontarHierarquiaRandre
```

Como podemos ver, o algoritmo é bem simples. Ele se baseia na tentativa exaustiva de casar os nós do grafo com o padrão contido nas RANDREs básicas, impondo que todas as regras contidas na definição da RANDRE básica sejam seguidas. Este algoritmo não fica em loop infinito, porque a cada iteração apenas duas coisas podem ocorrer: ou não existem

alterações no grafo ou houve alguma redução. O primeiro caso é a condição de parada do loop, portanto, se ocorrer o algoritmo pára. No segundo caso, pelo menos 2 RANDREs foram unificadas em 1, reduzindo assim, o número de RANDREs no grafo. Se houverem sempre reduções, o número de RANDREs no grafo diminuirá cada vez mais, tendo como limite um grafo contendo uma única RANDRE. Como não existem transformações que tem como fonte uma única RANDRE, nenhuma modificação é realizada, e o algoritmo pára.

O algoritmo possui uma complexidade relativamente alta. No pior caso, apenas uma transformação de duas RANDREs em uma é realizada a cada iteração. Neste caso, seja n o número de RANDREs no grafo inicial, o loop principal será percorrido n vezes. Em cada iteração, o número de RANDREs percorrido será reduzido em 1, resultando num total de iterações igual a (n + (n - 1) + (n - 2) + ... + 1) que é equivalente à soma de uma PA de primeiro termo n, último 1, possuindo n termos. Essa soma pode ser obtida através da fórmula $S = ((A_1 + A_n) * n) / 2$, que equivale no nosso caso a $S = (n + 1) * n / 2$. Este resultado nos diz que a complexidade deste algoritmo é $O(n^2)$. Esta ordem de complexidade, apesar de relativamente alta, não causa grande impacto no tempo total da simulação. Isso ocorre porque ele precisa ser executado apenas uma vez a cada simulação, sendo mínimo o impacto em uma simulação de 1000 escalonamentos, por exemplo.

No gráfico abaixo podemos ver o resultado obtido, passo a passo, através da execução deste algoritmo utilizando a rede de atividades mostrada na Figura 16:

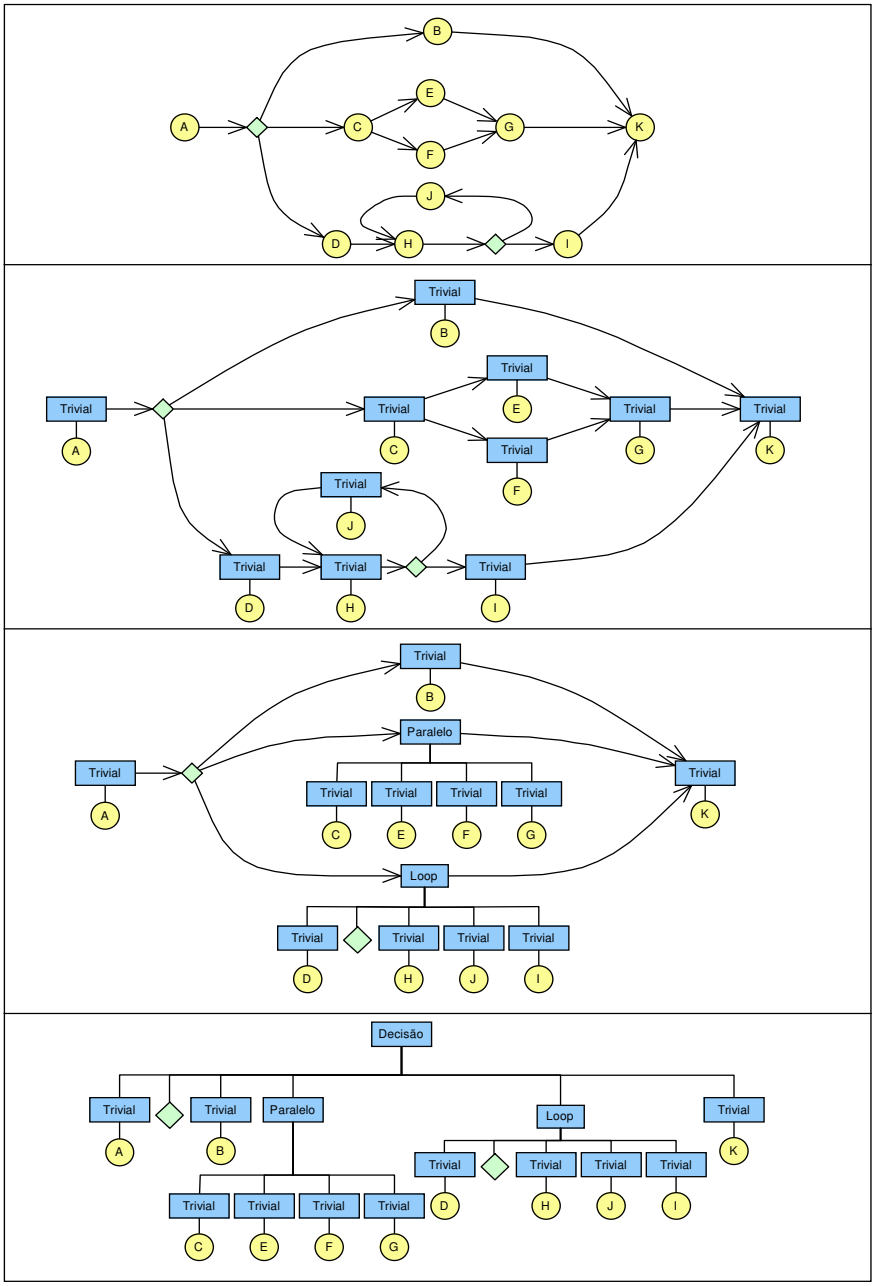

Figura 16: Execução passo a passo do algoritmo de hierarquização

4.2.2 - Algoritmo de seleção de cenário a partir da hierarquia

O algoritmo para a geração de um novo cenário de execução se baseia na hierarquia de RANDREs obtida à partir do algoritmo anterior para gerar múltiplos cenários de execução. Cada cenário é criado através da varredura em profundidade da hierarquia da árvore, gerando para cada nó um sub-grafo que irá compor o grafo de dependência de atividades do cenário. A entrada e a saída deste algoritmo podem ser definidas graficamente da seguinte maneira:

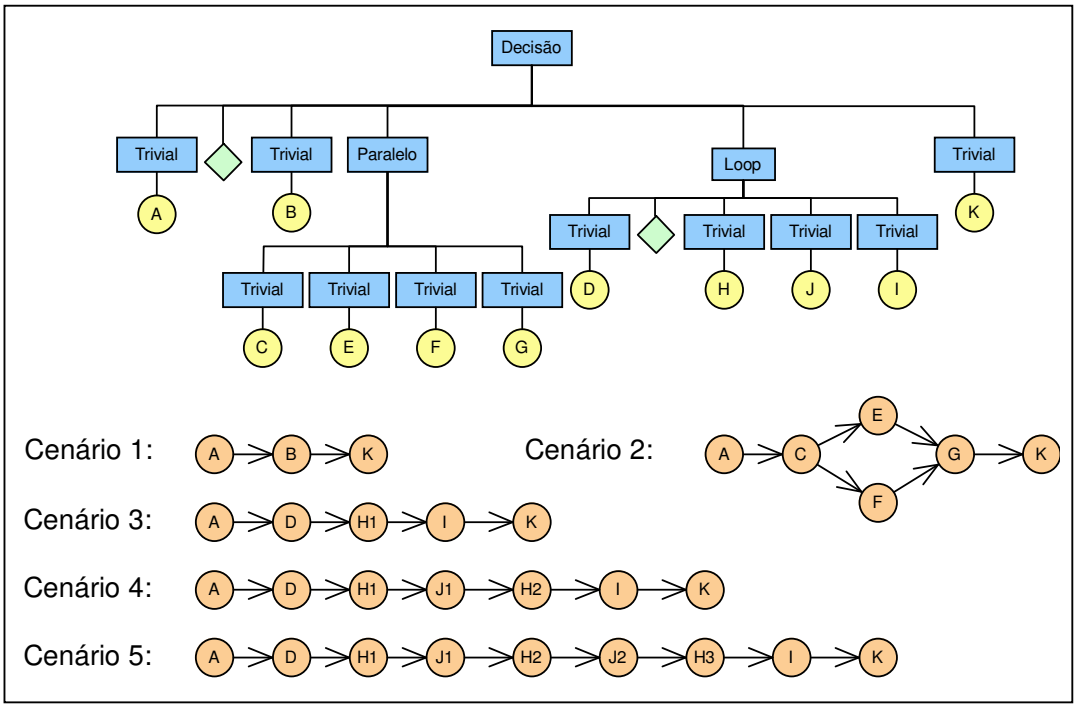

Figura 17: Saída esperada do algoritmo de seleção de um cenário

O algoritmo para seleção de um cenário é definido de forma recursiva, e opera sobre cada nó da árvore obtida a partir da rede de atividades. A cada visita a um nó um novo sub-grafo de tarefas é retornado, assim como a tarefa inicial e final deste sub-grafo. Damos o nome de tarefa aqui a uma instância de execução de uma atividade, por exemplo, a primeira execução da atividade A é a tarefa A1, a segunda, A2, e assim por diante. As tarefas são os elementos que irão compor o grafo resultante, assim como os relacionamentos de precedência entre elas. Cada tarefa tem uma duração fixa, diferentemente das atividades às quais estão associadas. O algoritmo opera encadeando os sub-grafos em suas tarefas iniciais e finais de acordo com a regra de formação definida para aquele nó da árvore. O algoritmo pode ser definido da seguinte forma:

GerarCenarioExecucao(NoArvore)

Caso o tipo de No Arvore seja:

//Caso da RANDRE Trivial:

Trivial:

Cria uma nova tarefa T_{ji} para a i-ésima execução da atividade A_j ligada a NoArvore;

Sorteia um possível valor t_j segundo a sua i-ésima distribuição de duração;

Cria um grafo G contendo apenas T_{ji};

Retorna G, T_{ji}, T_{ji};

//Caso da RANDRE Serial:

Serial:

G_j, Ini_j, Fin_j := GerarCenarioExecucao(AN_j) para todo j = 1, 2, ... n;

Cria um grafo G contendo todas as tarefas em cada G_j;

Criar em G a dependência entre Ini_{j+1} e Fin_j para j = 1, 2, ... n – 1;

retorna G, Ini_1, Fin_n;

//Caso da RANDRE Paralelo:

Paralelo:

G_j, Ini_j, Fin_j := GerarCenarioExecucao(AN_j) para todo j = 1, 2, ... n;

Cria um grafo G contendo todas as tarefas em cada G_j;

Criar em G a dependência entre Ini_j e Fin_1 para j = 2, ... n – 1;

Criar em G a dependência entre Ini_n e Fin_j para j = 2, ... n – 1;

retorna G, Ini_1, Fin_n;

//Caso da RANDRE Decisão:

Decisão:

G_1, Ini_1, Fin_1 := GerarCenarioExecucao(AN_1);

G_2, Ini_2, Fin_2 := GerarCenarioExecucao(AN_i) para um valor i sorteado entre 2 e n – 1, de acordo com as probabilidades da decisão D_1;

G_3, Ini_3, Fin_3 := GerarCenarioExecucao(AN_n);

Cria um grafo G contendo todas as tarefas em G_1, G_2 e G_3;

Criar em G a dependência entre Ini_2 e Fin_1;

Criar em G a dependência entre Ini_3 e Fin_2;

retorna G, Ini_1, Fin_3;

//Caso da RANDRE Loop:

Decisão:

Cria um grafo G contendo todas as tarefas em G_1, G_2 e G_3;

G_1, Ini_1, Fin_1 := GerarCenarioExecucao(AN_1);

Adicionar G_1 a G;

G_2, Ini_2, Fin_2 := GerarCenarioExecucao(AN_2);

```
        Adicionar G2 a G;
        Criar em G a dependência entre Ini2 e Fin1;


        Faça enquanto o valor i sorteado entre 3 e 4, de acordo com as
             Probabilidades da decisão D1 seja igual de 3:

             G3, Ini3, Fin3 := GerarCenarioExecucao(AN3);
             Adicionar G3 a G;
             Criar em G a dependência entre Ini3 e Fin2;

             G2, Ini2, Fin2 := GerarCenarioExecucao(AN2);
             Adicionar G2 a G;
             Criar em G a dependência entre Ini2 e Fin3;
        Fim Faça

        G4, Ini4, Fin4 := GerarCenarioExecucao(AN4);
        Adicionar G4 a G;
        Criar em G a dependência entre Ini4 e Fin2;

        retorna G, Ini1, Fin4;

Fim GerarCenarioExecucao
```

A definição indutiva da RANDRE implica na existência de apenas um nó inicial, e um nó terminal (que são o mesmo no caso de uma RANDRE Trivial). Por causa disso a combinação das instâncias de execução da RANDRE pode ser feito através da combinação destes nós. O algoritmo segue a definição de cada uma das RANDREs básicas, sendo possível verificar diretamente que eles geram as combinações de tarefas da maneira correta.

4.3 - Algoritmos para escalonamento de um cenário

Conforme vimos anteriormente, os algoritmos para seleção de um cenário produzem como resultado uma rede determinista de atividades, que representa um dos possíveis cenários de execução daquela RANDRE. Os algoritmos para escalonamento de um cenário são utilizados para obter o tempo total necessário para executar este cenário, definindo os tempos de início e fim de cada uma das tarefas contidas nele.

O grafo de entrada para os algoritmos de escalonamento de um cenário são idênticos aos grafos de dependência de atividades dos métodos CPM / RCPSP. Por este motivo, os mesmos algoritmos que resolvem este tipo de problema, podem ser utilizados para executar o escalonamento de um cenário de uma RANDRE. Estes algoritmos se encontram amplamente documentados na literatura existente, como em [20], [32], [33], e foram testados exaustivamente, o que é uma garantia de que eles funcionam bem.

Nesta seção serão mostrados dois algoritmos distintos para escalonamento de atividades. O primeiro é uma solução amplamente conhecida para o problema CPM. Ao utilizar este algoritmo para escalonar um cenário, as suas dependências de recurso são ignoradas, e o

cenário é escalonado como se existissem recursos infinitos. Isso se deve ao fato de que o algoritmo foi criado para resolver o problema CPM, que não possui recursos.

O segundo algoritmo é uma solução para o RCPSP que utiliza heurísticas para obter uma aproximação da solução ótima. Este algoritmo é do tipo de escalonamento baseado em regras de prioridade, conforme discutido na seção 2.3.2. Diferentemente do primeiro algoritmo, este implementa restrições de recursos, visto que o RCPSP possui restrições de recursos. Um cenário escalonado desta maneira estará restrito aos recursos disponíveis para execução descritos na RANDRE original.

4.3.1 - Algoritmo de escalonamento sem recursos

O primeiro algoritmo de escalonamento de tarefas é conhecido na literatura como a solução para o problema de obtenção do tempo mínimo para realização de uma rede CPM. Uma discussão completa sobre este algoritmo pode ser encontrada em [5]. As redes CPM são compostas por atividades com duração fixa, que possuem relacionamentos de precedência com outras atividades. Estes relacionamentos de precedência dizem que determinada atividade A só pode começar após o término da última das atividades antecessoras de A. O algoritmo de escalonamento obtém os tempos de início e fim, de modo a minimizar o tempo total, obedecendo a estas regras de formação da rede.

O algoritmo para escalonamento de uma rede de atividades CPM é composto pela execução de duas fases. Na primeira o grafo de dependências é varrido em profundidade, do início para o fim. Nesta fase o menor tempo de início possível para cada uma das tarefas é marcado. Na segunda fase o grafo é varrido do início para o fim, sendo calculado o maior tempo possível para início de cada atividade. Se o menor tempo de início é igual ao maior tempo de início a atividade se encontra no caminho crítico, pois não há folga para definição do seu tempo de início. No algoritmo apresentado abaixo, tomamos como tempo de início de cada atividade o menor tempo de início possível. Outras políticas, no entanto, podem ser usadas para selecionar o tempo de início entre o mínimo e o máximo possível. No quadro abaixo se encontra o algoritmo para escalonamento em linguagem estruturada:

```
EscalonarCenarioSemRecursos(Cenário)

    //Inicialização das variáveis
    minInicio(T) := −∞, maxInicio(T) := ∞ para toda Tarefa T ∈ Cenário;
    Tini := Tarefa sem predecessor contida em Cenário;
    Tfim := Tarefa sem sucessor contida em Cenário;

    //Execução da primeira passagem
    ExecutarIda(Tini, 0);

    maiorTempo := Maior valor de minInicio(T) + duracao(T)
                  para toda Tarefa T ∈ Cenário;

    //Execução da segunda passagem
    ExecutarVolta(Tfim, maiorTempo);

    //Calculo dos resultados
    inicio(T) := minInicio(T)
                 para cada T ∈ Cenário;
    camCritico(T) := Verdadeiro se minInicio(T) = maxInicio(T)
                 para cada T ∈ Cenário;
    tempoTotal := Maior valor de inicio(T) + duracao(T)
                  para toda Tarefa T ∈ Cenário;

Fim EscalonarCenarioSemRecursos

ExecutarIda(Tarefas, tempoIni)
     ∀ Tarefa T ∈ Tarefas ⇒
          (1) Se tempoIni > inicioIda(T) ⇒
               (1.1) minInicio(T) := tempoIni
               (1.2) ExecutarIda(Sucessores(T), minInicio(T) +
                                 duracao(T));
Fim ExecutarIda

ExecutarVolta(Tarefas, tempoFim)
     ∀ Tarefa T ∈ Tarefas ⇒
          (1) Se tempoFim < maxInicio(T) + duracao(T) ⇒
               (1.1) maxInicio(T) := tempoFim − duração(T);
               (1.2) executarVolta(Antecessores(T), maxInicio(T));
Fim ExecutarVolta
```

Para ilustrar a execução deste algoritmo vamos mostrar uma execução passo-a-passo do escalonamento de um cenário de execução. O cenário escolhido é o Cenário 2 da Figura 18. Nas figuras abaixo as variáveis utilizadas no algoritmo estão próximas às atividades às quais estão associadas. As variáveis alteradas a cada passo estão sublinhadas e em negrito:

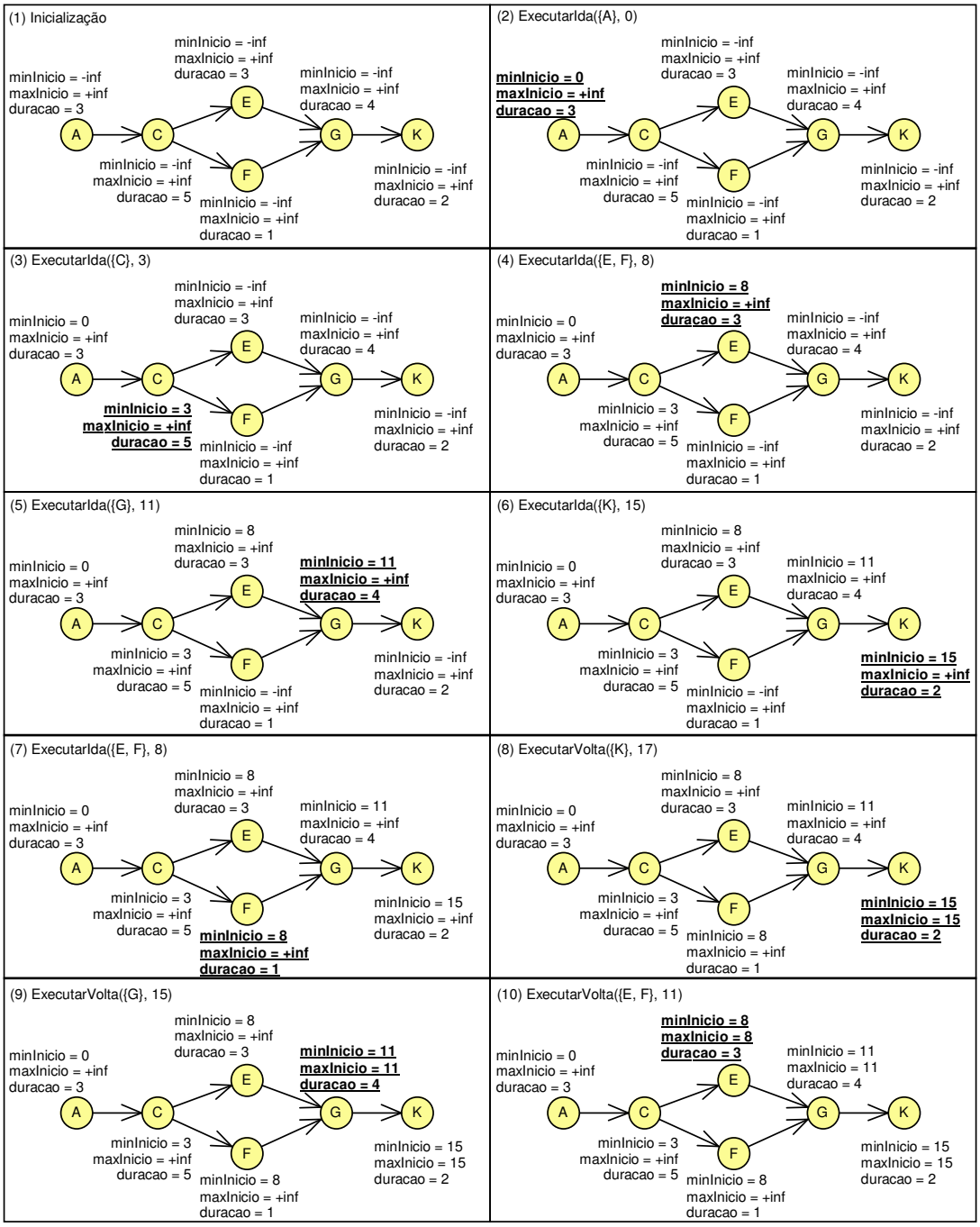

Figura 18: Exemplo de execução do algoritmo de escalonamento s/ recursos (I)

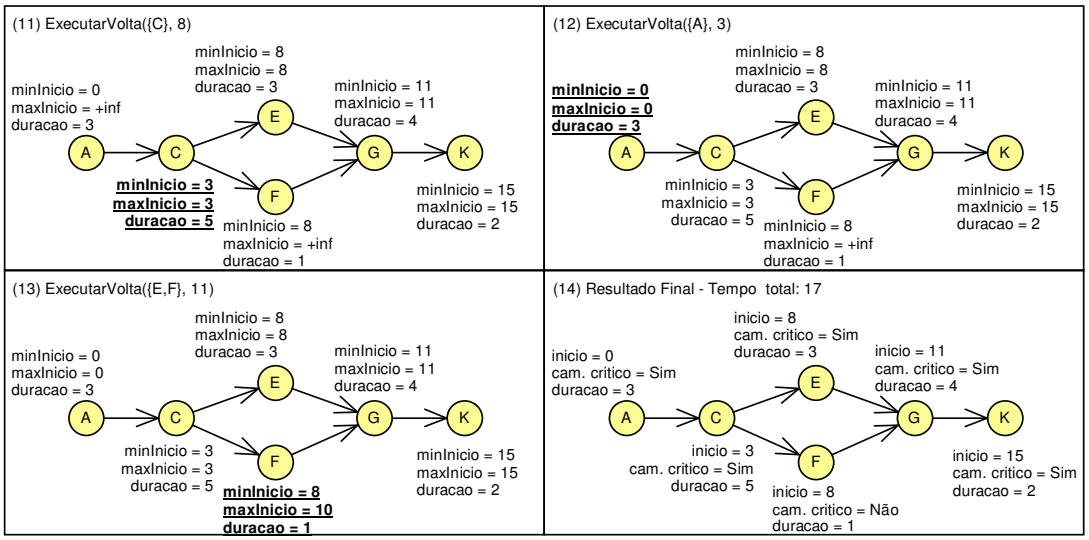

Figura 19: Exemplo de execução do algoritmo de escalonamento s/ recursos (II)

4.3.2 - Algoritmo de escalonamento com recursos

O segundo algoritmo para escalonamento de tarefas possui as mesmas entradas e saídas que o algoritmo anterior, mas pode produzir cronogramas com tempos maiores de execução. Isso acontece porque ele respeita as restrições de recursos contidas nas redes RANDREs, fazendo com que algumas atividades que poderiam executar em paralelo sejam impedidas de executar concorrentemente.

Conforme vimos, os recursos nas RANDRE são implementados utilizando uma lista de recursos necessários para execução, associados a cada atividade, e uma lista de recursos global, que representa a quantidade total de recursos disponível para execução. Quando as atividades são colocadas em execução, a lista de disponibilidade de recursos é decrementada da quantidade de recursos necessários para executar aquela tarefa. Se a quantidade de recursos necessários para execução é inferior à quantidade total disponível, a atividade não pode ser colocada em execução naquele momento. Quando uma atividade termina de executar ela libera novamente os recursos, incrementando a lista de recursos disponíveis. Por este motivo os recursos implementados na RANDRE podem ser classificados como renováveis.

Na sua constituição, a rede de atividades correspondente a um cenário da RANDRE é idêntica à rede de atividades da classe de problemas conhecida como RCPSP. Por isso, para escalonar um cenário levando em consideração as suas dependências de recursos, iremos utilizar um dos algoritmos mais conhecidos para escalonamento de rede RCPS. Uma definição completa deste algoritmo pode ser encontrada em [35]. Como o problema de encontrar a solução ótima é NP-Completo, este algoritmo tenta encontrar boas aproximações para este cronograma ótimo através da utilização de heurísticas. A palavra heurística tem sido utilizada no contexto da ciência da computação para definir qualquer técnica computacional de busca que tenha melhor performance na solução do caso médio de um problema, mesmo que não

tenha melhor performance no pior caso [36]. No caso específico do RCPS, podemos tomar duas medidas de performance: o tempo computacional para se chegar a uma solução, e a qualidade da solução.

As heurísticas aplicadas ao RCPS servem para selecionar qual atividade será colocada em execução quando existe mais de uma pronta para ser executada. A ordem em que essas atividades são escalonadas tem influência direta sobre o tempo total do cronograma, podendo produzir resultados mais próximos ou mais distantes do valor ótimo. Abaixo se encontram alguns exemplos de heurísticas também abordadas em [7]:

Sigla	Nome	Descrição
LPT	Longest Processing Time	Escalona primeiramente as atividades mais longas.
SPT	Shortest Processing Time	Escalona primeiramente as atividades com menor duração.
GRD	Greatest Resource Demand	Escalona primeiramente as tarefas que demandam maior quantidade de recursos.
LRD	Least Resource Demand	Escalona primeiramente as tarefas que demandam menor quantidade de recursos.
GRU	Greatest Resource Utilization	Escalona primeiro as tarefas que demandam mais recursos por mais tempo
LRU	Least Resource Utilization	Escalona primeiro as tarefas que irão demandar menos recursos, por menos tempo.
MIS	Most Immediate Successors	Escalona primeiro as atividades com maior número de sucessores imediatos.
LIS	Least Immediate Successors	Escalona primeiro as atividades com menor número de sucessores imediatos.

Tabela 1: Heurísticas de Escalonamento

O algoritmo de escalonamento utiliza quatro listas para armazenar as tarefas que são escalonadas. A lista TarefasEmExecucao contém as tarefas que estão sendo executadas em um determinado momento, TarefasExecutadas contém todas as tarefas que terminaram de executar, e TarefasExecutaveis contém todas as tarefas que já podem ser executadas de acordo com as restrições de dependências entre atividades. A lista AVL é construída a cada iteração, e contém as atividades contidas em TarefasExecutaveis que podem ser executadas de acordo com os recursos livres. Abaixo se encontra o código deste algoritmo em uma linguagem estruturada:

```
EscalonarCenarioComRecursos(Cenário, Heuristica)

    //Inicialização das variáveis
    TarefasEmExecucao := ∅;
    TarefasExecutadas := ∅;
    TarefasExecutaveis := Tarefa sem predecessor contida em Cenário;
    RecursosLivres := Recursos disponíveis para executar a RANDRE;
    tempoAtual := 0;

    //Execução do loop principal
    Faça enquanto 0 = 0

        //Retira da lista TarefasEmExecucao todas as tarefas T com
        //inicio(T) + duração(T) <= que tempoAtual. Devolve à
        //lista RecursosLivres os recursos usados por estas tarefas,
        //e adiciona à TarefasExecutadas todas as tarefas terminadas.
        //Adiciona em TarefasExecutaveis todas as tarefas que dependem
        //apenas de tarefas contidas em TarefasExecutadas
        RetirarTarefasTerminadas(tempoAtual, TarefasEmExecucao,
                                 TarefasExecutadas, TarefasExecutaveis,
                                 RecursosLivres);


        Faça enquanto 0 = 0

            //Retorna a lista de tarefas que podem ser executadas de
            //acordo com os recursos livres no momento.
            AVL := ObterTarefasElegiveis(TarefasExecutaveis,
                                         RecursosLivres);

            Se AVL = ∅ então
                 Sair do loop;
            Fim se

            //Seleciona uma das tarefas de acordo com a heurística
            T := SelecionarTarefa(AVL, Heurística);

            inicio(T) := tempoAtual;
            TarefasEmExecucao := TarefasEmExecucao ∪ T;
            TarefasExecutaveis:= TarefasExecutaveis - T;
            RetiraRecursosUtilizados(RecursosLivres, T);

        Fim faça

        Se #TarefasEmExecucao = ∅ então
             Sair do loop;
        Fim se

        tempoAtual := MenorTempoFinal(TarefasEmExecucao);
    Fim faça

Fim EscalonarCenarioComRecursos
```

Da mesma maneira que fizemos no algoritmo de escalonamento sem recursos, vamos mostrar um exemplo de execução passo-a-passo deste algoritmo. O cenário de exemplo é o

mesmo utilizado anteriormente, mas aqui, as tarefas precisam de recursos para executar. Vamos supor a existência de um determinado tipo de recurso chamado R1. Existe apenas um recurso do tipo R1 para executar a rede, e cada tarefa precisa de um recurso R1 livre para que possa ser executada. A heurística usada na execução é a SPT (a tarefa com menor duração é executada primeiro). As principais variáveis do algoritmo são mostradas em azul no canto inferior direito de cada passo:

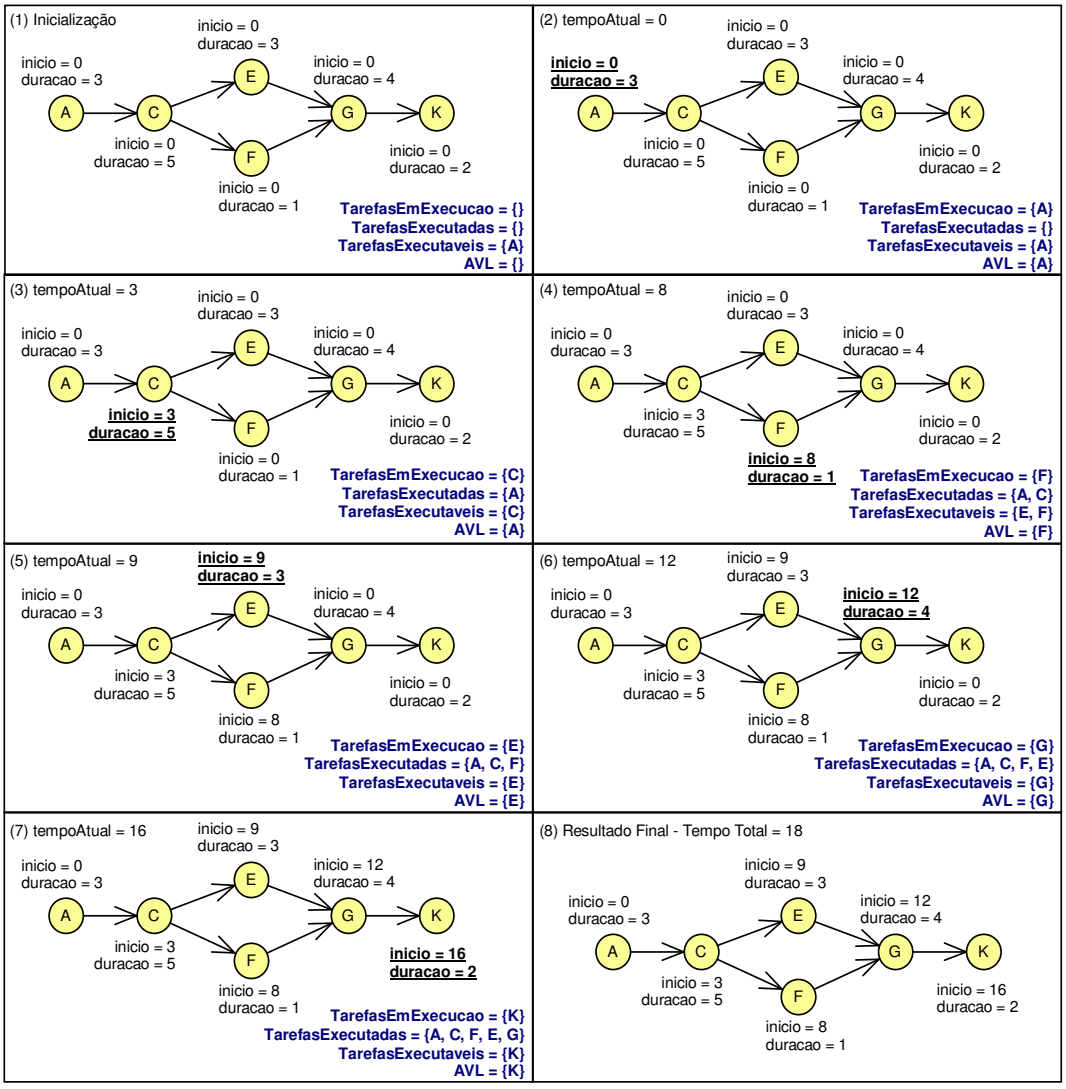

Figura 20: Exemplo de execução do algoritmo de escalonamento c/ recursos

Analisando os resultados obtidos utilizando os dois algoritmos, podemos ver que há diferenças, tanto no tempo total, quanto na ordem de execução das atividades. Devido à escassez de recursos, o segundo algoritmo gera um tempo de execução maior. Isso acontece devido à impossibilidade de executar as atividades E e F concorrentemente. Podemos observar estes fatos no quadro comparativo abaixo:

Atividade	Início Escalonamento sem recursos	Início Escalonamento com recursos
A	0	0
C	3	3
E	8	9
F	8	8
G	11	12
K	15	16
Tempo Total	17	18

Tabela 2: Comparação entre os algoritmos de escalonamento

4.4 - Algoritmos para simulação de múltiplos cenários

O mecanismo de simulação da RANDRE se baseia na utilização do método de Monte Carlo para obter uma aproximação da distribuição do tempo total do projeto através de um processo de amostragem. Através deste método, são gerados vários cenários à partir de uma RANDRE. Cada cenário é escalonado de modo a obter o tempo total em cada instância. O resultado obtido em cada instância pode ser acumulado em um histograma, permitindo obter as freqüências de ocorrência de cada um dos resultados.

Nesta seção o foco estará na montagem do histograma do tempo total. Por questão de simplicidade, falaremos apenas do histograma de duração total, apesar de outros histogramas também poderem ser obtidos (como tamanho do caminho crítico, tempo de início e fim de cada atividade, probabilidade de uma atividade estar no caminho crítico e etc.). Estes histogramas nos permitem responder questões como: Qual a probabilidade do projeto durar mais que x dias? Qual o prazo deve ser dado ao cliente de modo a ter x% de certeza de que o prazo será cumprido? Quanto tempo é necessário para que os casos de uso (por exemplo) estejam prontos, com x% de certeza?

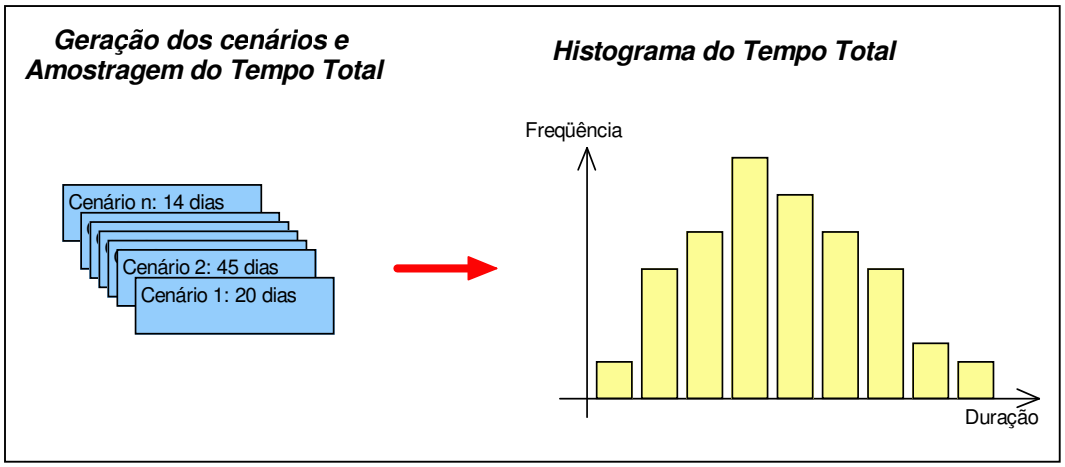

Figura 21: Processo de amostragem das variáveis

No processo de amostragem os valores da variável de controle (no caso o tempo total para realização do projeto) são obtidos para diferentes cenários. Esses cenários são gerados utilizando o algoritmo para seleção de um cenário, e escalonado utilizando algum dos algoritmos de escalonamento. O escalonamento produz o valor da variável que é armazenado no histograma.

Nesta seção apresentaremos dois algoritmos distintos para simulação de atividades. Todos os dois funcionam da mesma maneira, gerando cenários, escalonando-os e armazenando os valores das variáveis de controle em histogramas. A diferença principal entre eles é o modo como os escalonamentos são realizados. No primeiro algoritmo, apenas um cenário, ou instância, é executado por vez. Cada um é executado separadamente, não havendo conflitos por recursos entre diferentes instâncias.

No segundo algoritmo os cenários são disparados de tempos em tempos, podendo haver dois ou mais cenários executando concorrentemente. Os tempos de início de cada atividade são calculados à partir do momento em que um cenário começa a executar. Isto faz com que o cronograma de cada cenário tenha a mesma forma que a obtida no outro algoritmo. A principal diferença é que neste caso os recursos disponíveis são compartilhados entre os diferentes cenários, podendo haver escassez de recursos, o que pode causar atrasos.

O algoritmo para simulação de instância única possui maior utilidade na a simulação de projetos, enquanto que o de simulação de instâncias concorrentes é mais aplicável a processos. Um projeto pode ser definido como um esforço temporário feito com o objetivo de criar um produto ou serviço único [37]. O projeto é por natureza único, ou seja, é executado uma única vez. Um projeto é simulado para a previsão do tempo necessário para executá-lo. Neste caso não há a necessidade de disparo de múltiplas instâncias simultâneas. Já um processo pode ser definido como um conjunto de atividades que podem ser executadas de maneira repetitiva para produzir um determinado produto final. É intrínseco à natureza do processo a execução de múltiplas instâncias concorrentes de modo a se agilizar o processo como um todo. Por exemplo, não faz sentido em uma grande montadora de automóveis montar um carro por vez, mas também é impossível montar infinitos carros ao mesmo tempo. O algoritmo para simulação de múltiplas instâncias concorrentes permite analisar o comportamento de um processo de acordo com uma determinada demanda, permitindo identificar pontos onde os recursos se tornam insuficientes, ou escassos.

4.4.1 - Algoritmo para simulação de instância única

O primeiro dos algoritmos de simulação é chamado aqui de algoritmo para simulação de instância única. Este nome deve-se ao fato de que cada instância (ou cenário) é executada em separado, ou seja, em um dado momento da simulação existe sempre uma única instância executando.

O algoritmo consiste na execução de um número k de iterações dos passos de geração de um cenário, escalonamento deste cenário, e anotação dos resultados obtidos em histogramas. O algoritmo para simulação de instância única pode trabalhar tanto com um

algoritmo de escalonamento de com recursos quanto com um que ignora as restrições de recursos. O algoritmo selecionado para a simulação deve, portanto, ser fornecido como parâmetro, assim como o número de iterações a ser executada, e a heurística de escalonamento (no caso da utilização do algoritmo de escalonamento com recursos). O algoritmo pode ser descrito em uma linguagem estruturada da seguinte forma:

```
SimulaçãoInstanciaUnica(Randre, NumIteracoes, AlgEscalonamento,
                        Heuristica)

    //Inicialização das variáveis
    HierarquiaRandre := MontarHierarquiaRandre(Randre);

    //Execução do loop principal
    Faça de k = 1 até NumIteracoes

         Cenário := GerarCenarioRandre(HierarquiaRandre);
         Se AlgEscalonamento = Sem recursos então
              Resultado := EscalonarCenarioSemRecursos(Cenário);
         Senão
              Resultado := EscalonarCenarioComRecursos(Cenário,
                                                       Heurística);

         Adiciona o Resultado obtido ao Histograma;

    Fim Faça

Fim SimulaçãoInstanciaUnica
```

4.4.2 - Algoritmo para simulação de instâncias concorrentes

O último dos algoritmos que apresentaremos aqui serve para simulação de múltiplos cenários em execução concorrente. Neste algoritmo mais de um cenário pode estar executando ao mesmo tempo, criando uma competição por recursos entre diferentes cenários. Novos cenários são disparados de tempos em tempos, e são executados até que sejam terminados.

O tempo entre o disparo de duas instâncias é representado através de uma distribuição estatística. No mundo real é muito difícil saber com exatidão qual o intervalo de tempo entre duas requisições a um processo, por exemplo, em uma fábrica de aviões como podemos saber quanto tempo levará para que uma companhia aérea encomende um novo avião? Devido a esta incerteza, o tempo entre duas requisições é representado por uma distribuição estatística. Esta distribuição pode assumir qualquer forma, desde que não permita a seleção de números negativos. O tempo para a chegada da próxima requisição é sorteado no algoritmo obedecendo à distribuição de probabilidades descrita por esta variável aleatória, que é um dado de entrada da simulação.

O algoritmo para simulação de instâncias concorrentes consiste de algumas pequenas modificações no algoritmo de escalonamento com recursos. Este algoritmo é modificado para

permitir a existência de tarefas de diferentes cenários executando ao mesmo tempo. Quando a última tarefa de um cenário termina, os dados de escalonamento daquele cenário são coletados em histogramas da mesma forma como realizado no algoritmo de simulação de instância única. Os cenários são criados e colocados em execução de acordo com a chegada do seu tempo de início sorteado no começo do algoritmo. Esses tempos de início marcam os momentos em que novos cenários são disparados, e são sorteados de acordo com a distribuição do tempo entre disparos, recebida como parâmetro. Os tempos de início reais de cada tarefa são calculados à partir do tempo de disparo do cenário ao qual aquela tarefa pertence, para efeito de armazenamento nos histogramas de tempo de início. O algoritmo pára quando não existem mais tarefas a executar, nem novos cenários a disparar. No quadro abaixo podemos ver o algoritmo completo descrito em uma linguagem estruturada:

```
SimulaçãoInstanciasConcorrentes(Randre, NumIteracoes, Heuristica,
                                DistTempoDisparo)

    //Inicialização das variáveis
    HierarquiaRandre := MontarHierarquiaRandre(Randre);
    TarefasEmExecucao := ∅;
    TarefasExecutadas := ∅;
    TarefasExecutaveis := Tarefa sem predecessor contida em Cenário;
    RecursosLivres := Recursos disponíveis para executar a RANDRE;
    tempoAtual := 0;
    IniCenario1 := 0;
    IniCenarioi := IniCenarioi-1 + sortearValor(DistTempoDisparo)
                   Para i = 2 até NumIteracoes;

    //Execução do loop principal
    Faça enquanto 0 = 0

        Se IniCenarioi = tempoAtual para algum i então
            Cenárioi := GerarCenarioRandre(HierarquiaRandre);
            TarefasExecutaveis := TarefasExecutaveis ∪
                                  Tarefa sem predecessor contida
                                  em Cenárioi;
        Fim Se

        //Retira da lista TarefasEmExecucao todas as tarefas T com
        //inicio(T) + duração(T) <= que tempoAtual. Devolve à
        //lista RecursosLivres os recursos usados por estas tarefas,
        //e adiciona à TarefasExecutadas todas as tarefas terminadas.
        //Adiciona em TarefasExecutaveis todas as tarefas que dependem
        //apenas de tarefas contidas em TarefasExecutadas
        RetirarTarefasTerminadas(tempoAtual, TarefasEmExecucao,
                                 TarefasExecutadas, TarefasExecutaveis,
                                 RecursosLivres);

        Para todo Cenárioi cuja última tarefa terminou agora
            Adiciona o resultado obtido no escalonamento de
            Cenárioi ao Histograma;
        Fim Para
```

```
        Faça enquanto 0 = 0

             //Retorna a lista de tarefas que podem ser executadas de
             //acordo com os recursos livres no momento.
             AVL := ObterTarefasElegiveis(TarefasExecutaveis,
                                          RecursosLivres);

             Se AVL = ∅ então
                  Sair do loop;
             Fim se

             //Seleciona uma das tarefas de acordo com a heurística
             T := SelecionarTarefa(AVL, Heurística);

             inicio(T) := tempoAtual;
             TarefasEmExecucao := TarefasEmExecucao ∪ T;
             TarefasExecutaveis:= TarefasExecutaveis - T;
             RetiraRecursosUtilizados(RecursosLivres, T);

        Fim faça

        Se #TarefasEmExecucao = ∅ e
           IniCenario_i < tempoAtual para todo i então
             Sair do loop;
        Fim se

        tempoAtual := Menor tempo entre:
                      MenorTempoFinal(TarefasEmExecucao) e
                      Menor IniCenario_i maior que tempoAtual;
    Fim faça

Fim SimulaçãoInstanciasConcorrentes
```

Vale lembrar que se a distribuição do tempo entre disparo gera valores muito pequenos em relação ao tempo total para execução do projeto, muitos cenários podem ser disparados ao mesmo tempo. Isto pode ser um problema em termos de performance computacional, porque as listas de tarefas tendem a crescer muito.

Uma outra questão importante diz respeito às heurísticas para seleção da tarefa a executar. O algoritmo para simulação de instâncias concorrentes é uma extensão do algoritmo de escalonamento com recursos. Por isso, as heurísticas utilizadas naquele algoritmo podem ser usadas sem modificação aqui. No entanto, agora existem nas listas de tarefas a executar atividades pertencentes a diferentes cenários. Por isso surge uma questão: Será que o tempo total para realização do cenário poderia ser diminuído se as heurísticas levassem em consideração características do cenário ao qual a tarefa pertence? Por exemplo, se a heurística escalonasse primeiro tarefas pertencentes a cenários que estão mais próximos de terminar teríamos alguma melhora nos tempos totais? Várias questões deste tipo podem ser levantadas dando origem a várias heurísticas diferentes. Esta é uma discussão que pode se alongar, e não fará parte, portanto, do escopo deste trabalho. As heurísticas que utilizaremos aqui serão, por enquanto, as mesmas que as discutidas no algoritmo de escalonamento com recursos.

4.5 - Questões relevantes sobre os algoritmos de simulação de múltiplos cenários

Como podemos ver os algoritmos de simulação de múltiplos cenários são simples implementações do método de Monte Carlo na qual a geração do cenário e seu escalonamento são executados sucessivamente, gerando amostras das variáveis de controle a cada iteração. Quando analisamos implementações do método de Monte Carlo duas questões merecem atenção especial: 1) O gerador de números aleatórios utilizado como base é realmente uniforme? 2) O número de iterações utilizado foi suficiente para que o resultado obtido esteja realmente próximo da distribuição de probabilidade real?

Conforme mostra o experimento realizado pelo capitão O. C. Fox utilizando a "Agulha de Buffon", transcrito em [39], os resultados obtidos através do processo de amostragem podem ser incorretos se o método para geração dos eventos aleatórios for viciado. Naquele experimento o vício era introduzido pela posição utilizada para jogar a agulha, que fazia com que determinados eventos ocorressem com mais freqüência que outros. Nas simulações por computador que fazemos nos dias de hoje, este desvio pode ser introduzido por geradores de números aleatórios imperfeitos, que não são exatamente uniformes. Em aplicações de simulação é comum ocorrerem erros devido à utilização dos geradores de números aleatórios disponíveis nas linguagens de programação. Em grande parte das vezes estes geradores não funcionam com a uniformidade necessária. Por causa disso, na implementação o algoritmo de simulação será utilizado o gerador de números aleatórios descrito em [40] sob o nome de ran2. Segundo aquele texto, este método diminui a correlação entre as amostras de maneira considerável. O texto diz que, dentro dos limites da precisão da representação de ponto flutuante, ran2 gera números aleatórios perfeitos.

Para verificar o número de iterações necessárias para que o resultado final seja satisfatório, o teorema de Kolmogorov-Smirnov pode ser utilizado. Uma discussão completa sobre este teorema pode ser encontrada em [47]. Seja $S_n(x)$ a distribuição cumulativa com n amostras da variável aleatória X, e F(x) a sua função de distribuição cumulativa real.

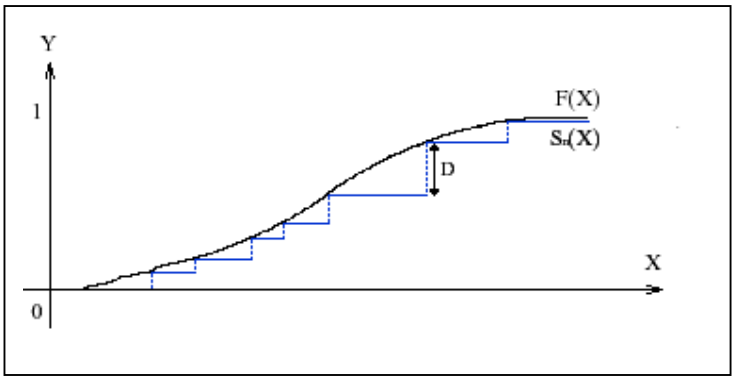

Figura 22: Distribuição cumulativa real (F(X)) e amostral $S_n(X)$

A diferença máxima D entre $S_n(x)$ e F(x) é dada por:

$$D = \max\left(\left|F(x) - S_n(x)\right|\right)$$

Seja $D_{n,\alpha}$ a máxima diferença entre o valor da função cumulativa real e o valor da cumulativa amostral com n amostras com um grau de confiança α. A probabilidade de D ser menor ou igual a $D_{n,\alpha}$ é dada por:

$$P\left(D \le D_{n,\alpha}\right) = 1 - \alpha$$
$$P\left(\max\left(\left|F(x) - S_n(x)\right|\right) \le D_{n,\alpha}\right) = 1 - \alpha$$
$$P\left(S_n(x) - D_{n,\alpha} \le F(x) \le S_n(x) + D_{n,\alpha}\right) = 1 - \alpha \qquad \forall x$$

$D_{n,\alpha}$ está limitado entre os valores 0 e 1, visto que essa é a altura máxima do gráfico da função de distribuição acumulada. Segundo o teorema de Kolmogorov-Smirnov há uma relação entre as variáveis n, α e $D_{n,\alpha}$, e ela pode ser tabelada da seguinte forma para n > 50:

A	0,1	0,05	0,01
$D_{n,\alpha}$	$\frac{1{,}07}{\sqrt{n}}$	$\frac{1{,}36}{\sqrt{n}}$	$\frac{1{,}63}{\sqrt{n}}$
N	$\frac{1{,}1449}{D_{n,\alpha}^{\ 2}}$	$\frac{1{,}8596}{D_{n,\alpha}^{\ 2}}$	$\frac{2{,}5949}{D_{n,\alpha}^{\ 2}}$

Na tabela acima estão mostrados os valores para os graus de confiança α de 90%, 95% e 99%. Esta tabela nos diz que para obter um erro máximo de $D_{n,\alpha}$% com 95% de certeza é necessário gerar (1,8596 / $D_{n,\alpha}^{\ 2}$) amostras. Por exemplo, se for necessário obter um erro máximo de 2% com 95% de confiança, devem ser feitas pelo menos 4649 amostragens.

O teorema de Kolmogorov-Smirnov permite ter uma idéia do número de iterações necessárias para se chegar a uma solução com determinado grau de confiança. Entretanto, analisando as ferramentas disponíveis no mercado pudemos encontrar um outro método para verificação de convergência que se baseia na estabilidade do histograma amostral obtido a partir da simulação. A ferramenta analisada é conhecida como @Risk, e é produzida por uma empresa chamada Palisade (www.palisade.com). Segundo [41] o @Risk é a ferramenta mais utilizada no mundo para análise de risco. O @Risk é uma ferramenta que funciona como um add-in do Excel, permitindo fazer simulações de Monte Carlo em modelos expressos em planilhas. O método que o @Risk utiliza para fazer a verificação de convergência se baseia na análise da variação percentual da média em cada iteração, permitindo verificar se o resultado final convergiu ou não.

Todo o método se baseia no princípio de que se a média convergiu para determinado valor, e se mantém estável, é provável que toda a distribuição tenha convergido para a sua forma final, e esteja estabilizada também. Isto quer dizer que a geração de mais amostras não produz mudanças significativas na forma da distribuição de probabilidade. De um modo geral a idéia do método é anotar a cada iteração a variação percentual obtida de alguma variável de controle (no caso a média da distribuição). O valor da variação percentual forma um novo histograma, possuindo um valor para cada iteração. Uma média pode ser obtida somando-se em módulo a variação percentual das últimas k iterações. Quanto mais próxima esta média estiver de zero, mais próximo o histograma está da estabilização. O motivo para fazermos a média em k iterações é que a variabilidade da variação percentual é alta, podendo gerar, em algumas vezes, uma variação pequena em uma única iteração. Esta acumulação permite evitar a ocorrência de falsas verificações positivas de convergência, por isso, quanto maior o valor de k maior a certeza da convergência.

Vamos tentar entender melhor como funciona esse método matematicamente. Seja $E[X]_n$ a média amostral da variável X após a iteração n e x_i o valor obtido para aquela variável na iteração i. $E[X]_n$ pode ser calculado usando-se a seguinte fórmula:

$$E[X]_n = \frac{\sum_{i=1}^{n} x_i}{n}$$

A variação percentual da média $VPM[X]_n$ da variável X na iteração n pode ser definida como o valor percentual que a diferença entre $E[X]_n$ e $E[X]_{n-1}$ representa em relação à $E[X]_{n-1}$. Utilizando regra de três, $VPM[X]_n$ pode ser definida da seguinte maneira:

$$\frac{E[X]_{n-1}}{E[X]_n - E[X]_{n-1}} = \frac{100}{VPM[X]_n}$$

$$VPM[X]_n = 100\left(\frac{E[X]_n - E[X]_{n-1}}{E[X]_{n-1}}\right) = 100\left(\frac{(n-1)\sum_{i=1}^{n} x_i - n\sum_{i=1}^{n-1} x_i}{n(n-1)E[X]_{n-1}}\right) = 100\left(\frac{x_n - E[X]_n}{(n-1)E[X]_{n-1}}\right.$$

$$VPM[X]_n = 100\left(\frac{x_n - E[X]_n}{\sum_{i=1}^{n-1} x_i}\right)$$

Seja c o maior valor possível para o tempo total de um projeto. Tanto x_n quanto $E[X]_n$ são menores ou iguais a c. Portanto: $-c <= x_n - E[X]_n <= +c$. Por definição x_i é sempre maior ou igual a 0, porque cada x_i representa a duração total de um cenário, que nunca é negativo. Podemos afirmar portanto que:

$$\lim_{n\to\infty} VPM[X]_n = 0$$

A média de VPM[X] nas k últimas iterações pode ser definida como:

$$E[VPM[X]]_n = \frac{\sum_{i=n-k}^{n} \left| VPM[X]_i \right|}{k}$$

Da mesma forma que VPM[X], E[VPM[X]] também tende a 0 com n tendendo a infinito, porque passa a ser uma soma do módulo de k termos que tendem a 0. Podemos dizer, portanto, que uma determinada variável X convergiu com determinado grau de confiança c em k iterações se E[VPM[X]] < c em k iterações.

Este critério de verificação de convergência fornece apenas uma indicação de que a simulação convergiu, não fornecendo absoluta certeza. É possível construir alguns casos de sorteios em que o método indica que a distribuição convergiu sem que ela tenha realmente convergido. No entanto, ao escolher um k suficientemente grande esta possibilidade pode ser minimizada. Ela pode ser diminuída também utilizando outras variáveis como indicadores de convergência no lugar da média, como por exemplo o desvio padrão. Este problema não parece, no entanto, ser tão grave, visto que este método tem sido utilizado por muitas pessoas em todo o mundo através do @Risk.

4.6 - O SimProcess

Com o objetivo de observar na prática todos os conceitos apresentados neste trabalho, foi construída uma ferramenta para simulação de redes RANDRE. Esta ferramenta recebeu o nome de SimProcess.

A aplicação foi desenvolvida na linguagem C# utilizando os recursos da infraestrutura Microsoft .Net 2003. Ele foi desenvolvido como uma aplicação “desktop” para Windows, podendo ser executada em Windows com versão igual ou superior ao do Windows 98. A aplicação possui um editor para a criação de redes de atividades e possui várias opções para a execução dos algoritmos de simulação e obtenção dos resultados.

Apesar dos algoritmos principais terem sido descritos através de uma linguagem estruturada, o código fonte do programa não se encontra exatamente daquela maneira. No Anexo A pode ser encontrada uma descrição da forma como o programa está estruturado. Esta

documentação é importante, porque permitirá que em trabalhos futuros a ferramenta possa ser estendida, sem que o trabalho feito aqui precise ser executado novamente.

O programa foi projetado com o objetivo de ser fácil de utilizar, possuindo uma interface que busca ser o mais amigável possível. No entanto, a complexidade do programa faz que alguns detalhes possam, às vezes, passar despercebidos a um usuário desatento. No Anexo B é possível encontrar uma descrição dos principais fatos que uma pessoa precisa saber para realizar uma simulação com sucesso utilizando o programa.

5 - Estudo de Caso – Fábrica de Software

Nesta seção será apresentado um estudo de caso que tem por objetivo mostrar como os principais recursos introduzidos pelo “SimProcess” podem ser utilizados na avaliação e otimização de processos reais.

Este estudo de caso se baseia em um modelo criado por um dos grupos de alunos que participaram da disciplina de Engenharia de Software ministrada pelo professor Eber Schimtz no segundo período letivo de 2004 do Mestrado de Sistemas de Informação do Núcleo de Computação Eletrônica (NCE) da Universidade Federal do Rio de Janeiro (UFRJ).

Durante o curso, os alunos eram apresentados a algumas das técnicas principais da área de Engenharia de Softwares. Alguns dos tópicos abordados na matéria foram os seguintes: Ciclos de Vida de Software; Elicitação e Especificação de Requisitos; Ambientes de Desenvolvimento de Software, Ferramentas CASE; Métodos de Desenvolvimento de Software, Métodos Orientados a Objeto; Análise e Projeto de Sistemas de Informação; Planejamento e Gerência de Projetos; Manutenção de Sistemas; Reutilização e Re-engenharia de Software; Avaliação de Software, Critérios de Qualidade; Tecnologias de Suporte ao Desenvolvimento.

Com o objetivo de tornar a aula mais produtiva, era exigido dos alunos um trabalho que deveria ser elaborado durante todo o período do curso. Neste trabalho os alunos deveriam desenvolver um plano para uma fábrica de software fictícia. À medida que eles iam aprendendo novos conceitos, estes eram aplicados ao projeto da fábrica, fazendo com que ele ganhasse corpo durante o curso. O objetivo principal do trabalho era que os alunos criassem um modelo das atividades necessárias em uma fábrica para produzir software. O objetivo desta fábrica é atender aos pedidos de desenvolvimento realizado pelos clientes, identificando as suas necessidades, e produzindo os softwares e artefatos que atendam à sua demanda.

No projeto desta fábrica, os alunos deveriam utilizar ao máximo os conceitos apresentados no curso. A fábrica foi criada aos poucos durante o curso. A cada etapa realizada os alunos faziam apresentações nas quais demonstravam o trabalho realizado naquela semana, e eram confrontados com perguntas do professor sobre as principais decisões tomadas no projeto. A turma discutia sobre cada um dos pontos, e tinham a oportunidade de trocar experiências e comparar os seus trabalhos ao dos outros.

Os alunos foram divididos em 5 grupos com aproximadamente 5 integrantes cada um. Cada um destes grupos produziu uma especificação diferente de uma fábrica. Entre estas especificações, uma foi selecionada para fazer parte desta dissertação. Este modelo foi produzido pelo grupo por Andréa Magalhães Magdaleno, Carla Tavares, Danilo Pestana de Freitas e Vanessa Tavares Nunes. Este modelo foi selecionado porque atendeu às principais exigências do trabalho, e apresentou uma riqueza de detalhes alta, além de uma organização clara. O modelo sofreu pequenas alterações para se adequar ao formato necessário para esta tese. Entretanto, ele não deve ser considerado um modelo ideal, nem um modelo completo para uma fábrica de software, afinal é apenas fictício.

Nesta secção mostraremos como algumas informações úteis podem ser obtidas sobre este modelo através da utilização do SimProcess, permitindo analisar o processo e identificar pontos onde ele pode ser melhorado.

5.1 - O Modelo

A atividade de criação de software pode ser muito complexa. Dependendo do nível de maturidade exigido do processo o número de atividades pode ser incrivelmente grande.

No trabalho era exigido que os alunos colocassem em suas fábricas grande parte das atividades exigidas em processos que podem ser classificados no padrão CMMI nível 2. Se os alunos conseguissem adicionar aos seus modelos atividades do nível 3, eles possuíam um diferencial e receberiam avaliações melhores ao final do curso. Todos os modelos se tornaram muito grandes, tornando difícil a análise e entendimento destes modelos. Para ilustrar as proporções que estes diagramas podem tomar, na Figura 23 se encontra o modelo da fábrica selecionado colocado todo em um único diagrama:

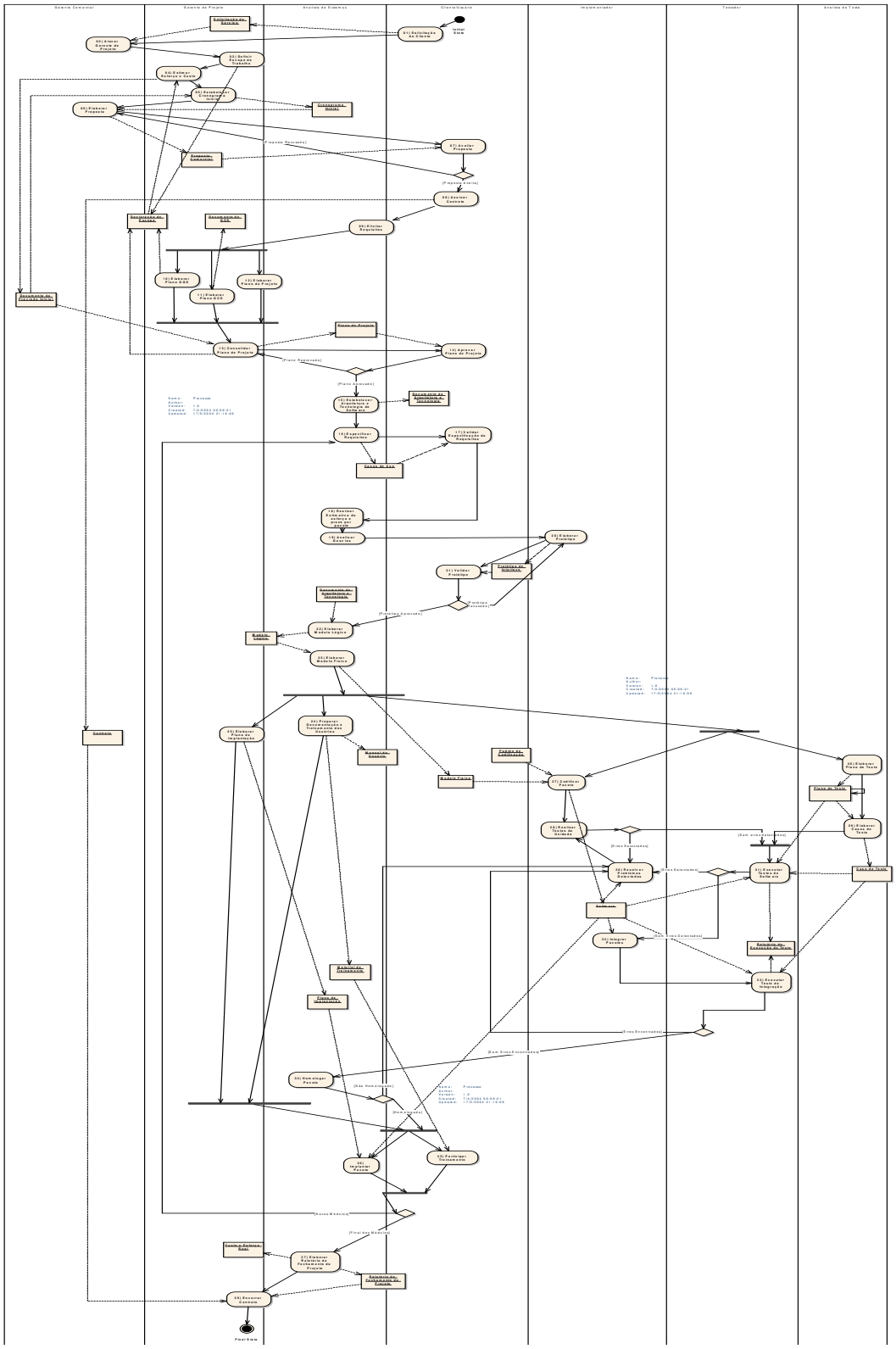

Figura 23: Modelo de atividades completo da Fábrica

Este modelo possui um total de 39 atividades, 11 barras de sincronismo e 8 decisões. é muito complicado explicar os detalhes de um modelo completo apresentado desta maneira. Por causa disso, estaremos usando uma abordagem de representar o modelo através de sub-modelos e macro-modelos. Neste método, um diagrama de alto-nível representa o relacionamento entre os principais grupos de atividades do modelo. Cada atividade deste modelo de alto nível pode possuir um sub-modelo associado, que representa o que realmente é feito quando aquela "macro-atividade" é executada. O macro-modelo da fábrica de software assume a forma da figura abaixo:

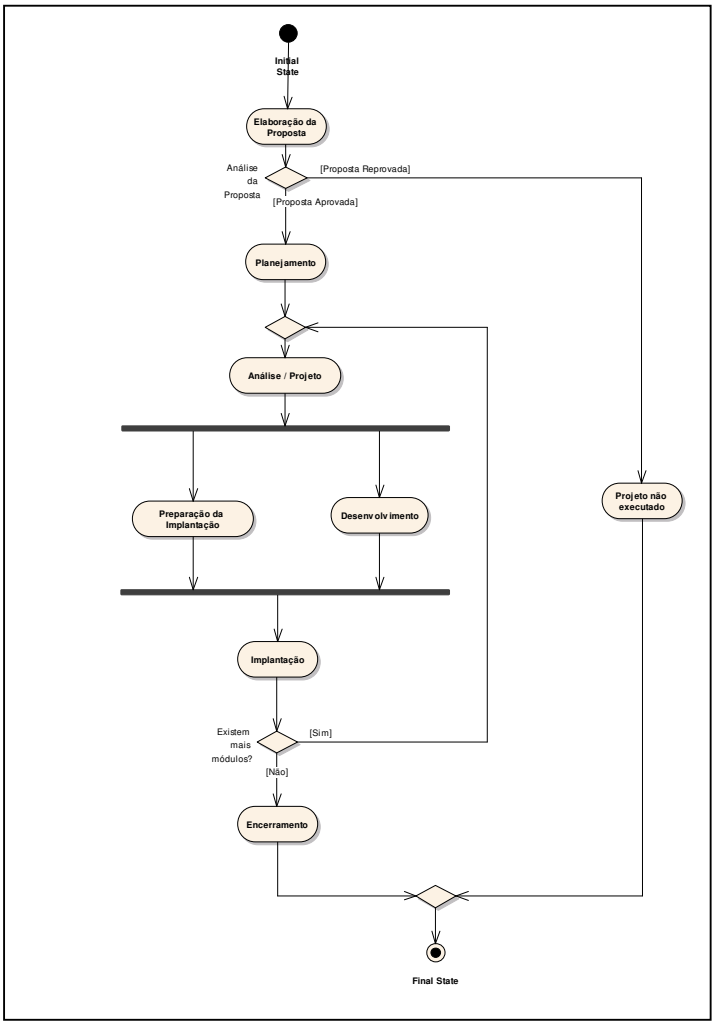

Figura 24: Diagrama de alto nível da Fábrica de Software

Observando este modelo de alto-nível podemos entender com maior facilidade o que está representado no diagrama anterior. O modelo é composto basicamente por 7 macro-atividades que vão desde o primeiro contato com o cliente à entrega final do produto. Quando uma nova execução do processo se inicia, a primeira macro-atividade executada é a "Elaboração da Proposta". Nela o primeiro contato com o cliente é estabelecido, e são

identificadas as suas necessidades, e como a fábrica de software pode atendê-las. Ao final uma proposta de trabalho é criada e negociada com o cliente.

Se a proposta inicial for aceita, o trabalho efetivo se inicia na macro-atividade de "Planejamento". Nela as definições dos requisitos são refinadas, e os planos de projeto são estabelecidos. Em seguida se inicia a fase de "Análise / Projeto", na qual os requisitos levantados são colocados no formato adequado, com a riqueza de detalhes desejada, e o sistema começa a ser montado. Desta fase surgem artefatos importantes para o resto do projeto, como os documentos de Casos de Uso, o Protótipo da Interface e o Modelo Lógico do software.

Uma vez terminada a fase de projeto, o fluxo de execução se divide em dois caminhos simultâneos. O primeiro é o do desenvolvimento do sistema em si, no qual as especificações são transformadas em código, e testadas. Na atividade "Preparação da implantação" algumas das tarefas necessárias para que o módulo possa ser implantado no cliente já começam a ser executadas.

Depois de terminadas essas duas atividades, o módulo já pode ser implantado no cliente. A macro-atividade "Implantação" contém todas as atividades a serem executadas com este fim. Em seguida, no diagrama, existe uma decisão que verifica se existem mais módulos a serem implementados. Se existirem o fluxo volta à "Análise/Projeto" para que ele seja construído, senão o fluxo segue para a atividade "Encerramento" no qual o contrato é fechado e o projeto termina. Note, no entanto, que existe uma simplificação importante neste modelo. Nele os módulos são implementados um por vez, um em seguida do outro, ou seja não existe a execução de diferentes módulos em paralelo. Esta é uma simplificação indesejável, já que não é o que acontece na maior parte das fábricas de software, no entanto é necessária para manter o diagrama em um nível de complexidade razoável.

Como veremos à frente, um quadro como o contido abaixo será utilizado para definir as probabilidades associadas a cada uma das decisões do diagrama. As probabilidades de transição das decisões do diagrama de alto nível estão apresentadas abaixo:

Decisão	Resultado	Probabilidades		
		1ª	2ª	3ª
Existem mais módulos?	Sim	50%		
	Proposta Necessita Reavaliação	50%		
Análise da Proposta	Proposta Aprovada	60%		
	Proposta Reprovada	40%		

Tabela 3: Resultado das decisões do macro-modelo

No *SimProcess*, um processo é representado tanto pelas atividades, quanto pelos recursos que executam estas atividades. Abaixo se encontra uma tabela contendo os principais recursos utilizados, e a quantidade disponível de cada um deles. A quantidade de recursos foi

selecionada de modo que a fábrica possa suportar até aproximadamente 3 projetos sendo executados ao mesmo tempo.

Recurso	Quantidade disponível
Gerente comercial	2
Gerente de projeto	2
Analista de sistemas	4
Implementador	5
Testador	2
Analista de Testes	1

Tabela 4: Recursos disponíveis para execução

Cada uma das macro-atividades apresentadas na Figura 24 possui um subdiagrama associado. Este subdiagrama contém cada uma das atividades que precisam ser executadas para que a macro-atividade produza seu objetivo final. Nas próximas secções mostraremos cada um destes diagramas e detalharemos cada uma das suas atividades, mostrando as suas durações e os recursos necessários para executá-las.

5.1.1 - Elaboração da Proposta

Neste sub-diagrama o primeiro contado é estabelecido com o cliente, e suas necessidades são identificadas. Um cronograma inicial é elaborado, baseado nesta previsão inicial, e o custo do projeto é estimado. Uma rodada de negociação é realizada até que seja possível fechar um contrato. Abaixo se encontra o diagrama de atividades relativo a este trecho:

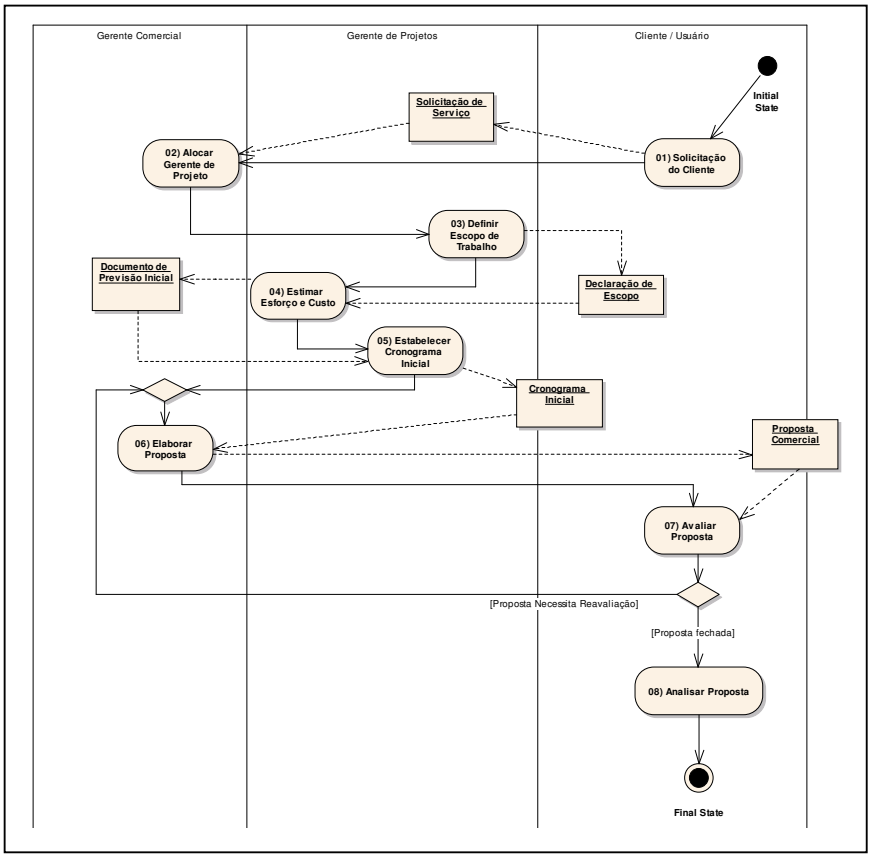

Figura 25: Diagrama de atividades - Elaboração da Proposta

O quadro abaixo apresenta cada uma das atividades que compõem esta sub-rede. Ao lado de cada atividade são colocadas informações sobre as suas durações e os recursos que ela necessita. Todas as durações utilizadas neste estudo de caso são dadas por distribuições de probabilidade Triangulares, cujos parâmetros são colocados na tabela. Caso a atividade possua mais de uma duração, ou seja, uma duração diferente para a sua segunda execução, esta é colocada em uma linha abaixo na tabela. À direita da duração se encontram os recursos necessários para executar cada atividade. O recurso é identificado pelo seu tipo e a quantidade necessária. Caso a atividade necessite de mais de um tipo de recurso diferente este é colocado na linha abaixo. Quadros deste tipo serão usados para representar todas as atividades dos sub-diagramas.

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
01) Solicitação do Cliente	0,5	2	7		
02) Alocar Gerente de Projeto	0,2	0,5	1	Ger. Comercial	1
03) Definir Escopo de Trabalho	7	12	25	Ger. Projetos	1
				Analista de Sist.	1
04) Estimar Esforço e Custo	2	4,5	8	Ger. Projetos	1
				Analista de Sist.	1
05) Estabelecer Cronograma Inicial	1	3	5	Ger. Projetos	1
06) Elaborar Proposta	1	2	4	Ger. Comercial	1
	0,5	1	2		

07) Avaliar Proposta	3	5	8
	1	3	4
08) Analisar Proposta	0,5	1	2

Tabela 5: Durações e recursos da Elaboração de Proposta

No quadro abaixo estão representadas as decisões contidas no diagrama e as suas respectivas probabilidades de transição. Cada linha da tabela contém uma decisão, e nas linhas abaixo dela estão cada um dos possíveis resultados destas decisões. Nestas tabelas, cada resultado pode estar associado a até 3 probabilidades, correspondentes às primeira, segunda e terceira passagens pela decisão. Outras tabelas neste formato serão usadas daqui para frente.

Decisão	Resultado	Probabilidades		
		1ª	2ª	3ª
Avaliação da Proposta	Proposta Fechada	40%	70%	80%
	Proposta Necessita Reavaliação	60%	30%	20%

Tabela 6: Probabilidades das decisões da Elaboração de Proposta

5.1.2 - Planejamento

Durante a etapa de "Planejamento" são elaborados os planos de projeto e de qualidade, e estes são enviados para o cliente para aprovação. Se o cliente não aprovar o plano pode ser revisto até que possa ser aceito. Abaixo se encontram o diagrama deste sub-diagrama e os dados das suas atividades e decisões:

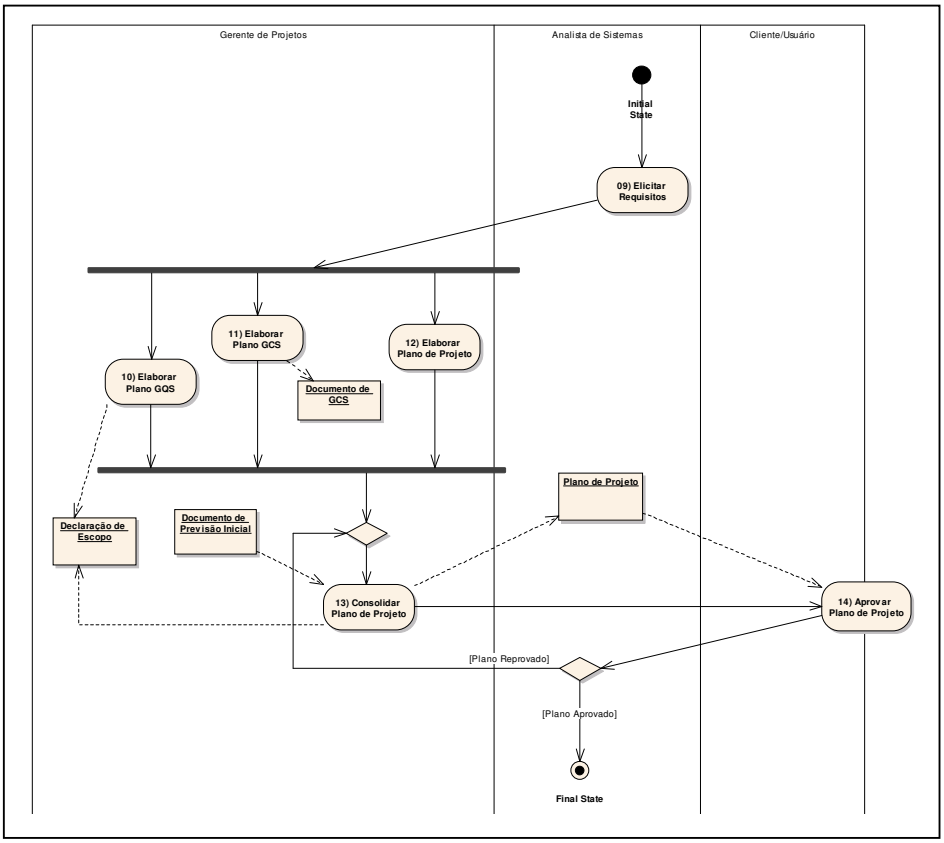

Figura 26: Diagrama de atividades – Planejamento

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
09) Elicitar Requisitos	10	21	42	Analista de Sist.	3
10) Elaborar Plano GQS	2	5	8	Ger. Projetos	1
11) Elaborar Plano GCS	4	6	8	Ger. Projetos	1
12) Elaborar Plano de Projeto	4	8	11	Ger. Projetos	1
13) Consolidar Plano de Projeto	1	3	5	Ger. Projetos	1
	0,5	1	2		
14) Aprovar Plano de Projeto	1	2	4		
	0,5	1	2		

Tabela 7: Durações e recursos do Planejamento

Decisão	Resultado	Probabilidades		
		1ª	2ª	3ª
Aprovação de planos	Plano Aprovado	40%	60%	80%
	Plano Reprovado	60%	40%	20%

Tabela 8: Probabilidades das decisões do Planejamento

5.1.3 - Análise / Projeto

Uma vez terminada a etapa de planejamento, iniciam-se as atividades que tem por objetivo produzir o software. Na fase de "Análise / Projeto" os requisitos do cliente são refinados e formatados em casos de uso. Estes casos de uso são utilizados para criar os modelos lógico e físico do sistema.

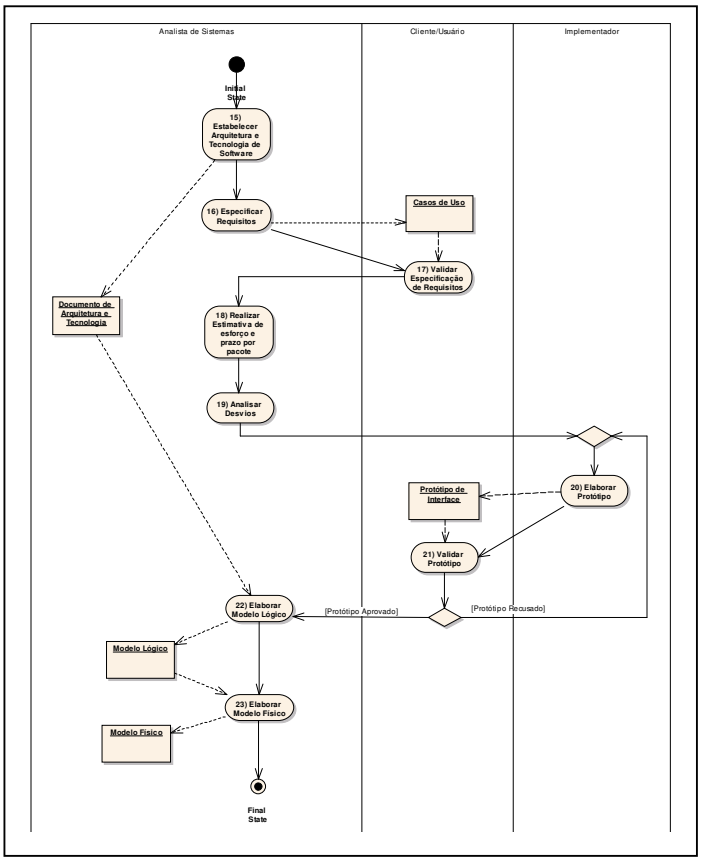

Figura 27: Diagrama de atividades – Análise / Projeto

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
15) Estabelecer Arquitetura e Tecnologia de Software	2	4	5	Analista de Sist.	1
16) Especificar Requisitos	5	11	20	Analista de Sist.	3
17) Validar Especificação de Requisitos	1,5	3	5,5		
18) Realizar Estimativa de esforço e prazo por pacote	0,5	2	6	Ger. Projetos	1
				Analista de Sist.	1
19) Analisar Desvios	0,5	2	2,5	Analista de Sist	1
				Ger. Projetos	1
20) Elaborar Protótipo	2	6	10	Implementador	3
	0,5	1,5	4		
21) Validar Protótipo	0,5	1,5	4		
22) Elaborar Modelo Lógico	5	8	12	Analista de Sist.	2

23) Elaborar Modelo Físico	1	4	6	Analista de Sist.	2

Tabela 9: Durações e recursos de Análise / Projeto

Decisão	Resultado	Probabilidades		
		1ª	2ª	3ª
Aprovação do protótipo	Protótipo Aprovado	40%	60%	80%
	Protótipo Reprovado	60%	40%	20%

Tabela 10: Probabilidades das decisões de Análise / Projeto

5.1.4 - Preparação da Implantação

Nesta parte se inicia a preparação para a implantação do sistema no cliente. Este é um diagrama simples composto por apenas duas atividades. Em uma delas é preparado o plano que será usado para a implantação do software, e na segunda o manual do usuário é preparado.

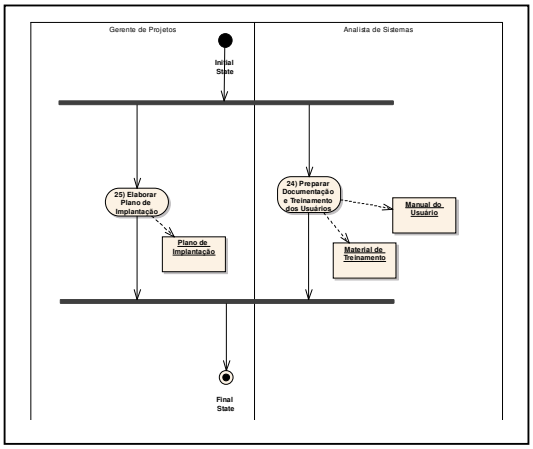

Figura 28: Diagrama de atividades – Preparação da implantação

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
24) Preparar Documentação e Treinamento dos Usuários	3	7	10	Analista de Sist.	1
25) Elaborar Plano de Implantação	1	3	6	Ger. Projetos	1

Tabela 11: Durações e recursos de Preparação da Implantação

5.1.5 - Desenvolvimento

É nesta etapa do projeto que o software é realmente desenvolvido. Nele o sistema é codificado, e os testes são executados respeitando rigidamente um plano de testes

previamente estabelecido. O sistema passa por testes unitários, testes de integração e pela homologação antes que seja considerado pronto. Caso algum erro seja encontrado ele é corrigido, e o sistema passa pelos testes novamente.

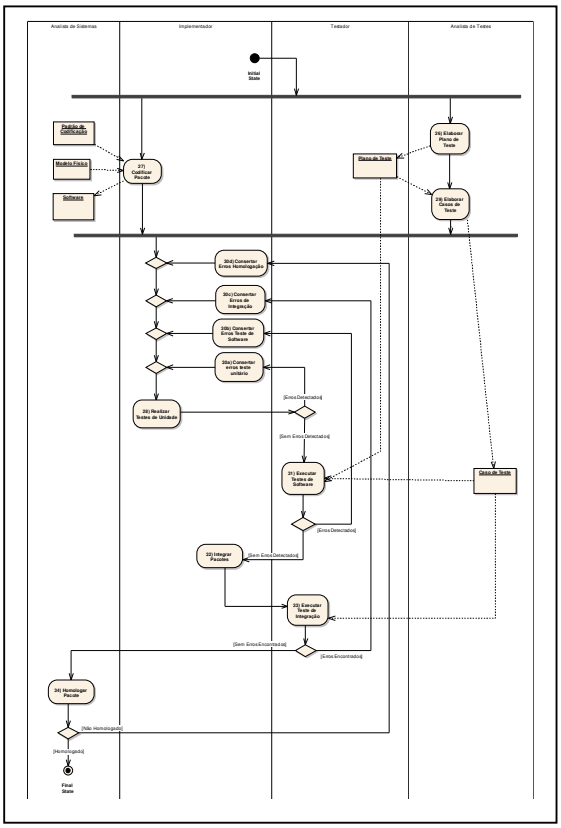

Figura 29: Diagrama de atividades – Desenvolvimento

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
26) Elaborar Plano de Teste	2	6	9	Analista Testes	1
27) Codificar Pacote	15	23	45	Implementador.	4
28) Realizar Testes de Unidade	1	3	4,5	Implementador	2
29) Elaborar Casos de Teste	4	9	13	Analista Testes	1
30a) Consertar erros teste unitário	0,5	2,5	4	Implementador.	1
30b) Consertar Erros Teste de Software	0,5	2,5	4	Implementador.	1
30c) Consertar Erros de Integração	0,5	2,5	4	Implementador.	1
30d) Consertar Erros de Homologação	0,5	2,5	4	Implementador.	1
31) Executar Testes de Software	1,5	3	5	Testador	2
32) Integrar Pacotes	1	2,5	3	Implementador	2
33) Executar Teste de Integração	1,5	3	5	Testador	2
34) Homologar Pacote	3	6	8	Analista de Sist.	1

Tabela 12: Durações e recursos de Desenvolvimento

Decisão	Resultado	Probabilidades		
		1ª	2ª	3ª
Teste unitário	Erros detectados	80%	40%	20%
	Sem erros detectados	20%	60%	80%
Teste de software	Erros detectados	80%	40%	20%
	Sem erros detectados	20%	60%	80%
Teste de integração	Erros detectados	80%	40%	20%
	Sem erros detectados	20%	60%	80%
Homologação	Não homologado	80%	40%	
	Homologado	20%	60%	

Tabela 13: Probabilidades das decisões de Desenvolvimento

5.1.6 - Implantação

Após ser desenvolvido, o sistema precisa ser entregue ao cliente. Nesta etapa são executadas as atividades necessárias para que o software possa ser instalado e disponibilizado para ao cliente. Além disso é necessário fornecer o treinamento adequado ao cliente para que ele saiba utilizar o novo sistema da maneira correta.

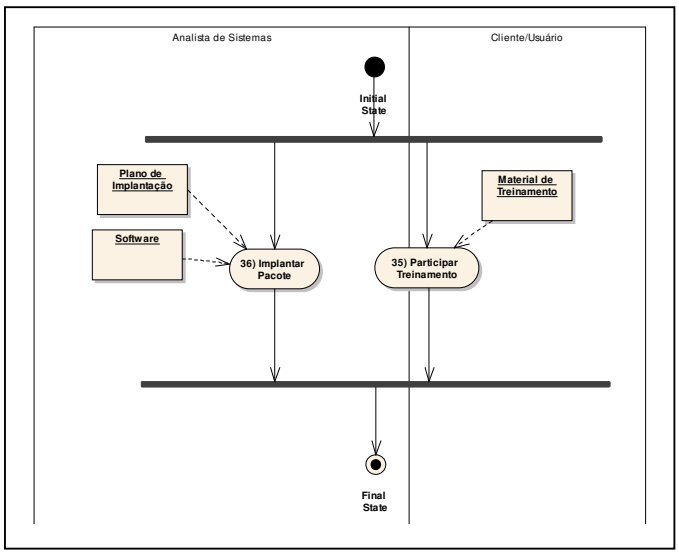

Figura 30: Diagrama de atividades - Implantação

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
35) Participar Treinamento	4	8	12	Analista de Sist.	1
36) Implantar Pacote	1	3	5	Analista de Sist.	2

Tabela 14: Durações e recursos de Implantação

5.1.7 - Encerramento

Ao final do processo o projeto precisa ser encerrado, garantindo que o cliente recebeu tudo o que foi contratado da forma correta.

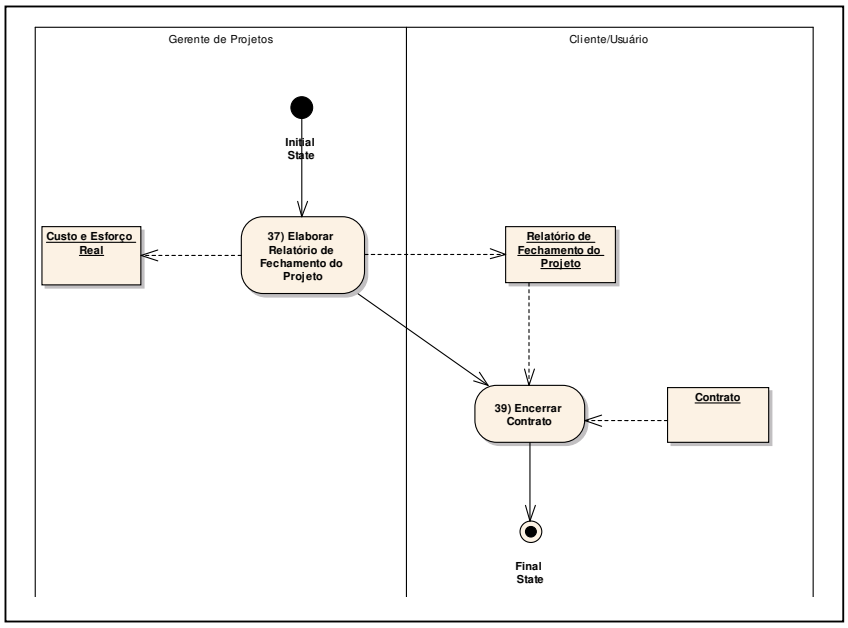

Figura 31: Diagrama de atividades - Encerramento

Atividade	Duração (em dias)			Recurso	
	Min	M.P.	Max.	Nome	Qtd.
37) Elaborar Relatório de Fechamento do Projeto	1,5	2	4	Ger. de Projetos	1
39) Encerrar Contrato	1	2	3		

Tabela 15: Durações e recursos de Encerramento

5.2 - Avaliação do Processo

O modelo detalhado na seção anterior será utilizado aqui para mostrar como informações relevantes sobre um processo podem ser obtidas a partir da utilização do SimProcess. Além de obter informações sobre o processo tentaremos também identificar pontos de melhoria, e utilizaremos o próprio SimProcess para verificar se essas modificações do processo surtiriam algum efeito.

A abordagem utilizada para realizar as análises se baseará em duas partes principais. Primeiramente faremos algumas simulações básicas do processo. Essas simulações gerarão dados, que serão analisados para identificar possíveis pontos de melhoria no processo. Esses pontos de melhoria se transformarão em perguntas, que tentaremos responder utilizando o próprio SimProcess. Por exemplo, se identificarmos que o número de analistas disponíveis

para o processo é muito pequeno, poderemos levantar a seguinte pergunta: Será que contratando dois novos analistas poderemos obter melhores resultados? Esta pergunta será verificada realizando uma nova simulação do processo, mas agora com dois analistas a mais. Os resultados dos dois casos serão então comparados para verificar a validade da conjectura.

5.2.1 - Análises básicas

O primeiro problema que encontramos antes de executar uma simulação é a determinação de quantas execuções dos escalonamentos são necessários para obter uma simulação na qual possamos nos basear.

Conforme vimos anteriormente, os algoritmos de simulação executam um determinado número de simulações, e os resultados obtidos em cada uma das variáveis de controle são anotados a cada escalonamento. Se poucos escalonamentos forem realizados, existe uma forte possibilidade de que não sejam contabilizados nos resultados alguns exemplares da rede que tem uma baixa probabilidade de ocorrer. Apesar de serem poucos prováveis, esses possíveis cenários da rede podem ser muito longas, e por isso ter um impacto grande nos resultados. Existe um limite quanto ao número máximo de escalonamentos que podemos realizar, já que a execução destas simulações leva tempo.

Nas simulações que serão feitas aqui serão feitas sempre 5000 execuções do projeto. Conforme mostrado no capítulo 4 o teorema de Komolgorov-Smirnov diz que com 5000 amostras a probabilidade do erro máximo ser inferior a 1,9% é de 95%. Este é um grau de confiança bastante satisfatório para estas simulações. Conforme discutido, o SimProcess possui um método de análise de convergência que se baseia na verificação da variação percentual da média e do desvio padrão a cada iteração. Este método será utilizado para mostrar que o resultado da simulação realmente convergiu.

Ao executar uma simulação o *SimProcess* exibe a janela na Figura 32. Nela são exibidas as variações da média e do d.p. a cada iteração. Existe ainda, à direita, um sinal que fica vermelho ou verde conforme a variação da média fica abaixo ou acima de um determinado valor. Este valor pode ser configurado utilizando a janela de configurações. Além desta análise no momento da execução, o sistema fornece um resumo para análise de convergência de cada uma das variáveis obtidas na simulação. Neste resumo o número de execuções é dividido em 10 partes iguais, e em cada um é exibida a média da variação percentual da média e do desvio padrão, por exemplo, do primeiro ao centésimo escalonamento a média da variação percentual foi de 0,1%, e assim por diante. Estes dados podem ser utilizados para acompanhar a variação percentual durante a simulação permitindo ter uma idéia da velocidade de convergência.

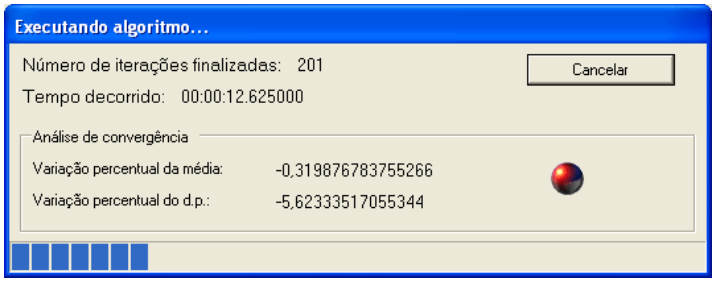

Figura 32: Tela de progresso do SimProcess

Vamos começar as nossas análises básicas pela execução do algoritmo Simulação de Instância Única com Recursos sobre a rede. Para não haver influência das diferentes heurísticas de escalonamento será utilizada sempre a mesma heurística em todas as execuções. A heurística de referência utilizada será a randômica, para evitar que possíveis vantagens de determinadas heurísticas induzam erros de interpretação. Esta heurística seleciona aleatoriamente a próxima tarefa a executar quando há empate. Foram executadas 5000 iterações do algoritmo de escalonamento, levando um tempo total de 3 minutos e 43 segundos em um AMD Athlon 2.0 Ghz com 512Mb de memória. A partir da metade da simulação a variação percentual da média do tempo total abaixo de 0,1% em praticamente todas as iterações. Abaixo se encontra o histórico de modificação da variação da média e do d.p. do tempo total conforme as execuções eram feitas:

Intervalo	Média da variação percentual da média	Média da variação percentual do desvio padrão
0 - 499	1,005	10,014
500 - 999	0,093	1,152
1000 - 1499	0,056	0,678
1500 - 1999	0,039	0,424
2000 - 2499	0,032	0,413
2500 - 2999	0,025	0,292
3000 - 3499	0,022	0,271
3500 - 3999	0,019	0,252
4000 - 4499	0,016	0,184
4500 - 4999	0,015	0,154

Tabela 16: Análise de convergência da simulação de instância única

Como podemos ver pelos dados da tabela, o histograma do tempo total parece estar convergindo. No final da execução podemos ver que a média da variação da média do tempo total ficou em 0,015% e a do desvio padrão em 0,154%, que são valores aceitáveis. Abaixo se encontram os dados obtidos nesta simulação separados por atividades:

Nome da Tarefa	MP(*)	MD(*)	QMT(*)	MTI(*)	MTF(*)	CC(*)
01) Solicitação do Cliente	100%	3,1	1,0	0,0	3,1	100%
02) Alocar Gerente de Projeto	100%	0,8	1,0	3,1	3,7	100%
03) Definir Escopo de Trabalho	100%	14,7	1,0	3,7	18,5	100%
04) Estimar Esforço e Custo	100%	4,8	1,0	18,5	23,3	100%

05) Estabelecer Cronograma Inicial	100%	3,0	1,0	23,3	26,3	100%
06) Elaborar Proposta	100%	3,3	1,8	26,3	33,5	100%
07) Avaliar Proposta	100%	7,6	1,8	28,7	37,2	100%
08) Analisar Proposta	100%	1,3	1,0	37,2	38,4	100%
Projeto não executado	40%	1,0	0,4	38,2	39,2	40%
09) Elicitar Requisitos	60%	24,4	0,6	38,5	62,8	60%
12) Elaborar Plano de Projeto	60%	7,7	0,6	64,1	71,8	24%
11) Elaborar Plano GCS	60%	6,0	0,6	64,7	70,7	42%
10) Elaborar Plano GQS	60%	5,1	0,6	65,0	70,0	52%
13) Consolidar Plano de Projeto	60%	4,0	1,1	74,2	80,0	60%
14) Aprovar Plano de Projeto	60%	3,4	1,1	77,2	81,6	60%
15) Estabelecer Arquitetura e Tecnologia de Software	60%	7,4	1,2	81,6	234,4	60%
16) Especificar Requisitos	60%	24,2	1,2	85,2	246,4	60%
17) Validar Especificação de Requisitos	60%	6,7	1,2	97,3	249,7	60%
20) Elaborar Protótipo	60%	10,5	1,9	105,1	260,6	60%
19) Analisar Desvios	60%	3,4	1,2	103,5	254,2	60%
18) Realizar Estimativa de esforço e prazo por pacote	60%	5,7	1,2	100,6	252,6	60%
21) Validar Protótipo	60%	6,4	1,9	111,2	262,6	60%
22) Elaborar Modelo Lógico	60%	16,8	1,2	116,8	271,0	60%
23) Elaborar Modelo Físico	60%	7,4	1,2	125,1	274,7	60%
29) Elaborar Casos de Teste	60%	17,4	1,2	134,5	289,1	1%
26) Elaborar Plano de Teste	60%	11,5	1,2	128,8	280,4	1%
27) Codificar Pacote	60%	55,7	1,2	128,8	302,5	60%
31) Executar Testes de Software	60%	27,0	5,1	165,7	360,0	60%
28) Realizar Testes de Unidade	60%	33,1	7,0	156,6	356,8	60%
32) Integrar Pacotes	60%	13,1	3,6	180,6	362,2	60%
33) Executar Teste de Integração	60%	19,1	3,6	182,8	365,4	60%
34) Homologar Pacote	60%	22,6	2,4	207,4	371,0	60%
30a) Consertar erros teste unitário	57%	7,6	1,9	166,8	283,9	57%
30b) Consertar Erros Teste de Software	56%	6,3	1,5	176,4	283,1	56%
30c) Consertar Erros de Integração	54%	5,2	1,2	194,4	284,2	54%
30d) Consertar Erros Homologação	51%	5,4	1,2	222,8	321,6	51%
35) Participar Treinamento	60%	16,1	1,2	249,0	379,0	60%
36) Implantar Pacote	60%	6,0	1,2	249,0	374,0	0%
25) Elaborar Plano de Implantação	60%	6,7	1,2	128,8	278,0	0%
24) Preparar Documentação e Treinamento dos Usuários	60%	13,4	1,2	128,8	281,4	0%
37) Elaborar Relatório de Fechamento do Projeto	60%	2,4	0,6	379,0	381,5	60%
39) Encerrar Contrato	60%	1,8	0,6	381,5	383,5	60%

(*) MP – Média da presença da atividade
MD – Média da duração da atividade
QMT – Quantidade média de tarefas daquela atividade (número de execuções)
MTI – Média do tempo de início
MTF – Média do tempo final
CC – Probabilidade da atividade estar no caminho crítico

Tabela 17: Resultado da simulação de instância única

Observando esses dados notamos que algumas atividades possuem médias de duração muito superiores às restantes. Apenas as atividades “31) Executar Testes de Software”, “32) Integrar Pacotes”, “28) Realizar Testes de Unidade”, “12) Elaborar Plano de Projeto”, “17) Validar Especificação de Requisitos” e “30a) Consertar erros teste unitário” possuem média de duração superior a 20 dias. Podemos observar que a presença média das atividades 01 a 07 é de 100%, enquanto todas as outras possuem média de presença inferior a 60%.

Abaixo se encontra o gráfico de distribuição de probabilidade acumulada do tempo total de projeto gerado pelo sistema:

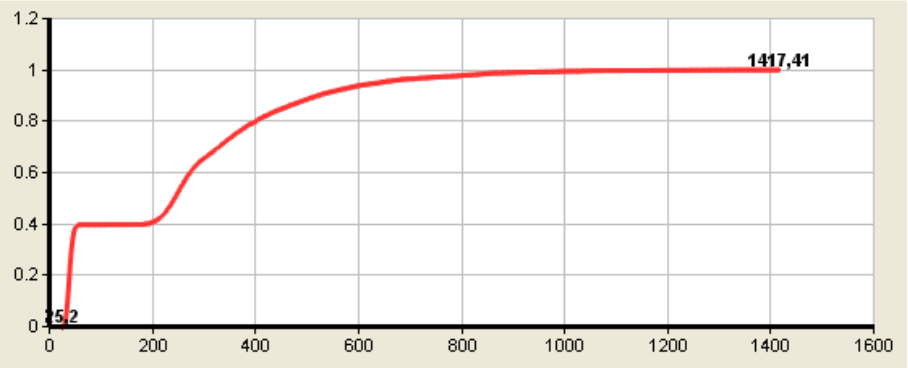

Figura 33: Resultado da simulação (I)

O gráfico mostra um "salto" que indica que praticamente não ocorrem projetos com tempo total entre 50 e 200 dias. Este "salto" ocorre por causa da possibilidade de o projeto acabar mais cedo caso a proposta não seja aceita pelo cliente. Em cerca de 40% dos projetos iniciados, não há acordo com o cliente e o projeto se encerra. Neste caso, levam 25 a 50 dias aproximadamente para se chegar à resposta final do usuário. Quando o projeto é aceito, ele leva pelo menos 200 dias para ser terminado, e pode levar até 1418 dias.

Vamos agora executar o algoritmo de simulação de processo, para que possamos comparar os seus resultados aos obtidos até aqui. Novamente serão usadas a heurística randômica e 5000 execuções ou disparos de processos. A distribuição do tempo entre a chegada de duas requisições à fábrica assumirá a forma de uma distribuição triangular com os parâmetros: mínimo, 30 dias (mínimo), 80 dias (mais provável) e 150 dias (máximo). Utilizando o mesmo computador usado para realizar a simulação de instância única, a simulação de instâncias concorrentes foi executada em 14 minutos e 21 segundos. Vamos analisar em primeiro lugar a convergência desta simulação. Abaixo se encontra um quadro contendo a média da variação percentual da média e do desvio padrão, da mesma maneira que no algoritmo anterior:

Intervalo	Média da variação percentual da média	Média da variação percentual do desvio padrão
0 - 499	7,359	64,665
500 - 999	0,381	4,554
1000 - 1499	0,236	2,952
1500 - 1999	0,171	2,589
2000 - 2499	0,0944	0,976
2500 - 2999	0,0897	0,902
3000 - 3499	0,0844	0,992

3500 - 3999	0,0714	0,771
4000 - 4499	0,0488	0,532
4500 - 4999	0,0476	0,457

Tabela 18: Análise de convergência da simulação de instâncias concorrentes

Da mesma forma que na simulação de instância única, a média e o desvio padrão indicam uma convergência. A variação da média nas últimas 500 iterações ficou em torno de 0,05%, enquanto o desvio padrão variou 0,5%. A queda, no entanto, não parece ser tão constante quanto o ocorrido com o algoritmo anterior. Existem algumas oscilações no ritmo da queda provocada principalmente pela natureza do algoritmo. Abaixo se encontram os resultados obtidos para cada uma das atividades:

Nome da Tarefa	MP(*)	MD	QMT	MTI	MTF	CC	MTEF
01) Solicitação do Cliente	100%	3,16	1	0	3,16	100%	0
02) Alocar Gerente de Projeto	100%	0,39	1	3,16	3,77	100%	0
03) Definir Escopo de Trabalho	100%	14,67	1	4,2	18,82	100%	0,42
04) Estimar Esforço e Custo	100%	4,87	1	18,82	23,68	100%	0
05) Estabelecer Cronograma Inicial	100%	3,03	1	23,68	26,64	100%	0
06) Elaborar Proposta	100%	3,31	1,8	26,64	33,58	100%	0
07) Avaliar Proposta	100%	7,45	1,8	28,97	37,36	100%	0
08) Assinar Contrato	60%	1,14	0,6	36,4	37,6	60%	0
Projeto não executado	100%	1	1	36,8	37,9	100%	0
09) Elicitar Requisitos	60%	24,43	0,6	40,07	64,46	60%	2,49
12) Elaborar Plano de Projeto	60%	7,69	0,6	68,93	76,58	52%	4,42
11) Elaborar Plano GCS	60%	5,98	0,6	65,55	71,54	26%	1,07
10) Elaborar Plano GQS	60%	4,91	0,6	64,91	69,87	47%	0,44
13) Consolidar Plano de Projeto	60%	5,66	1,14	77,85	85,43	60%	0,12
14) Aprovar Plano de Projeto	60%	3,41	1,14	80,81	87,08	60%	0
15) Estabelecer Arquitetura e Tecnologia de Software	60%	7,28	1,18	87,56	305,17	60%	0,63
16) Especificar Requisitos	60%	23,59	1,18	94,39	324,61	60%	7,17
17) Validar Especificação de Requisitos	60%	6,51	1,18	106,51	327,93	60%	0
20) Elaborar Protótipo	60%	10,54	1,91	118,74	347,17	60%	4,57
19) Analisar Desvios	60%	3,26	1,18	113,33	333,47	60%	0,32
18) Realizar Estimativa de esforço e prazo por pacote	60%	5,55	1,18	110,42	331,47	60%	0,73
21) Validar Protótipo	60%	6,44	1,91	124,83	349,17	60%	0
22) Elaborar Modelo Lógico	60%	16,37	1,18	137,98	362,62	60%	5,23
23) Elaborar Modelo Físico	60%	7,31	1,18	148,51	368,79	60%	3,13
29) Elaborar Casos de Teste	60%	16,94	1,18	158,52	385,17	1%	0,51
26) Elaborar Plano de Teste	60%	11,1	1,18	152,78	375,96	1%	1,46
27) Codificar Pacote	60%	54,66	1,18	156,9	406,12	59%	9,32
31) Executar Testes de Software	60%	22,77	4,27	201,15	491,77	60%	1,27
30a) Consertar erros teste unitário	60%	18,5	4,73	227,1	387,7	60%	0,09
30b) Consertar Erros Teste de Software	58%	7,5	2	170,5	297,3	58%	0,03
30c) Consertar Erros de Integração	56%	6,5	1,3	186,4	298,9	56%	0,32
30d) Consertar Erros Homologação	55%	5,5	1,1	224,7	309,6	55%	0,16
28) Realizar Testes de Unidade	60%	28,26	5,91	188,1	486,82	60%	4,62

32) Integrar Pacotes	60%	10,68	2,93	227,81	498,64	60%	4,41
33) Executar Teste de Integração	60%	15,58	2,93	230,71	503,92	60%	1,5
34) Homologar Pacote	60%	18,05	1,88	272,01	510,54	60%	0,82
35) Participar Treinamento	60%	15,83	1,18	323,85	519,8	60%	1,35
36) Implantar Pacote	60%	5,9	1,18	327,98	519,35	14%	6,11

(*) MP – Média da presença da atividade
MD – Média da duração da atividade
QMT – Quantidade média de tarefas daquela atividade (número de execuções)
MTI – Média do tempo de início
MTF – Média do tempo final
CC – Probabilidade da atividade estar no caminho crítico
MTEF – Média do tempo de espera na fila

Tabela 19: Resultado da simulação de instâncias concorrentes

Avaliando os resultados apresentados acima, podemos ver que não há grandes alterações nas médias da presença, duração, e número de etapas de cada uma das atividades em relação aos resultados do algoritmo anterior. Já as médias do tempo inicial e tempo final são bastante afetadas pela mudança de algoritmo. Isso acontece porque algumas atividades precisam iniciar mais tarde, porque faltam recursos para que elas comecem no momento que deveriam começar. O caminho crítico também é afetado, e o tempo médio de espera em fila passa a ser diferente de 0. Este tempo de espera em fila mede o tempo decorrido entre o momento em que a atividade passou a estar disponível para ser executada e o momento em que ela começou a executar. Estas variáveis são úteis para identificar quais são as atividades que estão sendo bloqueadas pela inexistência de recursos.

Abaixo se encontra o gráfico de distribuição de probabilidade acumulada do tempo total de projeto gerado pelo sistema:

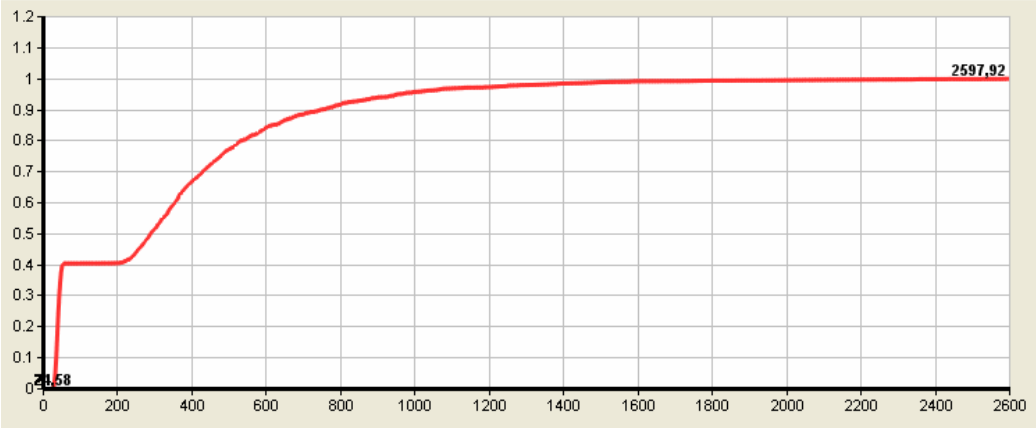

Figura 34: Resultado da Simulação (II)

O gráfico apresenta o mesmo formato básico obtido no algoritmo anterior. No entanto, chama atenção o fato da segunda metade do gráfico apresentar um comprimento muito maior. No outro algoritmo o valor máximo encontrado foi de 1418 dias, enquanto que neste algoritmo o tempo máximo foi de 2598 dias, quase o dobro. Abaixo se encontra uma imagem contendo os dois histogramas desenhados sobre o mesmo diagrama, para que possamos identificar as principais diferenças:

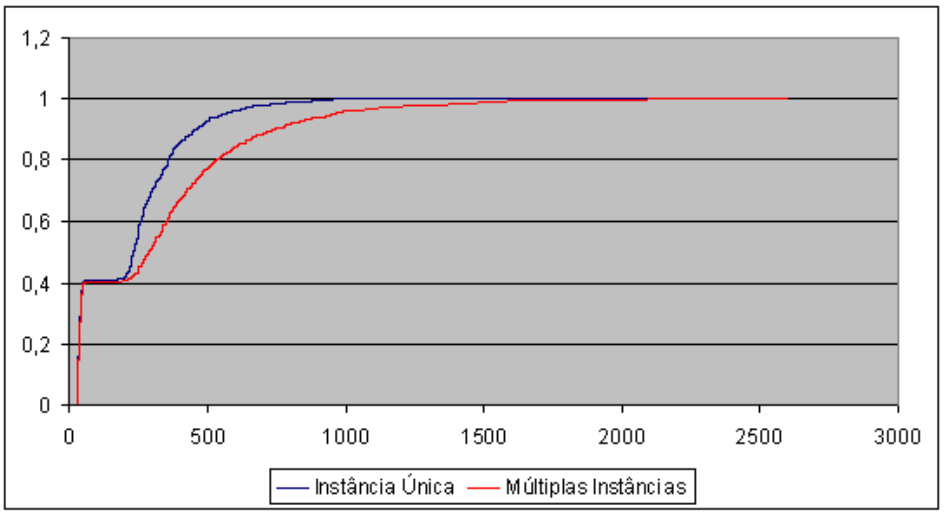

Figura 35: Comparação dos resultados da simulação

5.2.2 - Contratação de recursos para suprir demanda

Conforme observamos nos últimos resultados, existem diferenças grandes entre os resultados entre a simulação de instância única e a simulação de instâncias concorrentes. Isto acontece porque no primeiro as instâncias do processo são executadas separadamente, e não existe competição por recursos entre diferentes instâncias. Quando existe a competição entre instâncias a falta de recursos causa atrasos no término das instâncias do processo.

O gráfico que compara o tempo total nos dois algoritmos mostra que não há grandes diferenças entre os dois processos na fase inicial. Esta é a fase da elaboração e aprovação de proposta, que duram entre 25 e 50 dias. No entanto, na parte do gráfico que mostra términos superiores a 200 dias, que são os projetos que entram em fase de desenvolvimento, existe uma diferença grande no gráfico. A linha da instância única se aproxima mais rapidamente do 100%, indicando que mais projetos terminam em menos dias. Já a de instâncias concorrentes é mais alongada, indicando que existem mais instâncias que levam mais tempo. Enquanto o pior caso no de instância única é de, aproximadamente, 1400 dias, na de instâncias concorrentes é de 2600 dias.

Para identificar quais as atividades provocam essa discrepância, podemos utilizar uma variável chamada “Tempo de Espera em Fila”. Esta variável indica quanto tempo uma atividade ficou bloqueada, esperando para entrar em execução, devido ao bloqueio de algum recurso por outra instância do processo. No algoritmo de instância única esta variável é sempre igual a 0, visto que não há duas instâncias de processo executando ao mesmo tempo. No de instâncias concorrentes, quando a atividade está sendo bloqueada pela falta de algum recurso esta variável é diferente de 0. No quadro abaixo se encontram todas as atividades que possuem

essa variável diferente de 0 na Simulação, e os recursos utilizados por cada uma delas. Vamos tentar identificar à partir dele quais são os recursos que estão em falta.

Nome da Tarefa	MTEF	Recurso	Qtd. Necessária
03) Definir Escopo de Trabalho	0,42	Gerente de Projeto	1
		Analista de Sistemas	1
09) Elicitar Requisitos	2,49	Analista de Sistemas	3
12) Elaborar Plano de Projeto	4,42	Gerente de Projeto	1
11) Elaborar Plano GCS	1,07	Gerente de Projeto	1
10) Elaborar Plano GQS	0,44	Gerente de Projeto	1
13) Consolidar Plano de Projeto	0,12	Gerente de Projeto	1
15) Estabelecer Arquitetura e Tecnologia de Software	0,63	Analista de Sistemas	1
16) Especificar Requisitos	7,17	Analista de Sistemas	3
20) Elaborar Protótipo	4,57	Implementador	3
19) Analisar Desvios	0,32	Analista de Sistemas	1
		Gerente de Projeto	1
18) Realizar Estimativa de esforço e prazo por pacote	0,73	Gerente de Projeto	1
22) Elaborar Modelo Lógico	5,23	Analista de Sistemas	2
23) Elaborar Modelo Físico	3,13	Analista de Sistemas	2
29) Elaborar Casos de Teste	0,51	Analista de Testes	1
26) Elaborar Plano de Teste	1,46	Analista de Testes	1
27) Codificar Pacote	9,32	Implementador	4
31) Executar Testes de Software	1,27	Testador	2
30a) Consertar erros teste unitário	0,09	Implementador	1
30b) Consertar Erros Teste de Software	0,03	Implementador	1
30c) Consertar Erros de Integração	0,32	Implementador	1
30d) Consertar Erros Homologação	0,16	Implementador	1
28) Realizar Testes de Unidade	4,62	Implementador	4
32) Integrar Pacotes	4,41	Implementador	2
33) Executar Teste de Integração	1,5	Testador	2
34) Homologar Pacote	0,82	Analista de Sistemas	1
35) Participar Treinamento	1,35	Analista de Sistemas	1
36) Implantar Pacote	6,11	Analista de Sistemas	2
25) Elaborar Plano de Implantação	0,43	Analista de Sistemas	1
24) Preparar Documentação e Treinamento dos Usuários	0,61	Gerente de Projeto	2
37) Elaborar Relatório de Fechamento do Projeto	0,82	Gerente de Projeto	1

Tabela 20: Atividades com tempo de espera em fila não nulos

Para corrigir o atraso gerado pela falta de recursos é necessário contratar mais pessoas. No entanto, nem sempre é possível contratar todas as pessoas de que se precisa, por isso é necessário avaliar quais são as contratações mais necessárias. Analisando esta tabela podemos ver que as atividades que ficam em espera na fila dependem dos seguintes recursos: Gerente de Projeto, Analista de Sistemas, Implementador, Analista de Testes e Testador. A dependência de Gerente de Projeto ou de Analista de Sistemas aparecem em 70% das atividades contidas na tabela, e em, praticamente, todas as atividades que dependem de Implementador, o tempo médio de espera em fila é superior a 4 dias. Como esses recursos são importantes, vamos fazer contratações de pessoas com estes três perfis, e executar nova

simulação para verificar se as contratações surtiram o efeito desejado. O número de recursos a ser contratado é difícil de ser determinado, mas normalmente é possível obter uma aproximação a partir da quantidade necessária para executar essas atividades problemáticas. Vamos fazer uma nova simulação contratando os seguintes recursos:

Recurso	Quantidade contratada	Quantidade disponível
Gerente de projeto	1	3
Analista de sistemas	2	6
Implementador	2	7

Tabela 21: Novos valores para a quantidade disponível de recursos

Executando o algoritmo de simulação de múltiplas instâncias concorrentes utilizando esta nova configuração de recursos, obtemos um terceiro gráfico, que é mostrado abaixo em verde:

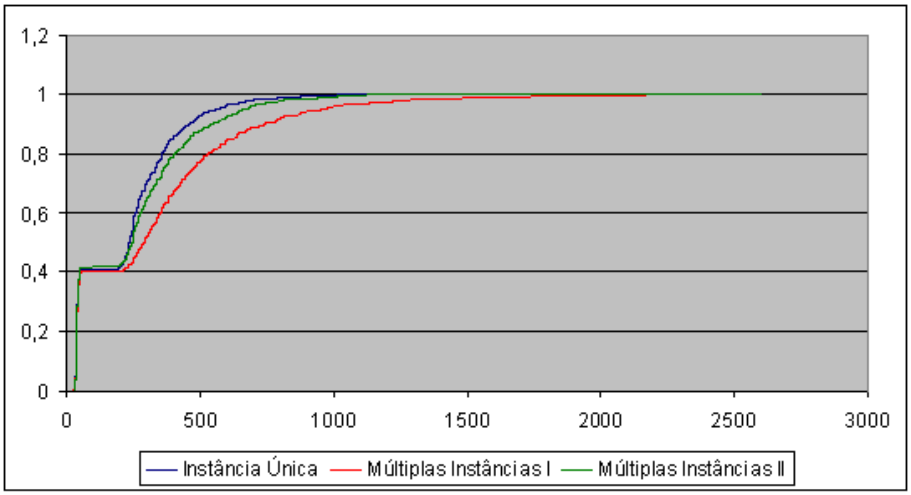

Figura 36: Comparação dos resultados da simulação (II)

Como podemos ver, a modificação nos recursos fez com que o gráfico do tempo total se aproximasse muito mais do gráfico do algoritmo de simulação de instância única. Ele ainda apresenta maiores concentrações em valores mais altos, mas a diferença não é mais tão grande quanto anteriormente. O pior caso obtido nesta nova simulação foi de 1553 dias, que está muito mais próximo dos 1400 dias da simulação de instância única do que os 2600 dias da primeira simulação de instâncias concorrentes. Este resultado ainda pode ser melhorado, através da contratação de novos recursos. No entanto, mais contratações podem aumentar o custo, e não produzir diminuições muito grandes de tempo. Pode ser também que o número de recursos adicionados nesta nova simulação seja mais que o necessário para produzir essa variação no tempo total. Para verificar isso seria necessário diminuir novamente os recursos e executar uma nova simulação. Abaixo se encontra um quadro semelhante ao mostrado anteriormente, com os tempos de espera em fila obtidos na nova simulação:

Nome da Tarefa	Tempo Espera Fila Anterior	Novo Tempo Espera Fila
03) Definir Escopo de Trabalho	0,42	0,14
09) Elicitar Requisitos	2,49	0,23
12) Elaborar Plano de Projeto	4,42	0,15
11) Elaborar Plano GCS	1,07	0,24
10) Elaborar Plano GQS	0,44	0,31
13) Consolidar Plano de Projeto	0,12	0
15) Estabelecer Arquitetura e Tecnologia de Software	0,63	0,1
16) Especificar Requisitos	7,17	0,64
20) Elaborar Protótipo	4,57	0,37
19) Analisar Desvios	0,32	0,06
18) Realizar Estimativa de esforço e prazo por pacote	0,73	0,2
22) Elaborar Modelo Lógico	5,23	0,33
23) Elaborar Modelo Físico	3,13	0,22
29) Elaborar Casos de Teste	0,51	0,51
26) Elaborar Plano de Teste	1,46	0,79
27) Codificar Pacote	9,32	5,84
31) Executar Testes de Software	1,27	0,25
30a) Consertar erros teste unitário	0,09	0,06
30b) Consertar Erros Teste de Software	0,03	0,02
30c) Consertar Erros de Integração	0,32	0,18
30d) Consertar Erros Homologação	0,16	0,14
28) Realizar Testes de Unidade	4,62	0,11
32) Integrar Pacotes	4,41	0,16
33) Executar Teste de Integração	1,5	0,41
34) Homologar Pacote	0,82	0,13
35) Participar Treinamento	1,35	0,25
36) Implantar Pacote	6,11	1,01
25) Elaborar Plano de Implantação	0,43	0,09
24) Preparar Documentação e Treinamento dos Usuários	0,61	0,15
37) Elaborar Relatório de Fechamento do Projeto	0,82	0,11

Tabela 22: Comparação dos tempos de espera em fila na primeira e segunda execução

Podemos ver pelo quadro anterior, que houve uma redução significativa do tempo médio de espera em praticamente todas as atividades, mostrando que as modificações foram bastante efetivas.

5.2.3 - Análise de aumento de demanda

Uma situação muito comum na análise de processo é a verificação da sua performance quando há um aumento na demanda. Os processos muitas vezes estão configurados para atender a determinada demanda, e quando ela aumenta, surgem atrasos devido à insuficiência de recursos.

As simulações feitas até agora supunham que a distribuição do tempo entre a chegada de duas requisições consecutivas à fábrica assumia a forma de uma Triangular de parâmetros 30, 80 e 150 dias. Vamos supor que, devido ao aquecimento da economia, a fábrica passe a sofrer requisições mais freqüentes. Por causa disso a distribuição do tempo entre as

requisições passa a ser uma triangular de parâmetros 10, 30 e 90 dias. Para essa simulação, manteremos a modificação feita ao processo na secção anterior. No gráfico abaixo se encontram a distribuição do tempo total da simulação de instância única, da simulação de instâncias concorrentes na situação anterior de demanda e situação atual:

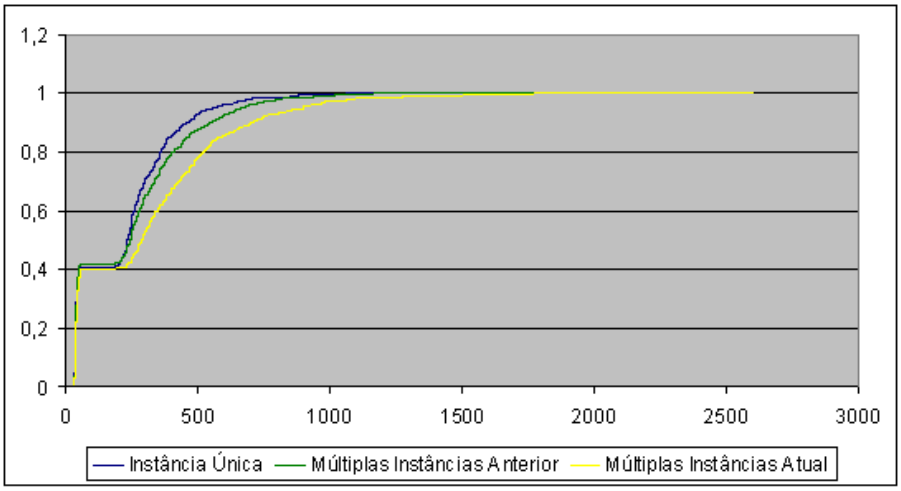

Figura 37: Comparação dos resultados da simulação (III)

Podemos ver no gráfico que o aumento da demanda faz com que o tempo total volte a apresentar valores maiores devido ao conflito de recursos. A linha amarela representa a simulação executada na nova situação de demanda, e é possível observar a sua diferença em relação à linha verde. Uma análise semelhante à feita na seção anterior pode ser feita para identificar quais são os recursos que estão em falta na nova simulação.

5.2.4 - Outras análises possíveis

Além das análises feitas aqui, inúmeras outras podem ser realizadas utilizando o SimProcess. Abaixo se encontram exemplos de algumas análises possíveis que não foram realizadas aqui:

- Comparação do tempo total em simulações com recursos e sem recursos
- Comparação do tempo total utilizando diferentes heurísticas de escalonamento
- Análise da performance em situação de redução de demanda
- Verificação de atividades que podem ser agilizadas se utilizados mais recursos
- Verificação de melhorias no processo através da modificação de suas atividades

6 - Conclusão

6.1 - Conclusões

Este trabalho apresentou um método para simulação de redes de atividades não deterministas com recursos escassos, através da proposição de um modelo para a representação deste tipo de rede de atividades, e criação de algoritmos de simulação que permitem a obtenção de resultados estatísticos a partir da rede.

Analisando os modelos de atividades existentes identificamos a necessidade de um modelo quantitativo que representasse transições e durações não determinista e restrições de recursos. Nenhum dos modelos é capaz de representar todas essas características.

Por este motivo, propusemos um novo modelo de redes de atividades chamado RANDRE. Este modelo pode ser considerado uma extensão do modelo de atividades conhecido como RCPS Estocástico através da adição de transições não determinista. O grafo de atividades passa a permitir a existência de "loops", o que não era possível na maior parte dos modelos predecessores. Uma RANDRE bem formada é definida de maneira recursiva, de modo que cada sub-rede possui sempre um nó inicial e um nó final. Esta regra de formação permite a definição de regras de transformação de redes descritas na forma do diagrama de atividades da UML em redes RANDRE. Estes diagramas, no entanto, devem seguir algumas regras de formação a fim de evitar a existência de redes de atividades patológicas. Esta regra de transformação faz com que a RANDRE se torne um método eficiente para a representação quantitativa de diagramas de atividades UML.

A partir da definição do modelo, foi possível criar os algoritmos necessários para a simulação das redes de atividades. Devido à natureza não determinista e a alta complexidade das redes expressas através do modelo, a maneira mais eficiente encontrada para obter os resultados foi a utilização do método de simulação de Monte Carlo. Através deste método os diversos cenários de execução possíveis à partir da rede são amostrados, e cada um deles é escalonado de acordo com determinadas regras de escalonamento. Os resultados obtidos podem ser armazenados em histogramas utilizados principalmente para o armazenamento do tempo total de projeto. Neste trabalho apresentamos os algoritmos necessários para a geração de um cenário à partir da RANDRE, escalonamento deste cenário obtendo tempos de início e fim de cada atividade e os algoritmos utilizados para fazer a repetição destes 2 passos.

Para testar a validade deste mecanismo foi produzido um software que implementa todos os conceitos apresentados aqui. Este programa recebeu o nome de SimProcess. Utilizando esta ferramenta foi possível executar um estudo de caso. Utilizamos neste estudo a rede de atividades produzida por um dos grupos que participaram da disciplina de Engenharia de Software. Nesta rede é representado o processo para geração de programas em uma fábrica de software fictícia. Apesar de fictício, o modelo é bastante complexo e foi construído tendo por objetivo se aproximar ao máximo do que ocorre na realidade. Utilizando a ferramenta

algumas análises puderam ser feitas sobre esta rede, permitindo identificar alguns pontos de melhoria e estudar o comportamento da rede em situações de modificação no ambiente global.

6.2 – Trabalhos Futuros

Devido à complexidade inerente à proposição do método de simulação, muitas questões não puderam ser respondidas através deste trabalho. Alguns destes problemas ficam aqui como sugestões para futuros trabalhos e análises mais profundas.

O primeiro deles diz respeito à verificação da validade do método. Infelizmente apenas um estudo de caso pode ser mostrado neste trabalho, e este estudo de caso não representa uma aplicação real, apesar de possuir muitas das características presentes em aplicações reais. Durante o desenvolvimento da ferramenta muitos testes foram feitos com diversas redes de atividades diferentes. No entanto, não foi possível incluir neste trabalho muitos destes experimentos. São necessários, portanto, estudos mais profundos que tenham como objetivo verificar a utilidade e a validade das simulações que podem ser feitas utilizando as redes RANDRE.

Vimos no texto que durante a simulação alguns dados são coletados sobre a execução do projeto, como tempo inicial, final, probabilidade de cada atividade estar no caminho crítico. As variáveis escolhidas foram selecionadas com base no que achávamos que fosse necessário para avaliar o projeto, ou o que ferramentas similares geravam. Um trabalho que apresente uma discussão sobre as principais variáveis de controle e que apresente novas propostas de variáveis é extremamente necessário para melhorar a qualidade dos resultados obtidos à partir das simulações.

No capítulo do estudo de caso mostramos alguns passos básicos para se fazer uma simulação de um projeto/processo utilizando o SimProcess. A ferramenta disponibiliza uma quantidade grande de informações, por isso muitas vezes é difícil descobrir onde se encontram os pontos de gargalo onde melhorias podem ser realizadas no processo. Seria importante que um trabalho estudasse em profundidade a metodologia necessária para se fazer uma análise de um processo utilizando a ferramenta, de modo a facilitar o trabalho do usuário final. Este trabalho poderia ter como resultado final um método automatizado para avaliação de processo, permitindo identificar automaticamente atividades críticas, sem necessitar de análises mais profundas por parte do usuário.

Uma questão muito importante quando se avalia durações em cadeias de atividades é a correlação entre as variáveis aleatórias. Em nosso modelo assumimos que não existe correlação entre as variáveis aleatórias que representam a duração das atividades, ou seja, assumimos que elas são independentes. No entanto, no mundo real essa regra nem sempre se aplica. Por exemplo, se forem gastos x dias para desenvolver um programa, é necessário que sejam gastos um número proporcional a x dias para testá-lo. Um trabalho que estude como adicionar essas correlações ao modelo, e como implementá-las pode tornar as análises muito mais precisas.

Neste trabalho utilizamos como função objetivo apenas o tempo total do projeto, mas existem outras funções de grande interesse na análise de projetos. Uma variável importante é o custo. Em muitos projetos o custo pode ser considerado proporcional ao tempo necessário para executá-lo, mas em muitos casos esta regra não se aplica. Seria importante que cada atividade estivesse associada a uma previsão de custo, e que simulações pudessem ser feitas de modo a se obter o menor custo possível.

Atrelada à questão da inclusão de um modelo de custo está a questão da múltipla modalidade das atividades. Esta questão vêm sendo muito estudada em modelos deterministas sob a designação de atividades multi-modais. Isto significa que uma atividade pode possuir múltiplos modos de execução tendo tempo e custo diferentes entre os diversos modos. Por exemplo, ao construir uma casa pode-se fazê-la em 1 ano usando 1 pedreiro ou em 8 meses utilizando dois. É óbvio que o custo de construir a casa utilizando 2 pedreiros é diferente do custo utilizando 1. Neste paradigma as análises ficam muito mais complicadas porque é necessário levar em consideração os diversos modos de execução durante a simulação. Os modos de execução influenciam também na utilização de recursos pela rede de atividades. Novas funções objetivo passam a ser também possíveis.

Finalmente existe a questão da qualidade dos recursos. No modelo RANDRE os recursos são considerados homogêneos. Isto significa, por exemplo, que todos os programadores em uma fábrica de software são considerados iguais e levam o mesmo tempo para realizar uma tarefa. Muitas vezes isso não é verdade na realidade. Utilizando um programador melhor em determinada tarefa seria possível reduzir o tempo necessário para realizar esta tarefa. Se este programador possui um salário diferenciado isto tem um impacto também na função de custo do projeto. Os recursos podem ser ainda representados de maneira hierárquica. Neste caso uma mesma tarefa poderia ser executada por dois tipos de recursos diferentes. Por exemplo, a tarefa de programação pode ser executada tanto por um programador quanto por um analista de sistemas, porque ambos atendem aos requisitos necessários para executar a tarefa que é saber programar. É óbvio que os tempos de execução e custo sofrem o impacto da escolha do recurso. É necessário um estudo que verifique a necessidade e a viabilidade de incluir este tipo de informação no modelo.

Referências Bibliográficas

[1] NEUMANN K., SCHWINDT C. – "Projects with minimal and maximal time lags: construction of activity-on-node networks and applications", Universitat Karlsruhe, Relatório Técnico, wior-447, 36p, 1995.

[2] LENSTRA J., RINNOOY K. – "Computational complexity of discrete optimization problems", Annals of Discrete Mathematics, 1979.

[3] LARSON, H. - "Introduction to Probability Theory and Statistical Inference" – 3rd Edition - Wiley Series in Probability and Mathematical Statistics, 1982

[4] MOORE L., CLAYTON E. – "GERT Modeling and Simulation – Fundamentals and Applications" – Petrocelli / Charter, 1976

[5] ELMAGHRABY S. E. – Activity Networks John Wiley & sons, 1977.

[6] WIEST J., LEVY L. – "A Management Guide to PERT/CPM" – Prentice-Hall, 1969.

[7] ARAÚJO R., COUTO R. – "Heurísticas para Escalonamento de Atividades com Durações Não Deterministas em Projetos com Recursos Limitados" – Projeto Final de Curso – Informática UFRJ, 1999.

[8] YANG B., GEUNES J., O'BRIEN W. – "Resource Constrained Project Scheduling: Past Work and New Directions" – *Research Report 2001-6*, Department of Industrial and Systems Engineering, University of Florida – 2001.

[9] DREZNER S., PRITSKER A. – "Network Analysis of a Count-Down" – The Rand corporation - 1966

[10] PRITSKER A. – "GERT: Graphical Evaluation and Review Technique" – The Rand Corporation - 1966

[11] "UML Press Releases – UML Primer" – Object Management Group – http://www.omg.org/news/pr97/umlprimer.html, 1997.

[12] ESHUIS R., WIERINGA R. – "A Formal Semantics for UML Activity Diagrams - Formalising Workflow Models" - CTIT Technical Report 01-04; University of Twente, February 2001.

[13] BADIRU, A. – "Quantitative Models for Project Plan, Scheduling and Control", 1993

[14] CARRUTHERS J., BATTERSBY A. – "Advances in critical path methods", Operations Research Quarterly, 17(4): 359-380, 1966.

[15] PATTERSON J., HUBER W. – "A horizon-varying, zero-one approach to project scheduling", Management Science, 20: 990-998, 1974.

[16] DEMEULEMEESTER E., HERROELEN W., ELMAGHRABY S. – "Optimal procedures for the discrete time/cost trade-off problem in project networks", European Journal of Operational Research, 88: 50-68, 1996.

[17] SIMPSON W., PATTERSON J. – "A multiple-tree search procedure for the resource-constrained project scheduling problem", European Journal of Operational Research, 1996.

[18] LOURENÇO F., SCHIMTZ E. – "Uma abordagem Branch and Bound para o RCPSP em um ambiente de computação colaborativa" – Dissertação de Mestrado em andamento, 2005

[19] BLACK P. – National Institute of Standards and Technology (NIST), www.nist.gov.

[20] KOLISH R., HARTMANN S. – "Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis", Advances in Project Scheduling, Elsevier, pp 113-134, 1996.

[21] BAAR T., BRUCKER P., KNUST S. – "Tabu-search algorithms for the resource-constrained project scheduling problem", Technical Report, Osnabrücker Schriften zue Mathematik, Fachbereich Mathematik/Informatik, Universität Osnabrücker, 1997.

[22] BOULEIMEN K., LECOCQ H. – "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem", Technical Report, Service de Robotique et Automatisation, Université de Liège, 1998.

[23] EGLESE R. – "Simulated annealing: A tool for operational research", European Journal of Operational Research, 46: 271-281, 1990.

[24] HARTMANN S. – "A competitive genetic algorithm for resource-constrained project scheduling", Naval Research Logistics, 45: 733-750, 1998.

[25] MERKLE D., MIDDENDORF M., SCHMECK H. – "Ant colony optimization for resource-constrained project scheduling", IEEE Transactions on Evolutionary Computation, 6:333-346, 2002.

[26] KOLISCH R., HEMPEL K. – "Finite scheduling capabilities of commercial project management systems", manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, Alemanha, no. 397, 15p, 1996.

[27] METROPOLIS, N. "The Beginning of the Monte Carlo Method." *Los Alamos Science,* No. 15, p. 125, 1987. http://jackman.stanford.edu/mcmc/metropolis1.pdf.

[28] ECKHARDT, Roger. "Stan Ulam, John von Neumann, and the Monte Carlo method", *Los Alamos Science*, Special Issue (15), 131-137, 1987.

[29] ARAÚJO R., COUTO R., SCHIMTZ E. - "Sim Project: Uma Ferramenta para Análise de Risco do Tempo de Realização de Projetos" - XIII Simpósio Brasileiro de Engenharia de Software Caderno de Ferramentas, 25-28, 1999

[30] ESHUIS R., WIERINGA R. – "A Comparison of Petri Net and Activity Diagram Variants" - In: Proceedings 2nd International Colloquium on Petri Net Technologies for Modeling Communication Based Systems, Berlin, Germany, 2001

[31] BRAVETTI M. – "Revisiting Interactive Markov Chains" - in Proc. of the 3rd Workshop on Models for Time-Critical Systems (MTCS 2002) , ENTCS 68(5), Brno (Czech Republic), 2002.

[32] OLAGUÍBEL R. GOERLICH, J. - "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis" – Advances in Project Scheduling – Edited by Roman Slowinski e Jan Weglarz – pg. 113-134, 1989.

[33] LAWRENCE, S.R. - "An Experimental Investigation of Heuristic Scheduling Techniques" - GSIA, Carnegie-Mellon University, Pittsburgh, 1984.

[34] BLAZEWICZ J., LENSTRA J., RINNOOY A. - "Scheduling Subject to Resource Constraints: Classification and Complexity" - Discrete Applied Mathematics 5, pg. 11-24, 1983.

[35] Ver [32]

[36] RUSSEL S., NORVIG P. "Artificial Intelligence: A Modern Approach" – Prentice Hall, 1995.

[37] DUNCAN W. "A Guide to the Project Management Body of Knowledge" - Project Management Institute (PMI) – Standards Committee - 1996.

[38] GRAHAM R., LAWLER E., LENSTRA J., RINNOOY K. – "Optimization and approximation in deterministic sequencing and scheduling theory: a survey", Annals of Discrete Mathematics, 1979.

[39] CONTIGENCY ANALYSIS CONSULTING – "Risk Glossary – Monte Carlo Method" - http://www.riskglossary.com/articles/monte_carlo_method.htm, 1996.

[40] PRESS W., TEUKOLSKY S., VETTERLING W., FLANNERY B. - Numerical Recipes in C - The Art of Scientific Computing – Second Edition – Cambridge University Press.

[41] PALISADE CORPORATION - "Guide to Using @Risk: Risk Analysis and Simulation Add-In for Microsoft Excel or Lotus 1-2-3".

[42] MICROSOFT CORPORATION – "Microsoft .NET Framework Version 1.1 Redistributable Package" - http://www.microsoft.com/downloads/details.aspx?FamilyID=262d25e3-f589-4842-8157-034d1e7cf3a3&displaylang=en

[43] SimProcess - http://equipe.nce.ufrj.br/eber/simprocess/

[44] OBJECT MANAGEMENT GROUP – "Unified Modeling Language (UML) 1.3 specification" –http://www.omg.org/cgi-bin/doc?formal/00-03-01.

[45] OBJECT MANAGEMENT GROUP – “XML Metadata Interchange (XMI) version 1.1 specification” – http://www.omg.org/cgi-bin/doc?formal/00-11-02

[46] SPARX SYSTEMS – “Enterprise Architect” – www.sparxsistems.com

[47] HOEL, P. – Introduction to Mathematical Statistics, John Wiley and sons, Third Edition, 1965.

Anexo A - Implementação do SimProcess

Neste anexo serão abordados alguns aspectos importantes sobre a implementação do programa, analisando o funcionamento dos seus componentes principais e mostrando algumas das decisões tomadas durante a sua codificação.

O SimProcess foi desenvolvido utilizando a plataforma .Net da Microsoft. O .Net é uma infraestrutura para criação de programas que conta com uma grande quantidade de bibliotecas de rotinas integradas à plataforma. Ela é composta por linguagens de alto nível (nas quais os programas são escritos) e compiladores que transformam o código em uma linguagem intermediária chamada CLR (Common Language Runtime). O .Net possui ainda uma máquina virtual que executa o "assembly" contendo código CLR.

A linguagem de alto nível utilizada para desenvolver o SimProcess é o C#. O C# possui uma sintaxe muito similar ao das linguagens C++ e Java, sendo orientada a objeto e possuindo gerenciamento automático de memória. Estas características fazem do C# uma linguagem ideal para o desenvolvimento do SimProcess, visto que permitem criar um código claro e eficiente. Mas para que estas vantagens possam ser realmente exploradas é necessário organizar o código de maneira correta.

A.1. Organização do código

A implementação do SimProcess foi feita buscando utilizar ao máximo os recursos da orientação a objeto, com o objetivo de tornar o código o mais claro e limpo possível. Tendo este objetivo em mente, as classes foram distribuídas em pacotes, agrupando aquelas que possuem finalidades similares. Os principais pacotes são mostrados no diagrama abaixo:

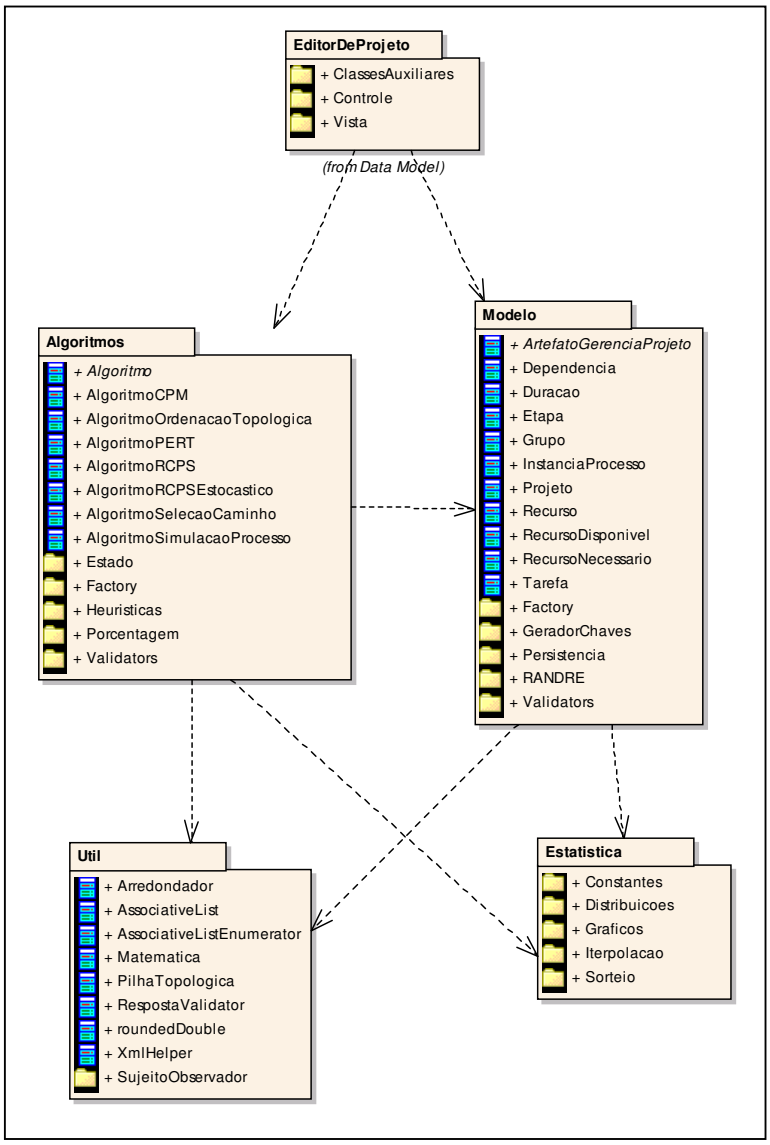

Figura 38: Pacotes de classes do SimProcess

Na figura 38 podemos ver 5 pacotes. O primeiro, EditorDeProjeto, contém as classes utilizadas para criar a interface gráfica com o usuário do programa. As classes deste pacote utilizam as classes do pacote Modelo para armazenar os dados referente ao projeto, assim como para fazer a sua persistência em arquivo. Utiliza também o pacote de Algoritmos, para executar os inúmeros algoritmos de simulação existentes no sistema. O pacote de Algoritmos também utiliza as classes contidas em Modelo, para obter informações referentes ao processo que está sendo simulado. Existe ainda o pacote de Estatística, que contém as classes que representam variáveis aleatórias, utilizadas em Modelo para representar as durações de atividades, e em Algoritmos para sortear valores e armazenar histogramas. Por último está o pacote Útil, que contém classes utilitárias usadas para armazenar listas, validação, arredondamento e etc.

Dentre estes pacotes os mais importantes são os pacotes Algoritmo e Modelo porque eles implementam todas as funcionalidades necessárias para executar as simulações.

A.2. Modelo

No pacote Modelo existem algumas classes que permitem representar a estrutura das redes RANDRE. Estas são as estruturas utilizadas nos algoritmos de simulação, e são importantíssimas para o sistema.

Na raiz do pacote Modelo, estão as principais classes deste pacote. São elas as responsáveis por representar as redes RANDRE. Abaixo se encontra um diagrama de classes que mostra com detalhe os relacionamentos entre estas classes:

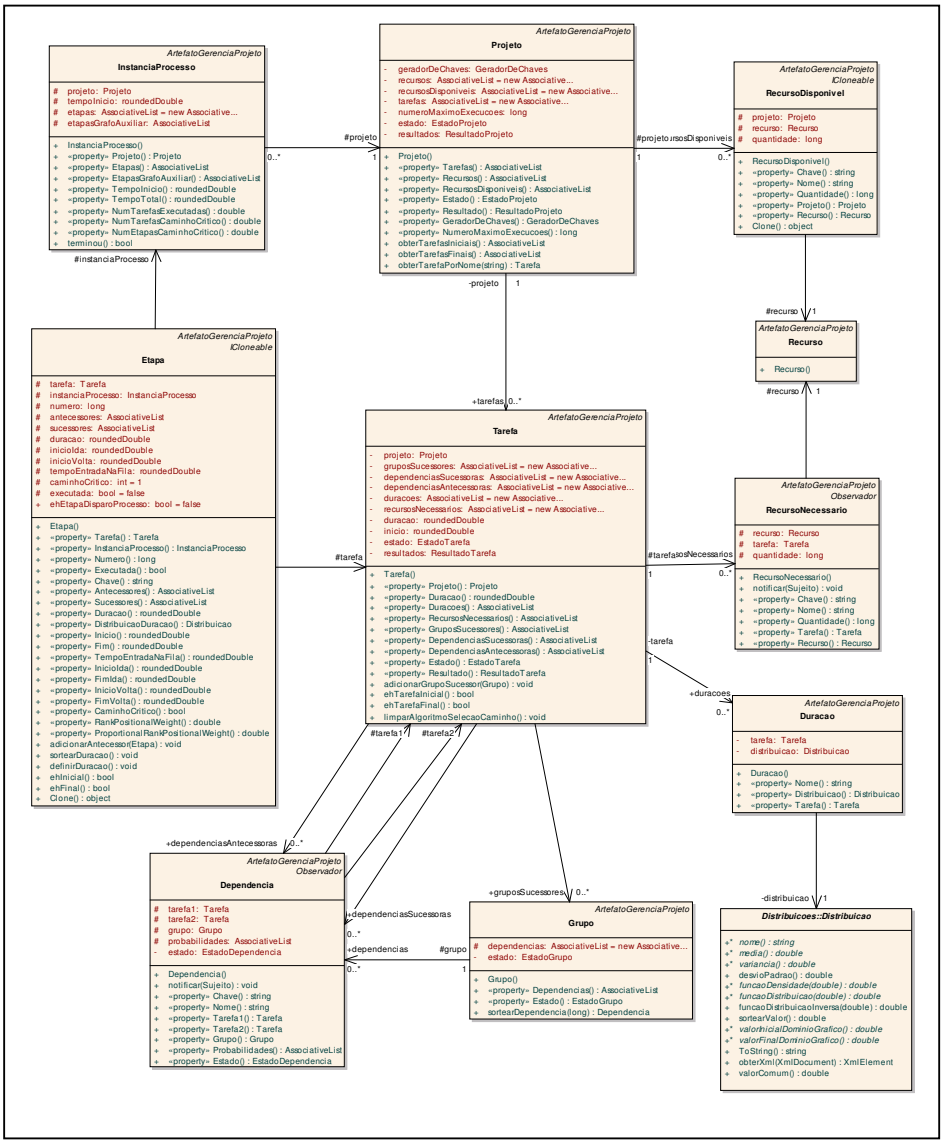

Figura 39: Diagrama de classes do pacote Modelo

Todas as classes contidas neste diagrama herdam da classe ArtefatoGerenciaProjeto. Esta classe não está sendo mostrada no diagrama porque tornaria mais difícil entender o que está representado nele. Esta classe contém dois atributos: chave e nome. A chave é um identificador único utilizado para identificar o objeto. Todas as listas utilizadas no sistema são indexadas por esta chave, permitindo que um objeto possa ser obtido à partir de sua chave em um tempo relativamente pequeno. Isto representa um ganho significativo em performance, já que muitas das operações executadas nos algoritmos são deste tipo.

A classe principal neste diagrama é a classe Projeto, localizada no topo, ao centro da figura. Ela representa um modelo de atividades, ou seja, uma rede RANDRE. Todas as atividades contidas nesta rede estão contidas no projeto assim como os recursos. As atividades são representadas no modelo pela classe Tarefa. Um projeto pode estar associado a n tarefas, e cada tarefa pertence a um único projeto. O projeto possui ainda um conjunto de objetos do

tipo Recurso, que representam os tipos de recursos existentes na rede. Além de possuir uma lista contendo os tipos de recursos, o projeto possui ainda uma lista de objetos que representam as disponibilidades de cada tipo de recurso. A classe que representa a disponibilidade é a classe RecursoDisponivel. Esta classe possui como atributos a referência para o Recurso, identificando o seu tipo, e um inteiro representando a quantidade disponível daquele recurso.

A classe Tarefa tem um papel crucial no modelo porque representa a atividade, e possui várias associações com outras classes representando a topologia da rede no modelo. Nós vimos que nas redes RANDRE cada atividade pode estar associada a um conjunto de recursos, que indicam quantas unidades de determinado recurso precisam estar disponíveis para que ela possa executar. Esta necessidade de recursos é representada no modelo pela associação entre Tarefa e a classe RecursoNecessario. Cada instância da classe RecursoNecessario está associada a uma instância de Recurso, e possui um atributo inteiro que indica quantas unidades daquele recurso são necessárias para que a atividade possa ser disparada. Uma instância de Tarefa pode possuir n RecursoNecessarios associados.

Cada atividade pode estar associada ainda a uma lista de durações, que representam o tempo necessário para executar a tarefa em cada vez que ela é executada. A duração é representada pela classe Duracao no modelo. Cada instância de Tarefa pode possuir de 0 a n instâncias de Duracao. Esta, por sua vez, está associada sempre a um objeto do tipo Distribuicao. Esta classe se encontra no pacote Estatistica/Distribuicoes, e é a super classe da qual herdam todas as classes que representam variáveis aleatórias. As classes implementadas atualmente que herdam de Distribuicao são: DistribuicaoValorFixo, DistribuicaoUniforme, DistribuicaoTriangular e DistribuicaoNormal.

Os objetos do tipo Tarefa estão associados ainda a 2 outras classes que servem para representar a estrutura da rede. A primeira das classes é chamada Grupo. Esta classe funciona aproximadamente da mesma maneira que a decisão, conforme descrito no modelo da rede, agrupando as arestas mutuamente exclusivas que saem de uma atividade. A existência desta classe em vez de uma classe que representasse a decisão, se deve ao modo como representávamos a rede quando o programa começou a ser implementado. Em vez de utilizarmos nós especiais no grafo (decisões) cujas arestas de saída eram mutuamente exclusivas, considerávamos que as arestas de saída de um grafo poderiam ser agrupadas, de modo que de cada grupo de saída apenas uma aresta era escolhida. A transição de uma atividade para outra poderia ser considerada, do mesmo modo, um caso especial de um agrupamento com uma única aresta. Os dois modos de representar o grafo são equivalentes podendo ser usados sem que haja qualquer perda de informação contida no grafo. Nós veremos que o SimProcess utiliza a notação antiga quando mostra a estrutura da RANDRE. Abaixo pode ser vista a diferença entre as duas representações:

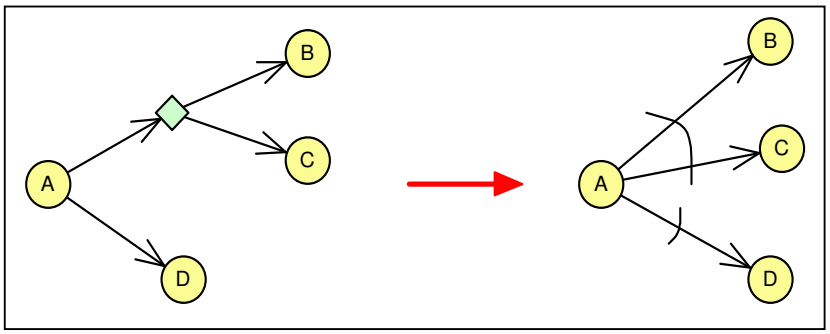

Figura 40: Modos de representação do grafo de dependência

Na figura podemos ver duas redes iguais expressas de duas formas diferentes. A primeira forma já conhecemos. Na segunda, as arestas exclusivas que ligam A a B, e A a C são marcadas com um traço que indica que se encontram em um mesmo grupo. A aresta que liga A a D também está em um grupo no qual é a única dependência. Neste caso a marcação de grupo pode ser omitida, mas foi colocada para mostrar o que acontece quando este grafo é representado usando o modelo de classes. No caso a Tarefa A possuirá dois grupos sucessores, um contendo as arestas B e C e o outro apenas a aresta que leva a D.

A classe Grupo possui, portanto, um conjunto de objetos do tipo Dependencia. A classe Dependencia representa um arco do grafo. Ela possui como atributos duas referências para objetos do tipo Tarefa. A primeira contém a atividade de origem do arco, e a segunda contém a atividade de destino. Cada dependência possui uma lista chamada probabilidades, que contêm probabilidades daquela dependência ser escolhida dentro do grupo. Cada probabilidade equivale a uma passagem por aquele arco. A soma das probabilidades de cada dependência em determinada passagem por um grupo deve sempre ser igual a 100%.

Para facilitar a execução de alguns algoritmos as Tarefas estão associadas as suas Dependências também diretamente, além de estarem ligadas pelos Grupos sucessores. Cada instância de Tarefa possui duas listas de dependências. A primeira contém as dependências cujo destino é aquela tarefa. Esta lista é chamada de dependenciasAntecessoras. Do mesmo modo, a lista dependenciasSucessoras contém todas as dependências que saem de determinada atividade.

Os últimos dois objetos existem apenas no momento da execução de algum dos algoritmos de simulação. O primeiro deles, InstanciaProcesso, representa um determinado cenário de um projeto obtido através da execução do algoritmo de seleção de um caminho. Ele está diretamente associado ao Projeto que o deu origem e possui uma lista contendo instâncias de objetos do tipo Etapa. A etapa é a instância de uma tarefa após a execução do algoritmo de seleção do caminho. A Etapa está associada diretamente à Tarefa que deu origem a ela, e tem como um de seus atributos a sua duração. Ela possui ainda dois auto-relacionamentos que permitem representar o grafo de saída obtido após a execução do algoritmo. O primeiro deles se chama antecessores, e contém todas as etapas que precisam ser executadas antes que esta etapa possa executar. O segundo contém as etapas que

passam a poder ser executadas depois que ela termina. Esta lista se chama sucessores. A etapa possui ainda uma série de atributos utilizadas durante a execução dos algoritmos.

Além destas classes principais existe ainda uma série de classes utilitárias contidas em pacotes dentro do pacote Modelo. Eles podem ser vistos na Figura 38. O pacote Factory contém classes responsáveis por construir o modelo de classes apresentado acima à partir dos diferentes tipos de arquivo que a aplicação permite importar. Existem implementações disponíveis para ler o modelo do formato de armazenamento interno da ferramenta (xml), arquivos contendo diagramas de atividades (xmi) e arquivos no formato dos benchmarks de redes de atividade RCPS (txt). O segundo pacote, GeradorChaves, contém a classe responsável por gerar a chave única que identifica cada objeto. Em Persistencia se encontram os objetos responsáveis por armazenar uma rede alterada no programa no formato padrão de armazenamento. Em RANDRE se encontram as classes responsáveis por representar o projeto como uma hierarquia de RANDREs básicas, conforme descrito nos algoritmos para seleção de um cenário. No pacote Validators existem classes capazes de verificar a validade do diagrama de acordo com algumas regras, como, por exemplo, verificar se a soma das probabilidades em um grupo é igual a 100%.

A.3. Algoritmos

O pacote Algoritmos contém as classes que fornecem a implementação dos algoritmos apresentados no capítulo anterior. Ele contém um conjunto de classes principais, que implementam estes algoritmos, e um conjunto de classes auxiliares utilizadas na execução deles. No diagrama abaixo estão representadas as classes principais deste pacote:

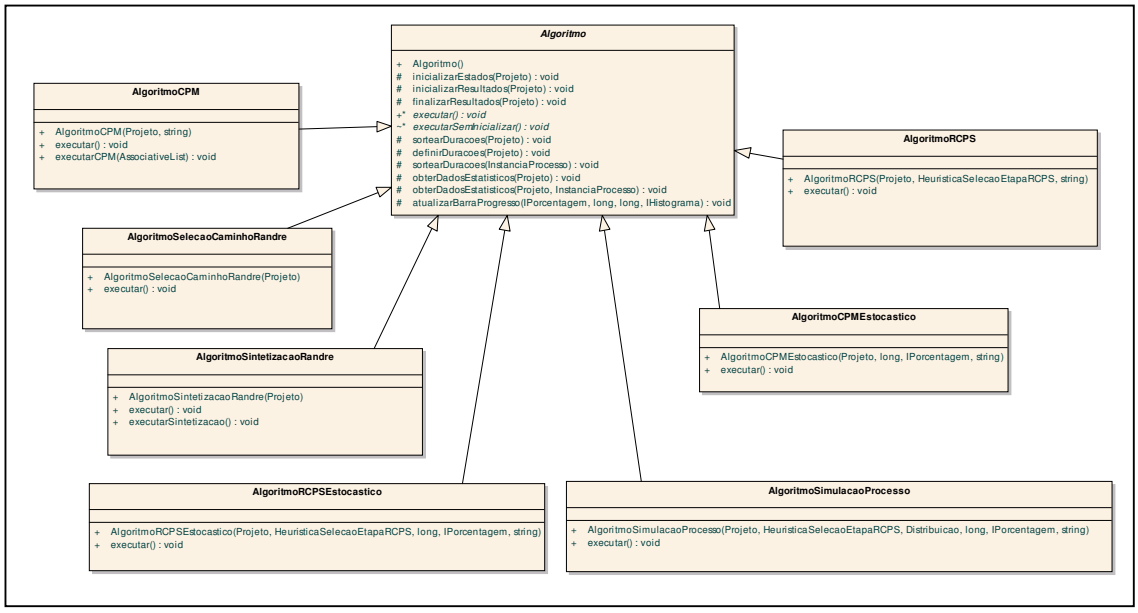

Figura 41: Diagrama de classes do pacote de algoritmos

Como podemos ver pelo diagrama, todas as classes que implementam algoritmos de escalonamento herdam de uma mesma classe. Esta classe se chama Algoritmo. Ela contém funções utilitárias usadas em grande parte dos algoritmos, como o método para obter os dados estatísticos ao final da execução ou sortear as durações de cada uma das etapas. Além disso, ela possui um método abstrato chamado executar, que deve ser sobrescrito nas classes filhas com a implementação do algoritmo que elas representam.

Cada uma das outras classes existentes no diagrama implementam um dos algoritmos apresentados anteriormente. As classes implementam os seguintes algoritmos:

1. AlgoritmoSintetizacaoRandre – Algoritmo para hierarquização da Randre;
2. AlgoritmoSelecaoCaminhoRandre – Algoritmo para seleção de um cenário;
3. AlgoritmoCPM – Algoritmo para escalonamento de 1 cenário sem recurso;
4. AlgoritmoRCPS – Algoritmo para escalonamento de 1 cenário com recurso;
5. AlgoritmoRCPSEstocastico – Algoritmo para simulação de instância única;
6. AlgoritmoSimulacaoProcesso – Algoritmo para simulação de múltiplas instâncias;

Anexo B – Utilização do SimProcess

Este anexo tem por objetivo mostrar os passos necessários para a simulação de uma rede de atividades no *SimProcess*. Serão mostradas todas as tarefas que devem ser executadas, desde a criação do diagrama de atividades da UML, até a obtenção dos resultados estatísticos da simulação. No diagrama de atividades contido na Figura 42 podemos ver todas as tarefas que precisam ser executadas para realizar uma simulação:

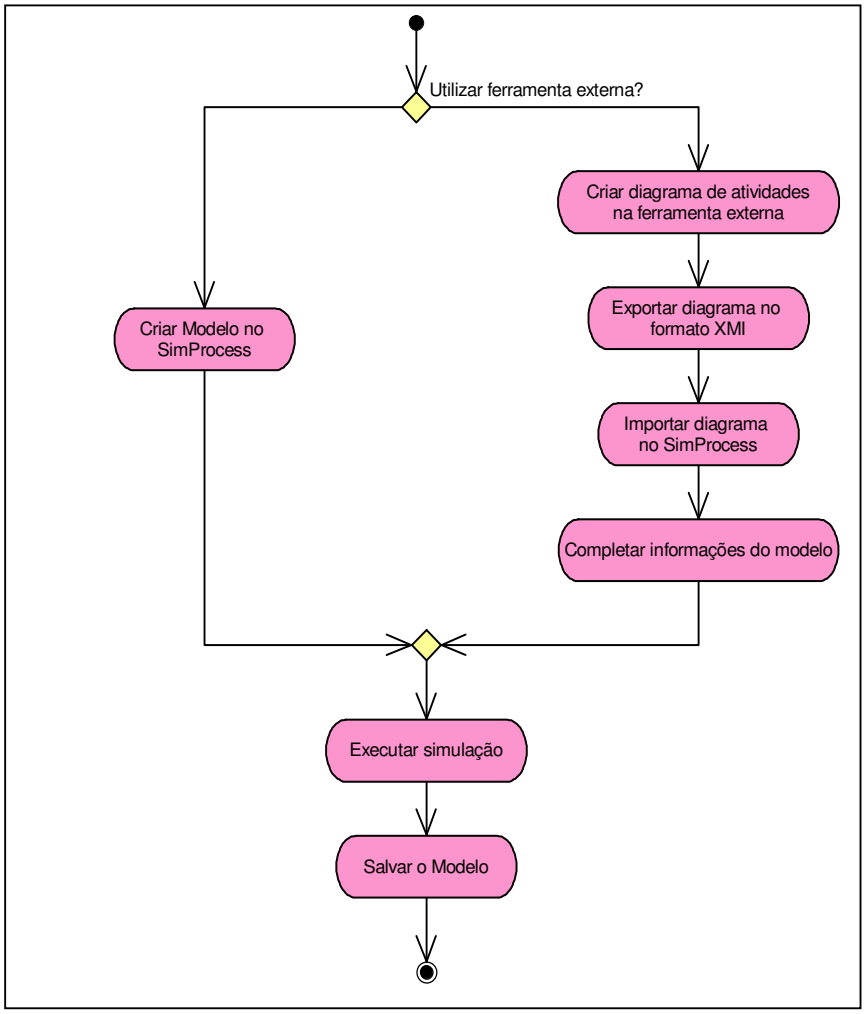

Figura 42: Diagrama de atividades do processo de simulação

Como podemos ver existem dois ramos principais no diagrama, dependendo do modo como o modelo é criado. No ramo da direita uma ferramenta externa é utilizada para criar o modelo, e no ramo da esquerda ele é criado diretamente no SimProcess. Neste texto daremos mais ênfase à criação do modelo usando uma ferramenta externa, já que o SimProcess não possui um diagramador, instrumento indispensável para montar o diagrama com maior facilidade.

Nas próximas secções serão mostrados em detalhes os passos necessários para completar cada uma das atividades mostradas na Figura 42. No entanto, antes disso é necessário entender como instalar o programa.

B.1. Instalação

O *SimProcess* foi desenvolvido utilizando Microsoft Visual C# .Net, e por isso necessita que seja instalada previamente a versão 1.1 do Microsoft .Net Framework Redistributable. O Framework .Net pode ser obtido gratuitamente à partir do site da Microsoft [42]. Existe uma cópia disponível também na página do *SimProcess* em [43]. Para instalá-lo basta executar o programa e seguir as instruções. Se você já possuir o Framework .Net 1.1 instalado, não precisa executar este passo.

O executável do *SimProcess* também pode ser obtido gratuitamente, através do endereço [43]. A ferramenta é disponibilizada em um arquivo com a extensão zip. Para utilizar o programa basta descompactar o arquivo em algum diretório, e executar o programa "*SimProcess.exe"*.

B.2. Criação do Modelo

Existem duas maneiras principais de se entrar com o modelo de atividades no programa: editar a rede de atividades diretamente nele ou criá-la em uma ferramenta externa. O *SimProcess* fornece uma interface que permite a edição de processos, permitindo adicionar atividades, recursos e alterar as suas propriedades.

O modo mais simples para criar um novo processo é criar o modelo em uma ferramenta externa e importá-lo no *SimProcess*. A ferramenta é capaz de importar modelos de atividades armazenados no padrão UML 1.3 [44] XMI 1.1 [45] tendo sido testada utilizando a ferramenta Enterprise Architect (EA) da Sparx Systems. Uma versão trial desta ferremanta pode ser obtida à partir de [46]. Vamos mostrar aqui como o Enterprise Architect pode ser utilizado para gerar o modelo de atividades.

Ao abrir o Enterprise Architect você tem a opção de criar um novo modelo, e pode adicionar diagramas de atividades a ele. Após criar um diagrama, as opções no menu do lado direito podem ser usadas adicionar atividades e outros elementos.

B.3. Exportando o Diagrama no formato XMI

Depois de completo, o diagrama deve ser exportado no formato XMI para que possa ser aberto no *SimProcess*. Para isto, basta clicar com o botão direito sobre o pacote que contém o diagrama no menu do lado direito da tela, e selecionar a opção “Import/Export” -> “Export package to XML file...”, como na figura abaixo:

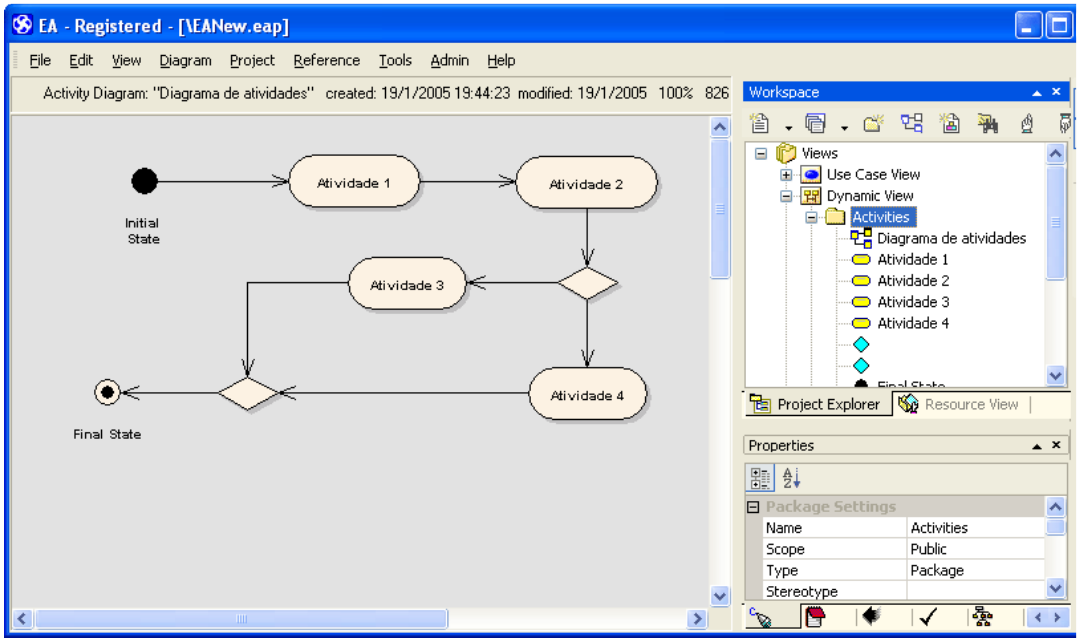

Figura 43: Exportando o diagrama de atividades da UML

Ao selecionar esta opção o EA exibe uma janela para seleção dos parâmetros para exportação. Para exportar o modelo, basta selecionar o arquivo onde o modelo será gravado na opção Filename, e clicar em Export, sem alterar qualquer das outras opções. **Importante:** Para que o *SimProcess* possa ler o arquivo, é necessário que a sua extensão seja XMI, e não XML como é o padrão do EA. Isto acontece porque a extensão xml é usada para os arquivos criados pelo *SimProcess*, sendo necessária essa diferença de extensão para que o programa possa identificar qual é qual.

B.4. Importando o XMI no SimProcess

Depois de exportado, o modelo pode ser importado no *SimProcess*. Para abrir um diagrama de atividades no *SimProcess*, basta selecionar a opção “Arquivo” - “Abrir”. O programa exibe, então, a janela padrão do sistema operacional para abertura de arquivo. Para abrir um arquivo gerado em uma ferramenta externa é necessário selecionar a opção “Diagrama de Atividades” na opção “Tipo de Arquivo”. Selecionado o arquivo, basta clicar em “Abrir” para importá-lo na ferramenta. A estrutura dos objetos contidos nos diagramas é exibida na tela principal do sistema, na árvore do lado esquerdo da tela. Para consultar ou modificar

dados associados a quaisquer destes itens, basta selecioná-lo na árvore para que a janela de edição deste item seja exibida no painel do lado direito, conforme a figura abaixo:

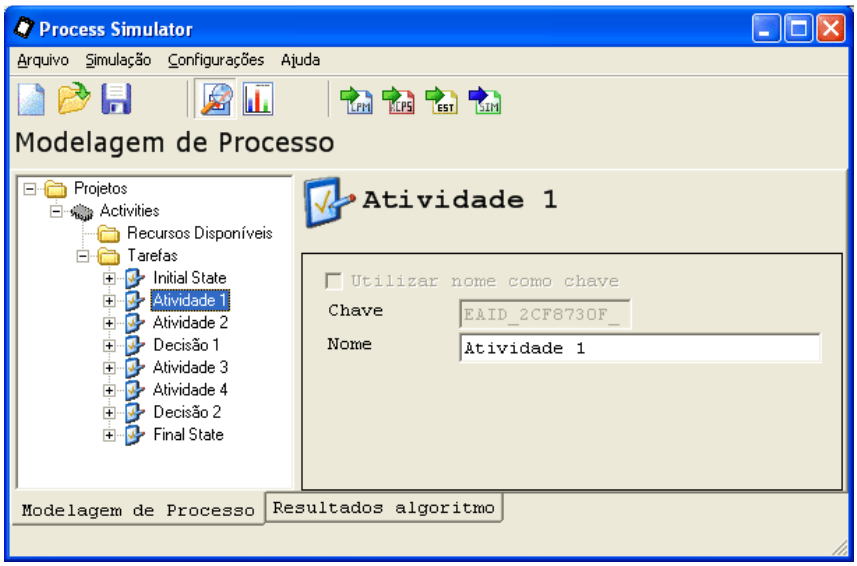

Figura 44: Diagrama de atividades importado

B.5. Completando as informações do modelo

Quando um diagrama de atividades é importado desta maneira, são importadas cada uma das suas atividades, assim como alguns objetos utilizados no diagrama, como decisões, barras de sincronismo, eventos e etc. No entanto, existem dados que são utilizados no programa, que não podem ser lidos diretamente do diagrama.

Importar o diagrama deste modo faz com que cada uma das atividades receba como duração o valor constante de 1, e os outros tipos de objetos recebem como duração o valor constante 0, ou seja, são importados como marcos. As durações das atividades precisam ser alteradas para refletir valores reais. Para alterar as durações de uma atividade, expanda o item correspondente à ela na árvore. São exibidos, então, os sub-itens “Recursos Necessários”, “Durações” e “Sucessores”. Expanda o nó “Durações”. O *SimProcess* permite que sejam fornecidas múltiplas distribuições para a duração de uma atividade. Cada distribuição é usada para cada ordem de execução da atividade. Por exemplo, na 1ª execução da atividade a 1ª distribuição é utilizada, na 2ª execução é utilizada a 2ª distribuição e assim por diante. Caso não existam mais distribuições, a última é utilizada. Para adicionar uma nova duração, clique com o botão direito sobre o nó “Durações” da atividade e selecione a opção “Adicionar Duração”. A janela para edição da duração é então exibida no painel do lado direito. Para alterar uma duração existente, clique sobre a duração na árvore que seus dados aparecerão no painel da mesma forma.

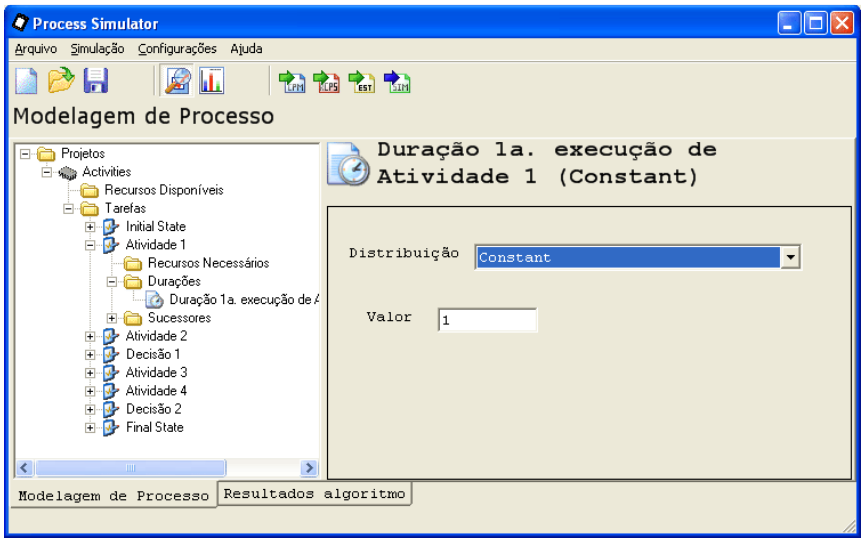

Figura 45: Modificando a duração de uma atividade

Como podemos ver, a duração da atividade é composta por dois elementos: a distribuição estatística e os seus parâmetros. Na versão atual do *SimProcess* existem três possibilidades para a distribuição: Valor Constante, Uniforme, Triangular e a Normal. Valor Constante na verdade não é uma distribuição, mas sim um valor fixo, que não é sorteado. A Uniforme distribui uniformemente a probabilidade de seleção entre todos os valores dentro de um intervalo. Já a Triangular distribui as chances entre 3 pontos, dando mais peso ao valor do meio. A distribuição Normal é definida pela sua média e desvio padrão, e o gráfico de sua função de distribuição tem a forma de um sino. Cada distribuição exige um conjunto diferente de parâmetros que são exibidos de acordo com a seleção da distribuição.

Alem das durações, os recursos também não podem ser lidos do diagrama de atividades. Por causa disso, devem ser manualmente preenchidos no *SimProcess* tanto os recursos que estão disponíveis para execução do processo, quanto quais são os recursos necessários para executar cada atividade. Os recursos disponíveis se encontram abaixo do item referente ao projeto na árvore. Para criar um novo recurso, basta clicar com o botão direito sobre o item “Recursos Disponíveis” e selecionar a opção “Adicionar Recurso Disponível”. A janela para edição do recurso é novamente exibida no painel da direita. Se desejar alterar um recurso, basta selecioná-lo no menu, que os seus dados aparecerão no painel da direita. É obrigatório fornecer um nome para o recurso e a sua quantidade disponível.

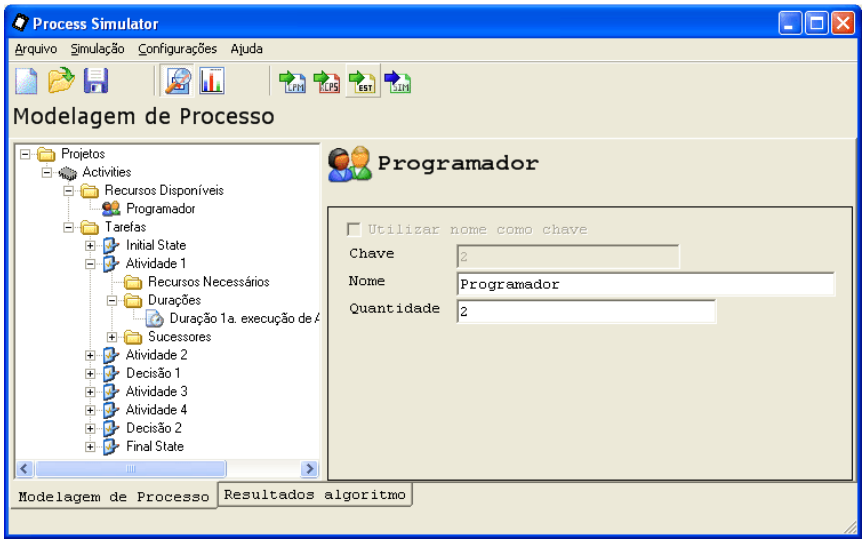

Figura 46: Modificando a quantidade de recursos disponíveis

Depois de dizer à ferramenta quais recursos estão disponíveis, é necessário informar quanto de cada recurso é necessário para executar cada atividade. Cada atividade pode depender da disponibilidade de diferentes tipos de recursos. Para adicionar uma nova dependência de recurso, é necessário utilizar o subitem “Recursos Necessários” situado abaixo da atividade na árvore. Para adicionar um novo recurso clique com o botão direito sobre “Recursos Necessários” e selecione a opção “Adicionar Recurso Necessário”. Uma janela é exibida, então, para a seleção do recurso a adicionar. Os recursos que são mostrados nesta lista são os mesmos cadastrados nos “Recursos Disponíveis” do projeto. Depois de selecionado um recurso, a tela para edição da quantidade do recurso é exibida no painel do lado direito. Se desejar alterar um recurso, basta selecioná-lo na árvore. No entanto, apenas a quantidade necessária do recurso pode ser alterada.

O último dos dados que não pode ser lido à partir do modelo é a função de probabilidade que diz qual caminho será selecionado após cada uma das decisões. Conforme foi dito anteriormente, as decisões são carregadas no *SimProcess* como atividades com duração constante, igual a 0. Por causa disso, elas aparecem na árvore de projeto como se fossem atividades. Na árvore abaixo da atividade, existe um nó que contém os sucessores de uma atividade, ou seja, as atividades que estão ligadas a ela no diagrama. No caso das decisões, esses sucessores são as atividades que são executadas após cada um dos seus resultados. Observe na figura abaixo os sucessores de uma decisão:

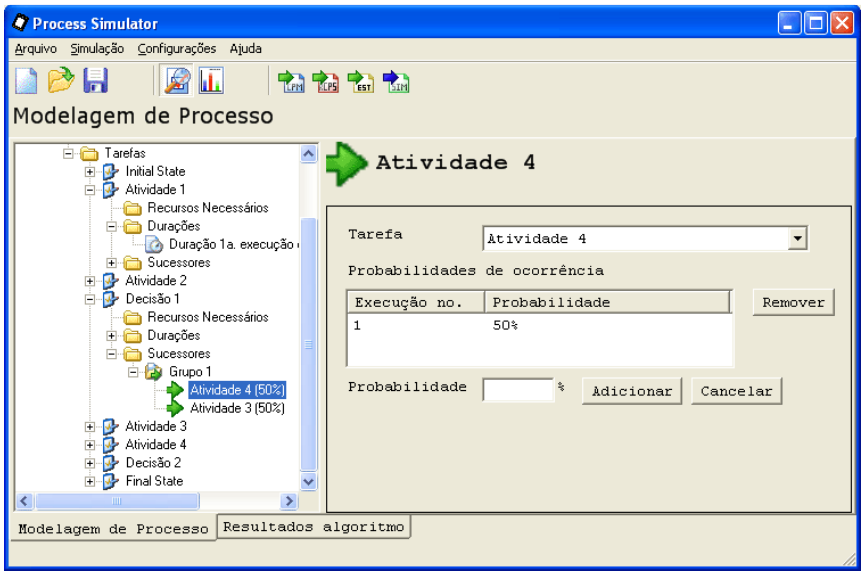

Figura 47: Alterando os grupos de atividades sucessoras

Como podemos ver, abaixo do nó “Sucessores” podem existir múltiplos grupos de atividades sucessoras. Dentro de um grupo, apenas uma das atividades é selecionada para execução. Se uma atividade possui mais de uma sucessora, ela possuirá múltiplos grupos sucessores. Dentro de um grupo, cada atividade está associada a probabilidades, que denotam a chance de determinado caminho ser selecionado. A soma das probabilidades de todas as atividades do grupo deve ser igual a 100%. Quando carrega um modelo de atividades criado em uma ferramenta externa, o programa divide igualmente entre as diferentes atividades a probabilidade de seleção. Para alterar estes valores, basta clicar sobre uma destas atividades dentro do grupo. A tela mostrada na Figura 47 é, então, exibida no painel da direita.

Cada uma das atividades do grupo pode estar associada a mais de uma probabilidade. Cada uma destas probabilidades se refere a uma passagem pela decisão, ou seja, na primeira passagem a primeira probabilidade é utilizada, na segunda a segunda probabilidade e assim por diante. Lembre-se que a cada ordem de execução, a soma das probabilidades deve ser igual a 100%. Para editar alguma das probabilidades da lista, basta selecioná-la. O programa colocará o seu valor no campo “Probabilidade” e modificará o botão para a opção “Alterar”. Para fazer a atualização, altere o campo “Probabilidade” e clique no botão “Alterar”. Para remover uma probabilidade, selecione-a na lista e clique em “Remover”. Para incluir uma nova edite o campo “Probabilidade” e clique em “Incluir”. Observe que o botão “Incluir” só aparece se o programa estiver fora do modo de alteração.

B.6. Executar a simulação

Uma vez que o modelo de atividades foi carregado e seus dados completados corretamente, o objetivo principal do programa pode ser atingido. As simulações podem ser executadas no *SimProcess* a partir da opção Algoritmos do menu principal:

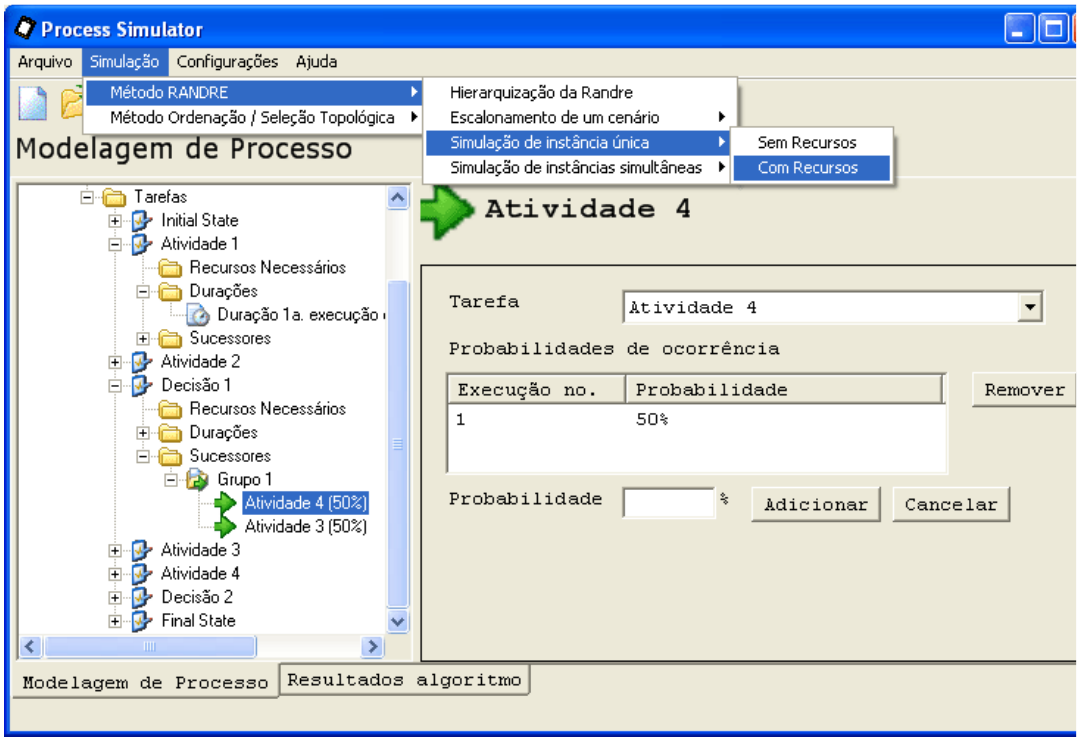

Figura 48: Executando uma simulação

No primeiro nível do menu existem duas opções principais. Elas dizem respeito ao método utilizado para geração dos cenários de execução da rede. Neste texto é apresentado apenas o Método RANDRE. Cada um desses menus apresenta a mesma estrutura de sub-menus, contendo os algoritmos que podem ser utilizados.

O primeiro algoritmo do menu não serve para simulação, mas sim para a montagem da hierarquia da RANDRE. Ao selecionar esta opção a rede de atividades selecionada será utilizada como parâmetro para o algoritmo de hierarquização da RANDRE, que apresentamos no capítulo anterior. O resultado é uma hierarquia em forma de árvore que é mostrada em uma janela separada.

O segundo menu (Escalonamento de um cenário) serve para a geração de um cronograma para um cenário de execução da RANDRE. Um novo cenário é gerado aleatoriamente, e ele é escalonado utilizando um dos algoritmos de escalonamento (com recurso ou sem recurso). O tipo de algoritmo utilizado pode ser selecionado no sub-menu. Se for selecionado um escalonamento com recursos, uma janela será exibida para a seleção da heurística.

A terceira opção (Simulação de instância única) dispara o mecanismo de simulação sem instâncias concorrentes. Novamente existe a opção para o escalonamento de atividades

sem recurso e com recurso. Ao executar a simulação uma janela é mostrada para a entrada do número de cenários que serão gerados na simulação. Se o escalonamento for com recursos, é necessário fornecer também a heurística para o escalonamento.

O último menu (Simulação de instâncias simultâneas) pode ser executado apenas sob o modo com recursos. Ela dispara o algoritmo para simulação de instâncias concorrentes apresentado no capítulo anterior. Para esta simulação é necessário fornecer o número total de cenários executados, a heurística de escalonamento e a distribuição do tempo entre disparo de cenários. Esta distribuição pode assumir a forma das mesmas distribuições disponíveis para a duração de uma atividade.

O resultado obtido à partir da simulação é exibido ao usuário da mesma forma em qualquer um dos métodos do *SimProcess*. Após a execução da simulação o programa altera automaticamente o modo de exibição para "Resultados de Algoritmo". O modo de exibição pode ser alternado utilizando as "tabs" no canto inferior esquerdo da tela. A tela que exibe os resultados de algoritmos possui o mesmo formato da tela de modelagem de processo. No lado esquerdo existe uma árvore no qual é mostrado o projeto e as suas atividades. Ao selecionar o projeto ou uma atividade uma tela é exibida no painel da direita contendo os resultados da simulação para aquele projeto/atividade. Ao selecionar um projeto a tela abaixo é exibida:

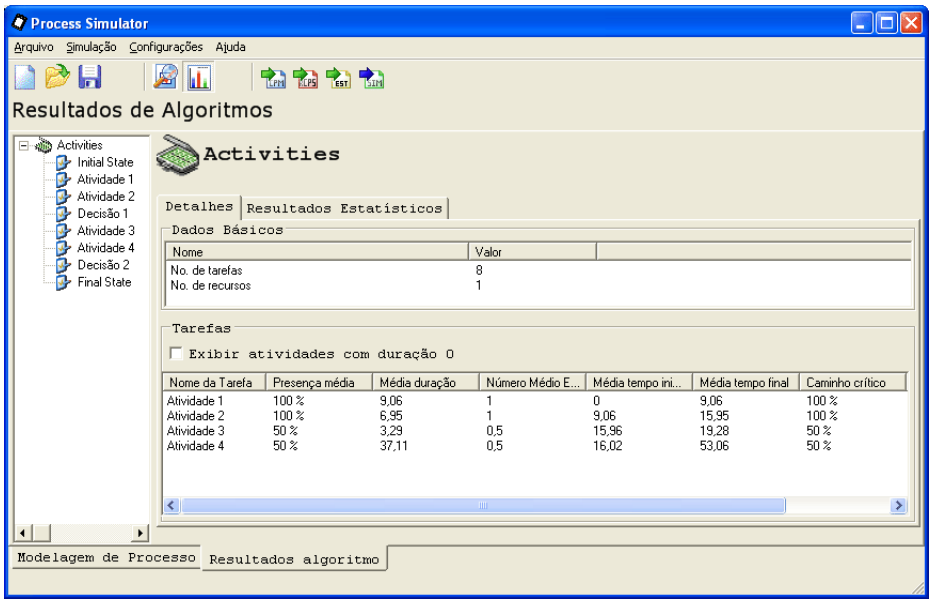

Figura 49: Resultados da simulação

Nesta tela existem duas "tabs": Detalhes e Resultados Estatísticos. Em "Detalhes" são exibidas informações sumarizadas para cada uma das atividades do projeto. Na lista "Tarefas" é exibida, para cada atividade: a porcentagem das execuções em que uma atividade esteve presente, a média da sua duração, o número médio de vezes em que ela foi executada, a média do tempo em que ela começou, a média do tempo final, a probabilidade de ela estar no caminho crítico e o tamanho médio da fila de espera. Ao clicar em "Resultados Estatísticos" é possível consultar dados específicos sobre cada um dos histogramas relacionados ao projeto.

Esta tela contém uma lista na qual o histograma pode ser selecionado, entre as seguintes opções: Duração, Número de Tarefas, Número de Tarefas no Caminho Crítico, Número de Etapas, Número de Etapas no Caminho Crítico. Existem três "tabs" que podem então ser selecionadas, contendo diferentes dados do histograma: Gráficos, Dados Estatísticos, Amostras Coletadas, Análise Convergência. Na "tab" "Gráficos" é possível plotar os gráficos de Distribuição, Densidade e Distribuição Invertida. Em "Dados Estatísticos" são exibidas medidas estatísticas comuns, como média, variância, desvio padrão. Em "Amostras Coletadas" é possível consultar o valor de cada amostra, sendo possível exportá-lo para um arquivo CSV. Em "Análise de Convergência" é mostrado o histórico de convergência da variação percentual da média e do desvio padrão.

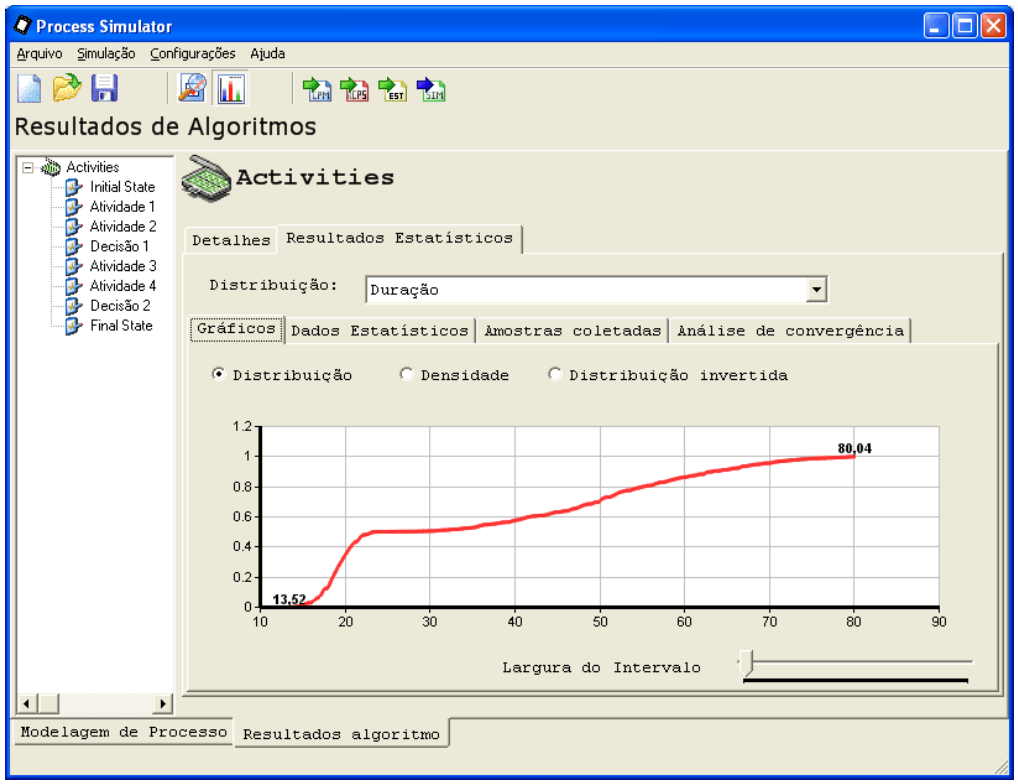

Figura 50: Resultados da simulação (II)

B.7. Salvar o Modelo

Depois de fazer todas essas alterações, é importante gravá-las em um arquivo para que não seja necessário fazê-las novamente. O formato XMI não pode armazenar todas as informações extras que foram adicionadas ao projeto. Por causa disso é necessário gravar o projeto em um arquivo cujo formato permita armazenar esses dados. O formato padrão de armazenamento do *SimProcess* é um arquivo XML. Dentro deste arquivo são armazenadas todas as informações do processo, de uma maneira organizada e legível. Para salvar o projeto

neste formato, selecione a opção do menu principal “Arquivo” - “Salvar Como”. No campo “Tipo de Arquivo” selecione a opção “Arquivos XML”, e forneça um nome para o arquivo.