O Módulo de Simulação da Ferramenta TANGRAM-II: Suporte para Medidas com Recompensas, Recursos de Eventos Raros e Aplicações a Modelos de Redes Multimídia

por

Daniel Ratton Figueiredo

UFRJ

Tese submetida para a obtenção do título de

Mestre em Ciências em Engenharia de Sistemas e Computação

ao Programa de Pós-Graduação de Engenharia de Sistemas e Computação

da COPPE/UFRJ

O Módulo de Simulação da Ferramenta TANGRAM-II: Suporte para Medidas com Recompensas, Recursos de Eventos Raros e Aplicações a Modelos de Redes Multimídia

Daniel Ratton Figueiredo

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Edmundo de Souza e Silva, Ph.D.

Prof. Berthier Ribeiro de Araujo Neto, Ph.D.

Prof. Paulo Henrique de Aguiar Rodrigues, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 1999

FIGUEIREDO, DANIEL RATTON

O Módulo de Simulação da Ferramenta TANGRAM-II: Suporte para Medidas com Recompensas, Recursos de Eventos Raros e Aplicações a Modelos de Redes Multimídia [Rio de Janeiro] 1999

XV, 124 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1999)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. simulação de evento raros
2. modelos de recompensa
3. ferramenta de modelagem

I. COPPE/UFRJ   II. Título (Série)

Aos meus pais com carinho.

Agradecimentos

Agradeço aos meus pais que contribuiram com muito carinho e compreensão, dando total apoio durante toda a minha formação. Agradeço e admiro o comprometimendo da minha noiva Sylvia em apoiar e compreender a minha dedicação durante a elaboração desta Tese.

Ao prof. Edmundo de Souza e Silva, meu orientador, que iluminou o caminho para o início e término desta Tese, e apoiou, sem medir esforços, a continuação dos meus estudos.

A todos os amigos do laboratório LAND (UFRJ), que direta ou indiretamente, ajudaram na confecção desta Tese. Ao amigo Carlos que, durante todo o trabalho de tese, discutiu e sugeriu modificações dando maior abrangência e clareza ao meu trabalho.

Esta Tese foi resultado de parte do trabalho realizado no projeto ALMADEM pertencente ao programa PROTEM-III financiado pelo CNPq e pelo programa PRONEX.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

O Módulo de Simulação da Ferramenta TANGRAM-II: Suporte para Medidas com Recompensas, Recursos de Eventos Raros e Aplicações a Modelos de Redes Multimídia

Daniel Ratton Figueiredo

Junho de 1999

Orientador: Edmundo de Souza e Silva
Programa: Engenharia de Sistemas e Computação

A modelagem de sistemas de computação e comunicação vem se tornando uma tarefa cada vez mais importante no processo de desenvolvimento de novas tecnologias. O uso de ambientes integrados e ferramentas de modelagem e análise facilitam este processo oferecendo métodos poderosos para obter resultados precisos. A ferramenta de modelagem TANGRAM-II permite a especificação de sistemas genéricos utilizando o conceito de objetos e mensagens. Medidas de interesse têm um papel importante no processo de modelagem, pois são esses números que irão caracterizar o comportamento do modelo. O método para especificar essas medidas deve ser o mais abrangente possível, e isto pode ser obtido através de "recompensas". Uma linguagem para especificação dessas medidas é definido e acoplado a ferramenta TANGRAM-II. Resolução do modelo e a obtenção das medidas de interesse é o último passo no processo de modelagem podendo ser feito de diversas maneiras. Apesar dos métodos analíticos e numéricos poderem ser utilizados na obtenção de soluções de sistemas mais simples (i.e. modelos Markovianos e alguns modelos não-Markovianos), simulação é a forma mais tradicional de resolver modelos e obter medidas de interesse. Um módulo de simulação genérico que calcula medidas de interesse de modelos com recompensa foi incorporado à ferramenta TANGRAM-II. A simulação de eventos raros representa um dos maiores desafios aos simuladores tradicionais podendo inviabilizar determinadas simulações. Neste caso, técnicas de simulação de eventos raros devem ser utilizadas por aumentarem a eficiência da simulação. Uma técnica bastante genérica e robusta, que pode ser aplicada a qualquer tipo de modelo, está acoplada ao módulo de simulação desta ferramenta.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

The Simulation Module in TANGRAM-II tool: Support for Reward Measures, Rare Event Simulation, and Applications to to Multimidia Network Models

Daniel Ratton Figueiredo

June 1999

Advisor: Edmundo de Souza e Silva
Department: Computer and System Engineering

Computer systems modelling and analysis has been considered as an important task in the process of developing new technologies. The performance analysis of systems being designed can be done using interest measures defined over a model. The use of frameworks and modeling tools makes this process easy and offers powerful methods to obtain accurate results. The modeling tool TANGRAM-II allows the specification of generic systems using the concept of objects and messages. Interest measures play an important role in the process of system analysis, since they are used to predict the behavior of the system. The specification of interest measures should be done in a generic enough way to allow the analyst flexibility during the modelling task, which can be achieved with the use of "rewards". A specification language for reward measures was developed and coupled to the TANGRAM-II tool. Model solution and the determination of interest measures are the last step in a modelling process and can be accomplished in many ways. Even though analytical and numerical methods can be used to solve simple models (i.e. Markovian and some non-Markovian models), simulation is still the most traditional form of solving models and obtaining interest measures. A generic simulation module that calculates interest measures through rewards specification was integrated to the TANGRAM-II tool. The simulation of rare events presents one of the greatest challenges to traditional simulators making some simulations impossible. In such cases, rare event simulation techniques should be used, turning simulations feasible and more efficient. A robust and generic technique, that can be applied to any kind of model, is incorporated to the simulation module of this tool.

Sumário

Resumo **v**

Abstract **vi**

1 Introdução **1**

1.1 Objetivos 3

2 Ferramentas de Modelagem e Simulação de Eventos Raros **5**

2.1 Ferramentas de Modelagem 5

2.1.1 A Ferramenta TANGRAM-II 6

Construindo um Modelo 9

2.1.2 A Ferramenta UltraSAN 10

2.1.3 TANGRAM-II *vs* UltraSAN 12

2.1.4 Uso de Recompensas nessas Ferramentas 14

2.2 Simulação 14

2.2.1 Método Monte Carlo 15

Integração de Monte Carlo 16

2.2.2 Simulação de Eventos Discretos 17

Sistema com Única Fila/Servidor 18

2.2.3 Simulação de Eventos Raros 19

Provocação de Eventos Raros 22

Importance Sampling 22

Splitting/RESTART 23

Descrição do Método 24

Eficiência e Robustez do Método 31

3 Especificação de Modelos com Recompensas 34

3.1 Modelos com Recompensas 34

Recompensa de Taxa Acumulada 36

Recompensa de Impulso Acumulada 37

3.2 Recompensas de Taxa 37

3.3 Recompensas de Impulso 41

3.4 Definição de medidas de interesse no UltraSAN *vs* TANGRAM-II . . 47

3.5 Solucionando as Medidas com Recompensas 48

4 Simulação de Eventos Discretos no TANGRAM-II 50

4.1 Arquitetura do Simulador 50

4.2 Possíveis Distribuições para Eventos 53

4.3 Clone de Eventos 55

4.4 Simulação Interativa 57

4.5 Definição de Medidas de Interesse 59

4.6 Exemplo 1: Modelo de fonte de vídeo + fila determinística 62

4.7 Exemplo 2: Modelo de fonte Poisson + fila com porta/ficha 65

5 Simulação de Eventos Raros no TANGRAM-II 70

5.1 Arquitetura do Simulador 70

5.2 Definição de Medidas de Interesse 74

5.3 Usando o Simulador 76

5.4 Exemplo: Fila M/M/1/k 77

5.4.1 Escolhendo Parâmetros para o Método RESTART 79

6 Algoritmo e Modelo para Recuperação de Pacotes de Voz na Internet 83

6.1 Transmissão de Voz em Tempo Real 84

6.2 Algoritmos para Recuperação de Perdas 85

6.2.1 Proposta de um Esquema de Redundância 88

6.3 Modelo para o Algoritmo Proposto 89

6.4 Resultados Obtidos 91

7 Conclusão 94

7.1 Sugestões para Trabalhos Futuros 96

Referências Bibliográficas 99

A TANGRAM-II Tutorial 103

A.1 Introduction 103

A.2 Modelling with TANGRAM-II 104

A.3 Constructing a Model 104

A.3.1 Declaration Section 105
A.3.2 Initialization Section 106
A.3.3 State variables Section 106
A.3.4 Events Section 106
A.3.5 Messages Section 107
A.3.6 Rewards Section 108
A.3.7 Conditions and Actions 108
A.3.8 Exchanging Messages 109
A.4 First Model 111
A.4.1 Reward Variables 114
A.4.2 Distribution Types 115
A.5 Interactive Simulation 115
A.5.1 Animation 116
A.6 Literal Models 119
A.7 Other examples 120
A.7.1 MMPP packet source, leaky bucket traffic shaper and server queue 120
A.7.2 Packet Recovery Mechanism 120

Lista de Figuras

2.1 Modelo do sistema M/M/1/k na ferramenta TANGRAM-II. 10

2.2 Modelo do sistema M/M/1/k na ferramenta UltraSAN. 11

2.3 Círculo e quadrado circunscrito: razão entre áreas é $\pi/4$. 16

2.4 Evolução de uma fila com chegadas e saídas de pacotes. 19

2.5 Duas formas de aumentar a eficiência em simulação. 21

2.6 Idéia básica da técnica de Splitting/RESTART. 24

2.7 Exemplo de uma execução de uma simulação utilizando a técnica de RESTART. 25

2.8 Exemplo de uma execução de uma simulação utilizando a técnica de RESTART. 28

3.1 Definição de uma simples recompensa de taxa. 39

3.2 Definição de recompensa de taxa para medir a utilização. 40

3.3 Exploração de estados no TANGRAM-II. 42

3.4 Definição de recompensa de impulso para contagem de pacotes do tipo 2. 43

3.5 Definição de recompensa de impulso para contagem dos pacotes enviados pela fonte. 44

3.6 Definição de recompensa de impulso para contagem de pacotes recebidos pela fila. 45

3.7 Definiçao de recompensa de impulso representando uma fila em uma fonte On-Off. 46

3.8 Parte da especificação de um modelo de rede de filas na ferramenta UltraSAN. 48

4.1 Arquitetura do Simulador de Eventos Discretos. 51

4.2 Exploração de estados no TANGRAM-II. 52

4.3 Lista de Eventos com diversos eventos clonados. 56

4.4 Servidor infinito determinístico utilizando a técnica de clonagem de eventos. 57

4.5 Fila animada utilizada na simulação interativa. 59

4.6 *Traces* das medidas $CR(t)$ e $CI(t)$ 60

4.7 Modelo On-Off com fila representada por recompensa de taxa 61

4.8 Cadeia de Markov do modelo de Nascimento e Morte de uma fonte de vídeo 63

4.9 Especificação do modelo de Nascimento e Morte de uma fonte de vídeo 64

4.10 Utilização *vs* Capacidade do servidor. Comparação entre resultado analítico e simulação 65

4.11 Tamanho médio da fila *vs* Capacidade do servidor. Comparação entre resultado analítico e simulação 66

4.12 Distribuição do tamanho da fila *vs* Capacidade do servidor. Comparação entre resultado analítico e simulação 67

4.13 Comportamento da fila/servidor ao longo do tempo 68

4.14 Resultados das medidas de interesse do modelo de fila com porta . . . 69

5.1 Arquitetura do simulador de eventos raros. 71

5.2 Pontos de parada de uma execução de simulação do método RESTART. 73

5.3 Medidas de interesse acumuladas durante a execução de uma simulação utilizando o método RESTART. 75

5.4 Comparação entre simulação de eventos raros e resultado analítico. . 79

6.1 Esquema de redundância com janela de 5 pacotes. 87

6.2 Janela de tamanho 6 com dois pacotes de redundância. 88

6.3 Fração de pacotes corretos para diferentes taxas de perda. 92

6.4 Fração de pacotes recuperados para diferentes taxas de perda. 93

A.1 The packet source class. 112

A.2 The server with queue class. 113

A.3 The M/M/1/k model. 113

A.4 The animated object representing queue size 117

A.5 The animated object representing packet movement 118

A.6 The MMPP packet source model with a leaky bucket traffic shaper and server queue 123

A.7 The Packet Recovery Model 124

Lista de Tabelas

5.1 Número de pontos de quebra e réplicas para diferentes cargas no servidor. 78

5.2 Função de distribuição de probabilidades para a fila com carga 65%. . 80

6.1 Média amostral do resultado da simulação 91

Palavras-chave

Ferramenta de Modelagem

Modelos de Recompensa

Simulação

Simulação de Eventos Raros

Método Splitting/RESTART

Recuperação de Pacotes

Capítulo 1

Introdução

A modelagem de sistemas de computação e comunicação vem se tornando uma tarefa cada vez mais importante no processo de desenvolvimento de novas tecnologias. A análise do desempenho e da confiabilidade de sistemas ainda sendo projetados é uma das grande vantagens do uso de modelagem. Novas tecnologias e algoritmos podem ser propostos e avaliados ainda na fase de projeto, antes de serem implementados. Para facilitar este processo, diversos ambientes integrados e ferramentas de modelagem e análise foram desenvolvidos [9]. Porém, essas ferramentas, em geral, aplicam-se a apenas uma determinada classe de modelos restringindo o número de sistemas que podem ser avaliados. Existem ferramentas mais genéricas que permitem ao projetista especificar e resolver praticamente qualquer tipo de sistema, dada uma determinada abstração necessária. A ferramenta TANGRAM-II, atualmente sendo desenvolvida pela UFRJ como continuação de um trabalho iniciado em 1991, é um exemplo. Esta ferramenta possui um paradigma de modelagem peculiar permitindo que modelos sejam descritos utilizando objetos genéricos. Esses objetos interagem entre si utilizando mensagens que podem ser enviadas e recebidas por objetos do sistema. Este paradigma assemelha-se à tecnologia de programação orientada a objetos.

Uma tarefa importante durante o processo de especificação do modelo é a definição das medidas de interesse. Medidas de interesse refletem o comportamento do modelo e são utilizadas para avaliar e caracterizar o sistema. Uma boa ferramenta de modelagem deve oferecer a seus usuários a possibilidade de definir medidas de interesse da forma mais abrangente possível. Assim, o projetista pode obter a informação desejada especificando uma medida qualquer sobre o modelo. Modelos com "recompensa" oferecem esta abrangência e permitem que medidas específicas possam ser definidas. Existem basicamente, dois tipos de recompensa: recompensas de taxa e recompensas de impulso. Recompensas de taxa são valores associados aos estados do sistema que indicam a taxa com a qual a medida de interesse está se acumulando. Recompensas de impulso são valores associados a transições entre estados do sistema que marcam a ocorrência de eventos de interesse. Utilizando estes dois tipos de recompensa é possível obter praticamente qualquer informação do sistema sendo modelado.

Uma vez descrito, um modelo precisa ser resolvido e as medidas de interesse especificadas calculadas. A ferramenta TANGRAM-II possui um poderoso módulo de solução que permite resolver sistemas Markovianos e uma classe de modelos não-Markovianos utilizando métodos analíticos e numéricos. Além desses, métodos sofisticados de solução de modelos com recompensas também integram este módulo. Porém, muitas vezes os modelos não podem ser resolvidos analiticamente, requisitando então o uso de simulação para resolvê-los. Simulação é a técnica mais utilizada para resolução de sistemas e análise de desempenho. Muitas vezes esta é a única forma de avaliar sistemas grandes e complexos. Porém, um dos grandes desafios da simulação tradicional é estimar a probabilidade de eventos raros. Este tipo de evento acontece poucas vezes durante o tempo de vida de um sistema e para estimá-lo, utilizando simulação, é necessário um enorme esforço computacional em termos de tempo de execução. Técnicas para simulação de eventos raros vêm sendo desenvolvidas com o objetivo de reduzir o tempo de simulação e aumentar o grau de confiança da estimativa.

1.1 Objetivos

Um dos objetivos desta tese é definir uma linguagem genérica para especificação de modelos com recompensas dentro da ferramenta TANGRAM-II. Recompensas associadas a estados e a transições podem ser especificadas utilizando o atual paradigma de modelagem da ferramenta. A forma para definir as recompensas deve ser a mais genérica possível, permitindo ao projetista dar o significado desejado à medida de interesse sendo especificada.

Um módulo de simulação de eventos discretos foi desenvolvido, permitindo que qualquer modelo especificado na ferramenta TANGRAM-II possa ser simulado. Diversas distribuições de probabilidades para eventos foram incorporadas ao simulador dando grande abrangência na obtenção da medida. O simulador obtem qualquer medida de interesse definida pelo usuário através de recompensas de taxa, associadas a estados, ou a recompensas de impulso, associadas a transições feitas pelo sistema. Essas medidas devem ser especificadas utilizando a linguagem definida para especificação de recompensas.

Acoplado ao módulo de simulação foi incorporado um algoritmo para simulação de eventos raros. Por ser flexível, robusta e se aplicar a qualquer classe de modelo, esta técnica foi recentemente abordada e vem sendo discutida amplamente na literatura. Permite que medidas de interesse com valores muito baixos (i.e. 10^{-30}) possam ser simuladas com precisão. As medidas de interesse calculadas pelo simulador são as mesmas medidas especificadas no simulador convencional. A interação do usuário com o método de eventos raros é pequena, não precisando que este seja conhecedor do assunto.

Esta tese está organizada da seguinte forma: o capítulo 2 apresenta os estudos sobre algumas ferramentas de modelagem e sobre técnicas de simulação. A ferramenta TANGRAM-II é também apresentada e uma comparação com outra ferramenta genérica de modelagem é feita. O capítulo aborda ainda questões de simulação de eventos discretos e sua forma mais tradicional de implementação. Finalmente, o capítulo investiga técnicas de simulação de eventos raros dando ênfase no algoritmo acoplado ao simulador da ferramenta TANGRAM-II.

O capítulo 3 apresenta a forma para definição de recompensas na ferramenta TANGRAM-II. Recompensas associadas a estados do sistema e a transições podem ser especificadas. Alguns exemplos são apresentados e discutidos.

O capítulo 4 apresenta o módulo de simulação de eventos discretos que foi desenvolvido e incorporado à ferramenta. Todas as funcionalidades do simulador são relatadas, e um exemplo, com medidas de interesse, é apresentado e discutido no final do capítulo.

O capítulo 5 introduz o módulo de simulação de eventos raros e seu funcionamento. Discute-se as medidas de interesse e seu uso genérico. Um exemplo completo é apresentado no final deste capítulo.

O capítulo 6 apresenta um exemplo de modelagem de um algoritmo para recuperação de pacotes desenvolvido durante os estudos do mestrado. Este modelo é simulado e medidas de interesse referentes à eficiência do algoritmo de recuperação de perdas são obtidas.

Finalmente, no capítulo 7 são feitas considerações finais sobre esta tese sugerindo possíveis temas para trabalhos futuros.

Capítulo 2

Ferramentas de Modelagem e Simulação de Eventos Raros

NESTE capítulo abordamos de forma geral os vários estudos desenvolvidos para realizar esta tese. Na seção 2.1 aborda-se o conceito de modelagem de sistemas e o uso de ferramentas para facilitar o processo. Destaca-se o funcionamento de duas delas: TANGRAM-II e UltraSAN. Na seção 2.2 discute-se a simulação de sistemas através de uma técnica tradicional. Na seção 2.3 apresenta-se a simulação de eventos raros e o que pode ser feito para amenizar os problemas dela decorrentes. Particularmente, descreve-se o método que foi incorporado ao módulo de simulação da ferramenta TANGRAM-II.

2.1 Ferramentas de Modelagem

Modelagem e análise são importantes partes no processo de desenvolvimento de sistemas complexos que permitem avaliar e comparar sua eficiência com outras alternativas de projeto. Especificamente, sistemas de computação e comunicação são frequentemente modelados para serem avaliados antes de serem implementados, permitindo assim prever seu desempenho e comportamento [11].

Atualmente, com a crescente necessidade de sistemas mais eficientes e confiáveis, é preciso dar respostas corretas e precisas a perguntas tais como: "Quanto trabalho o sistema é capaz de realizar?", "Qual a probabilidade deste sistema falhar em um determinado período?", "Qual a taxa de bloqueio de células deste *switch*?", "Qual o comportamento das filas deste *switch* sob alta/baixa carga?" [11]. Para encontrar respostas a essas perguntas, antes do sistema ser desenvolvido, recorremos a modelos. Um modelo consiste na abstração de um sistema real que possui simplificações com detalhes suficientes que permitam que o sistema seja avaliado de forma precisa dentro de uma determinada tolerância. Existem diversas vantagens em se criar um modelo de um sistema real. Por exemplo, se o sistema real não existe e o projetista deseja avaliar sua eficiência ou medir o comportamento do sistema sobre alguma condição, o uso de modelagem é, muitas vezes, a única forma de realizar tal tarefa.

Nos últimos anos, várias ferramentas de modelagem surgiram com o intuito de facilitar o processo de modelagem e análise de sistemas. Muitas delas são específicas e se aplicam apenas a determinadas classes de modelos. Por exemplo, a ferramenta SAVE (System Availability Estimator) que é talhada para modelos de confiabilidade [19]. Muitas vezes, o uso de uma ferramenta mais específica facilita a tarefa de modelagem porém, apenas uma determinada classe de modelos pode ser especificada e avaliada.

O TANGRAM-II [5] é uma ferramenta genérica que oferece um ambiente de modelagem e análise de sistemas de computação e comunicação através de uma interface gráfica amigável. Outra ferramenta, que também possui um ambiente gráfico para modelagem e análise é o UltraSAN [31]. Apesar de ambas oferecerem ambientes integrados para modelagem e análise, essas ferramentas adotam uma abordagem completamente diferente para descrição do sistema a ser modelado.

2.1.1 A Ferramenta TANGRAM-II

A ferramenta TANGRAM-II é um ambiente de modelagem desenvolvido para fins de pesquisa e ensino que provê uma interface flexível para descrição de sistemas de computação e comunicação. O paradigma de especificação de modelos é orientado a objetos e feita sobre uma poderosa interface gráfica de domínio público, o TGIF (TANGRAM Graphical Interface Facility) [6].

A idéia inicial deste paradigma de modelagem surgiu da necessidade de desenvolver modelos gerais para diferentes aplicações [2]. Uma versão rudimentar da ferramenta foi desenvolvida na linguagem Prolog em 1991 [12]. Posteriormente, uma versão com mais recursos foi desenvolvida em 1997 [5] sendo implementada em C/C++. Recentemente, parte da ferramenta TANGRAM-II foi reimplementada, oferecendo novas funcionalidades e incorporando novos módulos, dando-lhe maior flexibilidade e eficiência.

O paradigma de modelagem utilizado pelo TANGRAM-II foi proposto em [2]. Neste paradigma, o sistema a ser modelado é representado por uma coleção de objetos que interagem através do envio de mensagens. O estado interno de um objeto é representado por um conjunto de variáveis inteiras. O comportamento de um objeto é definido por eventos que possuem uma condição e um conjunto de ações. E também por ações que estão associadas ao recebimento de mensagens.

Eventos são gerados espontaneamente por objetos contanto que a condição associada ao evento seja satisfeita. A condição é uma expressão booleana que utiliza o estado atual do objeto para habilitar ou desabilitar um evento. Cada evento possui, associado a ele, um tempo de disparo caracterizado por alguma distribuição (i.e. exponencial, determinística, erlangiana, etc.).

Mensagens são apenas uma abstração utilizada para a interação entre objetos. Estas são entregues em tempo zero realizando uma determinada ação associada. Toda a comunicação entre os objetos deve ser feita sob a forma de mensagens.

Quando um evento ocorre ou quando uma mensagem é entregue, um conjunto de ações especificadas pelo usuário é executada com uma determinada probabilidade. O tempo para execução de ações e para a entrega de mensagens é zero. Durante este procedimento o sistema passa por um conjunto de estados evanescentes que não afetam seu comportamento. Ao final da execução de uma ação, um objeto pode ter o seu estado alterado e mensagens podem ter sido enviadas a outros objetos do modelo.

Para criar um novo modelo, o usuário pode utilizar uma biblioteca de objetos previamente definidos com comportamento determinado, ou iniciar a criação a partir de um objeto vazio (*template*). Objetos podem possuir constantes que devem ser atribuídas a valores numéricos quando forem utilizadas.

Após a criação de um modelo, o usuário precisa resolvê-lo para obter as medidas de interesse pertinentes à descrição do mesmo. No ambiente TANGRAM-II existem duas formas de se obter as medidas de interesse: através de simulação, ou através de solução analítica.

Uma vantagem da simulação é que ela pode ser feita sobre qualquer modelo especificado na ferramenta TANGRAM-II. Neste caso, os eventos dos objetos modelados podem ter distribuições genéricas pré-definidas e até ler distribuições de arquivos (*traces*). O capítulo 3 mostra como definir medidas de interesse para serem obtidas pelo simulador. Uma das vantagens da simulação é que o espaço de estados associado ao modelo não precisa ser gerado, uma vez que esta tarefa pode ser bastante custosa computacionalmente. O módulo de simulação foi desenvolvido como parte deste trabalho e está descrito detalhadamente no capítulo 4.

Para resolver o modelo utilizando métodos analíticos ou numéricos é necessário gerar por completo o espaço de estados e as transições associadas ao modelo. Dado o estado inicial do sistema, fornecido pelo usuário, o gerador de estados descobre a lista de todos os eventos habilitados que podem ser disparados naquele estado. Um evento é então escolhido para ser disparado e no final da ação correspondente, um novo estado é gerado possuindo uma possível lista de mensagens pendentes. Estados que possuem mensagens pendentes são chamados de evanescentes (*vanishing*) e não fazem parte do espaço de estados do modelo. Esses estados são apenas uma abstração utilizada para realizar a execução de ações e troca de mensagens em tempo zero. As mensagens pendentes são entregues aos seus objetos destino, uma a uma, e as ações correspondentes são executadas. Ao final, tem-se um estado tangível (*tangible*) que não possui mensagens pendentes e que faz parte do espaço de estados do modelo. Durante este processo, uma transição entre o estado inicial e o estado final também é calculada. Este algoritmo é repetido recursivamente fazendo uma busca em profundidade nos estados tangíveis.

Uma vez gerado o espaço de estados e a matriz de transição, o modelo pode ser resolvido. Na ferramenta TANGRAM-II, existe um poderoso módulo de solução capaz de obter diversas medidas de interesse com diferentes métodos de solução. O módulo de solução oferece suporte para análise em estado estacionário e transiente. Em [5] uma descrição do módulo de solução e dos métodos utilizados para análise é apresentada. Este módulo possui diversos métodos para modelos Markovianos, ou seja, modelos que possuem todos os eventos com distribuição exponencial. Existe ainda, um método de solução analítica que permite resolver modelos não Markovianos. Este método de solução suporta eventos determinísticos e exponenciais, porém com determinadas limitações. Uma de suas restrições está no número de eventos determinísticos que podem estar habilitados simultaneamente. Este método de solução permite apenas que um evento esteja habilitado [10], restringindo a classe de modelos que podem ser resolvidos.

Construindo um Modelo

O primeiro passo na construção de um modelo é a definição dos objetos que irão compor o sistema e o funcionamento de cada um desses objetos. O tutorial no apêndice A explica detalhadamente o processo de especificação de objetos e a construção de modelos. Nele, exemplos simples são utilizados para facilitar a explicação.

A Figura 2.1 ilustra o exemplo da modelagem do sistema de fila M/M/1/k [26]. Neste modelo, uma fonte emite pacotes a uma taxa com distribuição exponencial (fonte Poisson). Esses pacotes são enviados para uma fila. Se a fila tem tamanho menor que k, então ela é usada para armazenar pacotes que são servidos individualmente. Ao final do serviço o pacote é retirado da fila. Este é um modelo simples e bastante conhecido que ajuda a compreender a forma de especificar modelos na ferramenta TANGRAM-II. Uma explicação detalhada da especificação deste modelo é apresentada no apêndice A.

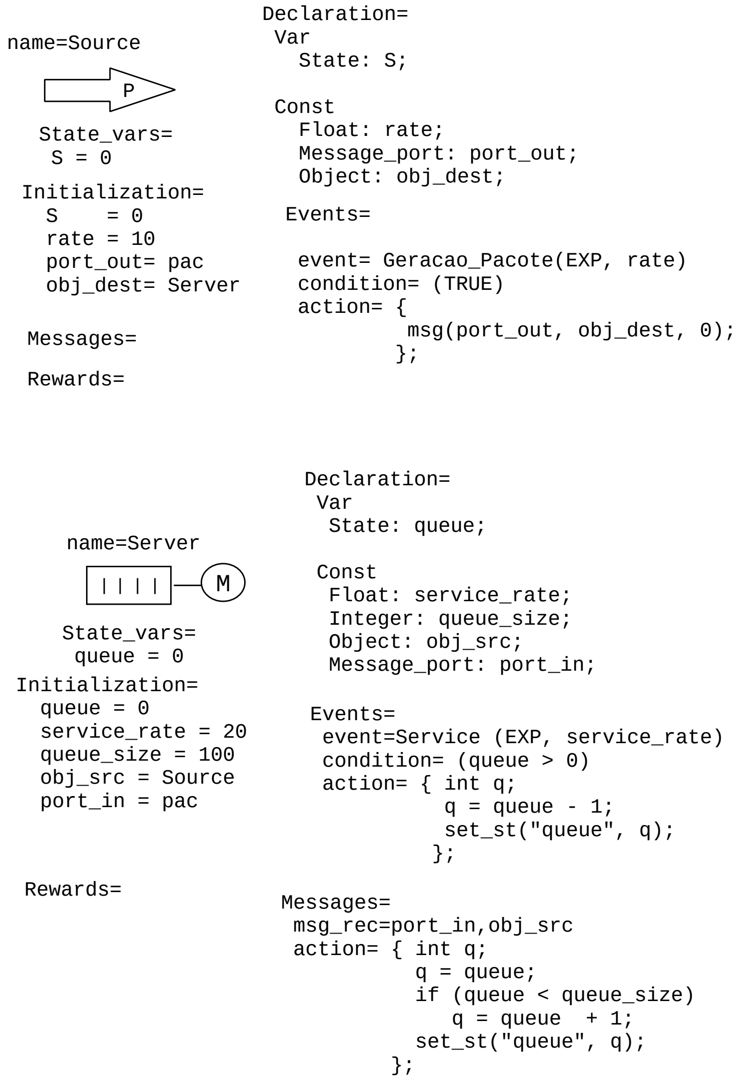


Figura 2.1: Modelo do sistema M/M/1/k na ferramenta TANGRAM-II.

2.1.2 A Ferramenta UltraSAN

A ferramenta UltraSAN [31] utiliza o conceito de redes de atividades estocásticas (Stochastic Activity Networks - SANs), concebido no início da década de 80, que é uma extensão a redes de Petri. Usando primitivas gráficas, SANs provêm um formalismo em alto nível com o qual modelos podem ser especificados.

SANs possuem quatro objetos primitivos que são utilizados para descrever o comportamento de um modelo.

- *Places* - compõem a representação do estado do sistema. Cada *place* contém um determinado número de tokens que representam o seu estado. O conjunto de todos os *places* formam o estado da rede de atividade que é denominado *marking*. São representados por círculos na especificação do modelo.

- *Activities* - representam ações feitas no modelo que possuem um determinado tempo (distribuição) para serem completadas. Além disso, *activities* podem possuir probabilidades associadas que permitem que ações diferentes, denominadas *cases*, sejam executadas. São representados por ovais achatados na especificação do modelo.

- *Input Gates* - controlam o disparo de ações. Estes são responsáveis em habilitar ou desabilitar as *activities* e também servem para definir o estado dos *places* ao final de uma ação. São representados por triângulos deitados com o vértice da ponta ligado à *activity* que ele controla.

- *Output Gates* - Assim como *Input Gates*, servem para definir a mudança a ser feita por um *place* ao término das *activities*. São representados por triângulos deitados que tem seu lado vertical ligado a *activities* ou a *cases*.

A Figura 2.2 ilustra uma SAN que modela o comportamento do tradicional sistema de fila M/M/1/k.

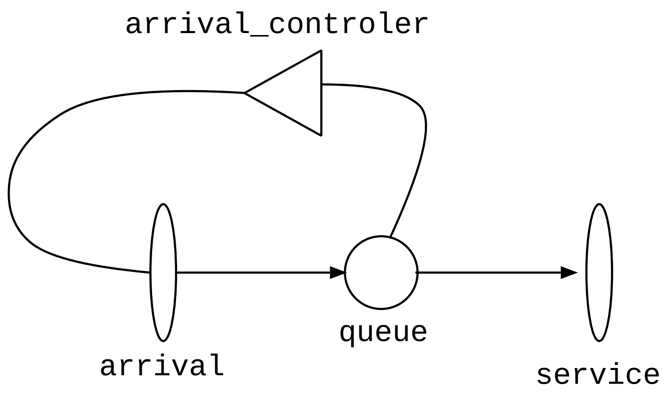


Figura 2.2: Modelo do sistema M/M/1/k na ferramenta UltraSAN.

Ao ser executada, uma SAN passa por uma sequência de *markings*. Durante esta execução a rede irá encontrar estados consistentes ou inconsistentes. Estados consistentes são aqueles que não possuem *activities* de tempo zero habilitadas. Apenas os estados consistentes contribuem para as medidas de *performance* do modelo o que equivale aos estado tangível do TANGRAM-II. A ferramenta UltraSAN possui um algoritmo que implementa esta busca de estados. Ao passar por todos as marcações obtém-se ao final um modelo matemático associado ao sistema especificado.

Existem duas formas de resolver o modelo descrito e obter as medidas de interesse especificadas: através de simulação ou através de solução analítica. No primeiro caso, um simulador é construído e em seguida executado. Esta forma de solução se aplica a qualquer processo estocástico associado a SAN. No segundo caso, todo o espaço de estado e a matriz de transição associada ao modelo SAN é gerada. Porém a solução analítica só se aplica em um dos dois casos abaixo:

- Todas as *activities* são exponencialmente distribuídas. Neste caso, a cadeia de Markov associada ao modelo é gerada e resolvida com métodos analíticos.
- Todas as *activities* possuem distribuição exponencial ou determinística. Porém, apenas uma *activity* com distribuição determinística pode estar habilitada por vez. Neste caso, um modelo semi-Markoviano é gerado e resolvido analiticamente.

2.1.3 TANGRAM-II *vs* UltraSAN

Apesar da semelhança aparente entre as duas ferramentas, elas são completamente distintas no processo de especificação do modelo e também na relação dos métodos de solução disponíveis. A ferramenta UltraSAN, carrega explicitamente o conceito e a forma de redes de Petri. Este conceito pode introduzir uma grande desvantagem no paradigma de modelagem para pessoas pouco familiarizadas com esta técnica. Além disso, modelos grandes e complexos se tornam incompreensíveis na sua forma gráfica devido à quantidade de informação incluída implicitamente no desenho. Em [2], é dado um exemplo onde uma rede de Petri estocástica se torna incompreensível sendo necessária uma descrição textual do comportamento do sistema sendo modelado para que se possa compreendê-lo. Esta descrição não é parte do modelo em si.

Na ferramenta TANGRAM-II todo o comportamento do objeto, e por sua vez do sistema, é descrito de forma concisa em uma linguagem de programação similar à linguagem C. Um diagrama gráfico pode ser, inicialmente, feito representado a abstração do sistema real e depois os objetos gráficos deste diagrama podem ser descritos através da linguagem de especificação da ferramenta. A descrição dos objetos não deixa dúvidas sobre o seu funcionamento e independe de outros objetos do sistema. Além disso, qualquer representação gráfica pode ser dada aos objetos tornando-os mais parecidos com o que se quer modelar. Os recursos gráficos utilizados na ferramenta TANGRAM-II não apresentam funcionalidade com relação á especificação do modelo, servindo apenas para melhor ilustrá-lo. Porém, o módulo de simulação interativa (a ser apresentada no capítulo 4) faz uso extensivo da parte gráfica.

Além de diferirem no paradigma para especificação dos modelos, as ferramentas também são distintas em seus módulos de solução. Apesar de alguns dos resultados apresentados serem sobre as mesmas medidas de interesse, a forma com que essas medidas são calculadas é bastante diferente. Os algoritmos e os métodos utilizados para calcular a solução analítica são distintos. Enquanto a ferramenta TANGRAM-II possui uma boa diversidade nos métodos analíticos e numéricos para solução do modelo, na ferramenta UltraSAN esta variedade não existe.

Os módulos de simulação também são distintos. Ambas possuem simuladores orientados a eventos discretos e suporte para simulação de eventos raros. Porém, na ferramenta UltraSAN, a técnica de Importance Sampling é utilizada para melhorar a eficiência da simulação de eventos raros. Observa-se na seção 3, que esta técnica é pouco robusta com relação aos seus parâmetros e ao modelo. Na ferramenta TANGRAM-II, utilizou-se a técnica de Splitting/RESTART para dar suporte a eventos raros. Constata-se também, que esta técnica é mais robusta com relação a seus parâmetros e pode ser aplicada a qualquer classe de modelo.

2.1.4 Uso de Recompensas nessas Ferramentas

Ambas as ferramentas oferecem suporte à definição de medidas de interesse. Essas medidas permitem ao usuário dar significado aos números que serão obtidos do modelo. Na ferramenta UltraSAN todas as medidas de interesse devem ser definidas a partir de recompensas de taxa ou de impulso. A ferramenta TANGRAM-II permite que o usuário defina não só medidas genéricas de recompensa como também medidas sobre a solução do modelo. No capítulo 3,x apresenta-se uma ampla discussão sobre modelos com recompensas e como utilizá-los no TANGRAM-II.

Na ferramenta UltraSAN a especificação de recompensas é feita em um módulo separado, e não no módulo de descrição do modelo. Isto impede que primitivas do sistema, como *places* e *activities*, carreguem em si definições para medidas de interesse. Ao serem reutilizadas, as medidas de interesse dessas primitivas precisam ser redefinidas, não podendo ser aproveitadas, pois são definidas fora do contexto das primitivas. Apesar disto, a filosofia para definir as recompensas é igual em ambas as ferramentas. No final do capítulo 3,x discute-se mais sobre as diferenças na especificação de medidas de interesse entre as ferramentas.

2.2 Simulação

Nos dias de hoje, grandes sistemas críticos precisam ser analisados de forma a se obter medidas de desempenho relevantes, o que muitas vezes torna a modelagem do sistema complexa e custosa. A necessidade de medir tais sistemas complexos é essencial para garantir um bom desempenho durante sua operação e para melhorar a relação custo-benefício durante o projeto e desenvolvimento.

Pesquisadores vêm, há muito tempo, desenvolvendo técnicas de modelagem e análise que permitem prever de forma relevante o comportamento de tais sistemas. Este esforço está limitado à solução analítica ou numérica, geralmente associadas a modelos Markovianos, ou menos freqüentemente, a modelos semi-Markovianos. O uso destas técnicas torna-se rapidamente inviável devido a problemas práticos como a explosão do número de estados do modelo e/ou à inadequação dos modelos Markovianos na representação de sistemas reais. Como o número de estados cresce exponencialmente de acordo com o número de componentes no sistema, e devido a limitações de armazenamento e de capacidade computacional, apenas sistemas pequenos podem ser analisados utilizando métodos numéricos para solução. Várias técnicas foram propostas para reduzir o espaço de estado de modelos Markovianos de tamanho médio ($10^4 - 10^5$ estados) e grandes (10^6 ou mais estados). Entre elas podem citar-se a técnica de *lumping* e o truncamento do modelo utilizando limites [11, 28]. Além disso, até mesmo sistemas médios do tipo *stiff* (geralmente, modelos que possuem taxas de transição com diferentes ordens de magnitude) tem sua solução numérica dificultada devido a esta diferença grande nas taxas de disparo de eventos. Porém, técnicas como decomposição e agregação [11] podem ser utilizadas para amenizar este problema.

Quando métodos numéricos e/ou analíticos de solução se tornam inviáveis para resolver o modelo, o uso de simulação passa a ser uma boa alternativa. A simulação apresenta vantagens claras como o baixo uso de armazenamento, uma vez que o espaço de estados do modelo não precisa ser gerado. Porém, medidas precisas obtidas através de simulação precisam ser interpretadas corretamente [30].

Apresenta-se a seguir, o método mais conhecido e utilizado para realizar simulações de modelos orientados a eventos.

2.2.1 Método Monte Carlo

O método Monte Carlo para solução de problemas matemáticos é utilizado em diversas áreas do conhecimento, como a matemática, a física, a química e a computação. O método consiste em fazer uma análise numérica sobre uma seqüência de eventos randômicos (i.e. um processo estocástico) com o intuito de observar algum comportamento interessante. Apesar de não estar ligado a computação, o método teve sua eficácia aumentada com a disponibilidade de computadores que são capazes de realizar simulações através da geração de eventos randômicos.

Segue agora um tradicional exemplo que mostra como o método Monte Carlo pode ser utilizado na resolução de problemas [24] [30]. Considere um círculo e um quadrado circunscrito de lado igual a L, conforme ilustra a Figura 2.3. A razão da área do círculo para a área do quadrado é $\pi/4$. É plausível que se pontos forem colocados randomicamente dentro do quadrado, a fração $\pi/4$ desses pontos também estará dentro do círculo. Se isto for verdade, então pode-se obter a medida de $\pi/4$ colocando um tabuleiro redondo com diâmetro L dentro de um quadrado com lado L e coletando pingos de chuva. Pode-se também programar o computador para gerar pares de coordenadas cartesianas aleatórias para representar pontos dentro do quadrado e contar a fração de pontos que estão dentro do círculo. Esta fração será uma estimativa para $\pi/4$. Este exemplo ilustra que a amostragem aleatória pode ser utilizada para resolver problemas matemáticos, apesar de que, neste caso, o problema possa ser resolvido mais rápida e precisamente com outros métodos.

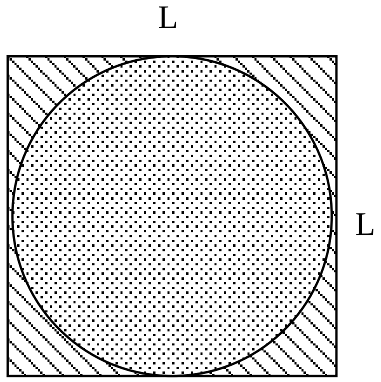


Figura 2.3: Círculo e quadrado circunscrito: razão entre áreas é $\pi/4$.

Integração de Monte Carlo

Uma outra aplicação do método Monte Carlo é na integração de funções [30]. Esta foi uma das primeiras aplicações que utilizavam números aleatórios em sua solução. Suponha que $g(x)$ é uma função e quer-se computar θ onde:

$$\theta = \int_0^1 g(x)dx$$

Pode-se utilizar a distribuição uniforme U sobre o intervalo $[0,1]$ para calcular o valor de θ. Pois,

$$\theta = E[g(U)] = \int_0^1 f(u)g(u)du = \int_0^1 g(u)du \text{ pois } f(u) = 1,\ 0 \leq u \leq 1$$

Se $U_1, U_2, \ldots, U_k$ forem variáveis aleatórias independentes com distribuição uniforme $[0,1]$ deduz-se que as variáveis aleatórias $g(U_1), g(U_2), \ldots, g(U_k)$ são independentes e identicamente distribuídas com média θ. Então, pela lei dos grandes números, segue que, com probabilidade 1,

$$\sum_{i=1}^{k} \frac{g(U_i)}{k} \rightarrow E[g(U)] = \theta, \text{ quando } k \rightarrow \infty$$

Logo, pode-se estimar θ gerando um grande número de amostras aleatórias U_i e fazendo a aproximação através da média dos $g(U_i)$.

O uso de números aleatórios para aproximar o valor de integrais se torna mais interessante no caso de integrais multidimensionais. Suponha que g é uma função de n dimensões e que deseja-se calcular

$$\theta = \int_0^1 \int_0^1 \cdots \int_0^1 g(u_1, u_2, \ldots, u_n)du_1, du_2, \ldots du_n$$

A idéia de usar a aproximação de Monte Carlo para estimar θ está no fato de que θ pode ser obtido através do seguinte valor esperado:

$$\theta = E[g(U_1, U_2, ..., U_n)]$$

onde $U_1, U_2, ..., U_n$ são variáveis aleatórias independentes uniformes $[0,1]$. Muitas vezes esta é a única forma eficiente de calcular o valor de tais integrais [24].

2.2.2 Simulação de Eventos Discretos

A simulação de eventos discretos é a técnica mais utilizada em simulações de sistemas. Esta técnica é a generalização do método Monte Carlo para simulação de modelos probabilísticos. A simulação de um modelo probabilístico envolve a geração do processo estocástico associado ao modelo e à observação do caminho amostral feito pelo processo ao longo do tempo. Durante a simulação, medidas de interesse definidas são acompanhadas e atualizadas. Porém, a evolução do modelo no tempo pode envolver complexas estruturas lógicas que definem seu comportamento, muitas vezes tornando difícil o acompanhamento de tais medidas de interesse. Um framework genérico, que foi desenvolvido em cima da idéia de "eventos discretos", permite seguir o comportamento do modelo ao longo do tempo facilitando o cálculo das medidas de interesse relevantes.

Os principais elementos em uma simulação de eventos discretos são as variáveis e os eventos. No desenrolar da simulação acompanha-se a evolução dessas variáveis que são modificadas por disparo de eventos. Genericamente, três tipos de variáveis são quase sempre utilizadas:

1. **Tempo de simulação** - Mede a quantidade de tempo simulado até o momento
2. **Contadores** - Contam o número de vezes que determinados eventos de interesse ocorreram
3. **Estado do sistema** - descreve o estado do sistema no instante atual

Sempre que um evento ocorre, os valores das variáveis acima são atualizados e informações relevantes às medidas de interesse são computadas. Para determinar o próximo evento a ser disparado, uma lista de eventos ordenada crescentemente no tempo é mantida. Os elementos desta lista informam o tipo do evento a ser disparado e seu tempo de disparo. O disparo de um evento pode escalonar novos eventos, que serão incluídos na lista de eventos. Desta forma é possível acompanhar a evolução do sistema no tempo.

Sistema com Única Fila/Servidor

Um exemplo clássico de simulação de eventos discretos aplica-se no modelo composto por uma única fila e um único servidor. Considere um sistema onde clientes chegam de acordo com alguma distribuição C. Ao chegar, se o cliente encontrar o servidor disponível ele entra em serviço imediatamente. O tempo que um cliente passa em serviço também possui uma distribuição qualquer S. Se ao chegar o cliente encontra o servidor ocupado, então ele se junta a uma fila. Esta fila é composta por clientes que estão aguardando serviço. Quando o servidor termina de servir um cliente, imediatamente outro cliente começa a ser servido, caso haja mais clientes na fila. A política de escolha do próximo cliente a ser atendido também é pré-determinada, como por exemplo a política FCFS (primeiro a chegar será o primeiro a ser servido). Descrito o sistema, suponha que há interesse no tamanho médio da fila e no tempo médio que um cliente espera até que seja atendido.

Para simular tal sistema serão utilizadas as variáveis tempo de simulação, número de chegadas N_c, número de saídas N_s e a variável de estado n que indica quantos clientes estão na fila. Essas variáveis serão atualizadas sempre que ocorrer a chegada ou a saída de um cliente. Logo, esses serão os dois eventos possíveis da simulação. A lista de eventos neste caso conterá apenas dois eventos que indicam o tempo da próxima chegada e o tempo da próxima saída.

A Figura 2.4 mostra a evolução do sistema e das variáveis. Não é difícil fazer um programa de computador que simule o processo descrito acima e acompanhe a evolução das variáveis no tempo, obtendo as respectivas medidas de interesse.

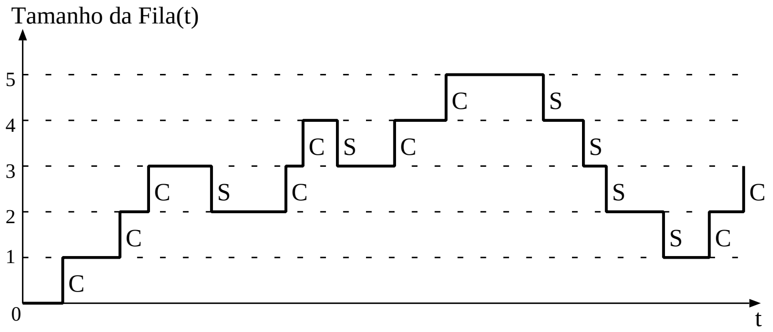


Figura 2.4: Evolução de uma fila com chegadas e saídas de pacotes.

2.2.3 Simulação de Eventos Raros

Apesar de ser considerada uma forma viável de se avaliar performance de sistemas complexos de computação e comunicação, a técnica de simulação possui algumas deficiências. Por exemplo, quando redes de computadores possuem valores para as medidas de serviço extremamente pequenas, o uso de simulação pode não ser eficiente. Isto ocorre pois a medida de interesse a ser calculada pelo simulador decorre de uma seqüência de eventos raros.

A necessidade de medir eventos que possuem probabilidades muito baixas aparece em diversos sistemas como redes de alta velocidade e sistemas ultraconfiáveis com redundância. Probabilidades na ordem de 10^{-10} são normalmente utilizadas para especificar a taxa de perda de células de comutadores em redes ATM. O uso de simulação convencional é impraticável para estimar probabilidades tão baixas. Para estimar probabilidades na ordem de 10^{-10} com uma confiança aceitável seria necessário simular pelo menos dez bilhões de eventos, o que corresponderia a ocorrência de aproximadamente 100 eventos raros. A um tempo computacional de 0.1 mseg por evento, esta simulação levaria 3 anos para terminar [35].

Estimar probabilidades relacionadas a eventos raros é um dos maiores desafios encontrados pelo método de simulação Monte Carlo. Surgem então, novas oportunidades para desenvolver estudos visando melhorar a eficiência da simulação através de novas técnicas.

Existem várias formas de aumentar a velocidade de uma simulação orientada a eventos discretos de um sistema. A Figura 2.5 ilustra duas abordagens que podem ser utilizadas para classificar essas técnicas. O gráfico A apresenta a evolução de uma simulação convencional. Os gráficos B e C ilustram duas abordagens utilizadas para aumentar a velocidade da simulação.

1. Aumento do número de eventos por unidade de tempo (Figura 2.5 parte B) - obtido através do uso de uma máquina com maior poder computacional ou explorando a computação paralela/distribuída em diversos processadores/computadores

2. Redução do *overhead* de simulação (Figura 2.5 parte C) - obtido aumentando o número de eventos de interesse através de alguma propriedade estatística do modelo

Dentro dessas duas classes, as técnicas para aumentar a eficiência da simulação

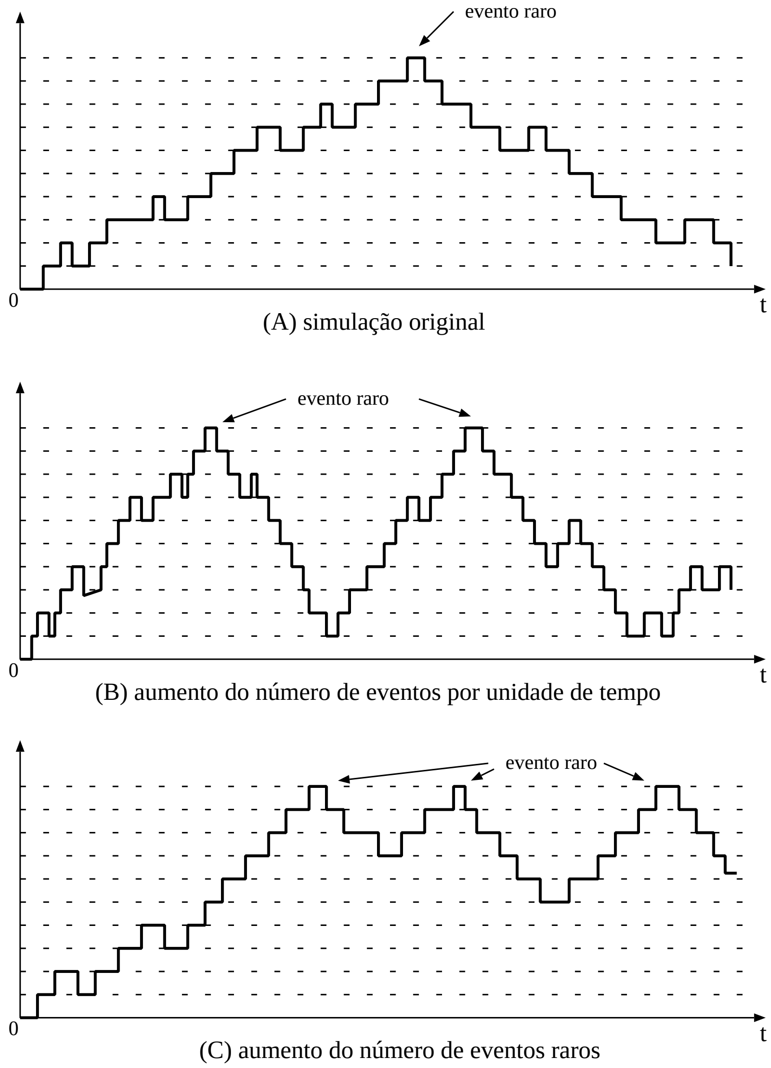


Figura 2.5: Duas formas de aumentar a eficiência em simulação.

podem ser assim divididas [21]:

- Simulação Paralela e Distribuída - Explora o uso de software e/ou hardware especial que permite paralelizar/distribuir a simulação.

- Técnicas Híbridas - Utilizam métodos numéricos ou analíticos para solucionar parte do modelo e combinam os resultados com simulação para resolver o sistema por inteiro.

- Minimização da Variância - Explora as características probabilísticas do sistema através de correlação entre medidas.

- Provocação de Eventos Raros - Aumenta a freqüência relativa com a qual os eventos raros de interesse ocorrem.

Vários estudos sobre todas essas técnicas foram feitos. Cada uma delas possui vantagens e desvantagens que são apresentadas e discutidas amplamente na literatura. A maioria dos estudos concentram-se em torno das técnicas de provocação de eventos raros, por apresentarem resultados otimistas e por poderem ser aplicadas a uma grande variedade de modelos.

Provocação de Eventos Raros

Algumas técnicas que aumentam a eficiência da simulação não se aplicam quando o cálculo da medida de interesse envolve a ocorrência de eventos raros. Pois estas técnicas não modificam a probabilidade de ocorrência do evento raro, sendo necessário um tempo muito longo de simulação para estimar com precisão adequada a medida de interesse.

Com a provocação de eventos raros, o comportamento estatístico do sistema sendo simulado é alterado fazendo com que os eventos raros ocorram com mais freqüência. Duas dessas técnicas são apontadas na literatura: *importance sampling* e *splitting/RESTART*.

Importance Sampling

Nos últimos anos, vários estudos bem sucedidos para realizar simulações baseadas em *importance sampling* foram feitos [29]. Esta técnica permite um aumento muito significativo na eficiência da simulação, e vem sendo utilizada, particularmente, na solução de sistemas de comunicação (sistemas de filas) e sistemas confiáveis.

A idéia básica é simples e consiste em simular o sistema com diferentes distribuições de probabilidade (ou dinâmica de probabilidade, uma vez que essas distribuições podem ser alteradas durante a simulação) para aumentar a possibilidade de uma seqüência típica de eventos ocorrer levando a um evento raro. A medida obtida através da observação deste sistema é então multiplicada por um fator de correção conhecido como *"likelihood ratio"*, para se obter uma estimativa sem bias (*unbiased*). Este fator é a razão entre as probabilidades do caminho amostral original e o modificado. A computação deste método é simples e pode ser feita recursivamente durante a simulação [29]. A escolha apropriada das novas distribuições de probabilidade do sistema a ser simulado permite uma redução significativa no tempo de simulação necessário para atingir a precisão especificada.

O desafio em usar IS (*Importance Sampling*) está em escolher as novas distribuições que minimizem a variância do estimador e levem a eventos raros. Se a distribuição escolhida estiver longe do ótimo então simulações ineficientes ocorrerão produzindo resultados piores que a distribuição original. Encontrar a mudança correta nas distribuições, geralmente requer identificar, pelo menos assintoticamente, as probabilidades do evento raro, muitas vezes descritas por resultados da teoria de *large deviation*. Este tipo de análise pode ser bastante difícil em modelos complexos restringindo o domínio de problemas onde esta técnica pode ser aplicada. Pesquisadores têm procurado encontrar heurísticas para determinada classe de modelos [22].

Splitting/RESTART

Recentemente, vários trabalhos na área de simulação revisitaram a técnica de *splitting* que foi originalmente descrita por Bayes [1]. Bayes chamou seu método de *Importance Sampling*, conflitando com a atual nomenclatura. A técnica ressurgiu com as modificações propostas por [34] que permitem simular modelos com eventos raros independentemente das características estatísticas do sistema. Villén-Altamirano et al chamaram a sua técnica de RESTART (REpetitive Simulation Trials After Reaching Thresholds). Uma das grandes vantagens desta técnica é a simplicidade da análise prévia do sistema a ser simulado. Assim sendo, esta pode ser aplicada a uma grande variedade de modelos.

A seguir a idéia básica deste método é apresentada. Suponha um dado evento raro A. Deseja-se estimar sua probabilidade, $P[A]$. Seja C um conjunto de eventos tal que $C \supset A$ e que $P[A] \ll P[C] \ll 1$.

Então $P[A] = P[C].P[A|C]$, pois $P[A|C] = P[A.C]/P[C] = P[A]/P[C]$, pois A está contido em C.

A Figura 2.6 ilustra o comportamento deste sistema. Em simulação convencional, $P[C]$ é melhor estimado do que $P[A|C]$ pois $P[C]$ é atingido mais freqüentemente do que $P[A|C]$. O método splitting consiste em melhorar a estimativa de $P[A|C]$ simulando repetidas vezes, a partir do ponto onde C ocorre. Isto leva a um aumento de confiança na estimativa de $P[A]$ com um custo computacional relativamente baixo.

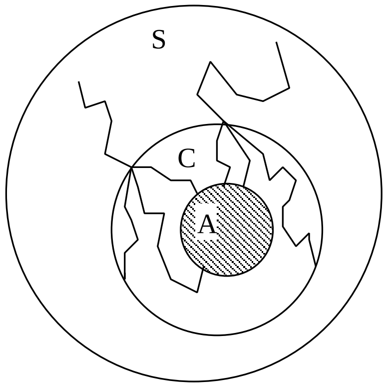


Figura 2.6: Idéia básica da técnica de Splitting/RESTART.

A técnica de RESTART foi expandida e definida com múltiplos níveis onde novas simulações são realizadas. Assim, supondo eventos múltiplos C_i que satisfazem $C_1 \supset C_2 \supset C_3 \supset \ldots \supset C_M \supset A$. Então segue que $P[A] = P[C_1].P[C_2|C_1]. \ldots .P[C_M|C_{M-1}]$

Em sua proposta original, o método de splitting possuía apenas um ponto de quebra, dando origem, conseqüentemente, a apenas dois níveis. Com a introdução de múltiplos níveis o desempenho do método sofreu significativas melhorias e várias pesquisas se voltaram para esta técnica.

Descrição do Método

A idéia básica do método RESTART é identificar subconjuntos dos quais se torna mais fácil atingir um determinado estado (ou conjunto de estados) de interesse, sendo este último o estado raro, ou o evento raro. Cada vez que a simulação atinge um novo subconjunto, o estado atual da simulação é replicado um certo número de vezes e todos continuam a evoluir de forma independente, seguindo caminhos amostrais distintos. Desta forma o número de eventos raros será proporcional à definição desses conjuntos e ao número de réplicas criadas em cada subconjunto.

A Figura 2.7 parte A ilustra um exemplo de simulação convencional. Esta evolui no tempo atravessando os diversos estados do sistema. Deseja-se estimar a probabilidade de ocorrência do evento A, que está definida pelo conjunto de estados do sistema que assumem valores maiores ou iguais a L. A Figura 2.7 parte B ilustra um exemplo da mesma simulação utilizando a técnica de RESTART. Nota-se que os eventos raros de interesse têm maiores chances de ocorrer em B.

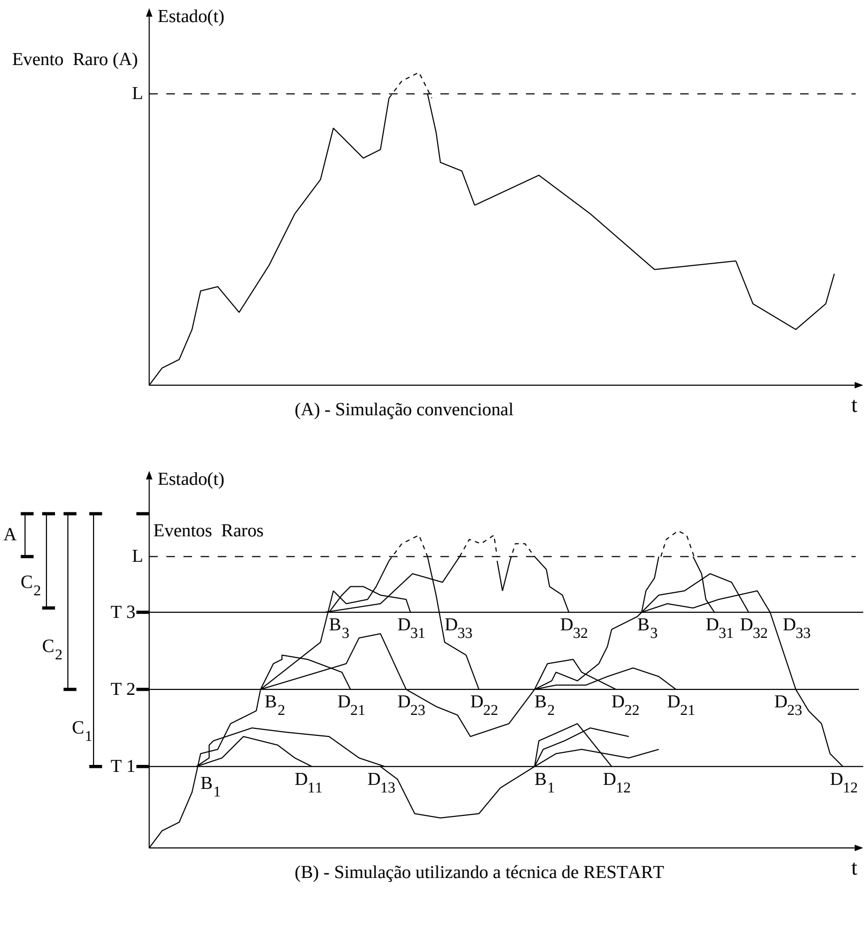


Figura 2.7: Exemplo de uma execução de uma simulação utilizando a técnica de RESTART.

Formalmente, define-se M conjuntos de eventos C_i e um conjunto de eventos raros de interesse A satisfazendo $C_1 \supset C_2 \supset \ldots \supset C_M \supset A$. Esta definição é feita associando estados do sistema aos extremos superior e inferior de cada um dos conjuntos. Desta forma, o espaço de estados é dividido por *thresholds* (pontos de quebra) que definem o limite entre um conjunto de eventos e outro. Dois pontos no tempo importantes que ocorrem durante a simulação podem ser identificados:

- B_i - instante da transição de um estado em $\bar{C}_i$ para um estado em C_i. Por exemplo, B_1 é o instante em que uma simulação transiciona de um estado em $\bar{C}_1$ para C_1.
- D_{ij} - instante da transição de um estado em C_i para um estado em $\bar{C}_i$ da j-ésima réplica ocorrida em B_i. Por exemplo, D_{12} é o instante em que a segunda réplica criada em B_1 transiciona de um estado em C_1 para $\bar{C}_1$.

O procedimento utilizado durante a execução do método segue:

- Quando um evento que acarreta em B_i ocorre, o estado do sistema é armazenado
- Quando um evento que acarreta em D_{ij} ocorre, o estado do sistema em B_i é restaurado e um novo intervalo $[B_i, D_{ij}]$ é simulado
- O processo acima é repetido R_i vezes (número de replicas para o *threshold* i). O ponto de partida de uma réplica é sempre em B_i enquanto os pontos de término são $D_{i1}, D_{i2}, \ldots, D_{iR_i}$
- Quando o evento D_{iR_i} ocorre, a simulação continua seguindo o procedimento acima até a de um novo B_1, ou de um D_{i-1j}. A escolha de continuar a última réplica faz parte desta especificação do método RESTART
- Durante a simulação do subconjunto C_i, um evento B_{i+1} pode ocorrer e R_{i+1} réplicas iniciando em B_{i+1} devem ser simuladas antes do simulador retornar ao nível i
- A ocorrência de eventos de interesse deve ser registrada durante todo o processo de simulação. Ao terminar, as estatísticas devem ser modificadas dividindo o

número de ocorrências de eventos de interesse pelo número de réplicas afetando sua medição

Este algoritmo é executado repetidamente até que a simulação atinja um certo limite de tempo. Se este limite de tempo for grande, então teremos uma estimativa do comportamento do modelo em estado estacionário.

Porém, em alguns modelos, o objetivo é analisar o comportamento do sistema em um determinado período de tempo, ou seja, em estado transiente. Neste caso, interessa estimar a probabilidade de um evento (raro) ocorrer no período $[0, t_e]$. Com simulação convencional o sistema é simulado repetidamente no intervalo $[0, t_e]$.

Uma pequena alteração do algoritmo permite utilizar a técnica de RESTART para realizar simulações em estado transiente. Para isto modifica-se a definição dos instantes D_i. Pois agora D_i ocorre no instante em que o simulador faz uma transição de um estado em C_i para um estado em $\bar{C}_i$ ou quando o tempo de simulação referente àquela réplica é maior ou igual a t_e. Neste caso a simulação termina quando não há mais réplicas para simular e o tempo de simulação, de todas as réplicas, é maior ou igual a t_e [35]

Observa-se que durante a simulação, todos os eventos de interesse são registrados independente da réplica que os atinge, e ao final da simulação a medida de interesse precisa ser ponderada para que o valor da estimativa esteja correto. Esta alteração na medida é feita da seguinte forma: o estimador da probabilidade de ocorrência do evento raro A no simulador convencional é dada por

$$P[A] = \frac{N_A}{N},$$

onde N_A é o número de ocorrências do evento A durante a simulação e N é o número total de eventos ocorridos durante a simulação.

Então, este estimador utilizando o método RESTART será dado por

$$P[A] = \frac{N_A}{N \prod_{i=1}^{M} R_i},$$

onde N_A é a soma do número de ocorrências do evento A de todas as réplicas e N inclui apenas o número de eventos simulados na primeira réplica de cada subconjunto.

R_i é o número de réplicas realizadas no nível i e M é o numero total de níveis.

Uma prova informal para o estimador acima será apresentada agora. Deseja-se estimar a $P[A]$ do modelo ilustrado pela Figura 2.8. Esta figura apresenta uma execução da simulação utilizando o método RESTART com apenas 1 ponto de quebra. Este ponto de quebra define os estados do conjunto $\mathcal{C}$. Os estados de interesse estão definidos pelo conjunto $\mathcal{A}$, de tal forma que $\mathcal{C} \supset \mathcal{A}$. O número de réplicas ao atingir o *threshold* T é R, e neste exemplo igual a 2. O valor t é o tempo total de simulação. O número de quebras que ocorreram durante a simulação é dado por N, e neste exemplo igual a 3.

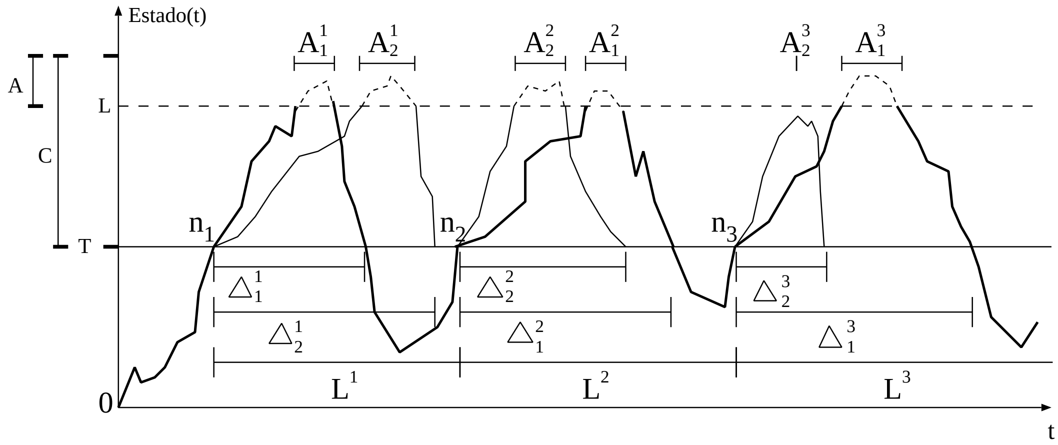


Figura 2.8: Exemplo de uma execução de uma simulação utilizando a técnica de RESTART.

Para desenvolver a prova precisa-se dos seguintes teoremas limites. Seja $\mathcal{X}(t)$ o processo simulado, ilustrado na figura 2.8 pelo caminho amostral em negrito. Seja η_i a i-ésima vez que o processo $\mathcal{X}(t)$ atinge o *threshold* T. Suponha que o estado de entrada no conjunto $\mathcal{C}$ é sempre o mesmo, porém, esta restrição pode ser facilmente generalizada e não é apresentada para simplificar esta demonstração.

Define-se então os pontos embutidos de $\mathcal{X}(t)$ os pontos $0, \eta_1, \eta_2, \ldots$. Seja L^i o intervalo de tempo entre os pontos embutidos η_i e $\eta_{i+1}, i \geq 1$. Seja Δ_1^i o tempo que o processo $\mathcal{X}(t)$ permanece no subconjunto $\mathcal{C}$ durante o intervalo L^i.

Seja $O^k = \sum_{i=1}^{k} L^i$ então tem-se que

$$\lim_{k \to \infty} \frac{O^k}{k} = E[L], \tag{2.1}$$

onde L é a variável aleatória que representa o intervalo de tempo entre a ocorrência de dois pontos embutidos. O mesmo se aplica a $C^k = \sum_{i=1}^{k} \Delta_1^i$ onde

$$\lim_{k\to\infty} \frac{C^k}{k} = E[C]$$

Considere agora o seguinte teorema:

$$P[C] \stackrel{c.p.1}{=} \lim_{t\to\infty} \frac{C(t)}{t} \tag{2.2}$$

onde $C(t)$ é o tempo total permanecido no conjunto $\mathcal{C}$ durante o intervalo $[0, t]$.

Seja $\mathcal{R}$ um subconjunto de estados quaisquer. Segue o teorema

$$P[R] = \frac{E[R]}{E[L]} \tag{2.3}$$

onde R é a variável aleatória que indica o tempo que o processo $\mathcal{X}(t)$ passa no subconjunto $\mathcal{R}$ entre pontos embutidos.

Voltando agora à figura 2.8, define-se agora os valores de tempo A_j^i como sendo a quantidade de tempo que a j-ésima réplica, referente a i-ésima quebra, passou dentro do conjunto $\mathcal{A}$. Os valores de tempo Δ_j^i definem a quantidade de tempo que a j-ésima réplica, referente a i-ésima quebra, passou dentro do conjunto $\mathcal{C}$.

Partimos da equação $P[A] = P[C|A].P[C]$ para estimar $P[A]$, estimando $P[C]$ e $P[A|C]$ separadamente. Desta forma, olhando para a simulação principal (em negrito na Figura 2.8) define-se que

$$P[C] = \frac{\sum_{i=1}^{N} \Delta_1^i + \varepsilon}{t}$$

pelo teorema definido na equação 2.2. O valor ε é o tempo permanecido no subconjunto $\mathcal{C}$ entre o último ponto embutido e t. Ainda, $P[C]$ também pode ser estimado da seguinte forma

$$P[C] = \frac{E[C]}{E[L]} = \frac{\frac{\sum\limits_{i=1}^{N} \Delta_1^i}{N}}{E[L]}$$

pelo teorema apresentado na equação 2.3.

Podemos agora calcular $P[A|C]$ da seguinte forma, utilizando os teoremas enunciados nas equações 2.1 e 2.3

$$P[A|C] = \frac{E[A]}{E[C]} = \frac{\dfrac{\sum_{i=1}^{N}\sum_{j=1}^{R} A_j^i}{NR}}{\dfrac{\sum_{i=1}^{N} \Delta_1^i}{N}}$$

Substituindo os dois termos de volta na equação da estimativa de $P[A]$ e arrumando os seus termos,x tem-se

$$\begin{aligned} P[A] = \; & P[A|C].P[C] \\ & \frac{\sum_{i=1}^{N}\sum_{j=1}^{R} A_j^i}{R\sum_{i=1}^{N} \Delta_1^i} \left(\frac{\sum_{i=1}^{N} \Delta_1^i}{t} + \frac{\varepsilon}{t} \right) \\ = \; & \frac{\sum_{i=1}^{N}\sum_{j=1}^{R} A_j^i}{Rt} + \frac{\varepsilon'}{t} \end{aligned} \tag{2.4}$$

O valor do último termo desta equação vai a zero pois podemos fazer t muito maior que ε. Este resultado é o cálculo utilizado para obter as medidas de interesse do simulador de eventos raros através da técnica de RESTART.

Este mesmo raciocínio pode ser estendido a simulações com múltiplos *thresholds*, sendo fácil concluir seu resultado. Porém, por possuir múltiplos *thresholds*, um novo termo aparece no denominador, e segue direto do desenvolvimento das equações que

$$P[A] = \frac{\sum_{j=1}^{N}\sum_{i=1}^{N} A_j^i}{t\prod_{i=1}^{N_t} R_i}$$

onde N_t é o número de *thresholds* definidos para a simulação. Este resultado é apresentado por [35] como sendo a estimativa correta para o simulador de eventos raros que utiliza o método de RESTART.

Eficiência e Robustez do Método

Vários estudos visando analisar o desempenho e a eficiência da técnica de splitting/RESTART foram feitos [35], [18], [20]. Esses estudos mostram que o método é bastante flexível e robusto, podendo ser aplicado a simulações transientes e em estado estacionário.

Em [35], uma expressão para o ganho da simulação de eventos raros é definida. Esta expressão é dada pela variância do trabalho-normalizado, que pondera o esforço computacional e a variância da medida sendo estimada. Esta expressão é dada pelo produto da variância do estimador pelo trabalho realizado em cada rodada de simulação. Em [18] uma análise deste critério é justificada. Desta forma, o ganho obtido com o método RESTART com respeito a simulação convencional é dada por

$$G = \frac{V(\hat{P})_{(C)}.C_{(C)}}{V(\hat{P})_{(R)}.C_{(R)}}$$

onde $V(\hat{P})$ é a variância do estimador de eventos raros $\hat{P}$, C é o custo da simulação em termos de tempo de processamento e os subíndices (C) e (R) indicam simulação convencional e RESTART, respectivamente.

Expressões para $V(\hat{P})$ e C foram obtidas em [35] dando origem a uma expressão para o ganho obtido com o método RESTART. Maximizando esta expressão em função de alguns parâmetros do método e assumindo que alguns outros parâmetros são constantes, chega-se a valores ótimos para esses parâmetros.

$$P_i = e^{-2}, R_i = e^2$$

onde P_i é dado por $P[C_i|C_{i-1}]$, $1 < i \leq M$, $P_1 = P[C_1]$; R_i, $1 \leq i \leq M$, é o número de réplicas feitas ao passar pelo *threshold* i, e M é o número de níveis. Esta solução ótima encontrada em [35] possui duas propriedades interessantes:

- Os valores de P_i e R_i são completamente independentes das características do sistema tornando o método aplicável à simulação de qualquer sistema
- Esses valores também independem de P_A (probabilidade do evento raro ocorrer). Desta forma o método pode ser aplicado em simulações onde a ordem de grandeza da probabilidade do evento raro é desconhecida

Uma abordagem completamente diferente é feita por [18]. Nesta análise, uma outra medida de interesse é definida juntamente com um estimador para esta medida. Uma variação do método RESTART é analisado e o ganho deste método é calculado. O estudo também apresenta valores ótimos para o números de réplicas e local dos pontos de quebra. Coincidentemente, as conclusões tiradas neste trabalho estão de acordo com as análise descritas acima em [35].

Em [21], uma generalização das análises é feita chegando a seguinte conclusão. Para maximizar o ganho do método e minimizar a variância do estimador $\hat{P}$, os parâmetros para a simulação devem seguir as regras:

- R_i - o número de réplicas em cada nível deve estar de acordo com $R_i = 1/P_i$
- M - o número de *thresholds* deve ser escolhido de tal forma que $P_i = P = e^{-2}$ para todo $i = 1, \ldots, M$. A definição dos conjuntos C_i deve respeitar estas equações e M será consequência deste cálculo
- T_i - os pontos de quebra (*thresholds*) devem ser definidos dentro dos critérios para definição dos níveis

O critério para definição dos pontos de quebra e para cálculo do número de réplicas apresentados acima não precisam ser seguidos a risca. Em [35] um estudo sobre a robustez do método é apresentado quando parâmetros não ótimos são utilizados, até mesmo quando valores para P_i estão longe de e^{-2}. Sua robustez quanto a variação de seus parâmetros é uma grande vantagem do método RESTART. Isto permite definir os pontos de quebra mesmo sem saber quais são os estados do sistema ótimos para definição desses pontos, ou mesmo quando as probabilidades associadas as transições entre os estados do sistema não permitem que P_i assuma o valor exato (ou muito próximo) de e^{-2}. Esta propriedade torna o uso do método RESTART simples e flexível.

A escolha do parâmetro R_i é mais delicada e o ganho obtido com o método possui uma maior sensibilidade a este parâmetro. Porém estudos em [25] mostram que mesmo para valores de R_i longe do ótimo o método ainda é viável e superior a simulação convencional. Além disso, não é difícil escolher valores para R_i que estejam próximos a seu valor ótimo. Dois métodos práticos para escolha desse parâmetro são:

- baseado em rodadas de testes. Defini-se valores iniciais para T_i e R_i e uma rodada rápida de simulação é feita. Os resultados obtidos são utilizados para reajustar os valores de T_i e R_i
- auto-ajustado. O próprio simulador adiciona, remove e move os valores de T_i e altera os valores de R_i ao longo da simulação em tempo de execução. Em [35], um esboço deste esquema é apresentado.

Entretanto, o ganho obtido pode ser significativamente reduzido se os parâmetros R_i estão longe dos valores ótimos. Além disso, a definição dos *thresholds* pode se tornar uma tarefa difícil quando a dimensão e o tamanho do modelo a ser simulado é grande e não existe nenhuma simetria que possa ser explorada. Se o espaço de estados do modelo for multidimensional então a definição dos pontos de quebra pode ser uma tarefa complicada resultando em pontos de quebra multidimensionais. Porém, em um modelo multidimensional é possível definir uma única variável de estado para ser observada que indicará a proximidade aos eventos raros.

Capítulo 3

Especificação de Modelos com Recompensas

MODELOS com recompensas têm sido muito utilizados ultimamente por serem flexíveis e permitirem a definição de medidas de interesse de forma genérica. O simulador desenvolvido e acoplado à ferramenta TANGRAM-II fará uso das medidas de interesse especificadas através de recompensas. Este simulador e sua arquitetura é descrito no próximo capítulo. Neste capítulo, serão discutidos modelos de recompensas e especificação de medidas de interesse. Na seção 1 expõe-se a idéia desta classe de modelos e possíveis medidas genéricas obtidas. Já nas seções 2 e 3 mostra-se como especificar e utilizar recompensas de taxa e de impulso na ferramenta TANGRAM-II, respectivamente. Na seção 4 apresenta-se algumas formas de solução dessas medidas.

3.1 Modelos com Recompensas

Modelos Markovianos com recompensa têm sido frequentemente empregados para obter medidas de performance de sistemas de computação e comunicação [9].

De início, a modelagem de sistemas concentrou-se em desenvolver técnicas para analisar exclusivamente um dos seguintes aspectos: (a) confiabilidade ou disponibilidade do sistema ao usuário; ou (b) performance do sistema, supondo que ele é perfeitamente confiável.

O modelo para confiabilidade/disponibilidade representa as alterações que podem ocorrer na estrutura do sistema. Entre as medidas mais importantes que podem ser obtidas estão a probabilidade de que o o sistema esteja operacional durante um período de tempo, e a fração de tempo em que o sistema pode executar trabalho.

O modelo de performance, por outro lado, representa a capacidade do sistema de executar trabalho, assumindo que não ocorrem mudanças estruturais. O modelo deve capturar a natureza probabilística das demandas do usuário para um conjunto de recursos do sistema, e a conseqüente contenção por estes recursos. O efeito da contenção por recursos é medido em termos de quantidades tais como *throughput*, tempo médio para execução de tarefas, etc.

Considere agora, a análise de performance que tem como objetivo obter informações e medidas sobre a quantidade de trabalho útil que um sistema pode oferecer ao usuário em sistemas que podem sofrer alterações em sua estrutura, devido a falhas e reparos de seus componentes. Após uma falha ou reparação, o sistema pode continuar a executar trabalho, possivelmente em um nível de capacidade diferente. Estes sistemas são chamados de sistemas degradáveis.

Para sistemas degradáveis, a performance não pode ser avaliada com precisão sem levar em conta o impacto das mudanças estruturais. Portanto, é necessário definir novas medidas que combinem confiabilidade e performance.

Meyer [27] introduziu um modelo geral para a definição e avaliação de novas medidas de interesse, chamadas de medidas de *performability*. Neste modelo, um processo estocástico descreve a evolução da estrutura do sistema em um intervalo finito de tempo. A cada caminho u do processo, é associada uma função $\gamma_t(u)$ que indica o nível de performance do sistema em u. Claramente, não é prático associar um nível de performance a cada caminho no modelo. A idéia é associar níveis de performance aos estados e às transições do modelo.

Assim, define-se o modelo chamado cadeia de Markov com recompensas. Neste modelo, uma cadeia de Markov de tempo contínuo descreve as mudanças na estrutura do sistema. Aos estados da cadeia são associadas recompensas, que representam a taxa com que o sistema acumula performance, quando se encontra neste estado. Também é possível associar recompensas a transições da cadeia, representando impulsos acumulados quando o sistema realiza uma determinada transição. A partir deste modelo, podem ser obtidas medidas como a distribuição da recompensa acumulada em um intervalo $(0, t)$.

Modelos com recompensas podem ser utilizados para calcular diversas medidas de interesse. Além das medidas de *performability*, várias outras características do modelo podem ser obtidas através do uso de recompensas. Por exemplo, medidas como a utilização, o *throughput*, tamanho médio de filas, etc. Uma definição genérica para medidas de recompensas é apresentada a seguir:

Considere um sistema (i.e. cadeia de Markov) com espaço de estados finito $S = \{s_i, i = 1, \ldots, M\}$. A cada estado $s \in S$ está associado uma taxa de recompensa de um conjunto $R = \{r_i, \ldots, r_{K+1}\}$, onde r_i representa a recompensa ganha por unidade de tempo em que o sistema se encontra no estado i. Seja a variável aleatória $IR(t)$ que representa a recompensa instantânea no tempo t.

Recompensa de Taxa Acumulada

A recompensa acumulada durante o intervalo $[0, t]$ é dada por

$$CR(t) = \int_0^t IR(\tau) d\tau$$

Dividindo o valor da recompensa acumulada pelo tempo de observação, obtem-se a média

$$ACR(t) = \frac{CR(t)}{t}$$

Pode-se também, definir a média ponderada pelo tempo da recompensa acumulada o que leva a

$$TCT(t) = \frac{\int_0^t CR(\tau) d\tau}{t}$$

Recompensa de Impulso Acumulada

Analogamente, é possível definir as medidas acima para recompensas associadas a transições feitas sobre o espaço de estado do sistema. Cada transição pode ou não, ter associado a ela um valor, ganho quando a transição é feita, definido como impulso.

Seja $\sigma_n = (s', s)$ a n-ésima transição feita pelo sistema e que ocorre do estado s' para o estado s. Define-se a variável ρ_{σ_n} como sendo o valor do impulso associado a σ_n.

Então a recompensa de impulso acumulada no período $[0, t]$ é

$CI(t) = \sum_{n=0}^{N(t)} \rho_{\sigma_n}$, onde N(t) é o número de transições feitas até o tempo t

Novamente, dividindo o valor da recompensa acumulada pelo tempo de observação, obtem-se a média

$$ACI(t) = \frac{CI(t)}{t}$$

Também é possível definir a média ponderada pelo tempo da recompensa acumulada o que leva a

$$TCI(t) = \frac{\int_0^t CI(\tau)d\tau}{t}$$

Essas medidas de interesse são bastante genéricas e podem ter diversas interpretações, dependendo da definição feita pelo usuário. Na ferramenta TANGRAM-II pode-se definir recompensas de taxa e de impulso dando significado às medidas apresentadas. As próximas seções mostram como fazer esta especificação de recompensas na ferramenta.

3.2 Recompensas de Taxa

Como já foi visto, para se obter medidas de interesse utilizando modelos com recompensas de taxa é necessário atribuir a cada estado do sistema um valor que indica a quantidade de recompensa que a medida está acumulando por unidade de tempo em que permanecer naquele estado. Por exemplo, em um modelo onde se quer medir o *throughput* total do sistema, a recompensa em cada estado pode determinar a quantidade de *throughput* que o sistema está obtendo naquele momento (estado). Assim, medidas como o *throughput* total médio durante um período de observação podem ser facilmente calculadas.

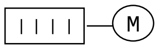
Na ferramenta TANGRAM-II as recompensas de taxa estão associadas aos estados dos objetos. Cada objeto pode possuir uma ou mais recompensas atribuídas a ele. Um objeto pode ter mais de uma recompensa, pois pode-se querer definir, sobre o mesmo objeto, diversas medidas de interesse que utilizam recompensas.

Para definir a recompensa o usuário especifica um nome para a sua medida de interesse, que exprime o significado da medida. Após a definição do nome, uma condição para que a recompensa seja atribuída ao estado do objeto é definida. Se esta condição for satisfeita, então um valor é atribuído à recompensa especificada, através de uma expressão aritmética definida pelo usuário. Esta condição é necessária, pois uma determinada recompensa pode não estar definida em todos os estados assumidos pelo objeto. Se a condição não for satisfeita, então a recompensa atribuída a este determinado estado é zero.

O identificador usado para dar nome à recompensa deve ser único no objeto. A condição deve ser uma expressão booleana e pode utilizar variáveis de estado e constantes do objeto. O valor atribuído à recompensa deve ser uma expressão aritmética de ponto flutuante (ou inteiro) que pode utilizar variáveis de estado, constantes e parâmetros definidos dentro do escopo do objeto.

A Figura 3.1 ilustra o exemplo da definição de uma recompensa simples. Neste exemplo, deseja-se medir a utilização do servidor de uma fila. Para isto, atribuímos uma recompensa com valor 1 aos estados onde o servidor realiza trabalho, e 0 aos estados onde o servidor está ocioso. Com isto, ao final de um tempo t, a recompensa "utilizacao" indicará a quantidade de tempo em que o servidor permaneceu ocupado. Logo, $ACR(t)$ é a fração de tempo em que o servidor realizou trabalho, ou seja, sua utilização. Essas e outras medidas que utilizam recompensa podem ser obtidas com facilidade através do módulo de solução e simulação da ferramenta TANGRAM-II. No final deste capítulo apresenta-se todas as medidas que podem ser obtidas pela ferramenta.

```
name=Servidor_Fila
```




```
State_vars=
 fila = 0
Initialization=
  fila = 0
  taxa_serv = 10
  tam_fila  = 2000
  obj_src   = Fonte
  port_in   = pac

Rewards=
 rate_reward= utilizacao
    condition= (fila > 0)
    value= 1;
```

```
Declaration=
 Var
  State: queue;
 Const
  Float: taxa_serv;
  Integer: tam_fila;
  Object: obj_src;
  Message_port: port_in;

Events=
 event=Servico (EXP, taxa_serv)
 condition= (fila > 0)
 action= { int f;
           f = fila - 1;
           set_st("fila", f);
          };

Messages=
 msg_rec=port_in, obj_src
 action= { int f;
           f = fila;
           if (fila < tam_fila)
              f = fila  + 1;
           set_st("fila", f);
        };
```

Figura 3.1: Definição de uma simples recompensa de taxa.

Para que a medida de interesse "utilizacao" acumule recompensa, é necessário que a fila não esteja vazia. Pela lógica do objeto, o servidor está sempre ocupado quando existem clientes na fila. Logo, a condição para acumular recompensa é a existência de clientes na fila. Uma vez satisfeita a condição, o valor da recompensa atribuída ao estado é sempre o mesmo; 1.

A ferramenta TANGRAM-II também permite que o usuário especifique pares de condições e valores para uma mesma recompensa, não sendo, necessariamente, mutuamente exclusivas. Isto permite ao usuário especificar valores distintos para as recompensas dependendo do estado do objeto. Não existe limite para o número de pares condição/valor que podem ser especificados.

A Figura 3.2 ilustra uma fila com multiservidores. Suponha que a utilização desta fila deve ser calculada utilizando recompensas de taxa. Para isto, defini-se a recompensa "utilizacao" de forma a receber valores $fila/N\_servidores$ enquanto existirem servidores ociosos e *1* quando todos os servidores estiverem ocupados. A variável de estado $fila$ indica o tamanho da fila e a constante $N\_servidores$ indica o número máximo de servidores que podem estar atendendo a fila simultaneamente. Logo, quando a fila estiver maior que $N\_servidores$, todos os servidores estão ocupados.

```
name=Servidor_Fila

State_vars=
  fila = 0
Initialization=
  fila = 0
  taxa_serv = 10
  tam_fila  = 2000
  N_servidores = 50
  obj_src   = Fonte
  port_in   = pac

Messages=
 msg_rec=port_in, obj_src
 action= { int f;
          f = fila;
          if (fila < tam_fila)
             f = fila  + 1;
          set_st("fila", f);
        };
```

```
Declaration=
 Var
  State: queue;
 Const
  Float: taxa_serv;
  Integer: tam_fila, N_servidores;
  Object: obj_src;
  Message_port: port_in;

Events=
 event=Servico_1 (EXP, tam_fila*taxa_serv)
 condition= ((fila > 0) && (fila < N_servidores))
 action= { int f;
          f = fila - 1;
          set_st("fila", f);
         };
 event=Servico_2 (EXP, N_servidores*taxa_serv)
 condition= (fila >= N_servidores)
 action= { int f;
          f = fila - 1;
          set_st("fila", f);
         };
Rewards=
 rate_reward  = utilizacao
    condition = (fila < N_servidores)
    value     = fila / N_servidores;
    condition = (fila >= N_servidores )
    value     = 1.0;
```

Figura 3.2: Definição de recompensa de taxa para medir a utilização.

Ao permitir que o usuário especifique vários pares de condições/valores podemos obter estados do objeto onde duas ou mais condições sejam satisfeitas, caso elas não sejam mutuamente exclusivas. Neste caso, existirão dois ou mais valores de recompensas para o mesmo estado. Porém, apenas um valor de recompensa pode ser atribuído a um determinado estado. A ferramenta prevê esta possibilidade e oferece ao usuário algumas opções que permitem definir o valor da recompensa nestes casos:

- média - define como valor da recompensa a média aritmética entre todos os valores das condições que foram avaliadas como verdadeira
- máximo - define como valor da recompensa o máximo entre todos os valores avaliados
- mínimo - define como valor da recompensa o mínimo entre todos os valores avaliados
- erro - gera um erro avisando ao usuário que mais de uma condição foi satisfeita em uma recompensa para o mesmo estado

Uma dessas opções deve ser escolhida pelo usuário antes de iniciar o processo de geração dos estados ou antes de iniciar a simulação. A opção *default* é gerar uma mensagem de erro quando duas ou mais condições forem avaliadas como verdadeiras sobre um mesmo estado do sistema.

3.3 Recompensas de Impulso

Além de recompensa associada aos estados, existe também a possibilidade de atribuir recompensas a mudança de estados no sistema. Ou seja, toda vez que o sistema realiza uma transição de um determinado estado para outro pode-se associar a esta transição um valor chamado recompensa de impulso. Assim, várias medidas de interesse genéricas podem ser definidas com o uso de recompensas de impulso. Por exemplo, para avaliar o número de falhas de um determinado componente de um sistema, pode-se atribuir ao evento "falha" um impulso que caracteriza o acontecimento de uma falha. Toda vez que este evento ocorrer será acumulada uma recompensa indicando mais uma falha. A medida de interesse $CI(t)$ desta recompensa fornece o número de falhas ocorridas até o instante t.

Para se obter a medida de interesse, é necessário marcar todas as transições que influem sobre a medida, atribuindo recompensas a essas transições. Logo, há necessidade de uma forma eficiente para marcar e dar valores a um conjunto de transições. Isto pode ser feito através de uma linguagem para especificação de recompensas de impulsos. A ferramenta TANGRAM-II, permite que o usuário especifique as transições de interesse associando recompensas de impulso ao disparo de eventos ou à chegada de mensagens.

Uma transição na ferramenta TANGRAM-II inicia-se quando um evento ocorre. Porém, esta transição pode desmembrar-se em vários caminhos, caso haja probabilidades associadas às ações que são realizadas quando o evento dispara. Isto ocorre porque cada uma dessas ações pode levar a distintos estados do sistema. O envio de mensagem também pode dar origem a várias transições com destinos diferentes, caso várias ações estejam associadas ao recebimento desta mensagem. Além disso, as transições podem ser "longas", pois uma ação pode enviar mensagens que serão entregues a outros objetos e que, por sua vez, podem enviar outras mensagens.

A Figura 3.3 ilustra este processo de disparo de um evento dando origem a diversas transições com diferentes estados destino através de múltiplas ações e mensagens. Os estados tangíveis do sistema, alcançados após o disparo do evento, não necessariamente, são distintos entre si. Um mesmo estado, após o disparo de um determinado evento, pode ser alcançado de duas ou mais maneiras diferentes, devido à especificação do modelo. Além disso, o estado alcançado pode ser igual ao estado que deu origem ao disparo do evento, criando assim uma transição de um estado para ele mesmo. Desta forma, associar impulsos a eventos e/ou mensagens possibilita a marcação de qualquer transição que o sistema descrito possa fazer.

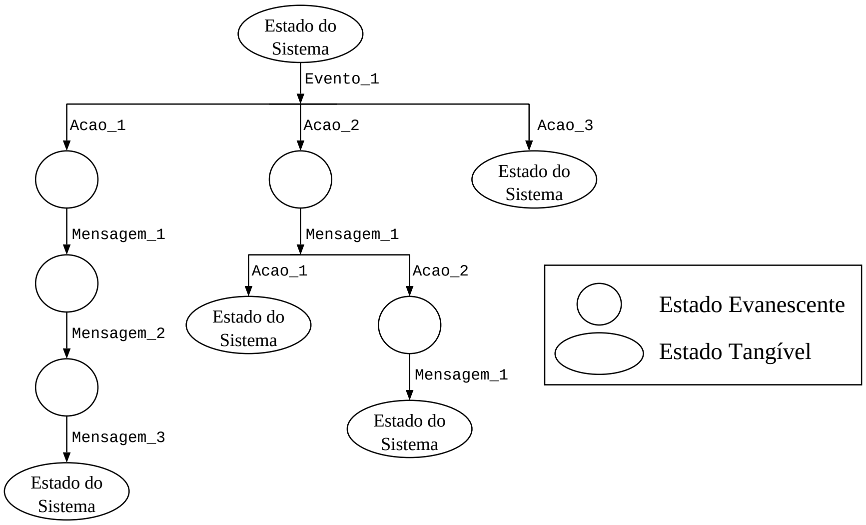


Figura 3.3: Exploração de estados no TANGRAM-II.

Apresenta-se agora como especificar os eventos/mensagens de interesse associando-lhes valores que representam a recompensa. Assim como na definição de recompensas de taxa, para definir recompensas de impulso, o usuário precisa especificar um nome para a recompensa. Este identificador deve ser único no objeto e servir para dar significado à medida de interesse sendo especificada. Em seguida, o usuário terá de especificar o evento (ou mensagem) a que este impulso está associado. Este evento terá que ser um daqueles descritos no objeto. Após a identificação, o usuário irá especificar uma ou mais ações do evento escolhido que ao serem executadas irão acumular recompensa. Esta identificação é feita pelo número da ação. As ações são numeradas na ordem especificada pelo usuário começando com 1. Por final, o usuário indica o valor que esta recompensa irá acumular ao ser avaliada. Este valor é o resultado de uma expressão aritmética que pode conter variáveis de estado, constantes e parâmetros do objeto.

A seguir, uma ilustração de um exemplo simples que utiliza recompensa de impulso é apresentado. A Figura 3.4 mostra um objeto que transmite dois tipos de pacote a ser enviado. Ao enviar um pacote, a fonte escolhe aleatoriamente, com probabilidades p_1 e p_2, o tipo do pacote. Havendo interesse em avaliar a quantidade de pacotes do tipo 2 que estão sendo transmitidos, basta definir uma medida de interesse que associa ao evento "envio de pacote do tipo 2" uma recompensa. Esta medida é obtida através da recompensa de impulso. Define-se então, a recompensa de impulso "tx_tipo_2" e associa-se a ela um único evento "transmissao". Porém, somente pacotes do tipo 2 devem ser considerados, logo apenas a segunda ação deste evento deve ser associada ao impulso. O valor desta recompensa é 1, pois o número de pacotes transmitidos está sendo contado..

```
name=Packet_Source

     P

 State_vars=
  S=0
 Initialization=
   S    = 0
   taxa = 10
   p1   = 0.3
   p2   = 0.7
   port_out= pac
   obj_dest= Queue

Rewards=

 impulse_reward= tx_tipo_2
   event = transmissao, 2
   value = 1;
```

```
Declaration=
 Var
   State: S;

 Const
   Float: taxa, p1, p2;
   Message_port: port_out;
   Object: obj_dest;

 Events=
   event= transmissao(EXP, taxa)
   condition= (TRUE)
   action= {
             msg(port_out, obj_dest, 1);
           }: prob = p1;
           {
             msg(port_out, obj_dest, 2);
           }: prob = p2;

    Messages=
```

Figura 3.4: Definição de recompensa de impulso para contagem de pacotes do tipo 2.

É possível que vários eventos estejam associados à mesma medida de interesse. Suponha que a fonte descrita acima seja modificada para enviar pacotes para dois receptores com diferentes taxas de envio. Existe o interesse em analisar a quantidade de pacotes (tipo 1 e tipo 2) que esta fonte está enviando aos seus receptores. Pode-se então definir uma medida de interesse “tx” definida como recompensa de impulso e associar a ela a ocorrência de qualquer um dos eventos de transmissão. A Figura 3.5 ilustra este exemplo. Note-se que há o interesse em contar todos os pacotes sendo transmitidos indiferentemente do destino e do tipo, logo torna-se necessário associar ao impulso “tx” os dois eventos de transmissão e as duas ações (transmissão do tipo 1 e transmissão do tipo 2) de cada evento. Isto é feito colocando uma lista de ações após a identificação do evento na especificação do impulso.

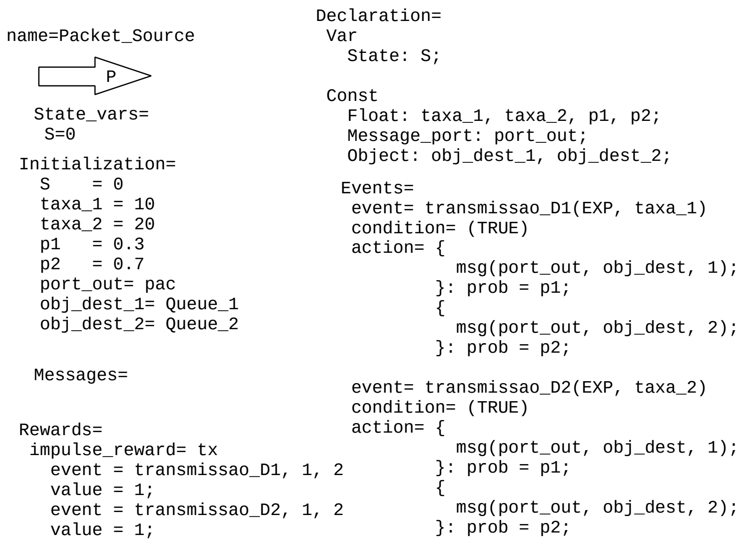


Figura 3.5: Definição de recompensa de impulso para contagem dos pacotes enviados pela fonte.

Além de associar recompensa de impulso a disparo de eventos, a ferramenta TANGRAM-II também é possibilita associar impulsos a chegada de mensagens. Ao receber uma mensagem um objeto pode acumular uma recompensa de impulso. Isto permite ao usuário definir recompensa de impulsos mais refinadas, dando maior abrangência às medidas de interesse que podem ser especificadas. Um exemplo claro do uso desta funcionalidade está ilustrado na Figura 3.6 onde uma fila, que recebe pacotes de diversas fontes, deseja medir o número de pacotes recebidos. Para obter esta medida de interesse o usuário deve especificar uma recompensa de impulso que esteja associada ao recebimento de pacotes. Esta associação é feita através da porta por onde a mensagem chega. Desta forma, todas vez que uma mensagem chega pela porta especificada na recompensa, o valor do impulso é acumulado. Da mesma forma que na definição de eventos, o recebimento de mensagem pode ter associado várias ações. O impulso a ser acumulado deve estar associada a pelo menos uma ação, o que é feito utilizando uma lista com os números das ações de interesse. O exemplo da Figura 3.6 ilustra uma fila que recebe pacote de duas fontes diferentes, Fonte_1 e Fonte_2. Os pacotes sendo enviados pelas fontes chegam à fila pela mesma porta e uma recompensa de impulso está associada a chegada de mensagem nesta porta. Assim, sempre que um pacote chega, independente da fonte, o valor da recompensa é incrementado em uma unidade.

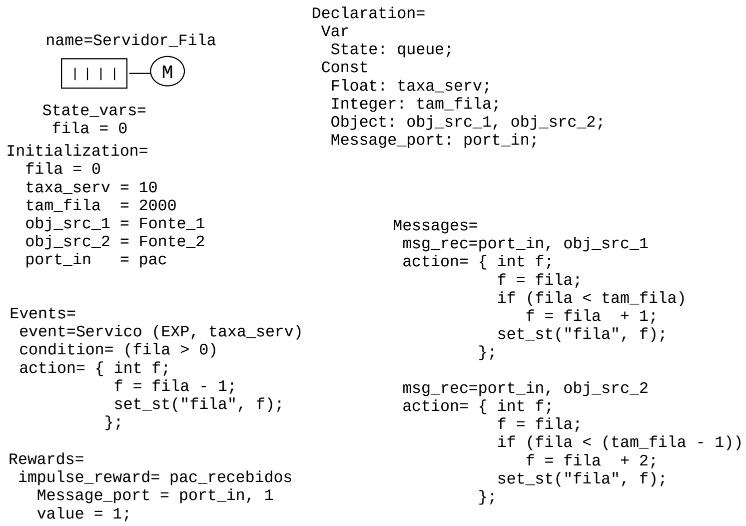
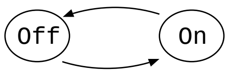


Figura 3.6: Definição de recompensa de impulso para contagem de pacotes recebidos pela fila.

Outro exemplo que utiliza recompensa de impulso como medida de interesse é apresentado na Figura 3.7, que apresenta a tradicional fonte On-Off [32] que transmite pacotes apenas durante sua permanência no estado On. Suponha que o objetivo é avaliar o comportamento de uma fila que recebe pacotes desta fonte e os serve a uma determinada taxa. Uma maneira de avaliar o comportamento desta fila seria criar um novo objeto para receber os pacotes desta fonte. Porém, pode ser definida uma medida de interesse utilizando recompensa de impulso que representa exatamente esta fila. Pode-se associar impulsos aos eventos transmissão de pacotes. Além disso, foi criado um novo evento neste mesmo objeto que representa o serviço realizado pela fila e associou-se, a este evento, o mesmo impulso, porém representando o serviço de pacotes. Desta forma, a medida de interesse representa o número de pacotes na fila. Toda vez que um evento de transmissão ocorre, acrescenta-se um elemento a esta medida (chegada na fila); e toda vez que um evento de serviço ocorre diminui-se um elemento desta medida (serviço de pacote na fila). Logo, é possível avaliar o comportamento da fila sem que ela realmente exista no modelo, pois nenhum objeto representando a fila é utilizado. Isto permite que sistemas com filas de dimensões realísticas (grandes) possam ser avaliados sem aumentar a complexidade de solução do modelo.

```
name=On_Off_Source
```




```
State_vars=
  status = 0

    Initialization=
      status     = 0
      OnOff_rate = 2
      OffOn_rate = 2
      tx_rate    = 2
      serv_rate  = 1

Messages=

Rewards=
 impulse_reward=buffer
   bounds= 0, 1000
   event=Transmit, 1
   value= 1;
   event=Service, 1
   value= -1;
```

```
Declaration=
 Var
  State: status;
 Const
  Float: OnOff_rate, OffOn_rate,
         tx_rate, serv_rate;

Events=
 event= On2Off (EXP, OnOff_rate)
 condition= (status == 1)
 action= { ; set_st("status", 0); };

 event= Off2On (EXP, OffOn_rate)
 condition= (status == 0)
 action= { ; set_st("status", 1); };

 event= Transmit (EXP, tx_rate)
 condition= (status == 1)
 action= { /* do nothing */ ; };

 event= Service (EXP, serv_rate)
 condition= (TRUE)
 action= { /* do nothing */ ; };
```

Figura 3.7: Definiçao de recompensa de impulso representando uma fila em uma fonte On-Off.

Porém, esta fila poderia crescer (ou decrescer) infinitamente, pois o estado da fila não é conhecido pelo modelo e sim representado em uma medida de interesse. Se a medida não tiver limites, é possível que o evento transmissão faça com que a fila atinja tamanhos muito grandes. Da mesma forma, o evento serviço pode "servir" mesmo quando a fila estiver vazia, gerando números negativos para o tamanho da fila.

O TANGRAM-II permite que o usuário limite as medidas de recompensas superiormente e/ou inferiormente. Desta forma as variáveis aleatórias $CR(t)$ e $CI(t)$ podem ser limitadas. Esta funcionalidade permite que o usuário resolva o problema da fila ilimitada acima, introduzindo limites para a medida da fila.

Para utilizar este recurso, o usuário deve introduzir a especificação dos limites logo após a definição do nome da medida de interesse. Os limites devem estar separados por vírgulas. O primeiro parâmetro especifica o limite inferior para a medida e o segundo o limite superior. Caso a medida seja ilimitada em um de seus extremos (ou em ambos) a palavra reservada INF deve ser utilizada. Um exemplo de especificação de limites também está ilustrado na Figura 3.7, dentro da caixa de texto utilizada para definição das recompensas.

O recurso de limite pode ser aplicado a medidas de interesse com recompensas de taxa e de impulso. Se o usuário omitir a especificação dos limites, então a medida se torna ilimitada.

3.4 Definição de medidas de interesse no UltraSAN *vs* TANGRAM-II

Conforme visto no capítulo 2, a ferramenta UltraSAN necessita que o usuário especifique medidas de interesse utilizando recompensas de taxa e de impulso. Este definição está ligada diretamente à forma com a qual o modelo é descrito. Foi visto também que esta especificação, no UltraSAN, é feita em um módulo separado, não sendo possível reutilizar objetos com recompensas. A filosofia por trás da especificação é bastante similar a do TANGRAM-II. Recompensas de taxa são associadas a estados do sistema, *markings* utilizando pares de condições e valores. Recompensas de impulso estão associadas ao disparo de *activities*, que, de certa forma, representam os eventos do modelo.

Apesar disto, a ferramenta UltraSAN não permite a especificações de limites para as recompensas restringindo a funcionalidade do uso de recompensas em modelos mais genéricos. Outra desvantagem é que, ao especificar recompensas de taxa, o usuário não possui opção quando mais de uma condição é satisfeita. A ferramenta então, define como o valor da recompensa a soma dos valores que tiveram suas condições satisfeitas.

A definição de recompensa de impulso também é limitada, pois está associada somente ao disparo de *activities* (eventos). Não é possível definir recompensas de impulso distintas para os casos onde o disparo de um evento pode levar a mais de um estado do sistema. Esses casos ocorrem sempre que se utiliza probabilidades, *cases*, associadas a execução de ações. A Figura 3.8 ilustra uma parte do modelo de rede de filas descrito na ferramenta UltraSAN. Neste modelo, a fila 1, ao servir um pacote, o transmite com diferentes probabilidades às filas 2 e 3. Existe interesse em medir apenas os pacotes que saem da fila 1 destinados à fila 2. Esta medida de interesse não pode ser definida no UltraSAN, o que pode ser feito na ferramenta TANGRAM-II, utilizando uma recompensa de impulso associada ao evento "transmissão" relacionada com a ação desejada.

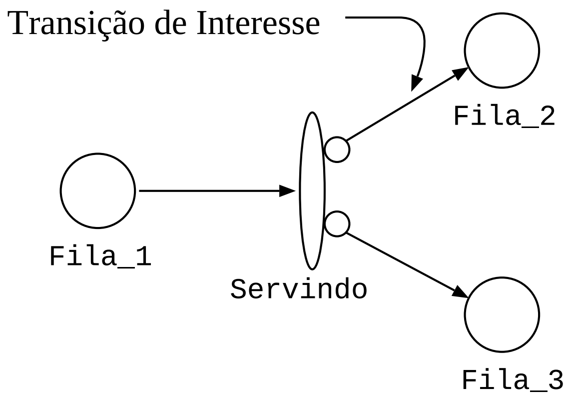


Figura 3.8: Parte da especificação de um modelo de rede de filas na ferramenta UltraSAN.

3.5 Solucionando as Medidas com Recompensas

Uma vez definidas, as recompensas de taxa e/ou impulso devem ser analisadas a fim de se obter resultados numéricos. Se a modelagem do sistema resultar em uma cadeia de Markov, então métodos de solução analíticos podem ser utilizados para calcular as medidas com recompensas. O módulo de solução da ferramenta TANGRAM-II possui alguns métodos para solução de recompensas. Várias medidas podem ser obtidas com esses métodos, entre elas estão:

- resultados em estado estacionário e em tempo transiente, $E[ACR(t)]$, $E[ACI(t)]$
- $P[CR(t) > k]$ e $P[CI(t) > k]$
- $P[ACR(t) > k]$ e $P[ACI(t) > k]$
- $E[CR(t)]$ e $E[CI(t)]$

Métodos sofisticados para solução de recompensas integram o módulo de solução da ferramenta. Um método de solução recentemente desenvolvido [8] permite que medidas de recompensas com limites sejam resolvidas dando maior flexibilidade ao usuário.

O simulador da ferramenta TANGRAM-II também permite solucionar medidas de recompensas de taxa e de impulso. Neste caso, o modelo não precisa ser Markoviano. Durante a simulação o módulo de medidas do simulador acumula as recompensas de acordo com o caminho amostral feito pela simulação. Ao terminar, o simulador apresenta as seguintes medidas:

- $E[CR(t)]$ e $E[CI(t)]$
- $E[ACR(t)]$ e $E[ACI(t)]$
- $E[TCR(t)]$ e $E[TCI(t)]$

Acompanhando essas medidas, o simulador apresenta a variância amostral e o intervalo de confiança de cada uma das medidas analisadas. No capítulo 4 o simulador é analisado com mais detalhes.

Capítulo 4

Simulação de Eventos Discretos no TANGRAM-II

CONFORME apresentado no capítulo 2, uma vez especificado um modelo, é preciso analisá-lo para obter as medidas de interesse desejadas. A simulação é a forma mais tradicional de realizar esta tarefa e estimar o comportamento do sistema.

Um módulo de simulação robusto, capaz de analisar qualquer modelo através de repetidas execuções, foi desenvolvido e acoplado à ferramenta TANGRAM-II. Neste capítulo, na seção 1, será discutida a arquitetura e o funcionamento deste módulo de simulação. Na seção 2 apresentam-se as funções de distribuição suportadas pelo simulador. Na seção 3 é apresentada uma funcionalidade denominada "clone de eventos". A seção 4 descreve o módulo de simulação interativa com suporte a animação. Finalmente, a seção 5 apresenta um modelo e a análise de seu comportamento através de simulação e de métodos analíticos.

4.1 Arquitetura do Simulador

A arquitetura do simulador da ferramenta TANGRAM-II está ilustrada na Figura 4.1, apresentado as principais estrutura de dados mantida pelo simulador durante sua execução.

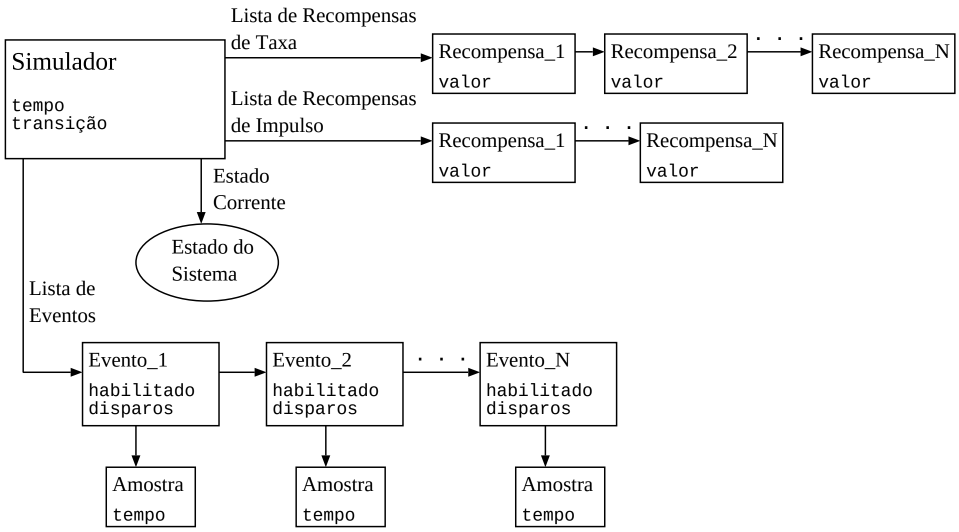


Figura 4.1: Arquitetura do Simulador de Eventos Discretos.

Uma das informações que o simulador armazena é o estado corrente da simulação. Este estado é alterado ao longo da execução, enquanto estados passados vão sendo removidos da memória. Desta forma, o simulador não apresenta problemas de armazenamento, podendo executar qualquer modelo por qualquer período de tempo. Durante a descrição do modelo, o usuário informa o estado inicial dos objetos que é utilizado como o estado inicial do simulador. O simulador parte deste estado inicial e percorre o sistema indo de um estado para outro através de transições.

O simulador mantém uma lista estática com todos os eventos descritos no modelo. Esta lista é construída apenas uma vez durante o processo de inicialização do simulador. Os eventos mantidos nesta lista são exatamente aqueles descritos pelo usuário em objetos de seu modelo. Estes estão agrupados por objetos na ordem em que aparecem no modelo.

Cada evento possui um *flag* para indicar se o evento está habilitado ou não, sob o atual estado do sistema. Se o evento está habilitado, então uma amostra daquele evento está escalonada no tempo e pode disparar. Essa amostra informa o próximo tempo de disparo deste evento. Inicialmente, o simulador varre todos os eventos habilitando-os ou não e gerando amostras de tempo para os eventos que estão habilitados. Os eventos também possuem um contador de disparos que contam o número de vezes que cada um deles ocorre que é apresentado no final da simulação. Este contador é utilizado no caso do término da simulação ser dada por um determinado número de disparos de um evento específico.

A cada passo, o simulador consulta a lista de eventos e identificando o evento que possui o menor tempo escalonado dentre os eventos habilitados. O tempo atual de simulação é atualizado para o tempo do próximo evento. Este é então disparado e sua ação correspondente é executada. Esta ação é tratada até que se atinja um estado *tangible*, ou seja, um estado que não possui mensagens pendentes. Este estado final é o próximo estado do simulador. Durante este processo, a lista de eventos é atualizada de forma a habilitar ou desabilitar eventos devido a mudança nas variáveis de estado do sistema. Se um evento for habilitado, então uma nova amostra de tempo é gerada para ele; se for desabilitado, a amostra de tempo é retirada da lista e o evento é marcado como desabilitado. Desta forma, ao final de um passo de simulação, a lista de eventos já está atualizada e o processo se repete.

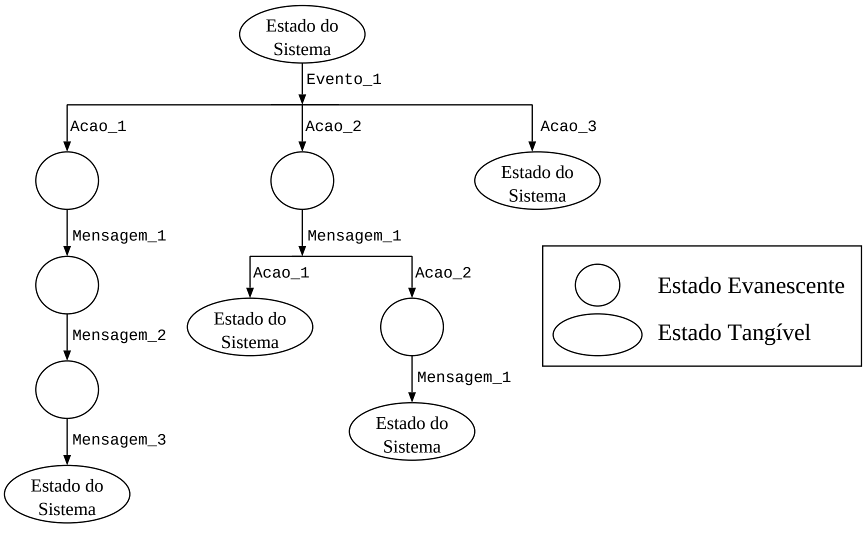


Figura 4.2: Exploração de estados no TANGRAM-II.

Se o evento a ser disparado possui mais de uma ação, então o simulador escolhe aleatoriamente, a ação a executar, seguindo as probabilidades especificadas pelo usuário, pois somente uma das ações associadas ao evento é executada. As probabilidade especificadas pelo usuário devem somar 1 sob qualquer valor que as variáveis de estado do objeto possam assumir. Esta verificação é feita pelo simulador, que gera um erro e aborta a simulação se a soma dessas probabilidades for diferente de 1. O mesmo acontece com mensagens que possuem várias ações associadas a diferentes probabilidades de execução. A Figura 4.2 ilustra o caso de um evento com diversas ações associadas e o envio de diversas mensagens. O simulador atinge apenas a um dos possíveis estados do sistema quando o evento é disparado e durante o passo de simulação nenhum outro evento é disparado.

A condição de parada do simulador deve ser fornecida pelo usuário. Existem diversas formas de indicar o término da simulação. O usuário pode instruir que o simulador execute até um determinado tempo de simulação ou, até que ocorra um determinado número de transições. É possível, também, especificar que o simulador execute até que um certo evento dispare um determinado número de vezes. Ou ainda, estabelecer um estado de parada para o simulador. Neste caso, o usuário especifica uma única variável de estado de algum objeto que será observada pelo simulador. Quando esta variável atingir um determinado valor, a simulação termina. Neste caso, é necessário especificar também o tempo de parada do simulador, para que seja utilizado caso, "nunca", o estado indicado pelo usuário seja alcançado.

O simulador também mantém as medidas de interesse especificadas pelo usuário. Duas listas de recompensas são mantidas: uma para recompensas de taxa, outra para recompensas de impulso. Durante um passo de simulação as recompensas de impulso são atualizadas. A cada ação executada e/ou mensagem entregue, o simulador verifica a existência de algum impulso associado, calculando e atualizando seu valor. As recompensas de taxa são tratadas no final do passo de simulação. Ao terminar um passo, o simulador percorre todas as recompensas de taxa verificando suas condições e calculando e acumulando seus valores, caso seja necessário.

4.2 Possíveis Distribuições para Eventos

Os eventos na ferramenta do TANGRAM-II são discretos. O intervalo entre dois eventos do mesmo tipo está relacionado a uma distribuição. As amostras dessas distribuições sempre se referem a instantes de tempo em que o evento irá ocorrer.

O usuário do TANGRAM-II possui uma variedade de distribuições, disponíveis para ser utilizada durante a simulação. Cada uma delas possui parâmetros específicos que devem ser fornecidos pelo usuário durante a especificação do sistema. A seguir, a lista de distribuições e seus respectivos parâmetros:

- Exponencial - taxa
- Determinística - valor
- Erlangiana - média, número de estágios
- Gaussiana - média, variância
- Lognormal - média, variância
- Uniforme - valor inferior, valor superior
- Pareto - média
- FBM - média, variância, nível máximo, escala de tempo, Hurst
- FARIMA - média, variância, número de amostras, escala de tempo, Hurst
- File - nome do arquivo que contém o valor das amostras

Essas distribuições podem ser utilizadas sem restrições e atribuídas a qualquer evento. Vale lembrar que todos os eventos têm associados valores de tempo, e que amostras de distribuições que tenham valores negativos ou iguais a zero não fazem sentido. Algumas distribuições, como a Gaussiana, quando utilizadas com parâmetros inapropriados podem gerar amostras negativas. Neste caso, o simulador produz um erro de execução e aborta a simulação.

As distribuições FBM (Fractional Brownian Motion), FARIMA (Fractional ARIMA - Auto Regressive Integrated Moving Average) e Pareto são específicas para modelos de tráfego que apresentam alguma dependência de longa duração. Essas distribuições são utilizadas em modelos de fonte multimídia, para representar tráfego de vídeo e voz. Os parâmetros para essas distribuições são específicos para esses tipos de modelo, como por exemplo o parâmetro de Hurst.

A distribuição File utiliza um arquivo que possui o tempo incremental de ocorrência dos eventos. Se durante a simulação todas as amostras do arquivo forem consumidas, volta-se à leitura da primeira amostra do arquivo. Desta forma, o arquivo é tratado como uma lista circular de amostras.

Todos os parâmetros de outras distribuições podem ser especificados com expressões aritméticas, excetuado-se as distribuições FBM e FARIMA. Tais expressões podem conter variáveis de estado e constantes do objeto. Desta forma, é possível definir distribuições com parâmetros variáveis. Para isto, basta especificar um dos parâmetros em função de uma (ou mais) variável de estado. Sendo assim, quando um evento é habilitado, os parâmetros da distribuição deste evento são calculados sobre o estado corrente do simulador antes da amostra ser gerada. Depois que a amostra é gerada, seu valor não sofre alterações. Esta amostra pode ser consumida, caso o evento ocorra (esta é a menor amostra na lista de eventos) ou removida caso o evento seja desabilitado. Mudanças nos parâmetros da distribuição não afetam amostras que já tenham sido geradas. Apenas novas amostras estão sujeitas a esta alteração.

4.3 Clone de Eventos

Até aqui, apenas uma amostra de cada evento era gerada e isto ocorria sempre que um evento era habilitado. Esta única amostra podia ser disparada (menor amostra da lista de eventos) ou removida da lista (caso o evento seja desabilitado). Porém, em alguns modelos existe a necessidade de disparar um mesmo evento diversas vezes, permitindo ao usuário descrever sistemas mais complexos e completos. Considere uma fila com o número de servidores infinito onde cada pacote que chega é colocado imediatamente em serviço, possuindo um determinado tempo para terminar. Neste exemplo, temos apenas um evento, que é o serviço de pacotes, porém, várias amostras devem ser geradas. Ainda nesta seção, apresenta-se formalmente este exemplo.

A capacidade de amostrar várias vezes um determinado evento será chamada aqui de clonagem. Ao clonar um evento, o usuário introduz uma nova amostra na lista de amostras deste evento. Desta forma, o evento poderá ser disparado duas (ou mais) vezes. Esta amostra é colocada na lista de amostras do evento correspondente em ordem crescente de tempo. Se o evento for desabilitado, todas as amostras são removidas da lista, pois deixam de fazer sentido, uma vez que o evento foi desligado. Caso o evento continue habilitado, as amostras vão sendo consumidas em seus respectivos tempos de simulação. A Figura 4.3 ilustra uma lista de eventos com diversos clones.

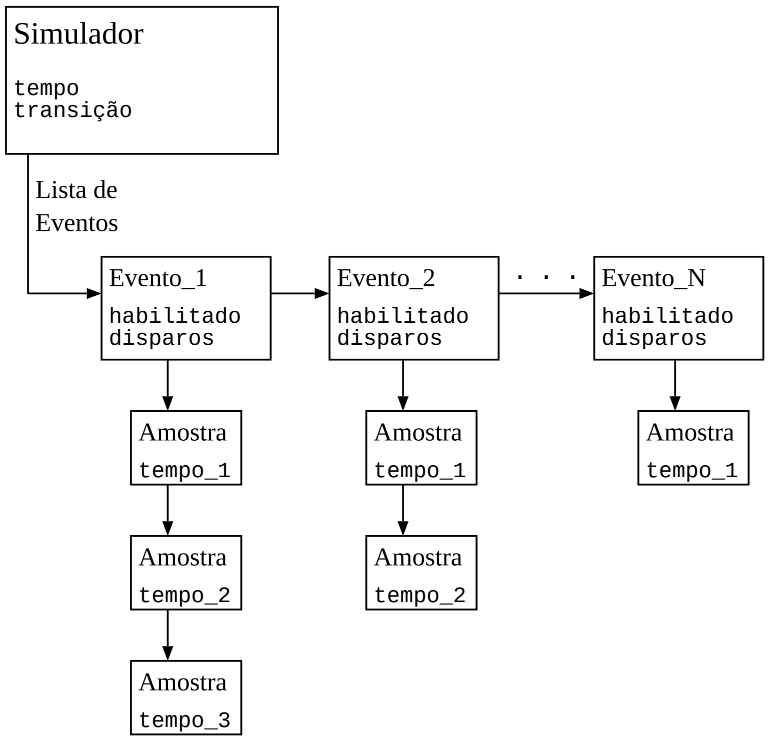


Figura 4.3: Lista de Eventos com diversos eventos clonados.

O usuário pode clonar um evento durante a execução de qualquer ação do objeto. Por exemplo, o disparo de um determinado evento pode levar a um clone de outro evento, ou a chegada de uma mensagem pode clonar algum evento. O simulador não permite que eventos desabilitados sejam clonados. No momento do clone, o simulador utiliza o estado do sistema corrente para avaliar a condição do evento.

Para clonar um evento o usuário deve utilizar a função *clone("nome do evento")*. Esta função pode aparecer em qualquer lugar dentro de uma ação e realiza exatamente um clone do evento passado como parâmetro. É permitido clonar várias vezes o mesmo evento e, para isto, a função acima deve ser chamada repetidas vezes.

O modelo ilustrado pela Figura 4.4 apresenta um servidor determinístico infinito com média variável. A cada nova chegada, uma nova amostra do evento "Service" é gerada tendo como média uma função do número de clientes sendo atendidos. Quando uma mensagem chega e o número de servidores ocupados é zero então o clone não é realizado, pois sob o estado "serving = 0" o evento "Service" não está habilitado. Assim, ao final da chegada apenas uma amostra será gerada.

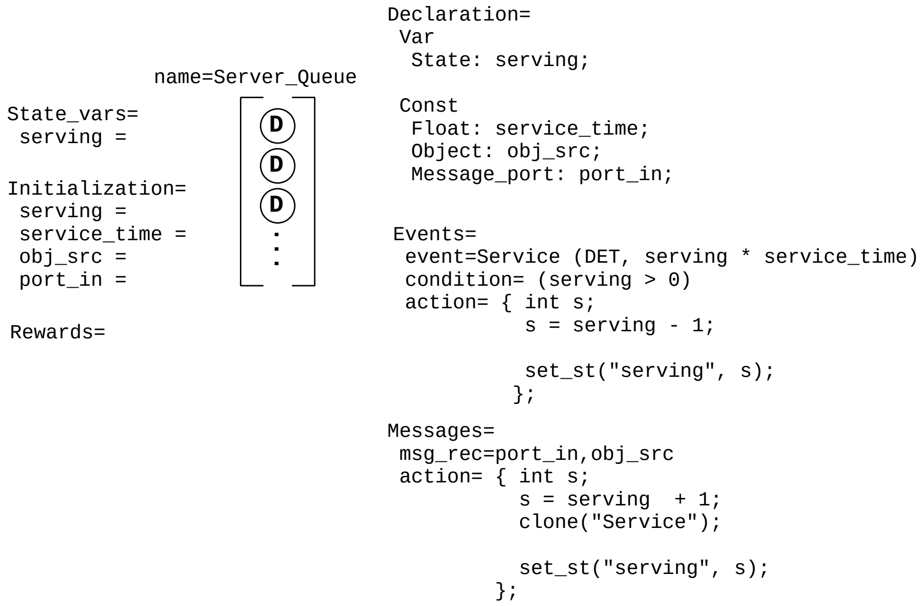


Figura 4.4: Servidor infinito determinístico utilizando a técnica de clonagem de eventos.

4.4 Simulação Interativa

A ferramenta TANGRAM-II oferece ao usuário a opção de realizar simulações interativas. Nestas, o modelo descrito na ferramenta gráfica TGIF é atualizado juntamente com a evolução do simulador. Isto permite que o usuário acompanhe, graficamente, o caminho amostral tomado pelo simulador, observando os estados do sistema.

Esta funcionalidade pode ser muito útil para fins de depuração do modelo, uma vez que permite que o usuário acompanhe a simulação a partir de qualquer estado que o sistema possa assumir. Basta que o usuário defina o estado inicial e execute a simulação passo-a-passo. A simulação interativa também pode ser utilizada para fins didáticos. Acompanhar a evolução do simulador graficamente, pode ajudar a compreender melhor o comportamento do modelo.

Ao iniciar a simulação interativa, a ferramenta gráfica TGIF conecta-se ao simulador que está executando em outro processo. Esta conexão é feita utilizando a API de *sockets* [7]. O estado inicial da simulação é dado pelo estado do sistema descrito no modelo quando o simulador é executado. Após esta conexão, o usuário pode dar um passo na simulação. Isto é feito através de uma mensagem de controle enviada pelo TGIF ao simulador. O simulador recebe esta mensagem, simula um passo e envia o estado corrente de volta ao TGIF. Este por sua vez, recebe esta mensagem com a descrição do novo estado do sistema e atualiza os valores das variáveis de estado na interface gráfica. Todo este processo de comunicação e atualização da interface gráfica foi implementado e incorporado ao TGIF, como sendo um módulo de simulação interativa.

O número de passos a ser dado pelo simulador, antes que um novo estado seja enviado ao TGIF, também é especificado pelo usuário. Desta forma, o usuário pode pedir ao simulador para realizar n transições antes de atualizar o estado do sistema. Também é possível colocar a simulação em modo contínuo, no qual o simulador executa um passo de simulação e atualiza a interface gráfica repetidamente. O usuário pode acompanhar o tempo de simulação e o número de transições efetuadas pelo simulador na interface gráfica.

Além de acompanhar a evolução do estado do simulador, o usuário pode definir animações que devem estar associadas à mudança dos estados do sistema. É possível criar diferentes animações para diferentes objetos e ilustrar graficamente o comportamento dos objetos. Por exemplo, um objeto que representa uma fila pode ter uma animação que está associada ao tamanho da fila. Assim, a fila pode crescer e diminuir graficamente com a chegada/saída de clientes. A Figura 4.5 ilustra um objeto fila/servidor animado.

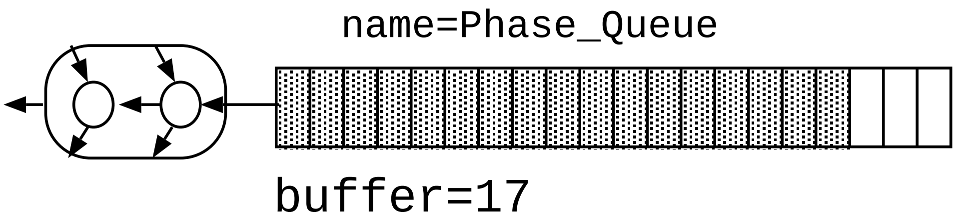


Figura 4.5: Fila animada utilizada na simulação interativa.

Toda a animação é definida pelo usuário. O simulador oferece um mecanismo para conduzir a animação, porém os objetos animados devem ser construídos pelo usuário utilizando recursos da ferramenta TGIF. A ferramenta gráfica TGIF oferece primitivas de desenho, como redimensionar ou mover um objeto, que podem ser utilizadas para criar uma animação. Para animar um objeto, o usuário deve escrever um trecho de código em TGIF que realiza a animação desejada. A sintaxe desta linguagem pode ser encontrada no manual do TGIF. Exemplos de animações já existentes são ótimas referências para criar novas, uma vez que o número de primitivas de animação é pequeno. No apêndice A a especificação da animação de um modelo simples é fornecida através de um exemplo.

4.5 Definição de Medidas de Interesse

Ficou claro assim, que o simulador oferece aos usuários medidas de interesse definidas através de recompensas de taxa e de impulso. Esta é a única forma de obter medidas do simulador. Ao final de uma simulação, todos os resultados sobre as medidas de interesse são apresentados. Também são relatados o número de disparos de cada evento, o número total de transições feitas pelo modelo, o tempo atual de simulação e o estado do sistema em que o simulador se encontra.

As recompensas especificadas pelo usuário são calculadas sobre o caminho amostral feito pelo simulador. A cada novo estado do sistema as recompensas são calculadas e acumuladas. Este cálculo é feito respeitando os limites definidos pelo usuário. As medidas $CR(t)$ e $TCR(t)$ são computadas ao final de cada passo (transição) de simulação. As medidas $CI(t)$ e $TCI(t)$ são computadas durante o passo da simulação, pois são influenciadas pela ocorrência de um evento e/ou entrega de mensagens.

O simulador permite que o usuário defina o número de vezes que a simulação deve ser executada, ou seja, o número de rodadas de simulação. Ao final das rodadas, a média amostral das medidas é apresentada juntamente com a variância e os valores para o intervalo de confiança de 95%. O intervalo de confiança pode ser especificado pelo usuário, porém seu valor inicial é de 95%. Esses dois parâmetros são especificados através da interface gráfica do módulo de simulação da ferramenta TANGRAM-II. Um exemplo completo é ilustrado na próxima seção.

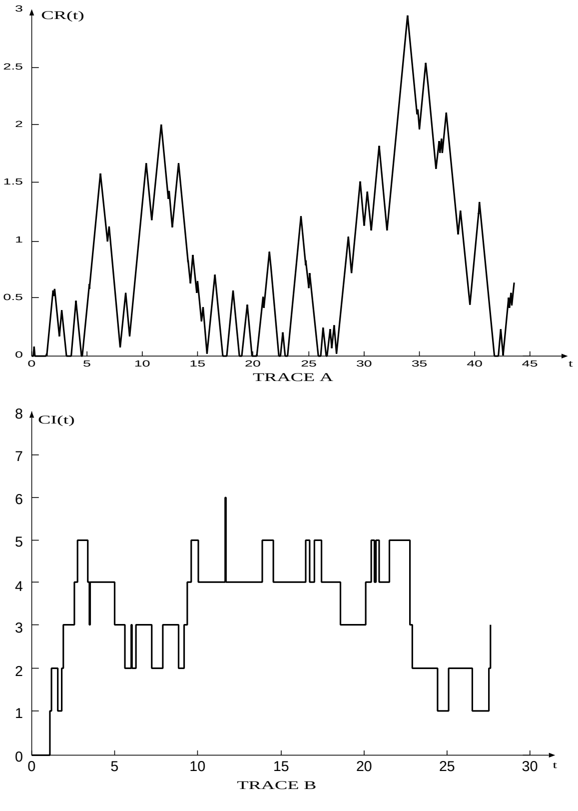
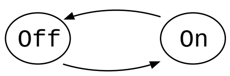


Figura 4.6: *Traces* das medidas $CR(t)$ e $CI(t)$

O simulador também é capaz de gerar um *trace* de uma medida de interesse especificada. Isto permite ao usuário acompanhar a evolução de uma determinada medida no tempo. Um gráfico pode ser facilmente gerado utilizando o *trace* de saída do simulador. Entre outras finalidades, este *trace* permite que o usuário depure o modelo ou compreenda melhor o comportamento da medida. O simulador gera *traces* das variáveis $CR(t)$ e $CI(t)$ para qualquer recompensa que tenha sido definida pelo usuário. Esta opção precisa ser definida pelo usuário antes de iniciar o processo de simulação.

A Figura 4.6 ilustra dois gráficos obtidos de *traces* que acompanham a evolução do tamanho de uma fila. A gráfico A ilustra um *trace* da medida $CR(t)$ e o gráfico B ilustra um *trace* da medida $CI(t)$, ambos indicando o tamanho de uma fila no tempo t. Os gráficos foram obtidos a partir dos modelos ilustrados na Figura 4.7 e 3.7, respectivamente. Esses modelos representam fontes de tráfego do tipo On-Off [32] que transmitem pacotes apenas enquanto permanecem no estado On. As duas recompensas representam uma fila cujo tamanho é determinado pelo comportamento da fonte. Na recompensa de taxa, o tamanho da fila está associado a taxas de transmissão da fonte e de serviço da fila. Esta associação é feita pelos estados que a fonte pode assumir, e em cada estado a recompensa possui um determinado valor. Na recompensa de impulso, o tamanho da fila está associado os eventos de transmissão da fonte e serviço da fila, representando a chegada e a saída de pacotes, conforme descrito no capítulo 3.3.

```
name=On_Off_Source
```




```
State_vars=
  status = 0
Initialization=
  status     = 0
  OnOff_rate = 2
  OffOn_rate = 2
  tx_rate    = 2
  serv_rate  = 1

Messages=
```

```
Declaration=
  Var
   State: status;
  Const
   Float: OnOff_rate, OffOn_rate,
          tx_rate, serv_rate;
Events=
 event= On2Off (EXP, OnOff_rate)
 condition= (status == 1)
 action= { ; set_st("status",0); };

 event= Off2On (EXP, OffOn_rate)
 condition= (status == 0)
 action= { ; set_st("status", 1); };
Rewards=

 rate_reward=buffer
   bounds= 0, INF
   condition= (status == 0)
   value=  -serv_rate;
   condition= (status == 1)
   value= (tx_rate - serv_rate);
```

Figura 4.7: Modelo On-Off com fila representada por recompensa de taxa

4.6 Exemplo 1: Modelo de fonte de vídeo + fila determinística

Ilustra-se agora a análise de um modelo feita com simulação comparando os resultados com os obtidos através da resolução da cadeia de Markov. Para poder comparar os resultados, este modelo utiliza apenas distribuições exponenciais e não faz uso de "clones" de eventos.

Para ilustrar o uso do simulador desenvolvido, foram realizados experimentos com um sistema composto por uma fonte de dados e um servidor com uma fila. A fonte de dados procura aproximar o comportamento de uma transmissão de vídeo codificado com algoritmos de compressão, como por exemplo, MPEG. Desta forma, o tráfego gerado por esta fonte possui taxa de envio de células variável e é conhecido como tráfego VBR (*Variable Bit Rate*). Os pacotes/células gerados pela fonte são enviados para o servidor que possui uma taxa exponencial (constante) de serviço, e armazena as células excedentes em uma fila de tamanho finito.

O modelo para a fonte de vídeo procura levar em consideração medidas estatísticas do filme StarWars (apenas os quadros do tipo I), tais como a média, variância, taxa de pico e auto-correlação. Este método considera que o tráfego de vídeo pode ser aproximado pelo tráfego gerado por um conjunto de fontes do tipo *OnOff* [32]. Desta forma é criada uma cadeia de Markov de nascimento e morte que representa esta fonte de vídeo, ilustrada na Figura 4.8. A cada estado da cadeia é associada uma taxa de transmissão. O modelo foi feito utilizando a ferramenta TANGRAM-II e consiste de dois objetos: a fonte de vídeo que transmite células e a fila que armazena e serve os pacotes. Segue agora os parâmetros para o modelo de fonte nascimento e morte retirados de medidas estatísticas do filme StarWars:

- 20 fontes *OnOff*
- Taxa de Nascimento: $\alpha = 0.0712$
- Taxa de Morte: $\beta = 0.1525$

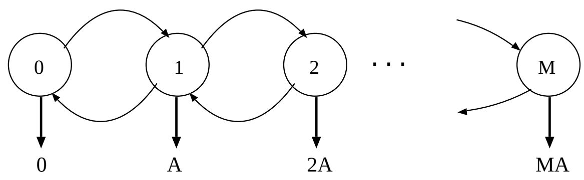


Figura 4.8: Cadeia de Markov do modelo de Nascimento e Morte de uma fonte de vídeo

- Taxa de transmissão (células/seg) em cada estado: $n * \lambda$, onde n é o número do estado e $\lambda = 50$
- Taxa média de transmissão da fonte: 318.28

Os pacotes são enviados a uma fila/servidor que os armazena e serve. O tamanho máximo da fila é de 500 pacotes. A capacidade do servidor será variada para que se obtenham curvas sobre o desempenho da fila enquanto as características da fonte permanecerão constantes. Os valores para a capacidade do servidor são 300 (carga média de 1.06), 335 (0.95), 375 (0.85), 450 (0.71), 550 (0.58), 650 (0.49) células por segundo, apresentando diferentes cargas médias sobre o servidor. A carga média maior do que um ilustra um caso de sobrecarga do servidor.

A seguir, a análise das seguintes medidas de interesse:

- utilização - fração de tempo que o servidor está ocupado
- tamanho médio da fila
- taxa média de transmissão - taxa de transmissão de pacotes da fonte
- $P[$ Tamanho da fila $\geq x\%]$ - fração de tempo em que a fila permanece com tamanho maior ou igual a $x\%$, com $x = [50\%, 75\%, 90\%, 99\%]$

Todas essas medidas foram definidas como recompensas de taxa e suas especificações podem ser vistas na Figura 4.9 que ilustra a descrição deste modelo.

Para obter os resultados analíticos, o espaço de estado do modelo foi gerado dando origem a uma cadeia de Markov. Esta cadeia foi resolvida em estado estacionário

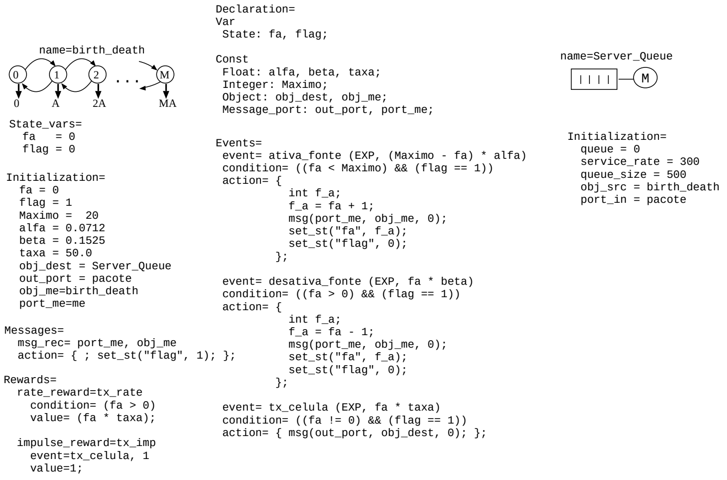


Figura 4.9: Especificação do modelo de Nascimento e Morte de uma fonte de vídeo

utilizando método de resolução GTH. O módulo da ferramenta TANGRAM-II para calcular medidas de interesse foi utilizado para calcular as medidas definidas.

A simulação do modelo foi feita por um tempo de simulação de 10000 segundos. Dez rodadas foram feitas obtendo no final a média amostral e o intervalo de confiança de 95% para cada medida. O processo foi repetido para cada uma das cargas médias.

Apresenta-se agora os resultados obtidos. A Figura 4.10 ilustra o gráfico da utilização obtida sobre as diversas cargas no servidor. A Figura 4.11 ilustra o gráfico do tamanho médio da fila e a Figura 4.12 ilustra o gráfico contendo alguns pontos da distribuição da fila. Neste último apresentamos a probabilidade da fila ter uma ocupação maior que $k\%$ com relação a sua capacidade de 500 pacotes.

Verfica-se que os resultados obtidos com o simulador casam com os obtidos através da solução analítica. Todos os valores exatos reportardos pelo método analítico estão dentro do intervalo de confiança dado pelo simulador. Isto pode ser observado para todos valores das medidas simuladas.

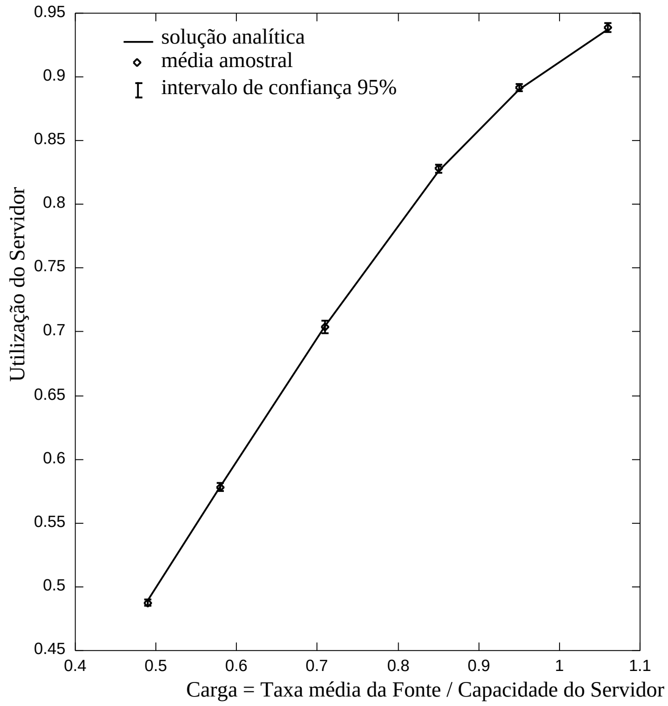


Figura 4.10: Utilização *vs* Capacidade do servidor. Comparação entre resultado analítico e simulação

4.7 Exemplo 2: Modelo de fonte Poisson + fila com porta/ficha

Apresenta-se agora um modelo mais complexo constituído de uma fonte de dados e de uma fila. A fonte se comporta como uma fonte Poisson e envia pacotes a uma taxa exponencial à uma fila que está acoplada a um servidor. A complexidade deste modelo está no comportamento da fila/servidor.

A Figura 4.13 ilustra o compotamento da fila e do servidor ao longo do tempo. Na parte (A) a fila recebe e armazena os pacotes transmitidos pela fonte. Neste instante o servidor não serve os pacotes. Na parte (B) o servidor ao receber uma ficha, fecha uma porta no final da fila e inicia o serviço. Apenas os pacotes que já estavam na fila na chegada da ficha podem ser servidos. Na parte (C) o servidor serve os pacotes dentro da porta enquanto outros pacotes, que são transmitidos pela fonte, chegam e são armazenados depois da porta. A fonte de pacotes nunca pára sua transmissão. Depois da parte (C), o sistema fila/servidor pode ter duas

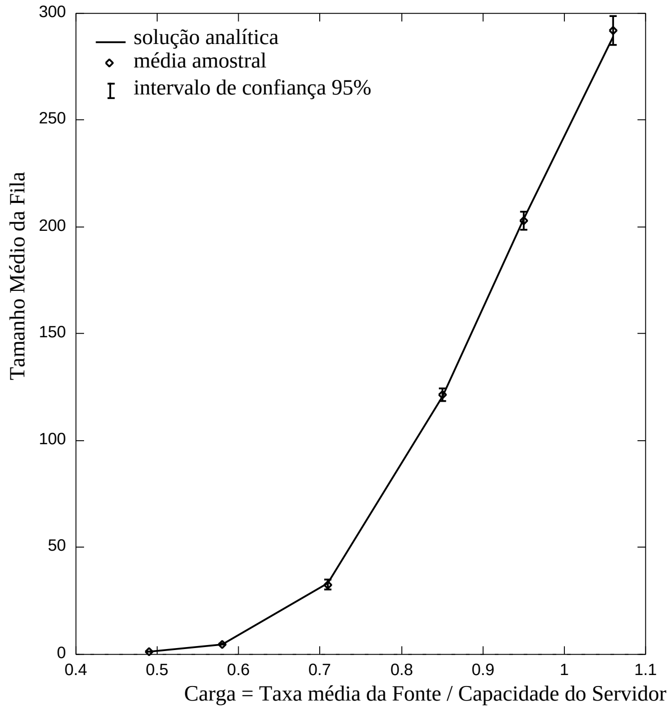


Figura 4.11: Tamanho médio da fila *vs* Capacidade do servidor. Comparação entre resultado analítico e simulação

configurações: (D) a ficha que o servidor possuía é devolvida antes do término do serviço dos pacotes dentro da porta; (E) todos os pacotes são servidos antes do servidor devolver a ficha, permanecendo ocioso até o momento de devolução da ficha. Ao devolver a ficha a porta é suspensa e os pacotes acumulados depois da porta se juntam aos pacotes dentro da porta, voltando a parte (A).

Duas medidas de interesse que caracterizam o desempenho deste sistema fila/servidor são: o tamanho médio da fila dentro da porta ao devolver a ficha, ou seja, o tamanho do resíduo da fila ilustrado na parte (D); quantidade de tempo que o servidor fica ocioso tendo a ficha porém com pacotes apenas atrás da porta, ou seja, quanto tempo o servidor segura a ficha após ter servidor todos os pacotes dentro da porta tendo pacotes para servir fora da porta, ilustrado pela parte (E).

Uma características dessas duas medidas de desempenho é que elas são inversamente proporcionais. Para reduzir o tamanho do resíduo da fila é necessário aumentar o tempo que o servidor fica ocioso, e vice-versa.

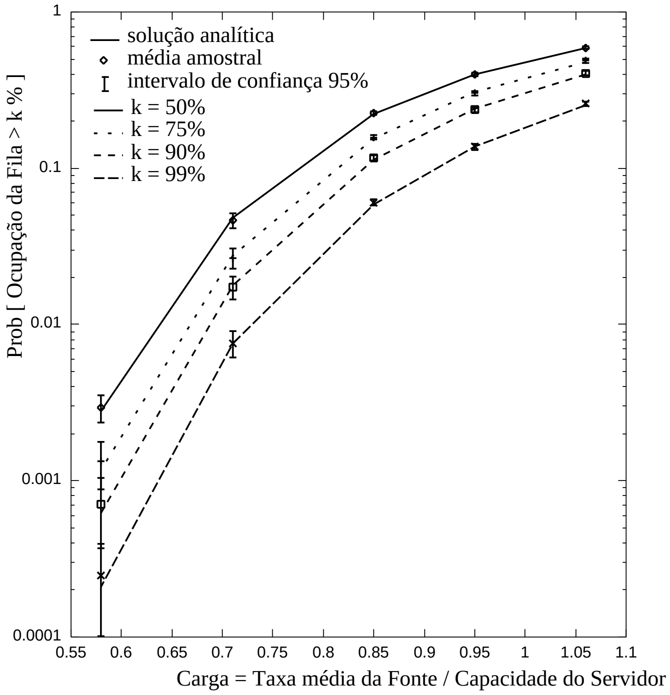


Figura 4.12: Distribuição do tamanho da fila *vs* Capacidade do servidor. Comparação entre resultado analítico e simulação

Essas medidas de interesse foram obtidas utilizando recompensas de impulso e de taxa respectivamente. Segue agora a especificação das recompensas utilizadas na obtenção dessas medidas:

```
Rewards=
  rate_reward = fracao_tempo_ocioso
    condition = ( (token == 1) && (gate == 0) && (queue > 0))
    value     = 1.0;

  impulse_reward = tamanho_residuo
    event        = timeout, 1
    value        = gate;
```

Observe que para marcar o tempo em que o servidor fica ocioso é necessário que o servidor possua a ficha, nenhum pacote esteja no resíduo da fila e a fila depois da porta possua pacotes. Para medir o tamanho do resíduo da fila, na ocorrência do evento de devolução da ficha um impulso é dado de comprimento igual ao tamanho da fila dentro da porta.

Uma simulação desse modelo foi feita por um período de 10000 unidades de

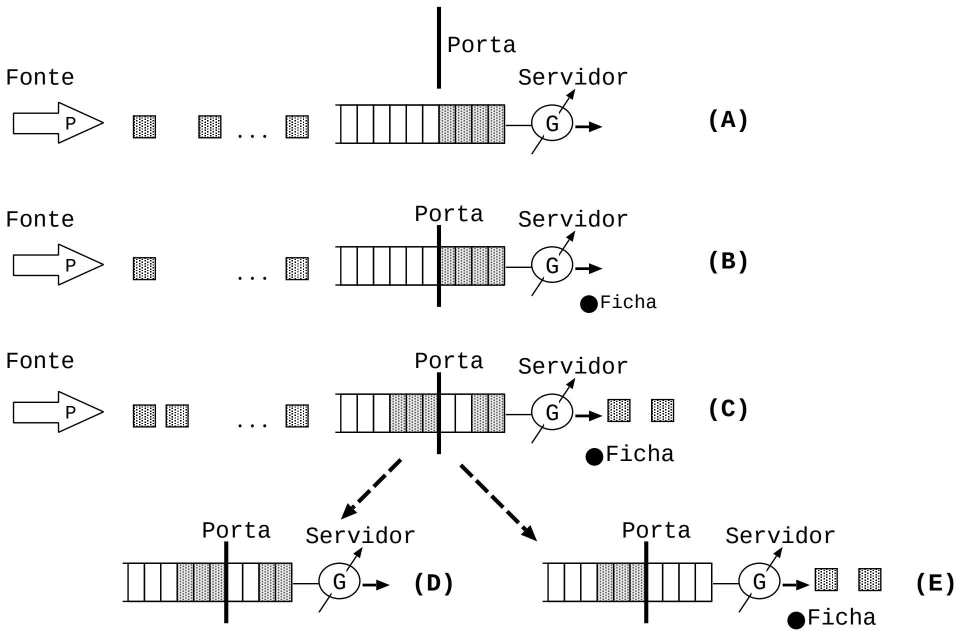


Figura 4.13: Comportamento da fila/servidor ao longo do tempo

tempo sendo executadas 10 rodadas. Os resultados obtidos tiveram um intervalo de confiança pequeno quando comparado com os valores das medidas. Apresentamos na Figura 4.14 os resultados obtidos através da simulação.

A carga média sobre o servidor foi variada para analisar o comportamento do modelo sobre diferentes situações. A carga média é dada pela taxa média de envio sobre a taxa média de serviço. Uma análise de sobrecarga (carga > 1) do servidor também foi feita.

No gráfico ilustrado na Figura 4.14 parte A ilustramos o tamanho médio do número de pacotes que ainda estão na fila dentro da porta quando o servidor devolve a ficha. Nota-se que este tamanho aumenta conforme o aumento da carga chegando ao tamanho máximo da fila que é de 1000 pacotes. Na parte B ilustramos a fração de tempo que o servidor fica ocioso tendo pacotes na fila fora da porta e segurando a ficha. Este tempo vai a zero quando a carga sobre o servidor aumenta. Pode-se notar que as duas medidas de desempenho são inversamente proporcionais e que um ponto de equilíbrio, que minimize ambas as medidas, pode ser encontrado.

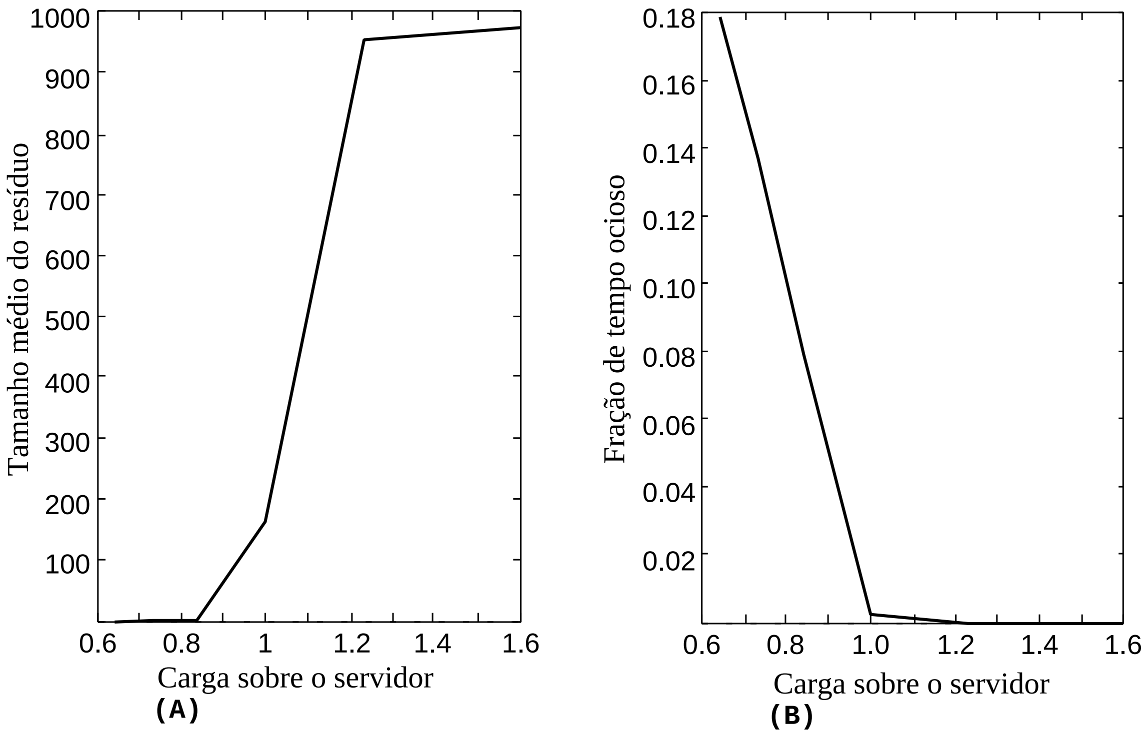


Figura 4.14: Resultados das medidas de interesse do modelo de fila com porta

Capítulo 5

Simulação de Eventos Raros no TANGRAM-II

NESTE capítulo apresenta-se uma poderosa funcionalidade desenvolvida sobre o módulo de simulação da ferramenta TANGRAM-II. O simulador dotou-se de um suporte a eventos raros que apresenta boa eficiência, implementando uma técnica recente de simulação de eventos raros chama-se splitting/RESTART apresentada no capítulo 2. Discute-se agora seu funcionamento e implementação dentro do módulo de simulação do TANGRAM-II.

5.1 Arquitetura do Simulador

A idéia básica do algoritmo de splitting/RESTART consiste em dividir o espaço de estados em vários níveis e simular esses níveis de maneira independente. Feita a divisão sobre o espaço de estados, um simulador inicia a simulação no nível 1 e evolui no tempo. Ao atingir um *threshold* o simulador pára. Se este ponto de parada for um *threshold* superior, então várias cópias do simulador atual são feitas (splits) e todas colocadas em execução no nível acima. Caso o ponto de parada seja um *threshold* inferior então, esta simulação termina e as medidas de interesse são coletadas.

Utilizando o simulador de eventos discretos, foi possível implementar a técnica de splitting/RESTART apenas controlando o funcionamento do algoritmo. Para isto, foi utilizado um escalonador de simuladores. A Figura 5.1 ilustra a arquitetura do simulador de eventos raros, apresentando as principais estruturas de dados.

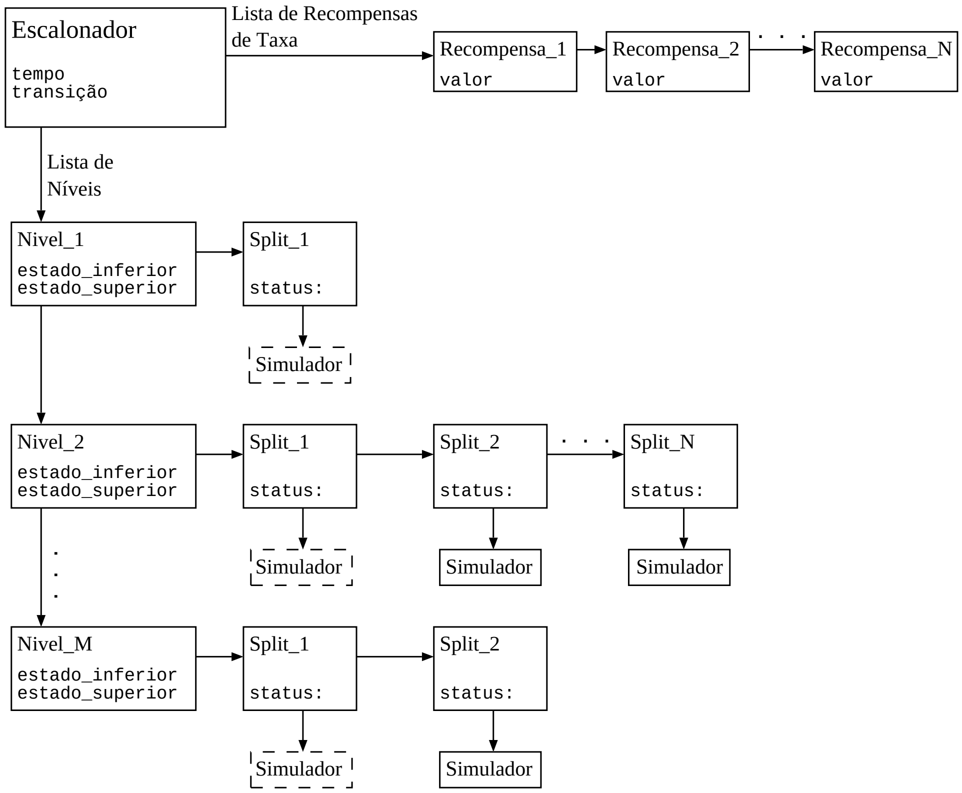


Figura 5.1: Arquitetura do simulador de eventos raros.

Define-se assim um escalonador (*Simulator Scheduler*) que é responsável por controlar toda a execução dos simuladores em todos os níveis. Este escalonador mantém uma lista de níveis. Esta lista é criada na inicialização do escalonador a partir da descrição do modelo feita pelo usuário. Cada nível é responsável por manter as múltiplas cópias dos simuladores sendo executadas, ou seja, os splits. Cada split possui um simulador de eventos discretos que executa até um determinado ponto. Ao terminar, o simulador que estava em execução avisa ao escalonador, que remove o split de seu nível e faz o tratamento adequado. Neste momento, as medidas de interesse também são atualizadas.

A arquitetura do escalonador é *multithreaded* e cada simulador em execução é mantido por uma única *thread* (i.e. linha de execução independente). As threads são geradas dinamicamente sempre que um novo simulador precisa ser executado, e destruídas logo após o seu término. Esta abordagem torna o simulador mais eficiente e facilita sua implementação e futura adaptação a um ambiente completamente distribuído (i.e. rede de computadores). O número de threads em execução é controlado pelo escalonador. Cada nível possui um limite para o número de threads que podem ser executadas simultaneamente. Desta forma, ao criar um novo simulador, um nível verifica se seu limite foi alcançado. Caso tenha sido, então a thread não é criada e este split fica aguardando o término de alguma thread deste nível. Assim que uma thread deste nível termina, o escalonador avisa ao nível correspondente, e um dos splits que estavam aguardando é colocado em execução. Caso o limite não tenha sido alcançado, então uma nova thread é criada para este split e sua execução se inicia imediatamente. Este controle é feito para não sobrecarregar a máquina com um número exagerado de threads, o que poderia levar o sistema operacional ao estado de *trashing* (i.e. máquina não realiza computação útil e fica apenas reescalonando processos). A implementação multithreaded é paralela e pode ser executada em máquinas com vários processadores, onde o ganho de performance será ainda maior. Futuramente, pretende-se adaptar esta implementação para suportar um ambiente completamente distribuído utilizando uma MPI (Message Passing Interface) padrão como PVM (Parallel Virtual Machine) [17]. O esforço necessário para fazer esta adaptação deverá ser pequeno, uma vez que o atual ambiente já é paralelo.

O algoritmo desta implementação foi tirado de [35] e funciona da seguinte forma. Inicialmente, o escalonador coloca um simulador para executar no nível 1 e aguarda até seu término, que pode ser alcançado por uma das seguintes formas:

- **alcance do threshold superior.** Neste caso, o simulador em execução é retirado do seu nível e k splits são escalonados para serem executados no nível acima. k é o número de replicações realizadas quando um simulador atinge este nível. A última réplica dos splits é marcada para ser executada quando terminar, seguindo a descrição do método. Os pontos I da Figura 5.2 ilustram esta condição de parada.

- **alcance do threshold inferior.** Neste caso, o simulador em execução é removido do seu nível e, caso seja a última réplica de um split (estará marcada para ser reiniciada), um novo simulador é escalonado para reiniciar a execução no nível inferior. Se esta réplica não estiver marcada, então as medidas de interesse são atualizadas e nenhuma continuação é feita. Os pontos IIa e IIb da Figura 5.2 ilustram estas duas condições de parada, respectivamente.

- **alcance do tempo máximo de simulação.** Neste caso, o limite máximo para o tempo de simulação especificado pelo usuário foi atingido. O simulador termina e as medidas de interesse são atualizadas. Os pontos III da Figura 5.2 ilustram esta condição de parada.

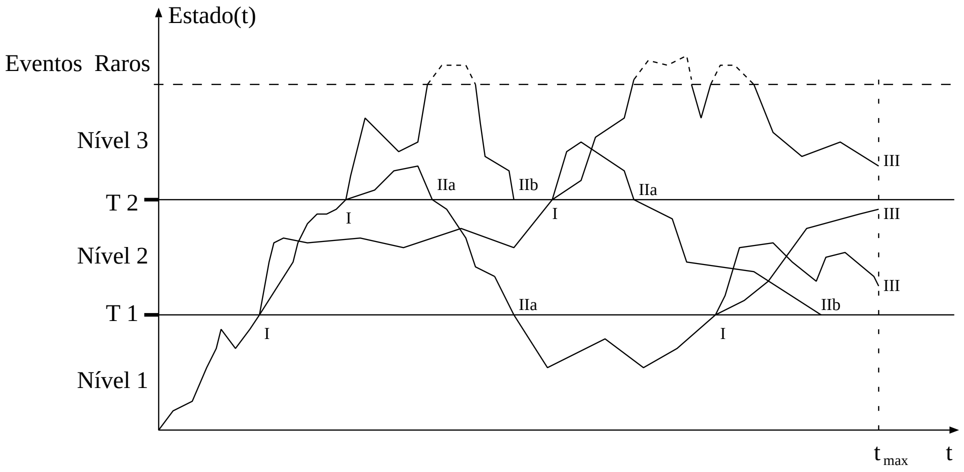


Figura 5.2: Pontos de parada de uma execução de simulação do método RESTART.

Um simulador sendo executado no nível 1 só termina quando atinge o próximo nível ou quando alcança o limite máximo de tempo especificado pelo usuário. Da mesma forma que um simulador executando no último nível só termina quando atinge o nível anterior ou quando alcança o limite de tempo especificado pelo usuário. Assim, a simulação prossegue até que todos os splits alcancem o limite máximo de tempo. Este algoritmo garante que pelo menos um simulador, em algum nível, estará sempre em execução.

5.2 Definição de Medidas de Interesse

A recompensa de taxa definida só deve ser acumulada quando o quando o estado do sistema estiver no subconjunto definido pelo último nível. O objetivo do simulador de eventos raros é provocar os eventos que estão no nível mais alto definido pelo usuário. Desta forma, apenas estados do nível mais alto devem ter associado a eles valores de recompensa.

Por exemplo, supondo que deseja-se calcular a fração de tempo que a fila de um sistema fonte/fila permanece acima de um determinado valor. Se a fila for grande e a carga da fonte sobre o servidor for baixa, o evento fila maior do que k é um evento raro. Podemos definir então níveis sobre a variável de estado que representa o tamanho da fila. Assim, se a simulação evoluir na direção da fila cheia, réplicas serão feitas. A fila com estado maior ou igual a k estaria no último nível.

A forma de especificar as medidas de interesse do simulador de eventos raros é exatamente a mesma utilizada no simulador convencional. Porém, o simulador de eventos raros suporta apenas medidas de recompensa de taxa. Essas recompensas são ponderadas pelo número de caminhos amostrais possíveis. Não é possível definir medidas de interesse com recompensa de impulsos no simulador de eventos raros. O cuidado para definir a medida de interesse, para que esta só seja alterada no último nível do simulador, deve ser tomado pelo usuário.

Ao definir uma recompensa de taxa para os estados de interesse, o simulador de eventos raros passa a monitorar esta medida. Toda vez que uma simulação do último nível atinge os estados do sistema onde existe recompensa, esta é acumulada. Ao sair do último nível o escalonador recolhe as medidas de interesse do simulador em transição e acumula o valor desta medida em seu acumulador. A Figura 5.3 ilustra um exemplo onde simulações que estão executando do último nível acumulam recompensas de taxa e ao terminarem passam esses valores ao escalonador.

No final da simulação o escalonador pondera o acúmulo da medida de interesse pelo número possível de simuladores que poderiam ter atingido o último nível. Toda simulação do último nível, ao terminar possui um CR e um TCR para as medidas

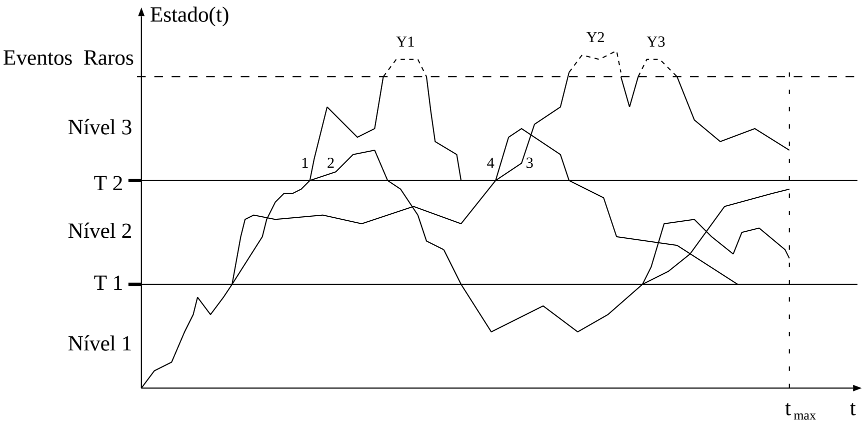


Figura 5.3: Medidas de interesse acumuladas durante a execução de uma simulação utilizando o método RESTART.

definidas. A valor de CR possui a recompensa de taxa acumulada pelo simulador. Esta medida foi definida no capítulo 2.

Essas variáveis são acumuladas em uma variável do escalonador. A equação seguinte mostra esta ponderação:

$CR = \sum_{k=1}^{N} CR_k,$

onde CR_k é o valor da medida de interesse do k-ésimo simulador no último nível.

$$CR(t_{max}) = \frac{CR}{\prod_{i=1}^{M} R_i}$$

onde R_i é o número de réplicas em cada nível e M é o número de níveis.

Observando a Figura 5.3 vimos que o valor da medida de interesse será dado por:

$$CR_1 = Y_1$$

$$CR_2 = 0$$

$$CR_3 = Y_2 + Y_3$$

$$CR_4 = 0$$

$$\text{Logo, } CR(t_{max}) = \frac{\sum_{k=1}^{4} CR_k}{\prod_{i=1}^{2} R_i} = \frac{Y_1 + Y_2 + Y_3}{4}$$

No final da simulação, ao atingir o tempo t_{max}, a medida CR indicará a recompensa acumulada no tempo $[0, t_{max}]$. O valor de $ACR(t_{max})$ da medida de interesse é obtido a partir de CR, dividindo seu valor por t_{max}. O análogo também se aplica à medida de interesse $TCR(t_{max})$.

5.3 Usando o Simulador

Para utilizar o simulador é preciso especificar a variável de estado que servirá para definição dos níveis, os *thresholds* para cada nível e o número de réplicas a ser feito em cada nível. Todas esses parâmetros devem ser especificados pelo usuário. Em geral, uma análise simples do modelo a ser simulado revela esses parâmetros com precisão adequada, principalmente se o modelo possuir alguma estrutura especial. Técnicas e algoritmos podem ser aplicadas para descobrir os valores para esses parâmetros. Existem estudos [35] que mostram como esses parâmetros podem ser calculados dinamicamente pelo simulador, porém, a variação entre os valores especificados pelo usuário e os valores ótimos desses parâmetros não prejudica a eficiência do método RESTART, como foi analisado por [34]. Sua eficiência é garantida, mesmo que parâmetros não ótimos sejam utilizados.

A ferramenta de simulação ASTRO [33], que implementa um biblioteca de funções para simulação de eventos raros utilizando o método de RESTART, também necessita que o usuário defina esses parâmetros.

O usuário, então, especifica esses parâmetros na interface gráfica que são passados ao simulador. O formato desta definição está ilustrado abaixo:

```
Server_Queue.queue
20 5
40 5
60 5
80 5
```

Neste exemplo, o usuário definiu que a variável de estado a ser observada pelo simulador de eventos raros é a variável "queue" do objeto "Server_Queue". Além disso, os *thresholds* são os estados 20, 40, 60 e 80, e em cada um deles, cinco réplicas deverão ser feitas. Logo, este exemplo define 5 níveis sobre a variável de estado "queue", entre $-\infty$-20, 19-40, 39-60, 59-80, 79-$+\infty$. Note que os extremos não possuem limites inferior ou superior, podendo a variável de estado assumir qualquer valor. O simulador utilizará essas informações para criar as estruturas de dados iniciais e coordenar a simulação nos níveis.

O resultado do simulador é apresentado da mesma forma que os resultados do simulador de eventos discretos. Assim sendo, é possível definir diversas rodadas de simulação e, neste caso, o simulador apresentará a média amostral, a variância e o desvio padrão da medida de interesse. Esta saída é idêntica a do simulador convencional.

A ferramenta TANGRAM-II não permite que os *thresholds* sejam definidos sobre mais de uma variável de estado. Isto aumenta a complexidade da definição da sequência de estados que leva aos estados raros. Os conjuntos de estados C_i, apresentado no capítulo 2, definidos pelo usuário, precisam estar contidos um no outro, podendo tornar difícil suas especificações. Porém, é possível que o usuário defina um objeto específico para delinear a sequência adequada para os estados raros. Este objeto pode receber mensagens de outros objetos do modelo indicando o incremento/decremento de sua única variável de estado. Desta forma, os *thresholds* podem ser definidos sobre a variável de estado deste objeto, mascarando o comportamento do modelo.

5.4 Exemplo: Fila M/M/1/k

Ilustra-se agora o uso do simulador para estimar probabilidades baixas e compara-se os valores obtidos através da simulação com resultados de métodos analíticos. Para realizar esta tarefa, foi utilizado o tradicional sistema de fila M/M/1/k [30] onde uma fonte de pacotes com distribuição exponencial envia pacotes a um servidor. O servidor possui uma fila onde armazena os pacotes excedentes enquanto realiza o serviço. Esta fila possui um tamanho máximo para o número de pacotes que podem ser armazenados, k.

Para calcular a probabilidade da fila estar cheia, ou seja, a fração de tempo

	55%	65%	75%	85%	95%	99%
thresholds	32	19	14	8	4	2
réplicas	6	9	9	7	5	5

Tabela 5.1: Número de pontos de quebra e réplicas para diferentes cargas no servidor.

em que o sistema passa com sua fila cheia, utiliza-se a ferramenta TANGRAM-II definindo um modelo para este sistema com dois objetos: fonte de pacotes e fila. Define-se uma medida de interesse utilizando recompensa de taxa associando ao estado onde a fila está cheia o valor 1. Logo, ao final de um determinado período de tempo, a variável $ACR(t)$ indicará a fração de tempo em que o sistema permaneceu com a fila cheia. Se a fila for grande e a carga sobre o servidor baixa, então esta medida está associada a um evento raro.

Neste exemplo, é utilizado um tamanho máximo de fila de 100 pacotes, fixando-se a capacidade de serviço do servidor em 1 pacote/segundo. A taxa de transmissão da fonte também é constante. Para variar a carga sobre o servidor, serão utilizados os valores 0.55, 0.65, 0.75, 0.85, 0.95, 0.99.

O cálculo dos *thresholds* e do número de réplicas em cada nível foi feito seguindo a orientação descrita no capítulo 2.2. Para escolher os parâmetros ótimos, resolveu-se o modelo analiticamente. Neste exemplo, valores para os parâmetros bastante próximos ao ótimo puderam ser utilizados. Por possuir uma estrutura regular e utilizando a solução analítica, o cálculo dos parâmetros de simulação para este modelo torna-se trivial. Os pontos de quebra estão igualmente espaçados pela fila e possuem o mesmo número de réplicas em todos os níveis. Porém, para cada utilização da fila, um conjunto de parâmetros é definido, uma vez que a dinâmica das probabilidades varia conforme a taxa de transmissão. A Tabela 5.1 mostra o número de *thresholds* definidos sobre a fila e o número de réplicas criadas em cada nível para as diferentes cargas sobre o servidor.

O simulador foi executado fazendo 10 rodadas por um tempo de simulação de 10000 segundos. Ao final, obteve-se a média amostral e a variância para a medida de interesse definida, no caso a fração de tempo de fila cheia. Para resolver este modelo analiticamente, foi gerada a cadeia de Markov associada sendo resolvida com o método de solução GTH. Os resultados obtidos são apresentados na Figura 5.4.

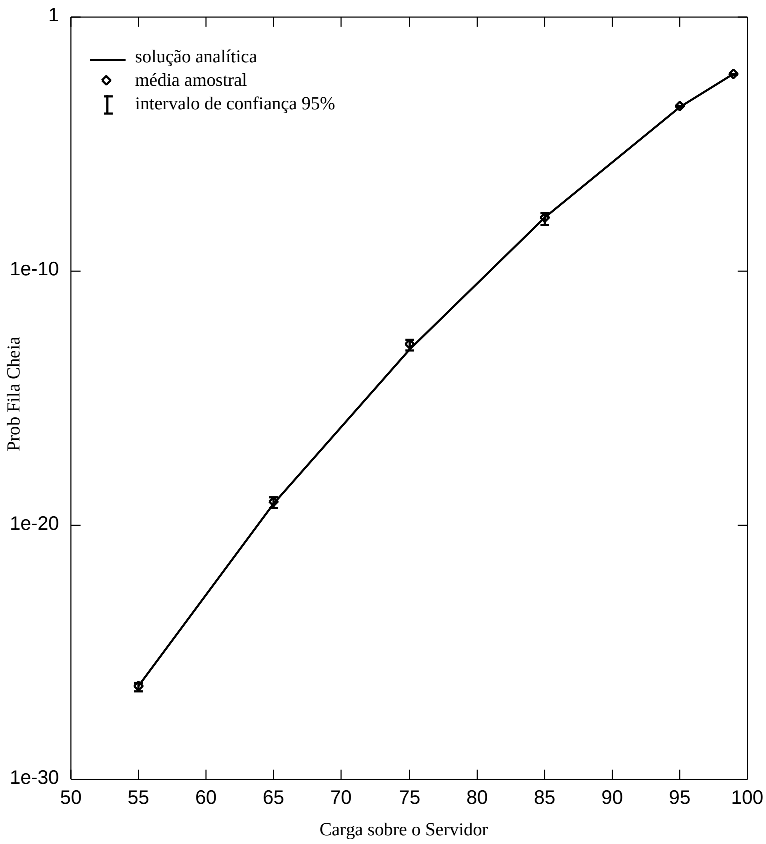


Figura 5.4: Comparação entre simulação de eventos raros e resultado analítico.

Observa-se que todos os valores obtidos utilizando o método analítico ficaram dentro do intervalo de confiança de 95% calculado pelo simulador. Até mesmo para medidas baixas, na ordem de 10^{-27} o intervalo de confiança foi pequeno, mostrando a eficiência do simulador.

5.4.1 Escolhendo Parâmetros para o Método RESTART

No exemplo apresentado na seção anterior, os parâmetros para o simulador de eventos raros foram obtidos através das equações apresentadas no capítulo 2. Desta forma, parâmetros próximos ao ótimo puderam ser utilizados. Apresenta-se agora a metodologia utilizada na escolha desses parâmetros.

	0	1	2	3	4	5
$P[(Fila \geq k]$	1.00	0.650	0.422	0.275	0.179	0.116

Tabela 5.2: Função de distribuição de probabilidades para a fila com carga 65%.

No capítulo 2 apresentamos a seguinte equação, abordada em [35], que define os parâmetros ótimos para o método RESTART. Os parâmetros do método (definição dos pontos de quebra e número de réplicas em cada ponto) devem ser estipulados de forma a se aproximarem dos valores definidos por estas equações.

$$P_i = e^{-2}, R_i = e^2$$

onde P_i é dado por $P[C_i|C_{i-1}]$, $1 < i \leq M$, $P_1 = P[C_1]$; R_i, $1 \leq i \leq M$, é o número de réplicas feitas ao passar pelo *threshold* i, e M é o número de níveis.

Lembrando que $C_1 \supset C_2 \supset \ldots \supset C_M \supset A$ são os conjuntos definidos sobre o espaço de estados do modelo. No exemplo apresentado, apenas uma variável de estado é necessária para representar o estado do modelo e esta indica o número de pacotes na fila. Logo, C_i é dado por *número de pacotes na fila* $\geq i$. Calcula-se agora o primeiro ponto de quebra para este modelo. Para isto é necessário encontrar o ponto onde $P[C_1|C_0]$ seja igual ou próximo de e^{-2}. Desta forma tem-se que

$$P_1 = P[C_1|C_0] = P[(Fila \geq k_1)|(Fila \geq 0)] = P[(Fila \geq k_1)] \simeq e^{-2}$$

Basta então, encontrar o ponto onde o número de pacotes na fila satisfaça (ou se aproxime) da equação acima. No simples exemplo apresentado, o modelo pode ser resolvido utilizando um método analítico, obtendo então os valores das probabilidades sobre a fila, ou seja, $P[(Fila = k]$.

Utilizando o resultado obtido analiticamente, o cálculo da função de distribuição torna-se trivial. Para o exemplo onde a carga sobre o servidor é igual a 65a tabela 5.2 apresenta a função de distribuição de probabilidades para alguns tamanhos de fila.

Como $e^{-2} = 0.135$ a melhor escolha para o valor de k_1 neste caso é 5. Então, define-se o primeiro ponto de quebra quando a fila possui 5 pacotes.

O segundo parâmetro a ser definido é o número de réplicas a serem feitas ao atingir este ponto de quebra. O valor ótimo para o número de réplicas é dado pela seguinte equação:

$$R_i = 1/P_i$$

Então $R_1 = 1/0.116 = 8.62$. Como o número de réplicas é inteiro, aproxima-se este valor para $R_1 = 9$. Com isto, define-se os dois primeiros parâmetros para o método.

Para definir o segundo ponto de quebra e o número de réplicas para este ponto, a mesma metodologia deve ser empregada. Seguindo a equação para definição dos pontos ótimos de quebra, temos

$$P_2 = P[C_2|C_1] = P[(Fila \geq k_2)|(Fila \geq k_1)] = P[(Fila \geq k_2)|(Fila \geq 5)] \simeq e^{-2}$$

O valor de k_2 deve ser encontrado respeitando a equação acima. Porém, por este modelo ser simples ao condicionar em Fila $\geq$ 5 a função de distribuição da fila permanece igual. Ou seja, dado que a fila é maior que cinco, a probabilidade da fila ser maior do que 6 é igual a probabilidade da fila ser maior do que 1. Desta forma, o próximo ponto de quebra será dado por $k_2 = 10$, pois $P[(Fila \geq 10)|(Fila \geq 5)] = 0.116$ que é a mesma probabilidade de P_1. Da mesma forma, o número de réplicas a serem feitas neste ponto de quebra é dado por $R_2 = 9$.

Seguindo esta metodologia é possível definir todos os pontos de quebra e o número de réplicas a serem feitas em cada ponto. O número total de pontos de quebra é obtido quando a variável de estado indicando o tamanho da fila atinge seu valor máximo. Assim sendo, os pontos de quebra para este exemplo ocorrem quando o tamanho da fila é $5, 10, 15, 20, \ldots, 95$ e o número de réplicas em cada ponto é sempre $R_i = 9$. O número total de pontos de quebra, M, para este exemplo é 19.

Neste exemplo, utiliza-se a solução analítica para obtenção dos parâmetros ótimos. Porém, nem sempre é possível resolver o modelo analiticamente. Nesses casos, o usuário adquirir um conhecimento do processo estatítico da variável sendo utilizada para definir os pontos de quebra. Uma forma de conhecer melhor o processo estatítico desta variável é através da solução de um modelo simplificado. O usuário pode reduzir a complexidade do seu modelo tornando a solução analítica possível. Desta forma, os parâmetros para o método RESTART podem ser melhor especificados. Outra forma de se conhecer o processo estatístico da variável em questão é através da observação de pequenas simulações. O usuário poderia adquirir conhecimento do processo estatístico fazendo simulações rápidas e com isto definir parâmetros para seu modelo. Lembrando que uma das vantagens do método RESTART é sua robustez com relação a escolhas dos parâmetros a serem utilizados na simulação. Não sendo assim, necessário definir parâmetros ótimos para ter um ganho significativo de eficiência.

Capítulo 6

Algoritmo e Modelo para Recuperação de Pacotes de Voz na Internet

DURANTE os estudos do mestrado, a área de transmissão de voz sobre a Internet foi abordada e alguns trabalhos foram elaborados. Esses estudos visavam o entendimento do processo de perda de pacotes de voz com o objetivo de avaliar diferentes algoritmos que fossem capazes de recuperar essas perdas. Surgiu assim, uma proposta de um algoritmo para recuperação de pacotes que utiliza redundância. Este se mostrou mais eficiente que outros algoritmos descritos na literatura [15].

Um modelo de um esquema simples do algoritmo proposto foi recentemente elaborado na ferramenta TANGRAM-II. Neste capítulo apresenta-se parte deste estudo, juntamente com a descrição do algoritmo e a especificação do modelo. Medidas de interesse sobre o comportamento do modelo foram obtidas através de simulações e são reportadas no final deste capítulo. Este exemplo mostra a versatilidade do simulador da ferramenta TANGRAM-II para modelar sistemas complexos.

6.1 Transmissão de Voz em Tempo Real

Em um aplicativo de voz de tempo real, a perda de pacotes e o atraso entre eles são fatores fundamentais que influenciam na recuperação do sinal transmitido e, conseqüentemente, na qualidade da voz que chega ao destinatário. Pacotes que são perdidos na rede causam falhas no sinal de voz no aplicativo que os recebe. Essas falhas podem prejudicar a compreensão da conversa caso as perdas sejam longas (i.e. de vários pacotes consecutivos) ou muito freqüentes. Porém, falhas raras que estejam muito espaçadas são imperceptíveis e não comprometem a conversa. Por isso, para manter uma boa qualidade de voz, o número de pacotes que chegam entre duas falhas deve ser grande e o número de falhas deve ser pequeno.

O segundo fator fundamental para garantir uma boa qualidade de voz, está relacionado com o atraso aleatório inserido pela rede entre os pacotes. Este atraso, conhecido como *jitter*, mede a variação do tempo de chegada entre pacotes consecutivos em relação ao tempo de geração (constante) desses pacotes na fonte. Se atrasos entre pacotes são freqüentes e longos, então falhas na voz irão ocorrer no aplicativo destino. Para garantir uma boa qualidade de voz, é necessário manter o *jitter* baixo, em torno de zero.

A rede é a maior responsável pela má qualidade do áudio, por não oferecer a qualidade de serviço adequada. Para solucionar este problema, duas abordagens são consideradas na literatura. A primeira é a modificação do protocolo de transporte e das políticas de escalonamento de pacotes nos roteadores, para que suportem tráfego de tempo real. Com isto seria possível garantir a qualidade de serviço exigida pelo aplicativo. Para implementar esta modificação será necessário que a rede (protocolo + roteadores) suporte controle de admissão, reserva de banda, alocação prévia de recursos e disciplinas de escalonamento sofisticadas. Estes mecanismos ainda não foram implementados na Internet, sendo esta uma área de pesquisa onde novas propostas e análises de desempenho estão surgindo.

A segunda abordagem é implementar nos aplicativos de tempo real, algoritmos que melhorem a qualidade do serviço oferecido pela rede. O objetivo desses algoritmos é suavizar o impacto da mudança das características da rede ao longo do tempo, sobre a qualidade de serviço. Nos aplicativos de voz, esses algoritmos tentam reduzir o *jitter* e a taxa de perda de pacotes, que são os principais responsáveis pela qualidade do áudio.

Jitter existe em quase todas as redes, e várias propostas para solucionar ou minimizar o problema foram sugeridas. Em uma delas, foi visto que a utilização de um algoritmo simples, que utiliza um buffer na saída da rede, reduz significativamente o *jitter* [16, 13]. Neste algoritmo, os pacotes que chegam da rede não são consumidos imediatamente pelo aplicativo destino, mas armazenados em um buffer até que um certo número de pacotes seja acumulado. Só então o primeiro pacote que chegou é entregue ao destino. Desta forma, elimina-se por completo o *jitter* negativo e o *jitter* positivo pode ser minimizado. Porém, o atraso fim a fim pode prejudicar a interatividade da conversa. Por isto, o número de pacotes inicialmente acumulados no buffer não deve ser grande. Em [16] mostrou-se que este número pode ser pequeno e mesmo assim uma redução significativa do *jitter* é obtida.

O segundo tipo de algoritmo tenta recuperar pacotes perdidos durante a transmissão da voz. Esta recuperação pode ser feita de diversas maneiras através de diferentes algoritmos. Na próxima seção, apresentam-se os algoritmos mais utilizados.

6.2 Algoritmos para Recuperação de Perdas

A qualidade do áudio em aplicativos de voz é afetada, principalmente, pela perda dos pacotes na rede. Com isto é essencial que pacotes perdidos sejam, de alguma forma, recuperados. Alguns aplicativos de transmissão de voz já implementam esquemas de recuperação de pacotes perdidos. Recentemente, foi lançada uma versão do FreePhone [4] que utiliza redundância para recuperar pacotes perdidos. O Internet Phone, também possui um esquema de redundância, porém proprietário, por ser um aplicativo comercial. O vat [23] ainda não possui nenhum esquema para recuperação de perdas. Nesta seção são abordados alguns algoritmos para recuperação de pacotes.

Existem, basicamente, duas formas de recuperar pacotes perdidos em transmissões de dados. A primeira, consiste em retransmitir o pacote de dados perdido. A segunda, consiste em reconstruir os pacotes perdidos utilizando informação redundante que é enviada juntamente com o fluxo de dados.

O esquema de retransmissão dos pacotes perdidos é bastante conhecido e utilizado por diversos protocolos como, por exemplo, o protocolo ARQ (Automatic Repeat Request) (e.g. [3]). Todos esses esquemas utilizam a técnica de loop fechado para fazer a retransmissão, e somente os pacotes que não chegam ao destino são retransmitidos. Porém, uma retransmissão só pode ser disparada de duas maneiras. Ou por um *nack* vindo do aplicativo destino, informando que um pacote se perdeu, ou por esgotamento de um temporizador acionado no aplicativo origem que esperava por um *ack* do aplicativo destino, confirmando a chegada do pacote.

Este mecanismo de recuperação de pacotes perdidos, em geral, não funciona adequadamente, no caso de transmissão de voz em tempo real. A retransmissão de um pacote perdido pode demorar significativamente, o que prejudica a interatividade da conversa, causando súbitas interrupções na fala. O atraso na retransmissão do pacote perdido está relacionado com o *round-trip time* da conexão. Em redes como a Internet, onde este tempo pode ser muito alto em relação ao tempo de geração dos pacotes, esses algoritmos se tornam inviáveis.

A segunda forma de recuperar os pacotes perdidos, conhecida como FEC (Forward Error Correction), é baseada em loop aberto. Neste mecanismo, redundância é enviada juntamente com os pacotes de dados, e em caso de perda, o pacote é reconstruído. A vantagem deste esquema é que não existe retransmissão, porém, sua desvantagem é que nem todos os pacotes perdidos podem ser recuperados. O número de perdas que o algoritmo irá recuperar, está diretamente relacionado com a quantidade de redundância que o mesmo irá enviar. Além disso, o *overhead* introduzido pela redundância pode ser grande, diminuindo a eficiência da transmissão, e portanto, a quantidade de informação redundante deve ser minimizada.

Existem diversas variações de esquemas do tipo FEC para recuperar erros. Um esquema bastante conhecido, e utilizado em outras áreas da computação, é a utilização do ou-exclusivo para compor a redundância. Neste esquema, divide-se a seqüência de pacotes em janelas. Em cada janela, a redundância é obtida através do ou-exclusivo de todos os pacotes pertencentes à janela. A redundância, neste caso, possui o mesmo tamanho dos pacotes de dados e é enviada juntamente com o primeiro pacote da próxima janela. Desta forma, o primeiro pacote de cada janela possui duas vezes o tamanho de um pacote de dados. O *overhead* deste esquema é dado por $1/k$, onde k é o tamanho da janela. A Figura 6.1 ilustra este exemplo onde o tamanho da janela é de 5 pacotes.

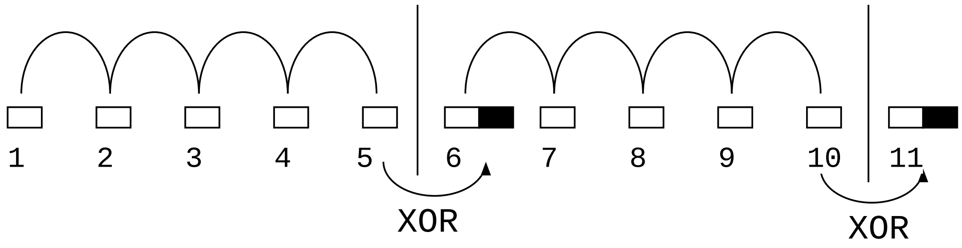


Figura 6.1: Esquema de redundância com janela de 5 pacotes.

Utilizando este esquema, pode-se recuperar até 1 pacote em cada janela. Se mais de um pacote for perdido na mesma janela então nenhum deles pode ser recuperado. Outra desvantagem deste esquema é o aumento do retardo fim-a-fim quando o tamanho da janela é aumentado, pois, para recuperar um pacote perdido, é necessário que cheguem todos os pacotes da janela atual e o primeiro pacote da próxima janela (que possui a redundância) para que a reconstrução possa ser feita.

Uma de suas vantagens é a simplicidade com que pode ser implementado. Além disso, conforme foi visto na seção 1, para eliminar o *jitter*, um buffer na saída da rede já é utilizado, e este mesmo buffer pode também ser usado para reconstruir os pacotes perdidos. Mantendo o tamanho da janela pequeno, não haverá retardos adicionais.

É importante ressaltar que a redundância em si, não está coberta por nenhum ou-exclusivo e que somente os pacotes contendo áudio podem ser recuperados. Desta forma, na figura 6.1, se os pacotes 6 e 3 se perdessem, apenas o pacote 6 seria recuperado. O pacote 3 não poderia ser recuperado, pois a redundância que protege esta janela foi perdida.

6.2.1 Proposta de um Esquema de Redundância

Apresenta-se agora uma modificação do esquema simples de redundância, apresentado na seção anterior, que utiliza ou-exclusivo para compor a redundância.

A modificação consiste em aumentar a redundância que protege cada janela. Desta forma, teremos mais de uma seqüência de pacotes dentro da mesma janela protegidos por um ou-exclusivo. A redundância referente a cada seqüência continua sendo enviada nos primeiros pacotes da próxima janela. Chamaremos de cadeia uma seqüência de pacotes protegida pela mesma redundância. Para formar o ou-exclusivo, as cadeias são intercaladas dentro da janela. A Figura 6.2 ilustra um exemplo de janela com 6 pacotes e duas redundâncias. Com isto o número de pacotes que pertencem a uma cadeia é inferior ao tamanho da janela.

O *overhead* deste esquema é dado por k/w onde k é o número de redundâncias (ou cadeias) em cada janela e w é o tamanho da janela.

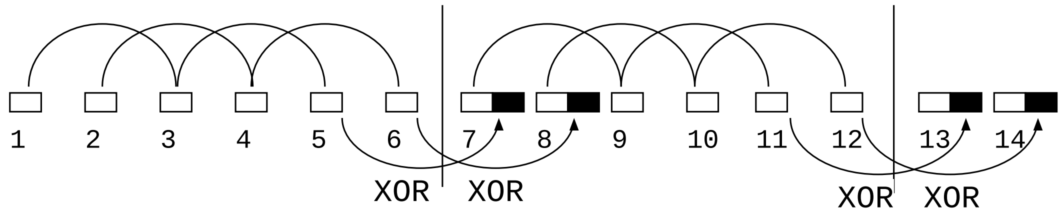


Figura 6.2: Janela de tamanho 6 com dois pacotes de redundância.

A primeira janela no exemplo da Figura 6.2 possui duas cadeias de pacotes. A primeira é formada pelos pacotes 1,3,5 e o ou-exclusivo desses pacotes é enviado juntamente com o pacote 7. A segunda cadeia é formada pelos pacotes 2,4,6 e sua redundância é enviada juntamente com o pacote 8. Neste exemplo, os pacotes 7 e 8 possuem o dobro de seu tamanho original, pois carregam, além da amostra de áudio, a redundância da janela anterior. Entretanto, como foi visto em [14], o aumento no tamanho do pacote não afeta significativamente sua probabilidade de perda. Utilizando esta modificação, é possível recuperar perdas de tamanho maior que um. Na Figura 6.2, podemos recuperar qualquer perda (consecutiva) de tamanho dois. Isto só é possível pois as cadeias que compõem a redundância estão intercaladas, e uma perda de tamanho dois irá afetar somente um pacote em cada cadeia.

É importante notar que este esquema possui o mesmo *overhead* que o esquema de ou-exclusivo simples 1:3 (uma redundância e janela com três pacotes). Porém esses dois mecanismos têm comportamento completamente diferente. O esquema 1:3 nunca recupera perdas (consecutivas) de tamanho dois. O esquema 2:6 (duas redundâncias e janela com seis pacotes) não recupera duas perdas de pacote que ocorram na mesma cadeia (i.e. se os pacotes 1 e 5 se perdessem, o esquema 1:3 recuperaria ambos, enquanto o 2:6 não recuperaria nenhum).

Vários esquemas com o mesmo *overhead* podem ser propostos, e a eficiência de cada um deles será diferente de acordo com as características da rede. A escolha do esquema mais eficiente para recuperar os pacotes perdidos depende, diretamente, do processo de perda de pacotes.

Uma segunda modificação foi feita de forma que dois esquemas distintos de redundância sejam misturados. O objetivo é recuperar perdas simples, de um pacote, e perdas de tamanho maiores do que um. Este esquema está descrito em [14].

6.3 Modelo para o Algoritmo Proposto

Para modelar o algoritmo proposto uma especificação da versão mais simples do algoritmo foi feita na ferramenta TANGRAM-II. Foi criado um modelo para o esquema de redundância 2:6, ilustrado na Figura 6.2. Este modelo consiste de 4 objetos. O gerador de pacotes de voz; um descartador de pacotes; um recuperador de pacotes e um contador de pacotes recuperados.

O gerador de pacotes simula uma fonte de voz ininterrupta que envia pacotes ao objeto descatador a uma taxa constante. A taxa de transmissão dos pacotes não influi sobre as medidas de interesse, pois neste modelos estamos avaliando medidas relacionadas com a recuperação de pacotes. O objeto descartador de pacotes marca os pacotes como perdidos ou não. O processo de descarte está especificado no objeto. Para este modelo utilizamos um processo de descarte simples onde cada pacote possui uma probabilidade p de ser descartado independente dos outros pacotes. Este objeto envia então os pacotes para o recuperador de pacotes. Os pacotes chegam ao recuperador marcados com 0 ou 1 indicando se o pacote foi perdido ou não. O recuperador acumula uma janela de pacotes e a redundância necessária. Após montar a janela o recuperador executa o algoritmo recuperação e verifica quantos pacotes podem ser recuperados. Para cada pacote recuperado, uma mensagem é enviada ao objeto contador de recuperações. Após processar toda a janela, o objeto recuperador de pacotes zera suas variáveis e aguarda a chegada de novos pacotes. A descrição deste objeto é apresentada em um dos exemplos no apêndice A.

Definiu-se as seguintes medidas de interesse utilizando recompensa de impulso:

- **pacotes transmitidos** - número de pacotes enviados pela fonte
- **pacotes perdidos** - número de pacotes descartados
- **pacotes recebidos** - número de pacotes que chegaram ao objeto recuperador
- **pacotes recuperados** - número de pacotes recuperados pelo algoritmo

Essas medidas foram obtidas associando recompensas de impulso ao disparo de eventos, como é o caso de pacotes transmitidos, ou associando impulsos à chegada de mensagens como é o caso das outras três medidas. Utilizando essas medidas é possível obter diversas outras relações como por exemplo:

- fração de pacotes recuperados = número de pacotes recuperados / número de pacotes perdidos
- fração de pacotes corretos = (número de pacotes recebidos + pacotes recuperados) / número de pacotes transmitidos

Prob. Perda	Recebidos	Perdidos	Recuperados
0.01	49491.9	508.1	492.6
0.05	47502.7	2497.3	2137.2
0.10	45013.3	4986.7	3636.5
0.15	42481.4	7518.6	4612.0
0.20	40015.5	9984.5	5133.3
0.25	37466.7	12533.3	5263.5
0.30	34974.9	15025.1	5143.1
0.35	32506.6	17493.4	4799.2
0.40	29971.3	20028.7	4291.7
0.50	25019.7	24980.3	3139.4

Tabela 6.1: Média amostral do resultado da simulação

Para avaliar o algoritmo de recuperação de pacotes, o processo de perda foi variado utilizando as seguintes probabilidades de perda: 1%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40% e 50%. Para cada valor da probabilidade de perda, dez rodadas de simulação foram feitas e em cada rodada 50000 pacotes foram transmitidos.

6.4 Resultados Obtidos

Após 10 rodadas de simulação obteve-se a média amostral entre as rodadas e o intervalo de confiança de 95%. Os resultados obtidos estão apresentados nos gráficos abaixo. O modelo simulado, que representa o esquema 2:6 do algoritmo proposto, apresentou resultados satisfatórios para baixas taxas de perda. A tabela 6.1 mostra o resultado da simulação deste modelo, onde são apresentados as médias amostrais para todas as medidas definidas.

O gráfico da Figura 6.3 apresenta a fração dos pacotes corretos para diferentes taxa de perda. O gráfico ilustrado na Figura 6.4 apresenta a fração de pacotes que foram recuperados pelo algoritmo. Podemos observar que este esquema possui um bom desempenho para taxas de até 15%. Estudos mais detalhados sobre diferentes

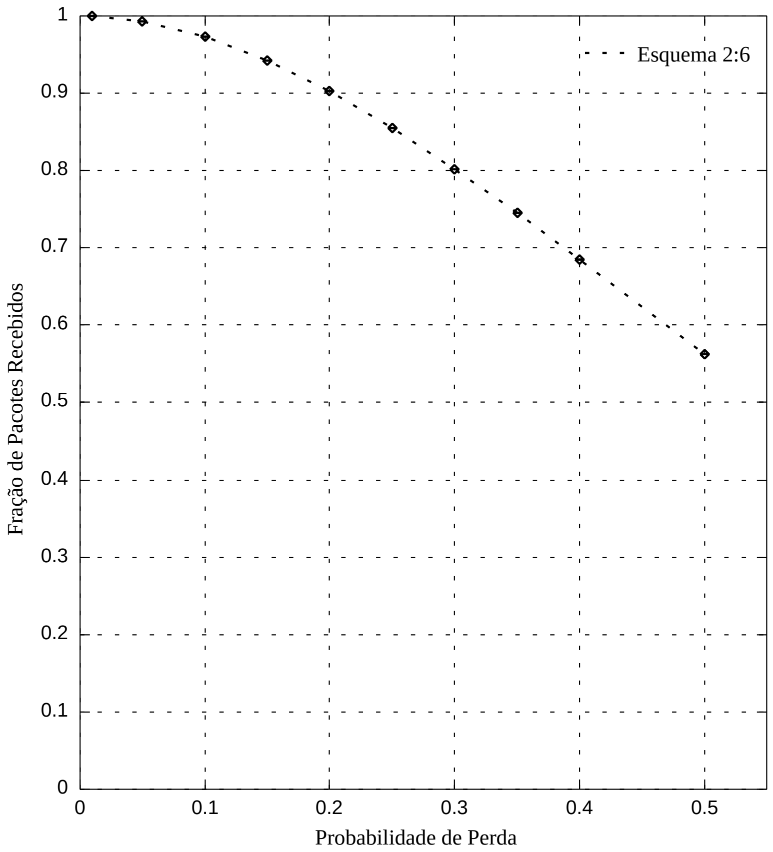


Figura 6.3: Fração de pacotes corretos para diferentes taxas de perda.

esquemas de redundância estão sendo feitos sobre modelos específicos para cada esquema.

Além de simulado, este modelo pode ser analisado analiticamente colocando o evento de transmissão de pacotes com distribuição exponencial. Neste caso, a cadeia de Markov associada ao modelo é gerada, podendo ser resolvida com um método de solução analítico. Os valores para as recompensas de impulso podem ser obtidas através da solução em estado estacionário, obtendo assim, o valor médio para cada uma das medidas fracionárias.

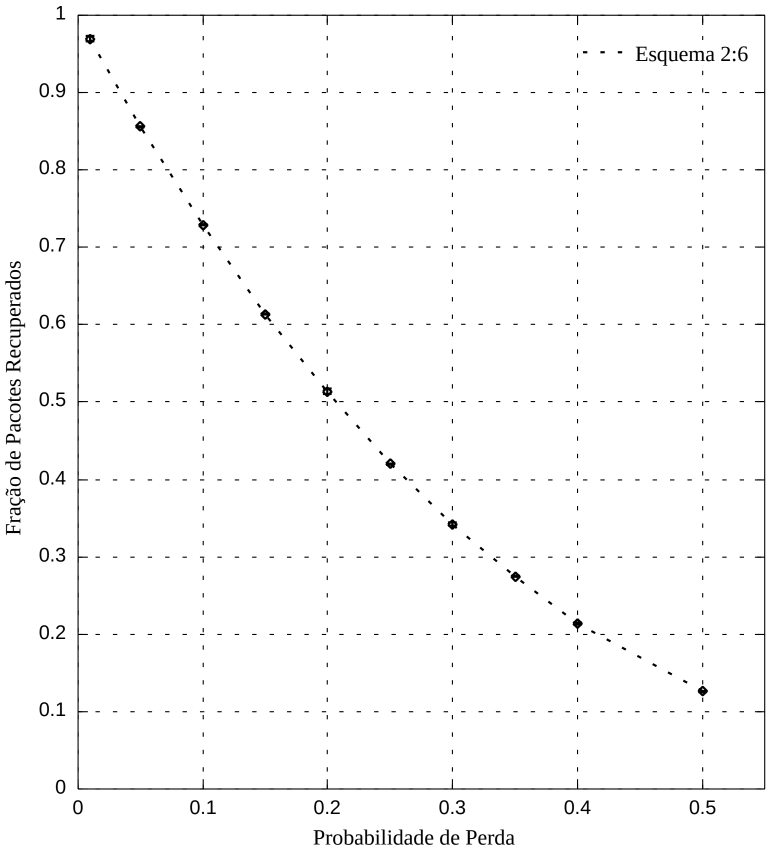


Figura 6.4: Fração de pacotes recuperados para diferentes taxas de perda.

Capítulo 7

Conclusão

A modelagem de sistemas de computação e comunicação é uma das tarefas mais importantes no processo de projeto e desenvolvimento de qualquer nova tecnologia. Além de modelar, é preciso obter medidas de interesses precisas sobre o sistema, pois são esses números que refletem e aproximam o comportamento do sistema real. Uma das dificuldades em se obter informações relevantes sobre um modelo está no processo de especificação das medidas de interesse. Uma forma genérica e bastante flexível de especificar essas medidas é através do uso de modelos com recompensas.

Esta tese apresenta a metodologia desenvolvida para definição de medidas de interesse através de recompensas de taxa e de impulso na ferramenta TANGRAM-II. Uma sintaxe para especificação de recompensas foi definida dentro do paradigma de modelagem da ferramenta. Esta linguagem oferece uma forma genérica para definir recompensas, permitindo a especificação de medidas complexas e refinadas. A sintaxe para especificação foi comparada com o módulo de definição de recompensas da ferramenta UltraSAN, apresentando maior funcionalidade e flexibilidade.

Após definidas as medidas de interesse, um modelo precisa ser resolvido para que suas medidas possam ser analisadas. A solução de um modelo pode ser feita através de métodos analíticos/numéricos ou através de simulações. Muitas vezes, métodos analíticos não se aplicam à solução devido às determinadas características modelo, tornando esses métodos inapropriados ou inpraticáveis. Quando isto ocorre, a simulação, em geral, é a única técnica que pode ser utilizada para resolver esses modelos. Um módulo robusto de simulação de eventos discretos foi desenvolvido e acoplado a ferramenta TANGRAM-II. Este simulador é capaz de simular qualquer modelo especificado na ferramenta. Todos os modelos que podem ser resolvidos analiticamente podem ser simulados, oferecendo uma boa maneira de validar e comparar resultados entre as duas técnicas de solução. O simulador oferece funcionalidades de modelagem adicionais em comparação ao método de geração do modelo matemático. Por exemplo o uso de diferentes funções de distribuição de probabilidade para definição dos eventos, e amostragem múltipla de um mesmo evento (clonagem de eventos). Uma das funções de distribuição é a "FILE", onde o usuário define um arquivo que possui os tempos de ocorrência do evento. Isto permite, por exemplo, que um objeto modelando uma fonte de vídeo possa ser construído a partir de um *trace* real com as informações dos quadros do filme. Com estas funcionalidades, modelos mais completos e detalhados, que se aproximam mais do sistema real, podem ser especificados e simulados. Além disso, a ferramenta permite que simulações interativas sejam feitas, sendo possível acompanhar na interface gráfica a evolução do caminho amostral. O simulador evolui sobre os estados do sistema, que têm os valores atualizados na interface gráfica onde o modelo foi especificado. É possível ainda, definir animações gráficas para ilustrar o comportamento do sistema ao longo da simulação. Essas animações são definidas pelo usuário e executadas na interface gráfica da ferramenta. Utilizando esses dois recursos o projetista pode compreender melhor e validar o funcionamento do modelo, uma vez que a simulação é feita passo-a-passo.

Apesar do uso de simulação ser bastante genérico, a técnica de simulação tradicional perde eficácia se eventos raros precisam ser estimados. Medidas de interesse definidas sobre um conjunto de eventos raros podem levar anos para serem simuladas com precisão em simuladores convencionais. Para amenizar este problema, técnicas de simulação de eventos raros vem sendo desenvolvidas e aprimoradas. Uma recente técnica, bastante robusta e flexível, é o método conhecido como Splitting/RESTART. Este permite simular modelos genéricos e estimar medidas de interesse com probabilidades extremamente baixas (i.e. 10^{-30}). Esta técnica foi adaptada e acoplada ao módulo de simulação da ferramenta TANGRAM-II. Com isto, modelos com medidas de interesse que tenham valores extremamente baixos podem ser simuladas eficientemente. Em sua implementação, um ambiente *multi-threaded* foi utilizado dando maior flexibilidade e desempenho ao método. Uma das vantagens do módulo de simulação de eventos raros é que as medidas de interesse são definidas da mesma forma que as utilizadas no simulador convencional.

Durante o trabalho de pesquisa para a tese, estudos relacionados a perda de pacotes de voz na Internet foram feitos. Dessa pesquisa surgiu um proposta de um algoritmo simples para recuperação de pacotes de voz que utiliza redundância. Uma análise comparativa entre diferentes esquemas deste algoritmo e outros mecanismos de recuperação foi realizada. Porém, nesta análise apenas *traces* de pacotes obtidos na Internet foram utilizados. Um modelo detalhado deste algoritmo foi especificado na ferramenta TANGRAM-II. Este modelo foi analisado através de simulação e diversas medidas de interesse sobre o algoritmo proposto puderam ser obtidas. Este modelo permitirá uma melhor avaliação do algoritmo proposto.

7.1 Sugestões para Trabalhos Futuros

Com o desenvolvimento do módulo de simulação, e o suporte a simulação de eventos raros na ferramenta TANGRAM-II, diversas possibilidades para possíveis trabalhos futuros apareceram. Entre essas tarefas podemos citar a expansão do ambiente paralelo, multithreaded, existente hoje no simulador, para um ambiente distribuído. Desta forma, simuladores podem ser distribuídos por uma rede de computadores através de algum padrão para troca de mensagens e gerenciamento da computação distribuída (i.e. PVM).

Com o uso de simulação, o usuário poderia ter acesso aos valores das variáveis de recompensa durante a especificação das ações associadas a eventos ou mensagens. Ele poderia consultar o valor de uma determinada recompensa para tomar uma decisão e alterar seu valor. Desta forma, a variável de recompensa passa a ser uma variável contínua que compõem o estado do sistema, podendo ser tratada exatamente como uma variável de estado. Isto irá permitir modelos ainda mais detalhados e completos, dando ao usuário maior flexibilidade, sem introduzir custos adicionais durante a simulação. Este trabalho foi realizado após a defesa da tese. O simulador da ferramenta TANGRAM-II permite agora que o projetista acesse as variávies de recompensa acumulada podendo tirar proveito desta informação.

Outro trabalho importante estaria relacionado com o simulador de eventos raros. Atualmente, o usuário precisa informar os pontos de quebra e o número de réplicas em cada ponto. Esta especificação deve ser o mais próximo da ótima possível, pois assim, o simulador apresenta sua maior eficiência. Existe na literatura trabalhos relacionados com a definição automática desses parâmetros feitas pelo próprio simulador. Isto facilitaria o uso do simulador tornando-o ainda mais robusto, uma vez que os parâmetros são adaptados em tempo de execução tendendo aos seus valores ótimos. Além disso, um trabalho relacionado com a definição dos conjuntos que definem o caminho para a ocorrência dos eventos raros poderia ser feito. Atualmente, os pontos de quebra são feitos sobre uma das variáveis de estado do sistema. É abordado na literatura, que pontos de quebra possam ser definidos sobre uma medida de interesse, ao invés de estados do sistema. Isto facilidade irá permitir aplicar esta técnica de eventos raros a uma gama ainda maior de modelos.

Para utilizar o simulador de eventos raros com eficiência é necessário conhecer em parte o processo estatístico que rege o modelo. Com isto, o usuário pode especificar os parâmetros pontos de quebra e número de réplicas próximos ao ótimo. Porém, nem sempre é fácil avaliar as características de um modelo a fim de definir esses parâmetros. Um interessante trabalho futuro seria especificar um modelo que represente o simulador de eventos raros implementado no TANGRAM-II. Este modelo levaria em consideração as principais características da implementação do simulador e seria utilizado para saber como o simulador de eventos raros se comportaria. Antes de iniciar uma simulação de eventos raros de um modelo qualquer, o usuário utilizaria este modelo do simulador para ter uma idéia de como definir os parâmetros de sua simulação. O modelo do simulador pode ser resolvido através de métodos analíticos ou através de simulação convencional. Além das características de implementação, o modelo do simulador deve levar em consideração caracteríticas do modelo do usuário que irá utilizar o simulador. Este trabalho facilitaria a especificação dos parâmetros do simulador de eventos raros, permitindo avaliar a sensibilidade dos mesmos antes de iniciar a simulação propriamente dita.

Referências Bibliográficas

[1] BAYES, A. J. Statistical techniques for simulation models. *The Australian Computer Journal 2(4)* (1970), 180–184.

[2] BERSON, S., DE SOUZA E SILVA, E., E MUNTZ, R. R. *Numerical Solution of Markov Chains*. Marcel Dekker, Inc., 1991, ch. An Object Oriented Methodology for the Specification of Markov Models, pp. 11–36.

[3] BERTSEKAS, D., E GALLAGER, R. *Data Networks*, 2 ed. Prentice Hall, 1992.

[4] BOLOT, J.-C., E VEGA-GARCÍA, A. Control mechanisms for packet audio in the Internet. In *Proceedings of the IEEE Infocom* (março de 1996).

[5] CARMO, R. M. L. R., DE CARVALHO, L. R., DE SOUZA E SILVA, E., DINIZ, M. C., E MUNTZ, R. R. Performance/Availability Modeling with the TANGRAM-II Modeling Environment. *Performance Evaluation 33* (1998), 45–65.

[6] CHENG, W. C. Tangram graphic interface facility. URL http://bourbon.cs.umd.edu:8001/tgif/.

[7] COMER, D. E., E STEVENS, D. L. *Internetworking with TCP/IP*, 2 ed., vol. 3 - Client-Server Programming and Applications. Prentice Hall, 1996.

[8] DE BRITO, C. E. F. Medidas transientes para recompensa acumulada limitada: Novas técnicas de solução e aplicações para redes de alta velocidade, junho de 1999. Tese de Mestrado - UFRJ - COPPE/Sistemas.

[9] de Souza e Silva, E., e Gail, H. R. Performability analysis of computer systems: From model specification to solution. *Performance Evaluation 14* (1992), 157–196.

[10] de Souza e Silva, E., Gail, H. R., e Muntz, R. R. Efficient solutions for a class of non-Markovian models. In *Computations with Markov Chains*, W. J. Stewart, Ed. Kluwer Academic Publishers, 1995, pp. 483–506.

[11] de Souza e Silva, E., e Muntz, R. R. *Métodos Computacionais de Solução de Cadeias de Markov: Aplicações a Sistemas de Computaçãoo e Comunicação.* VIII Escola de Computação, 1992.

[12] Diniz, M. C., e de Souza e Silva, E. Especificacao e Geracao de Modelos Markovianos para Analise de Desempenho e Confiabilidade de Sistemas. *Revista Brasileira de Computacao 6*, 3 (1991), 23–42.

[13] Diniz, M. C., e de Souza e Silva, E. Análise de mecanismos de controle de jitter em redes ATM. In *XXIII Seminário Integrado de Software e Hardware* (agosto de 1996), pp. 303–314.

[14] Figueiredo, D. R., Brandi, B. R., e de Souza e Silva, E. Mecanismos para recuperação de perdas de pacotes de voz na internet. In 16º *Simpósio Brasileiro de Redes de Computadores* (Rio de Janeiro, Brasil, maio de 1998), pp. 52–67.

[15] Figueiredo, D. R., Brandi, B. R., e de Souza e Silva, E. Efficient mechanisms for recovering voice packets in the internet. In *To appear in the proceeding of IEEE Globecom'99* (Rio de Janeiro, Brasil, dezembro de 1999).

[16] Figueiredo, D. R., Duarte, F. P., e de Souza e Silva, E. Implementação de um aplicativo de voz com análise e caracterização de seu tráfego. In *XXIV Seminário Integrado de Software e Hardware* (Brasília, Brasil, agosto de 1997), pp. 531–540.

[17] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., e Sunderam, V. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[18] Glasserman, P., Heidelberger, P., Shahabuddin, P., e Zajic, T. Splitting for rare event simulation: Analysis of simple cases. In *Proceedings of the 1996 Winter Simulation Conference* (Coronado, California, USA, dezembro de 1996), pp. 302–308.

[19] Goyal, A., Carter, W. C., de Souza e Silva, E., Lavenberg, S. S., e Trivedi, K. H. The system availability estimator. In *Proceedings of the* 6^{th} *Annual International Symposium on Fault-Tolerant Computing Systems (FTCS-16)* (Viena, Austria, julho de 1986), pp. 84–89.

[20] Heegaard, P. E. Comparison of speed-up techniques for simulation. In *The 12th Nordic Teletraffic Seminar (NTS-12)* (Trondheim, Norway, agosto de 1995), pp. 407–420.

[21] Heegaard, P. E. Speed-up techniques for simulation. *Telektronikk ISSN 0085-7130 91(2-3)* (1995), 195–207.

[22] Heidelberger, P. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation 5*, 1 (janeiro de 1995), 43–85.

[23] Jacobson, V., e McCanne, S. URL http://www-nrg.ee.lbl.gov/vat. Lawrence Berkeley National Laboratory - University of California, Berkeley.

[24] Kalos, M. H., e Whitlock, P. A. *Monte Carlo Methods*, vol. I. John Wiley & Sons, 1986.

[25] Kelling, C. A framework for rare event simulation of stochastic petri nets using RESTART. In *Proceedings of the 1996 Winter Simulation Conference* (Coronado, California, USA, dezembro de 1996), pp. 317 – 324.

[26] Kleinrock, L. *Queueing Systems*, vol. 1 - Theory. John Wiley & Sons, 1975.

[27] Meyer, J. F. On evaluating the performability of degradable computing systems. *IEEE Journal on Selected Areas in Communications 29*, 8 (1980), 720–731.

[28] Muntz, R. R., de Souza e Silva, E., e Goyal, A. Bounding availability of repairable computer systems. *IEEE Trans. on Computers 38*, 12 (1989), 1714–1723.

[29] Nicola, V., Heidelberger, P., e Shahabuddin, P. Techniques for fast simulation of highly dependable systems. In *Proceedings of the Second International Workshop on Performability Modelling of Computer and Communication Systems* (França, junho de 1993), p. paper 5.

[30] Ross, S. M. *A Course in Simulation.* Prentice Hall, 1990.

[31] Sanders, W. H., e Meyer, J. F. Reduced base model construction methods for stochastic activity networks. *IEEE JSAC - special issue on Computer-Aided Modeling, Analysis and Design of Communication Networks 9* (1991), 25–36.

[32] Schwartz, M. *Broadband Integrated Networks.* Prentice Hall, 1996.

[33] Villén-Altamirano, M., Martínez-Marrón, A., Gamo, J., e Fernández-Cuesta, F. Enhancement of the accelerated simulation method RESTART by considering multiple thresholds. In *Proceedings 14th International Teletraffic Congress* (1994), North-Holland, pp. 797–810.

[34] Villén-Altamirano, M., e Villén-Altamirano, J. RESTART: a method for accelerating rare event simulations. In *13th International Teletraffic Congress - Queueing, Performance and Control in ATM* (Copenhagen, 1991), P. C. Cohen J. W., Ed., North-Holland, pp. 71–76.

[35] Villén-Altamirano, M., e Villén-Altamirano, J. RESTART: a straightforward method for fast simulation of rare events. In *Proceedings of the 1994 Winter Simulation Conference* (Lake Buena Vista, Florida, USA, dezembro de 1994), pp. 282 – 289.

Apêndice A

TANGRAM-II Tutorial

NESTE apêndice apresenta-se um tutorial em inglês da ferramenta TANGRAM-II. Este tutorial contém todas as fases necessárias durante a especificação de um modelo. Todas as funcionalidades de especificação são reportadas ao longo do tutorial através de exemplos.

A.1 Introduction

TANGRAM-II is a modelling environment developed for research and educational purpose that provides a flexible user interface to describe computer and communication system models. It is based on an object oriented description language and has a sophisticated graphical interface built from the public domain software package TGIF (TANGRAM Graphic Interface Facility). The current version is implemented in C/C++ and has several solvers for transient and steady state analysis of performance and availability metrics. A robust simulator is also part of the solution methods and supports rare event simulation. Several examples that we present highlight the modelling capabilities of the environment.

A.2 Modelling with TANGRAM-II

The system being modelled is represented by a collection of objects which interact by sending and receiving messages. We represent the internal state of each object by a set of integer valued variables. The behavior of an object is defined by events and messages and their associated conditions and actions.

Events are generated spontaneously by an object, provided that the conditions associated with these events are satisfied. These conditions are boolean expressions evaluated using the current state of the object.

Messages are just an abstraction used to represent the interaction among the objects, and are delivered (and reacted to) in zero time.

When either an event occurs or a message is received, the set of actions specified by the user is executed with a given probability distribution. As a result of an action execution, the object state may change and messages may be sent to other objects in the model.

After the model is built, a Markov Chain can be generated and an appropriate solution technique can be used to obtain steady state and/or transient measures of interest. If the model is not Markovian or too large, then the simulator can be used to obtain different measures of interest.

A.3 Constructing a Model

The definition of an object consists, basically, of six distinct sections:

- The Declaration section
- The Initialization section
- The State variables section
- The Events section

- The Messages section
- The Rewards section

A.3.1 Declaration Section

In this section, we declare variables, constants and specify any literal parameters used in the object. The Declaration Section is divided into three distinct subsections where these three types of declaration (variables, constant and parameters) are specified. All variables must be declared having the State type; constants can be of Integer, Float, Object or Message_port types; literal parameters can be of Integer or Float types.

A brief explanation of the pre-defined types follows:

- State: used to declare the variables that represent the internal state of the object.
- Integer, Float: used to specify numerical constants or literal parameters.
- Object: used to reference other objects of the model.
- Message_port: to define ports used in the communication between objects.

The variables and constants can be used in the conditions and actions of the Events, Messages and Reward sections. Parameters can be used only when specifying the rate of the events or the probability of an action. Only the State type can have its value modified during the execution of an action. All constants and literal parameters have a fixed value inside an object. Here is presented an example of a declaration section:

```
Declaration=
    Var
      State  : queue;
    Const
      Integer: max_queue_size;
      Float  : service_rate;
```

```
    Object : packet_src;
    Message_port: port_in;
  Param
    Float  : lambda;
```

A.3.2 Initialization Section

In this section, constants are defined, and state variables receive their initial values. All state variables and constants must appear in this section and have a value attibuted to it. The value of literal parameters are given afterwards, when solving the model. This would be the initialization section of all variables and constants of the above declaration. The constant type Object must be initialized with the name of an existing object in the model.

```
Initialization=
  queue          = 0
  max_queue_size = 100
  service_rate   = 10.0
  packet_src     = Poisson_Source
  port_in        = packet
```

A.3.3 State variables Section

This section is used by the interactive simulator. The state variables of the model must be placed in this section if interactive simulation is to be used. When simulating, the state variables of the object are updated in this box. This section is much similar to the initialization section, however, only state variables appear in this box. Following our previous example, an illustration of this section is:

```
State_vars=
  queue=0
```

A.3.4 Events Section

In this section, all the events are declared. An event is defined by a name, a distribution type and its parameters. An event must also has a condition and at least one action associated to it. The state of the object can be modified by the execution of the associated actions. At the end of an action the state variables of the object can be modified with the *set_st(<var_name>, <value>)* function, which sets the value of the variable *<var_name>* to *<value>*. An event can only trigger if its condition is satisfied.

```
Events=
  event = service (EXP, service_rate)
    condition= (queue > 0)
    action= {
              int q;

              q = queue - 1;

              set_st("queue", q);
            };
```

A.3.5 Messages Section

In this section, all messages that can be received by the object are specified. When a specific message is received an action associated with that message is taken to process it. The reception of a message is defined by a Message_port constant and an Object constant. The Object constant is used to identify the object that is allowed to send messages to this port. The Message_port constant indicates the port used to deliver the message.

```
Messages=
  msg_rec = port_in, packet_src
    action= {
              int q;

              if (queue < max_queue_size)
                 q = queue + 1;
              else
                 q = queue;

              set_st("queue", q);
            };
```

A.3.6 Rewards Section

In this section all rate or impulse rewards are specified. Rewards can be used to obtain interest measures together with specific solution methods or simulation. Below we have an example of a rate and impulse reward definition.

```
Rewards=
   rate_reward= utilization
     condition= (queue > 0)
     value= 1.0;

   impulse_reward= arrived_departured
     Message_port= port_in, 1
     value =  1.0;
     event = service, 1
     value = -1.0;
```

A.3.7 Conditions and Actions

Conditions are boolean expressions used to indicate wether an event is enabled or not. Conditions are also used when specifying rate rewards indicating which states will receive the given reward value. When a condition evaluates to true the event can trigger. A condition expression can have variables and constants declared in the object, numerical values, and logical/arithmetic operators. The condition must be enclosed in parenthesis.

```
condition= ( (queue > (max_queue_size / 2)) && (server == 1) )
```

Actions are used to modify the state of the object, and to send messages to other objects. An action is executed when an event triggers or a message is received. Actions are described in C like code and can use the variables and constants declared in the object. An action can also contain local variables declared within its scope. The state variables cannot have its value changed during the action. To change the value of the state variables, the funtion *set_st()* must be used at the end of the action section. The function *set_st(<state_var_name>, <st_var_value>)* sets the state variable *<state_var_name>* to the value *<st_var_value>*. The *<st_var_value>*

can be an integer value or a local integer variable. Actions must be enclosed in curly braces followed by a semi-colon. If an action section contains no action statements then a semi-colon should be used to characterize this. Several actions may be associated with an event or message, each having a different probability of occurrence (in this case, the probabilities must add to one). The probabilities are specified using the reserved word prob followed by an arithmetic expression.

- Simple action:

```
action= {
          int temp;

          if (queue > (max_queue_size / 2) )
             temp = queue - 2;
          else
             temp = queue - 1;

          set_st("queue", temp);
        };
```

- Multiple actions:

```
action= {
          /* a correct packet has arrived */
          int q;
          q = queue + 1;
          set_st("queue", q);
        } : prob = p;
        {
          /* an incorrect packet has arrived */
          /* do nothing */
          ;
        } : prob = 1 - p;
```

A.3.8 Exchanging Messages

Objects can communicate with each other exchanging messages. To send a message, an object uses the msg(...) command inside the action code. The parameters of this command are the Object receiveing the message, the Message_port to be used for delivery and the payload value. The Object type constant identifies the destination object of a message.

A message is always sent through an output port of the source object and received in an input port of the destination object. The ports are declared as constant having Message_port type and must have the same value in order to exchange messages.

To receive a message, an object must have declared this message in the Message section of its description. A message that is received by an object is identified by the input port and source object.

It is necessary to match the output port of the sending object with the input port of the receiving object. This is done by giving the same value to the port constants in both objects. If an object sends a message to another one through a port that doesn't exist in the receiving object an error is generated and the process is aborted.

Every message has an integer valued payload. The payload is set by the sender in the msg(...) command, and can be referenced in the action of the receiving object using the reserved word msg_data.

When executiong a msg(...) command inside an action, TANGRAM-II stores this message for later delivering. If other msg(...) commands are issued than a list of pending messages is created. After finishing the execution of the action, the messages stored in the pending message list starts to be delivered. The first message to be delivered is the first message sent, so the order of delivery remains the same. If while delivering a message (execution of an action), more messages are sent, then these messages are appended at the end of the pending message list.

Sending a message to the object associated with the Object constant packet_dest, and Message_port port_out, with payload value of three:

```
action= {
          ...
          msg (port_out, packet_dest, 3);
          ...
        };
```

Receiving a message from the object associated with the Object constant packet_src, and Message_port port_in, and checking the payload value:

```
msg_rec= port_in, packet_src
  action= {
             ...
             if (msg_data == 0)
                xyz = xyz - 1;
             ...
          };
```

A.4 First Model

In this section we illustrate the specification of a new model through a simple example. In this example we have two classes of objects: source of packets, and a server with a queue. Several objects of the same class can be instantiated to create the complete model.

The source object generates packets with a Poisson distribution, sending them to the server object. The server object has a limited queue and serves a single type of packet in FCFS order with exponentialy distributed time. If a packet arrives to the server when the queue is full the packet is dropped.

Once defined the objects that will compose the model, we have to create the description of each class.

An object template can be used to aid the user in creating a new class. This template has all the sections necessary to describe an object: Declaration section, Initialization section, Events section, Messages section and Rewards section.

We will now create the packet_source class. The behavior of an object is represented by events and messages and their associated actions. The single event of the packet_source class is Packet_Generation. Each time this event triggers the action specified is executed, in this case, a message is sent. The distribution of the event is defined by the reserved word EXP (which indicates an exponential distribution). The rate at which this event occurs is defined in the second parameter with the constant pkt_rate.

There are no conditions to this event to occur, this is stated by the boolean expression “TRUE”. No messages are received by a packet_source class object, so the Messages section is empty. All variables and constants used in the object specification have to be declared in the Declaration section. Since each object must have at least one State type variable, the packet_source class has a dummy variable. After the class has been defined, the user can instantiate an object of this class, give initial values to the variables and define the constants. Figure A.1 shows the complete packet_source class specification.

Figura A.1: The packet source class.

Having specified the packet_source class, we will now describe the server class. Only one event is associated with the server class, Packet_Service. The rate at which this event occurs is defined by the constant service_rate. The Packet_Service event can only occur if the queue is not empty, so a condition is needed. When the event Service_Packet fires a packet is removed from the queue.

A server class object receives messages from other objects. The message represents the arrival of a packet to the server. In this model the message is received from an instance of the packet_source class. If the queue is not full the packet is stored in it to be served later. Otherwise, the packet is dropped. Figure A.2 shows the server class specification.

With the two classes defined above we will create a model for the well known

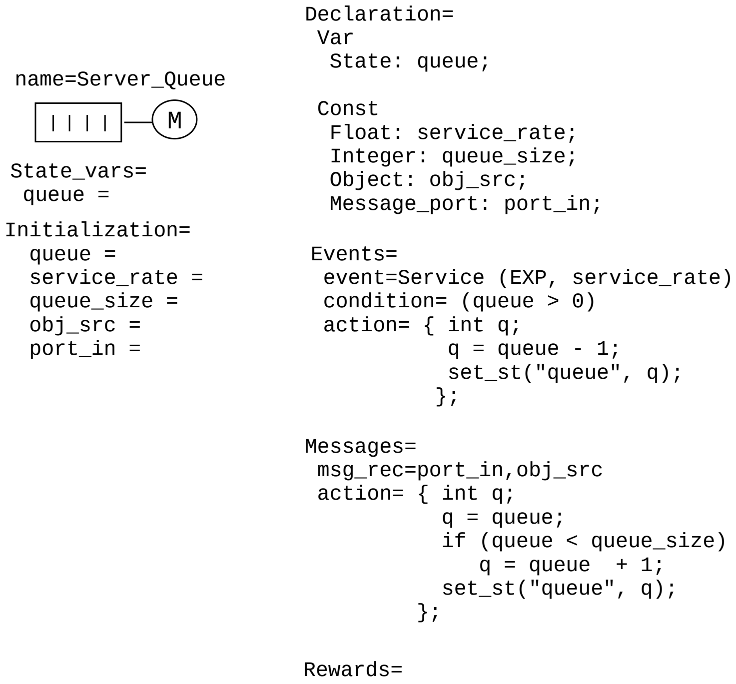


Figura A.2: The server with queue class.

M/M/1/k queue. There is one instance of each class in this model. When a new object is instantiated a unique name must be given together with inital values for its constant and state variables. In this example, the objects were named Poisson_Source and Exp_Server. Figure A.3 shows the final model with the variables and constants initialized.

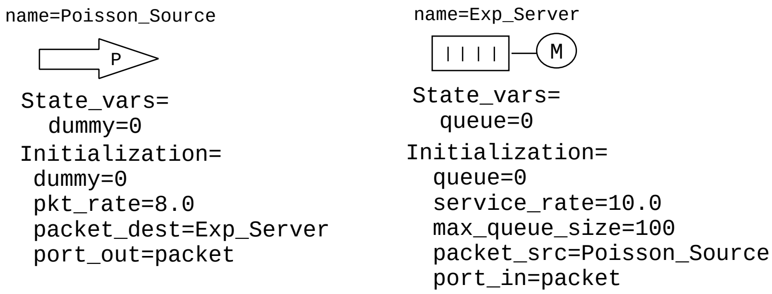


Figura A.3: The M/M/1/k model.

After the model is built the user has to generate the mathematical model using the option Model Generation from the Tangram-II graphical interface. Once the mathematical model is generated the user has to solve the model using an appropriatte solution technique. This can be done using the option Analytical Model Solution from the Tangram-II menu. After solving the model, the user can specify different interest measures and plot their results, which can be done using the option Interest Measures in the graphical interface.

A mathematical model can only be generated if the system described is Markovian, and in this case, all events must have an exponential distribution, generating a continuous time Markov chain. A semi-Markovian model can also be generated if only exponential and deterministic events are used. In this case special solution methods must be used in order to solve the model.

If a mathematical model cannot be generated for the specified system, then simulation must be used to solve the model. To simulate the model the option Simulation in the graphical interface should be used. A rare event simulator is also section of the tools that are available for a user when solving a model. This option can also be set through the graphic interface.

A.4.1 Reward Variables

After describing a model a user can specify different interest measures through the use of reward variables. Rewards variables are values that can be associates with system states (rate rewards) or system transitions (impulse rewards). These variables have are generic enough to permit the definition of a wide range of measures of interest. Its specific meaning is given only after its definition.

The rate reward variable has a simple interpretation and it is just the association of system states to values. When the model enters a state, say i the reward variable starts to accumulate at the rate R_i. For as long as the system stays in state i the reward accumulates. When the system transits to another system state, j, another rate value R_j start to be accumulated. After a period of time t, the variable $CR(t)$ expresses the cumulated reward in $[0, t]$.

Impulse rewards are associated with system transitions. When the model transits from state i to state j a value can be assigned to this transition. Transitions are made in zero time, so the impulse rewards are not related to time, and serve as an counting mechanism. An impulse reward variable can be associated with the triggering of an event or with the reception of a message. This are the two situation were a transition can be characterized.

We will now illustrate the use of reward variables in the model specified in the previous section. Suppose we are interested in calculating the utilization of the queue in the server object. We can define a reward variable "utilization" that can express this measure of interest. In the Rewards section, the user should specify the name of the reward, used as an identifier. The conditions for the reward to be satisfied and its value. If the condition is not satisfied a zero value reward is given in those states.

```
Rewards=
   rate_reward= utilization
     condition= (queue > 0)
     value= 1.0;

   impulse_reward= served
     event = Service_Packet
     value = 1;
```

Suppose now we are interested in counting, or marking, all transitions related to the act of serving. An impulse reward can be defined and associated with the event Service_Packet to have a value of 1 when this event triggers. This is done in the previous example, for the reward served.

A.4.2 Distribution Types

When specifying events the user must indicate the distribution probability function

A.5 Interactive Simulation

After its being specified, any model can be simulated. One of the features of the simulator module is interactive simulation. This is accomplished with together with the TGIF tool, which is responsible for updating the State variables section during the simulation. When starting the interactive simulation, the simulator connects to TGIF through socket API and establishes a communication channel. Every time the user steps in the simulation, a message is sent from TGIF to the simulator. The simulator received this message, processes it, and sents back the final system state for that transition. TGIF receives this message and updates the State variable section in all objects. The user can then see the evolution of the sample path being taken by the simulator.

A.5.1 Animation

Besides being able to watch the evolution of the system state, the user can define animations that are related to the simulation. Here, the object can have pre-defined animations that are executed according to specific states that the object might reach.

The animation has to be specified by the user, introducing a new section in the object, the Animation section. In this section, the user should specify the animation of the object and related objects. The TGIF object that being used for model specification, should not be used for simulation. A separate TGIF graphical object should carry the animation. This new object is simply an animated object that is called from the Animation section of its related model object. These procedires, of calling an animated object and of specifing an animation, has to be done using TGIFs internal commands, that are similar to a programming language. Together with the internal commands, TGIF supports a few animation primitives that can be used to do the animations. A good reference of all supported commands can be found in TGIF manual. After a simulation step, the animation starts and a pre-defined number of animation steps are executed before resuming to simulation. This allows "simultaneous" animation of different objects, permitting each animation to do something only at the specific animation steps.

We will now show an example of an animation done over the M/M/1/k model. This animation consists of two parts. The packet being sent from the source to the queue and the increase/decrease of the queue size. The animation of the queue size is done by an rectangle object named _BAR_. This object streches its size to the appropriate value of its variable q. The object Exp_Server has the following Animation section:

```
Animation=
  get_line_in_attr(result1, State_vars, 1);
  tokenize(result2, $(result1), "=");
  get_line_in_attr(q, result2, 2);
  assign(_BAR_.n, $(q));
  if("$(__SIMULATION_STEP__.Ani_T_Steps)"==
     "$(__SIMULATION_STEP__.Ani_C_Step)",
       _BAR_.animate, NULL);
```

This code gets the current value of the queue state variable and sets the variable n in object _BAR_ to this value. If it is the last step in the animation loop then it calls the funcion _BAR_.animate which animates the object _BAR_. This object is illustrated in Figures A.4. It has an animation function "animate" that is called by the object Exp_Server, that checks the size of the variable n and executes the corresponding function that resizes the BAR. This is done only at the last animation step.

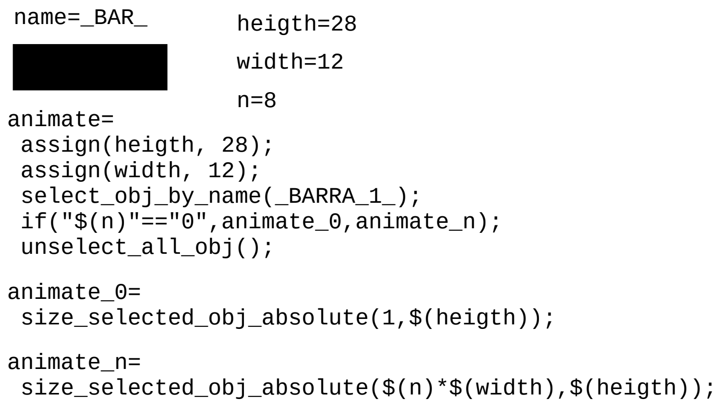


Figura A.4: The animated object representing queue size

To animate the movement of the packet from the source to the queue it is necessary to modify the behavior of the object Packet_Source. The animation can depend only on the state variables of the object. In order for the animation to occur only when a packet is transmitted, this must be reflected on its state variables. We modify the dummy state varible S to which between states 0 and 1. Every time a packet is transmitted the state changes. By doing this, the Animation section can decide when to play the animation of the packet movement.

```
Animation=
  get_line_in_attr(result1, State_vars, 1);
  tokenize(result2, $(result1), "=");
  get_line_in_attr(s, result2, 2);
  if("$(s)"!="$(s_ant)", _PAC_.animate, NULL);
  if("$(__SIMULATION_STEP__.Ani_T_Steps)"==
     "$(__SIMULATION_STEP__.Ani_C_Step)",
       Reset_S, NULL);

Reset_S=
  assign(s_ant, $(s));
```

This code checks to see if the state variable S has changed its value. If it has, then it calls the animation function in object _PAC_ for every animation step. At the last animation step, it stores the new value for the state varible S. The object _PAC_, ilustrated in Figure A.5, has the animate function which moves the object horizontally to the left. At the first animation step, this object records its position. For every step it uses the number of the animation step to calculate the new x coordinate and shitf the object to the left. At the last step of the animation it restores its original position.

```
name=_PACOTE_

dx=-30  Ox=663  ltx=663  rbx=687
dy=0    Oy=328  lty=328  rby=336

animate=
 select_obj_by_name(THIS);
 if("$(__SIMULATION_STEP__.Ani_C_Step)"=="1",get_origin_position,NULL);
 if("$(__SIMULATION_STEP__.Ani_C_Step)"=="$(__SIMULATION_STEP__.Ani_T_Steps)",animate_n,animate_other);
 unselect_all_obj();

get_origin_position=
 get_selected_obj_bbox(ltx,lty,rbx,rby);
 assign(Ox,$(ltx));
 assign(Oy,$(lty));

animate_other=
 move_selected_obj_relative($(dx),$(dy));

animate_n=
 move_selected_obj_absolute($(Ox),$(Oy));

exec=
 select_obj_by_name(THIS);
 move_selected_obj_absolute($(Ox),$(Oy));
 unselect_all_obj();
```

Figura A.5: The animated object representing packet movement

Many other complex animations can be specified. The user can basically define any possible graphic command supported by TGIF. Sometimes the user will need to modify the behavior of the object in order to get the desired animation. However, since animation is mainly for educational purposes, there is no drawback in increasing the complexity of the model.

A.6 Literal Models

As mentioned in section 3, the tool supports the use of literal parameters. These parameters can be used only in expressions that specify parameters of a distribution, in expressions that specify the probabilities of an action or in expressions that define the value of a reward. For example, the exponential distribution (EXP) needs the parameter rate, which can be specified using an expression that contains a literal parameter. In the same way, in a multiple action example a probability can have an expression with literal parameters. Also when specifing rate or impulse rewards, the value field of the reward can have an expression with a literal parameter. Literal parameters cannot be used in conditions or inside actions.

Literal parameters are usefull for doing sensibility analysis. In this case, the Markov chain associated with the model is generated only once, with literal expressions in its transitions. Afterwards, the value of these literal parameters are specified and a numeric chain can be easily calculated. Many differente numeric chains can be calculated for different values of the literal parameters. This can be done in the option Analytical Model Solution of the graphic interface, without the need of generating the whole model again. In this option of the interface, the user can define the numeric values for its literal parameters and calculate the numeric chain. The numerical rewards are also calculated after the specification of the parameters. The solution module deals only with numerical chains and numerical rewards, so it is necessary to specify all parameteres before solving the model.

If simulation is to be used, then no literal parameters can be specified. When simulating a model, the state space is not generated and the simulator calculates the samples for the events and the probabilities of the action based on model description.

A.7 Other examples

A.7.1 MMPP packet source, leaky bucket traffic shaper and server queue

In this model we have three different classes of objects. A three state MMPP packet source which transmits packets at different rates in each state, a leaky bucket traffic shaper that has a bucket of credits and a queue of packets, and a server with a finite size queue of packets. Figure A.6 shows the description of this model.

A.7.2 Packet Recovery Mechanism

In this example we present the model of a novell algorithm for packet recovery that uses redundancy. This scheme was proposed in [14] and recently a model for evaluating its performance has been developed. The model consists of four objects, a packet source, a packet dropper, a packet recover and a counter. The packet source simply transmits packets at a constant rate emulating a voice source. These packets are delivered to the object packet dropper that discards a packet with probability p and transmits a packet with probability 1-p. The discard is done by setting the payload of the packet in 0 while the delivery is done by setting the payload in 1. The object packet recover receives the stream of packets marked and stores them in a "window". After receiving all packets in the window, the recovery mechanism is executed possibly recovering some lost packets. Every time a lost packet is recovered a message is sent to the packet counter. This counter simply counts the number of packets recovered.

Many possible interest measures can be defined over this model. For example, impulse rewards can be used to count (mark) the packets transmitted, the packets lost, packets delivered and packets recovered. All these impulse rewards are specified in this example.

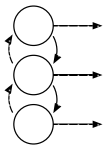
Figure A.7 ilustrates the whole model and its four objects. Now, we show the specification of the packet recover object.

```
State_vars=

Initialization=
  P0 = 1  P1 = 1
  P2 = 0  P3 = 0
  P4 = 0  P5 = 0
  P6 = 0  P7 = 0
  counter = 2
  N = 6
  K = 2
  obj_src = Looser
  port_in = pac2
  obj_dest = Counter
  port_out = pac3

Events=

Messages=
  msg_rec= port_in, obj_src
  action={
            int lost, recovered, redun_ok, i, j;
            int P0_, P1_, P2_, P3_, P4_, P5_, P6_, P7_, c_;
            P0_=P0; P1_=P1; P2_=P2; P3_=P3;
            P4_=P4; P5_=P5; P6_=P6; P7_=P7;

            if (counter == 2)
               P2_ = msg_data;
            if (counter == 3)
               P3_ = msg_data;
            if (counter == 4)
               P4_ = msg_data;
            if (counter == 5)
               P5_ = msg_data;
            if (counter == 6)
               P6_ = msg_data;

            c_ = counter + 1;

            /* analyse the window received */
            if (counter == 7) {
               P7_ = msg_data;

               recovered = 0; i = 0;
               while (i < K) {
                 lost = 0; j = 0;
                 while (j < (N/K)) {
                   if ( (i + (j * K)) == 0)
                      if (P0 == 0) lost = lost + 1;
                   if ( (i + (j * K)) == 1)
                      if (P1 == 0) lost = lost + 1;
```

```
            if ( (i + (j * K)) == 2)
               if (P2 == 0) lost = lost + 1;
            if ( (i + (j * K)) == 3)
               if (P3 == 0) lost = lost + 1;
            if ( (i + (j * K)) == 4)
               if (P4 == 0) lost = lost + 1;
            if ( (i + (j * K)) == 5)
               if (P5 == 0) lost = lost + 1;
            if ( (i + (j * K)) == 6)
               if (P6 == 0) lost = lost + 1;
            if ( (i + (j * K)) == 7)
               if (P7_ == 0) lost = lost + 1;

            j = j + 1;
          }
          if ( (i + N) == 6)
             if (P6 == 1) redun_ok = 1;
             else redun_ok = 0;
          if ( (i + N) == 7)
             if (P7_ == 1) redun_ok = 1;
             else redun_ok = 0;

          if ((lost == 1) && (redun_ok ==1)) {
             recovered = recovered + 1;
             msg(port_out, obj_dest, 0);
          }

          i = i + 1;
        }

        /* update the values for the next window */
        P0_ = P6;  P1_ = P7_;
        P2_ = 0;   P3_ = 0;
        P4_ = 0;   P5_ = 0;
        P6_ = 0;   P7_ = 0;
        c_ = 2;
     }

     set_st("P0", P0_); set_st("P1", P1_);
     set_st("P2", P2_); set_st("P3", P3_);
     set_st("P4", P4_); set_st("P5", P5_);
     set_st("P6", P6_); set_st("P7", P7_);
     set_st("counter", c_);
  } ;

Rewards=
```

```
Generate Chain
setup  debug_level=0
       max_states=0
       output=/dev/stderr
       file_name=MMPP
       max_values=
        MMPP_Source.S          2
        Leaky_Bucket.buffer   50
        Leaky_Bucket.credits  50
        Server_Queue.queue   200

Start Simulator
End Simulator
Simulation Step

Transitions=0
Time=0.0
Mode=I
Step=1
Delay=0
Animation=OFF
```

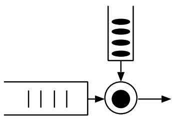
name=MMPP_Source



```
State_vars=
 S=0
Initialization=
 S=0
 tx0=10.4
 tx1=31.2
 tx2=132
 tr0_1=1.14
 tr1_0=9.78
 tr1_2=0.636
 tr2_1=1.52
 dest=Leaky_Bucket
 port_out=packet_1

Messages=

Rewards=

 rate_reward=utilization
  condition= (S == 0)
  value = 1;
```

```
Declaration=
  Var
   State: S;
  Const
   Float : tx0, tx1, tx2,
           tr0_1, tr1_0,
           tr1_2, tr2_1;
   Object : dest;
   Message_port : port_out;

Events=
 event= Transicao0_1 (EXP, tr0_1)
 condition= (S == 0)
 action= { ; set_st("S", 1); };

 event= Transicao1_0 (EXP, tr1_0)
 condition= (S == 1)
 action= { ; set_st("S", 0); };

 event= Transicao1_2 (EXP, tr1_2)
 condition= (S == 1)
 action= { ; set_st("S", 2); };

 event= Transicao2_1 (EXP, tr2_1)
 condition= (S == 2)
 action= { ; set_st("S", 1); };

 event= Tx_0 (EXP, tx0)
 condition= (S == 0)
 action= { msg(port_out, dest, 0); };

 event= Tx_1 (EXP, tx1)
 condition= (S == 1)
 action= { msg(port_out, dest, 0); };

 event= Tx_2 (EXP, tx2)
 condition= (S == 2)
 action= { msg(port_out, dest, 0); };
```

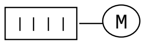
name=Leaky_Bucket



```
State_vars=
  buffer=0
  credits=0
Initialization=
 buffer=0
 credits=0
 credit_rate=17.5
 max_credits=50
 max_buffer=50
 src=MMPP_Source
 dest=Server_Queue
 port_in=packet_1
 port_out=packet_2
```

```
Declaration=
 Var
   State : buffer, credits;
 Const
   Float  : credit_rate;
   Integer: max_credits, max_buffer;
   Object : src, dest;
   Message_port : port_in, port_out;

Events=
 event=credit_arrival (EXP, credit_rate)
 condition= (credits < max_credits)
 action= {
           int b, c;
           b = buffer ; c = credits;

           if ( (buffer > 0) && (credits == 0)) {
              b = buffer - 1;
              msg(port_out, dest, 0);
           } else
              c = credits + 1;

           set_st("credits", c);
           set_st("buffer", b);
         };
Messages=
 msg_rec=port_in, src
 action= {
          int b, c;
          b = buffer ; c = credits;

          if (credits > 0) {
            c = credits - 1;
            msg(port_out, dest, 0);
          }
          else if (buffer < max_buffer)
            b = buffer + 1;

          set_st("buffer", b);
          set_st("credits", c);
        };

Rewards=
```

name=Server_Queue



```
State_vars=
  queue=0
Initialization=
  queue = 0
  service_rate = 10
  queue_size = 100
  obj_src = Leaky_Bucket
  port_in=packet_2
```

```
Declaration=
 Var
  State: queue;

 Const
  Float: service_rate;
  Integer: queue_size;
  Object: obj_src;
  Message_port: port_in;

Events=
 event=Service (EXP, service_rate)
 condition= (queue > 0)
 action= { int q;
           q = queue - 1;
           set_st("queue", q);
          };

Messages=
 msg_rec=port_in,obj_src
 action= { int q;
           q = queue;
           if (queue < queue_size)
              q = queue  + 1;
           set_st("queue", q);
         };

Rewards=
```

Figura A.6: The MMPP packet source model with a leaky bucket traffic shaper and server queue

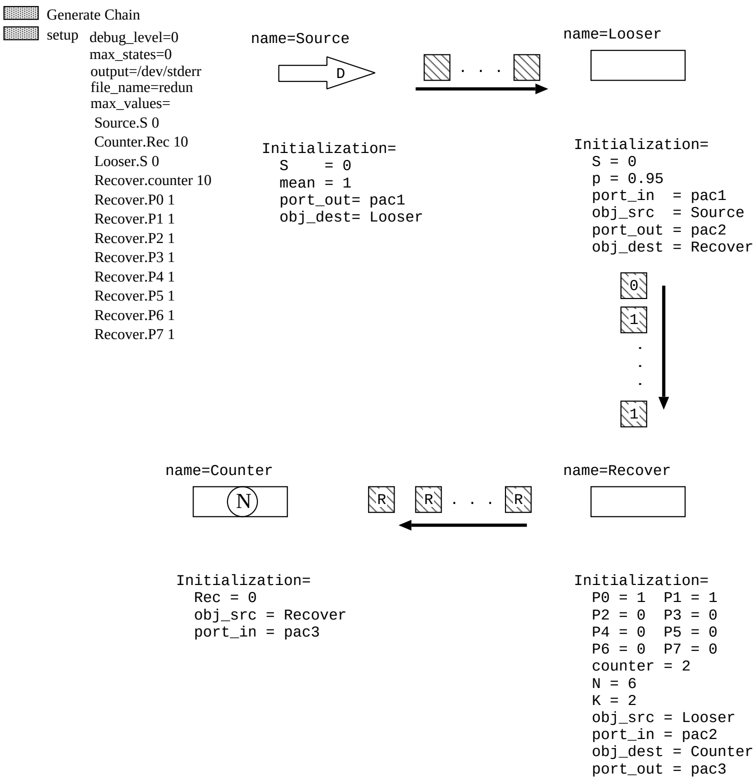


Figura A.7: The Packet Recovery Model