Pontifícia Universidade Católica
do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Técnica para Obtenção de Redes de Autômatos Estocásticos Baseada em Especificações de Software em UML

Felipe Barp Neuwald

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

Porto Alegre, 7 outubro de 2008.

Dados Internacionais de Catalogação na Publicação (CIP)

N498t Neuwald, Felipe Barp
Técnica para obtenção de redes de autômatos estocásticos baseada em especificações de software em UML /Felipe Barp Neuwald. – Porto Alegre, 2008.
90 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

1. Redes de Autômatos Estocásticos. 2. Engenharia de Software. 3. Software – Avaliação. 4. Orientação a Objetos. 5. UML (Informática). I. Fernandes, Paulo Henrique Lemelle. II. Título.

CDD 005.1

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Reconhecido pelo Parecer No. 930/98.C.N.E. Homologação Publicada no D.O.U. de 30/12/98.

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada **"Técnica para Obtenção de Redes de Autômatos Estocásticos Baseadas em Especificações de Software UML",** apresentada por **Felipe Barp Neuwald,** como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Área de Computação Científica, aprovada em 28/03/2005 pela Comissão Examinadora:

Prof. Dr. Paulo Henrique Lemelle Fernandes – Orientador — PPGCC/PUCRS

Prof. Dr. Duncan Dubugras Alcoba Ruiz – — PPGCC/PUCRS

Prof. Dr. Jorge Luiz Nicolas Audy – — PPGCC/PUCRS

Prof. Dr. Marcelo Blois Ribeiro – — PPGCC/PUCRS

Prof. Dr. Marcelo Hideki Yamaguti – — FACIN/PUCRS

Homologada em 15./10./2008, conforme Ata No. 21/08 pela Comissão Coordenadora.

Prof. Dr. ***Avelino Francisco Zorzo***
Coordenador.

AV. IPIRANGA, 6681 - PRÉDIO 16/SALA 106
90619-900 - PORTO ALEGRE -RS -BRASIL
TELEFONE: 55 51 33203611 FAX 55 51 3320.3621
http: //www.inf.pucrs.br e-mail: ppgcc@inf.pucrs.br

Dedico este trabalho a meus pais Ivete e Rovaldo e a meu grande amor Marla.

Agradecimentos

Agradeço a Rovaldo e Ivete pelo apoio,
a Marla pela dedicação e compreensão,
ao Prof. Dr. Paulo H. L. Fernandes pela orientação e
ao Prof. Dr. Marcelo Blois Ribeiro pelo auxílio

Resumo

A avaliação de desempenho de um *software* orientado a objetos, hoje, pode ser feita basicamente de três maneiras: com testes de desempenho, simulações ou utilizando métodos analíticos como Redes de Autômatos Estocásticos e Rede de Filas de Espera. Os teste de desempenho são eficientes, porém podem ser aplicados apenas nas fases finais do desenvolvimento de *software*, quando o produto está praticamente pronto. As simulações podem ser realizadas nas etapas iniciais do processo de desenvolvimento de *software* porém, geralmente, são custosas, pois dependem da criação de um simulador. Os métodos analíticos podem ser aplicados nas fases iniciais do desenvolvimento, mas exigem a criação de um modelo de avaliação de desempenho.

No desenvolvimento de *software* orientados a objetos já se constrói um modelo, o qual possui informações detalhadas sobre o sistema. Estas informações podem ser utilizadas para gerar um modelo de avaliação de desempenho que represente o *software* já que muitas das informações contidas no modelo do *software* são comuns ao modelo de avaliação de desempenho. Utilizando esta idéia foram criados alguns métodos de conversão de UML para formalismos de avaliação de desempenho como redes de filas de espera. Porém, ainda inexistem métodos ou técnicas que convertem UML para formalismos como SAN, que é mais abrangente que, por exemplo, redes de filas de espera. Este trabalho demonstra uma técnica de conversão de UML para SAN, onde explora-se a representação da arquitetura lógica de *softwares* orientados a objetos.

Abstract

Performance evaluation of an object oriented software nowadays can be done, basically, by three different kinds of methods: Stress Testing, Simulations or Analytic Methods, like Stochastic Automata Networks or Queuing Networks. Stress Testing is an efficient method, but could only be performed on final phases of a software development process when the final product is almost finished. Simulations can be performed in initial phases of a software development process, but usually at a high cost, because it depends on a creation of a simulator. The Analytic Methods could be performed at the earlier phases of the software development process too, but they require a creation of a performance evaluation model.

In the object oriented software development a model is created, which contains detailed information about the system. This information could be used to generate a performance evaluation model, since many of the information containing on each other are common. Following that idea researches were made generating methods for conversion of UML diagram into performance evaluation methods, for example, to Queuing Network. However, does not exist a method or technique that converts UML to formalisms like SAN, which has a higher representative power then Queuing Networks. This work will present a technique for converting UML diagrams into Stochastic Automata Networks focusing on the representation of the logical architecture of object oriented softwares.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

EG	Execution Graph - Grafos de Execução
PEPS	Performance Evaluation of Parallel Systems - Avaliação de Desempenho de Sistemas Paralelos
QN	Queue Networks - Redes de Filas de Espera
SAN	Stocastic Automata Networks - Redes de Autômatos Estocásticos
SPE	Software Performance Engineering - Engenharia de Desempenho de Software
UML	Unified Modeling Language - Linguagem de Modelagem Unificada

Sumário

Capítulo 1

Introdução

Com o aumento da complexidade dos *softwares* orientados a objeto e o crescimento do número de usuários dos mesmos, torna-se, cada vez mais, indispensável avaliar o desempenho destes e evitar surpresas em tempo de implantação, quando o software já foi concebido [4]. As avaliações de desempenho mais comuns, hoje, feitas são: testes de desempenho, o que ocorre após a codificação do *software* e simulações, que geralmente implicam em um grande custo. Uma alternativa não tão custosa, como simulações, e que pode ser feita nas fases iniciais do desenvolvimento de um *software*, e nem tão exploradas hoje em dia são os métodos analíticos. Os métodos mais conhecidos são redes de filas de espera [8], redes de autômatos Estocásticos, cadeias de Markov. Estes métodos utilizam-se de um modelo, que deve seguir as regras definidas em cada metodologia.

No desenvolvimento de um *software*, geralmente, é criado um modelo de especificação que será utilizado como base para a implementação e implantação do mesmo. No caso dos orientados a objetos, os modelos são criados em UML que é a linguagem padrão de modelagem de *softwares* orientados a objeto, criada por Ivar Jacobson, James Rumbaugh e Grandy Booch, e aceita pela OMG [9]. Os modelos em UML contêm detalhes sobre a estrutura lógica do *software* (com representação de classes, métodos e seus relacionamentos), estrutura comportamental do usuário (com a modelagem de atores e casos de uso) e de estrutura física de implantação do sistema (nós e suas relações).

As informações compreendidas no modelo UML descrevem minúcias que podem ser utilizadas para criar modelos de avaliação de desempenho, como o relacionamento entre as classes e métodos, evitando a criação de um segundo modelo matemático, a fim de avaliar o desempenho por um método analítico, para o mesmo *software*. Existem alguns métodos que utilizam esta ideologia e produzem modelos de avaliação de desempenho baseados nas informações contidas nos modelos em UML [4, 7, 5]. Entre estes métodos, o apresentado por

Cortellessa e Mirandola [4] se destaca por utilizar-se dos principais diagramas que abrangem o comportamento, estrutura lógica e física para gerar um modelo de avaliação de desempenho em redes de filas de espera. Neste trabalho, os autores propõem que sejam utilizados os diagramas de casos de uso e seqüência para gerar dados e o diagrama de implantação para gerar a estrutura do modelo de avaliação de desempenho. Sua principal contribuição é a avaliação do desempenho em diferentes arquiteturas físicas (máquinas e suas ligações), pois a estrutura da rede de filas de espera representa o hardware e a estrutura de rede que o liga.

O método de Cortellesa e Mirandola é bem eficiente para avaliar o desempenho do hardware e a necessidade de modificarmos sua estrutura [4]. Porém, a arquitetura de *software* [1], descrita pelas estruturas lógicas do UML, também pode influenciar o desempenho do *software*. Se os diagramas de seqüência, além de gerarem dados, forem utilizados para a criação da estruturas, há a possibilidade de se avaliar, também, o desempenho das estruturas lógicas. Com isso, poder-se-á obter dados para decidir por uma arquitetura de *software* ou outra, e mesmo verificar eventuais problemas de desempenho em arquiteturas de *software*. A avaliação desta arquitetura facilita a revisão de estruturas de componente, pois deixa mais claras, ao analista do sistema, as potênciais chamadas a possuírem piores desempenhos e corrobora as decisões acertadas na estruturação dos componentes.

No método de Cortellessa e Mirandola, não há a preocupação em avaliar a estrutura lógica, a arquitetura de *software* é, somente, utilizada para gerar um *execution graph* (EG) [2] [14] e a partir deste são extraídos dados para a rede de filas de espera. Um EG, no trabalho deles, representa as seqüências de chamadas de métodos de um sistema, o qual por conseqüência representa a arquitetura lógica do software. Portanto, o EG pode ser utilizado como base para representar as arquiteturas *software*. Se for considerado que cada chamada de método representa um estado do sistema e cada nodo do EG representa uma chamada, neste caso o EG representa os estados do sistema. Com esta abordagem o EG pode ser facilmente convertido em um autômato de estados finitos e adicionado a uma rede de autômatos estocásticos (*Stochastic Automata Network* - SAN)[1], e por conseqüência ser utilizado para avaliar o desempenho do *software* e dos detalhes de sua arquitetura lógica.

Para representar a arquitetura lógica de um *software* orientado a objetos, este trabalho propõe uma técnica de conversão de UML para SAN, onde serão utilizadas as estruturas

[1] Arquitetura de *software* ou lógica neste trabalho representa a estrutura de classes e métodos que compõem um *software*. Não representando estruturas físicas, como máquinas e redes. Estas últimas sendo denominadas da Arquitetura do Sistema ou Física

[2] os detalhes da notação do EG não serão demonstrados neste trabalho. Estes detalhes podem ser obtidos nos trabalhos de Smith e de Cortellesa e Mirandola

básicas do método de Cortellessa e Mirandola, gerando um resultado em SAN e não em Rede de Filas de Espera. Na Sessão 2, serão detalhados a linguagem de modelagem UML, o método de Cortellesa e Mirandola e o método de Redes de Autômatos Estocásticos. Na Sessão 3, será apresentada a técnica proposta para a geração de uma SAN a partir de um modelo UML utilizando um exemplo fictício de uma agência bancária. Na sessão 4, será apresentada uma breve verificação do método, através de um estudo de caso, com exemplos de alguns casos de uso de um sistema. Na Sessão 5, serão apresentadas as conclusões deste trabalho e os trabalhos futuros visionados.

Capítulo 2

Fundamentação Teórica

2.1 UML

A linguagem de modelagem de *software* UML foi especificado por Grandy Booch, James Rumbaugh e Ivar Jacobson baseada em metodologias existentes, são elas *Object-Oriented Software Engineering* - OOSE [6], *Object Modeling Technique* - OMT [12] e Booch [2]. Esta linguagem foi aceita pela OMG (*Object Management Group*) [9] como padrão para modelagem de *software*. A UML procura unificar as vantagens de cada um das metodologias e apresenta inovações como novos diagramas, e na forma de como descrever o *software* atráves de várias visões ou níveis de abstração diferentes: comportamental, lógico e físico. Cada visão pode ser projetada na fase mais adequada do desenvolvimento de *software*, enriquecendo assim o modelo ao longo do projeto de *software*. Um mesmo diagrama pode ser usado em diferentes visões e assim representar, em mais ou menos, detalhes as regras de negócio, a implementação e a implantação do *software*.

A UML define nove tipos de diagramas: diagrama de casos de uso, de seqüência, de atividades, de estados, de classes, de objetos, de implantação, de colaboração e de componentes [3]. Para montar um modelo de avaliação de desempenho, este trabalho utiliza informações de apenas três deste conjunto de diagramas, os quais representam os níveis de abstração comportamental, lógico e físico e são eles: os diagramas de casos de uso, seqüência e de implantação. Por este motivo somente esses serão definidos. A definição dos demais diagramas, assim como os detalhes da modelagem em UML, podem ser encontrados em "UML Guia do Usuário"[9, 3].

2.1.1 Modelagem Comportamental

Na modelagem comportamental de um *software* são descritas suas funcionalidades e as relações com elementos externos, na forma de casos de uso e atores. Nesta modelagem, os principais elementos e diagramas utilizados são os casos de uso, atores, diagramas de casos de uso, digramas de estado e atividades. Conforme mencionado anteriormente, apenas o diagrama de casos de uso e os elementos casos de uso e atores serão utilizados neste trabalho e, portanto, apenas estes serão descritos em detalhes.

2.1.1.1 Atores:

Os atores modelam elementos externos ao sistema, os quais interagem com este. Um ator pode representar uma pessoa, um outro sistema, ou um *hardware*. Estes representam tudo o que troca informações, ou seja, entrada e saída de dados e estímulos. Eles modelam os diferentes papéis assumidos pelos usuários, que o compõem [6].

2.1.1.2 Caso de Uso:

Um caso de uso é uma descrição de um conjunto de seqüências de ações executadas por um sistema para produzir um resultado de valor para um ator [3]. Um caso de uso, geralmente, é formado por:

- Um nome: é uma seqüência alfanumérico que é único no modelo.
- Objetivo: é uma descrição textual da funcionalidade, que o caso de uso representa.
- Pré-condições: são requisitos que devem ser satisfeitos antes da execução da funcionalidade do sistema, a qual o caso de uso representa. Podem não existir pré-condições em um caso de uso.
- Pós-condições: são condições válidas após a execução da funcionalidade, a qual o caso de uso representa.
- Fluxos de Execução: Um fluxo de execução é uma seqüência de passos, descritos textualmente, que devem ser executados para se atingir o objetivo do caso de uso.

Um mesmo caso de uso pode possuir mais de um fluxo de execução, que descreve a funcionalidade que representa. Estes são chamados fluxos alternativos. O caso de uso pode conter, ainda, informações adicionais como, por exemplo, as exceções a serem tratadas. Estas informações podem ser adicionadas em formas de novos elementos do caso de uso.

2.1.1.3 Diagrama de Casos de Uso:

Um diagrama de casos de uso é a estrutura onde estão representados, graficamente, casos de uso, atores e as relações entre eles. Podem existir relações entre casos de uso e atores, atores com atores ou casos de uso com casos de uso. As relações entre caso de uso e ator são chamadas de associação. As relações entre casos de uso pode ser de três tipos: inclusão, extensão e generalização. Existe, ainda neste diagrama, a relação entre atores chamada generalização ou especialização.

2.1.2 Modelagem Lógica ou Estrutural

A modelagem lógica em UML representa a estrutura em termos de classes, objetos e seus relacionamentos, agrupados em diagramas de classes, objetos e seqüências.

2.1.2.1 Classes:

Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica [3].

2.1.2.2 Diagrama de Seqüência:

Diagramas de seqüência representam a relação entre classe, objetos e atores. Estes diagramas podem ser utilizados em diversas partes do modelo UML representando diferentes visões. Na visão (ou modelagem) lógica, os diagramas de seqüência modelam as chamadas de métodos entre as classes, objetos e atores. Cada chamada representa um método de uma classe ou objeto. Estes métodos, na modelagem lógica são os mesmos implementados na linguagem de orientação a objeto alvo deste *software*.

2.1.3 Modelagem da Arquitetura

Na modelagem da arquitetura, são definidos fisicamente como o *software* sendo modelado, estará disposto em termos de componentes do mesmo e sua distribuição em servidores e máquinas cliente.

2.1.3.1 Nós:

Um nó é um elemento físico que existe em tempo de execução e representa um recurso computacional, geralmente tendo alguma memória e capacidade de processamento. [3].

2.1.3.2 Componentes:

Um componente é a parte física de um sistema, o qual se adapta e fornece a realização de um conjunto de interfaces [3]. Componentes representam bibliotecas, pacotes, etc, pertencentes ao sistema. Os componentes são a realização física das abstrações lógicas representadas por classes. Na avaliação de desempenho, estes serão utilizados para identificar quais classes deverão ser executas e em quais nós do sistema.

2.1.3.3 Diagrama de Implantação:

Os diagramas de implantação são formados por nós e as ligações entre eles. Este demonstra a arquitetura física do sistema, provendo informações de ligações existentes entre os nós. Além disso, demonstra a relação entre componentes e os nós.

Na Figura 2.1 são demonstrados os diagramas de Use Case, Seqüência e Componentes para um *software* de um microprocessador, que faz a leitura da temperatura de uma fornalha, através de um sensor e repassa esta a um controlador.

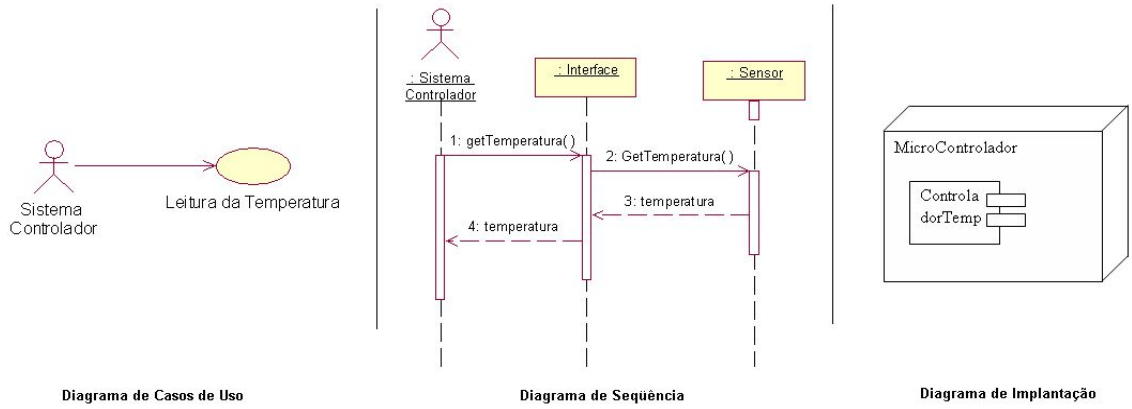


Figura 2.1: Diagramas UML

2.2 Metodo de Conversão UML para QN

O método aqui apresentado está dividido em cinco passos que serão detalhados nas próximas seções deste capítulo:

Obter o Perfil dos Usuários do Sistema: onde são associados dados probabilísticos, que identificam as chances de um usuário acessando o sistema ser de um determinado tipo e,

também, as chances de cada funcionalidade do sistema (use case) ser executada.

Obter as linhas de execução do sistema (EG): neste passo, os diagramas de seqüência são analisados para se obter as possíveis linhas de execução do sistema e estas são representadas em um Execution Graph. Tipicamente, existirá uma linha de execução, saindo do nodo Init (nodo inicial do execution graph), para cada use case, já que os diversos diagramas de seqüência da mesma use case, na maioria das vezes, diferem em apenas algumas chamadas uns dos outros, como por exemplo, nas situações de falha de autenticação, onde a mudança ocorre, somente, no resultado apresentado.

Construir a topologia da Rede de Filas de Espera : O diagrama de deployement representa o hardware a ser utilizado para rodar a aplicação. E, é deste que são obtidas as filas e as conexões entre elas da rede de fila de espera.

Assinalar os tempos de processamento ao EG: Nos passos anteriores, foram criadas as estruturas do EG e da rede de filas de espera, neste devem ser assinalados os dados de tempo de comunicação das transferências de controle entre os módulos.

Resolver o EG e assinalar os valores obtidos a Rede de filas de Espera: O último passo, antes da resolução da rede de filas de espera,é a obtenção dos dados de tempo de serviço das filas e comportamento dos clientes (dados dos terminais). Estes são extraídos do execution graph.

2.2.1 Obter o Perfil dos Usuários do Sistema

O perfil de um usuário, mencionado neste trabalho está relacionado a definição de quais são as probabilidades de um usuário ser de um tipo específico (ator) ao utilizar o *software* e as probabilidades dele acessar cada funcionalidade do sistema.

As relações entre o sistema e seus usuários são expressas pelo diagrama de *use cases*. Portanto, este é o diagrama onde os valores probabilísticos devem ser relacionados com funcionalidades (*use cases*), e usuários (atores). Este passo do método consiste em obter estas probabilidades e enriquecer o diagrama de *use cases* com as mesmas.

As probabilidades dos tipos de usuários acessarem o sistema, ou seja, de um usuário acessando o sistema ser de um tipo, serão assinaladas aos atores do diagrama de *use cases*. Sendo A o número de atores no sistema, *p(i)* (sendo i um valor entre *(1..A)*) é a probabilidade do ator i ator ser o usuário que está acessando o sistema. O somatório de todas probabilidades p(i) deve ser 1, ou seja, as probabilidades dos atores devem estar normalizadas.

$$\sum_{i=1}^{a} p(i) = 1$$

Para cada ator i no diagrama de *use cases* existirá uma probabilidade *Pij* deste ator i acessar a funcionalidade contida na use case j, caso não haja uma relação entre ator e use case, esta probabilidade é zero, mas caso haja o diagrama de *use cases* deverá ser enriquecido com estas probabilidades. Sendo u o número de *use cases* do sistema, o somatório da probabilidades *Pij*, sendo $j = (1..u)$, é igual a 1, para um mesmo ator i, em outras palavras, estas probabilidades devem estar normalizadas para cada ator.

$$\sum_{j=1}^{u} P(ij) = 1$$

Na Figura 2.2 podem ser observado um diagrama de *use cases* enriquecido com as probabilidades. Ao lado do ator, está a probabilidade do mesmo acessar o sistema. Como ele é um ator único no sistema recebe a probabilidade igual a 1. Nos arcos estão representadas as probabilidades do ator acessar as *use cases* com que se relaciona.

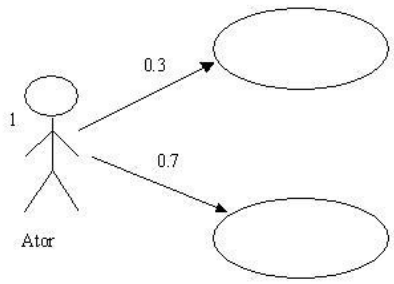


Figura 2.2: Diagrama de *Use Cases* enriquecido com probabilidades

Como estes valores no diagrama de *use cases* pode-se obter, agora, a probabilidade consolidada de uma funcionalidade (use Case) ser executada (considerando todos os atores que a acessam), ou seja, verificar qual a chance de uma funcionalidade ser acessada, quando um usuário qualquer utilizar o sistema. Essa probabilidade é calculada através do somatório das probabilidades dos i atores (p(i)),sendo i= 1 até A , multiplicado pela probabilidade deste *iésimo* ator acessar a use case x (P(ix)).

$$\sum_{i=1}^{A} p(i) * P(ix)$$

2.2.2 Obter as linhas de execução do sistema (EG)

Um execution graphs, como já mencionado, representa as diferentes linhas de execução de um sistema. Neste passo, serão obtidas estas linhas, para tanto, serão utilizados os diagramas

de seqüência, que expressam exatamente as relações entre as chamadas aos componentes do sistema.

Foi definido em [4] um algoritmo para montar a estrutura de um *execution graph* a partir dos diagramas de seqüência do UML.

O algoritmo cria um nodo de inicial chamado **Init** que é um nodo de laço de onde todas as linhas de execução do sistema partem. Cada diagrama de seqüência irá contribuir com parte do EG, portanto para cada diagrama de seqüência o algoritmo irá avaliar as chamadas entre os elementos do sistema e transformá-las em um nodo básico do execution graph. Cada nodo básico será identificado por uma tupla (l, A1, A2), onde l é o nome da transação, A1 é o componente de origem da transação e A2 é componente de destino da mesma. Existirá, ainda, um tupla (l', A2, A1) que expressa o retorno de controle para o elemento A1.

Os diagramas de seqüência podem possuir dois tipos de transações: as simples, onde apenas uma chamada é feita e um retorno é esperado e as múltiplas, onde mais de uma chamadas é realizada e então é aguardado o retorno das mesmas para continuar o processamento no componente de origem [4].

O algoritmo, em questão, é demonstrado na apêndice A. Na primeira etapa do algoritmo 1 são considerados os nodos que ainda não foram traduzidos, ou seja ainda não apareceram em nenhum diagrama de seqüência já avaliado. Neste caso deve ser avaliado se a transação é múltipla ou simples.

Caso seja uma transação simples é avaliado se existe uma conexão em aberto no EG, onde este novo nodo seria o próximo da linha de execução. Se existir, conecta-se a mesma ao novo nodo simples. Caso não exista conexão em aberto, a única possibilidade é de uma seleção em um linha de execução (uma nova sub linha de execução será criada).

No caso de novas chamadas múltiplas, o algoritmo se comporta de maneira semelhante ao caso da simples, com um principal ponto a ser adicionado que é os nodos de *fork* e *join* como é observado no "*else*"que indica a chamada múltipla no algoritmo 2 descrito no apêndice A

Ao final do algoritmo, a topologia do Execution Graphs estará completa. Nos passos a seguir, serão descritas as formas de se obter dados do mesmo e obter a estrutura da rede de filas de espera

2.2.3 Usar o diagrama de deployment para obter a Rede de Filas de Espera

Após utilizar as informações dos diagramas de casos de uso e de seqüência nos passos anteriores chega o momento de extrair os dados do diagrama de implantação.

As informações retiradas deste diagrama são a topologia da rede de filas de espera e os dados de comunicação. A topologia da rede de filas de espera é nada mais que a representação do hardware em filas de espera. Os dados de comunicação serão tratados na seção 2.2.4.

A obtenção é relativamente simples, pois cada uma das unidades de processamento ou de armazenamento existentes no diagrama de implantação, e utilizadas pelo sistema são mapeadas para uma fila de espera que irá integrar a rede.

O caminho entre as filas é determinado pelas conexões do hardware no diagrama de deployment. Se estas forem redes de computadores devem ser adicionados elementos delay na conexão, que representam o atraso provocado pela estrutura de rede de computadores entre os processadores (latência). A topologia pode sofrer alguns ajustes, que são determinados de forma empírica pelo projetista para melhor representar o *hardware*.

Por este mapeamento ser muito simples, fica facilitada a avaliação de outras estruturas de *hardware* para o mesmo sistema, bastando apenas criar-se novos diagramas de implantação. Na Figura 2.3 é demonstrado um diagrama de implantação a sua tradução, para uma rede de filas de espera.

2.2.4 Obter os valores numéricos para o EG

Após gerar a topologia da rede de filas de espera o diagrama de implantação, também, fornece informações para preencher dados no EG. Estes dados são os tempos médios gastos em comunicação e processamento de cada transação dos diagramas de seqüência (cada nodo básico do EG).

Para cada nodo básico será adicionado um tempo de comunicação e processamento que leva em conta o local físico onde cada componente envolvido na chamada está. Se um componente X faz uma chamada a um componente Y no mesmo computador, o tempo de chamada é menor do que o mesmo componente X chamar um componente Z em um computador remoto.

O diagrama de implantação vai auxiliar exatamente na definição dos tempos de cada chamada. Os tempos em si são valores que o projetista terá que obter ou estimar.

Após assinalados os tempo de processamento devem ser agora assinaladas a probabilidades de cada linha de execução ocorrer. Se cada linha de execução equivale a um diagrama de seqüência de uma use case pode-se, então, utilizar os dados probabilísticos obtidos no passo 1, para se obter as probabilidade de cada diagrama de seqüência ser executado.

No passo 1 foram identificadas as probabilidades consolidadas de cada caso de uso. Para se obter as probabilidade de cada diagrama de seqüência o projetista deve avaliar a freqüência

com que cada diagrama de seqüência ocorre no caso de uso. Se essa freqüência for normalizada, dentro de um mesmo caso de uso, e assim conter valor onde sua soma seja 1, pode-se afirmar que, esta é a probabilidade do diagrama de seqüência ser executado quando a use case for executada. A probabilidade consolidada de cada diagrama de seqüência será, então, a probabilidade consolidada da use case, obtida no passo 1, multiplicada pela freqüência normalizada (probabilidade) do diagrama de seqüência, recém assinalada.

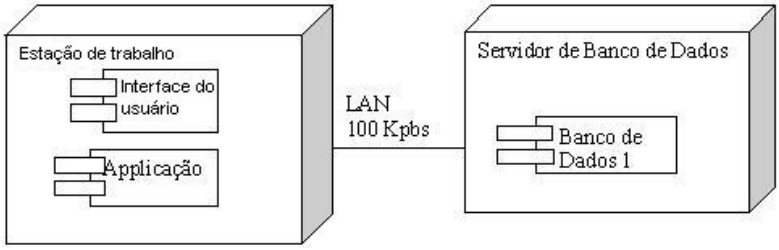
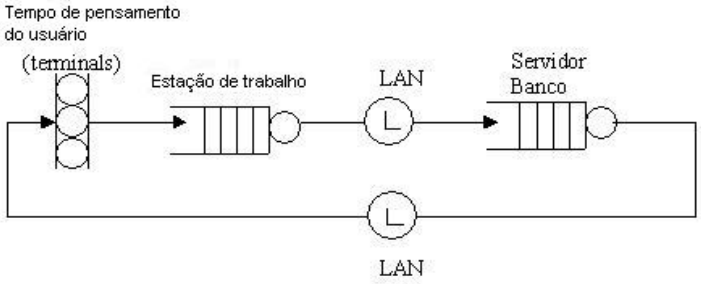


Diagrama de componentes / implantação



Rede de Filas de Espera Resultante

Figura 2.3: Obtenção da rede de filas de espera

Para assinalar estes valores ao EG deve-se observar, que os diagrama de seqüência de um mesmo caso de uso tem nodos básicos em comum nas linhas de execução, porém em algum ponto da linha de execução haverá uma seleção e é neste ponto onde serão assinaladas as probabilidade dos diagramas de seqüência. Após assinaladas todas as probabilidades aos nodos de seleção, para cada nodo de seleção deve-se normalizar as probabilidades, para que se tenham de fato, probabilidades de um caminho ou os outros.

O ponto crítico a ser observado, nas seleções, é a seleção logo seguir do nodo **Init**, como haverão linhas que tem execuções iniciais em comum a probabilidade, que deve ser assinal-

ada a cada linha deste nodo de seleção, é o somatório da probabilidades consolidadas dos diagramas de seqüência que seguirão seu fluxo pelo ramo da seleção, e como mencionado no parágrafo anterior, nos próximos nodos de seleção esta probabilidade será novamente ramificada e normalizada.

2.2.5 Combinar o EG e a Rede de Filas de Espera gerados

Com o Execution Graph podem-se obter, neste momento, os dados de tempo médio de serviço de cada fila da Rede de Filas de Espera. Para tanto o Execution Graph deve ser resolvido. A resolução de execution graphs apresentada aqui pode ser encontrada em [15] e consiste em verificar a ocupação de cada recurso (fila da rede de filas) por cada nodo básico do EG e somá-las ponderando os totais com as probabilidades de cada linha de execução (nodo de seleção), como será descrito a seguir.

O primeiro passo na resolução do EG é a criação de uma tabela, que contém a relação entre os nodos básicos e a ocupação de tempo de cada recurso modelado como uma fila na topologia da rede de filas de espera obtida no passo 2.2.3. Note, que existirão nodos que utilizarão apenas alguns recursos, neste caso os tempos dos recursos não utilizados são zerados. Na Figura 2.4 pode-se observar um execution graph simples e a tabela de tempos relacionados a ele.

Com esta tabela, agora, deve-se percorrer os nodos básico de cada linha de execução, partindo do último nodo básico da linha, que é exatamente o nodo que conecta-se ao nodo *Init*, em direção ao nodo pai, o que passa o controle ao nodo atual, somando os valores de cada recurso na tabela, até chegar a um nodo de seleção, onde o somatório será ponderado ou ao nodo *Init* onde os valores estarão finamente calculados.

Quando é encontrado um nodo de seleção deve-se calcular todos os demais ramos da seleção antes de continuar, pois nestes pontos os valores de cada ramo são multiplicados pela probabilidade do ramo, o que gera um valor médio de ocupação dos recursos em todos os ramos da seleção.

O nodo *Init* contém todos os pontos de saída das linhas de execução, e portanto quando as iterações o alcançarem o somatório estará concluído chegando aos valores de taxa de serviço para cada fila da rede de filas de espera.

Assim são obtidos os tempos médios de serviço. Os valores de delay e dos terminais são obtidos também EG e são exatamente os valores adicionados no passo anterior com atrasos provocados pela comunicação, com exceção do número de clientes dos terminais, onde este deve ser estimado pelo projetista.

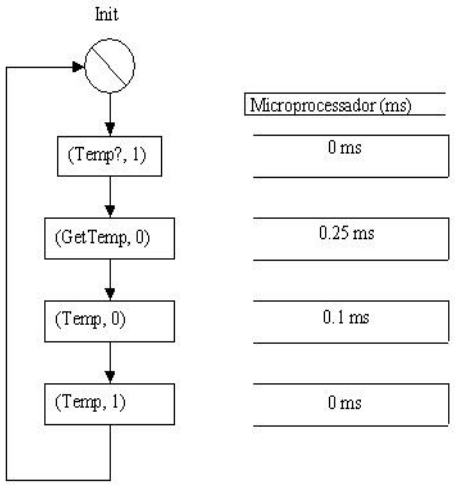


Figura 2.4: Tabela de ocupação dos recursos

No exemplo da Figura 2.4 a taxa de serviço da única fila da rede que representará o microprocessador teria um tempo de serviço média de 0.35 ms. As taxas dos terminais (freqüência em que o usuário acessa) estimada pelo projetista é 1 ms que é exatamente o valor do nodo logo a seguir do Init e o último nodo antes do retorno ao Init, que se for observado no diagrama de seqüência da Figura 2.1 são exatamente as iterações com o Sistema Controlador (usuário do sistema). Neste caso o número de clientes dos terminais deve ser considerado igual a 1, pois só existe um sistema controlador.

2.3 Redes de Autômatos Estocásticos

Redes de Autômatos Estocásticos (SAN) é um formalismo baseado em cadeias de Markov [1]. O formalismo de SAN é capaz de modelar diversos problemas, e através dessa modelagem obter índices de desempenho sobre os mesmos. Uma Rede de Autômatos Estocásticos é composta por um conjunto de autômatos que possuem dependências entre si.

Cada autômato é composto por um conjunto de estados e transições. A composição dos estados locais de cada autômato da rede define a corrente situação da rede, também chamado de estado global da rede. Cada autômato da rede estará em um e apenas um estado. Transições definem a mudança de um estado para outro. Uma transição é disparada pela ocorrência de um evento.

Os eventos podem ser de dois tipos: locais ou sincronizantes. Os eventos locais afetam apenas o estado de um autômato da rede. Os eventos sincronizantes sincronizam autômatos

da rede, isto é, afetam estados de dois ou mais autômatos da rede. Um evento sincronizante só pode acontecer, quando todos os autômatos envolvidos com este evento estiverem em um estado de onde o evento possa ser disparado.

Cada evento tem associado uma taxa, em escala de tempo contínua, ou uma probabilidade em escala de tempo discreta. Estas taxas ou probabilidades podem ser funcionais ou constantes. As taxas e probabilidades funcionais são definidas por uma função (que depende dos demais autômatos da rede), ou seja, a taxa varia conforme a situação em que a rede se encontra.

Um evento pode ser associado a duas ou mais transições partindo de um mesmo estado. Neste caso, deve-se associar uma probabilidade de rotação ao par (evento, transição). Se apenas um caminho é possível, a probabilidade de rotação é igual a 1.0 e pode ser omitida.

Uma SAN possui um conjunto de estados globais, onde cada estado global equivale a uma combinação de estados locais dos autômatos da rede. Supondo que $S^{(i)}$ seja o conjunto de estados locais do autômato $\mathcal{A}^{(i)}$, o conjunto de estados globais de uma rede composta por N autômatos $\mathcal{A}^{(i)}$, onde i ϵ [1..N] é dado pelo produto cartesiano dos espaços de estados $S^{(i)}$, ou seja: $\Pi\ S^{(i)}$. Porém, devido as taxas funcionais e eventos sincronizantes, alguns dos estados globais podem não ser atingíveis, por isso, define-se um subconjunto dos estados globais como atingíveis [1].

Na Figura 2.5, pode-se observar um exemplo de uma SAN com dois autômatos. O autômato $\mathcal{A}^{(1)}$ possui os estados 'a', 'b' e 'c' e o autômato $\mathcal{A}^{(2)}$ que possui os estados 'x' e 'y'. Os eventos nesta rede são os seguintes:

- e1, e2 e e3, eventos locais envolvendo somente $\mathcal{A}^{(1)}$ com taxas constantes λ_1, λ_2 e λ_3;
- e4, um evento sincronizante envolvendo $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(2)}$, com um taxa constante λ_4;
- e5, evento local envolvendo $\mathcal{A}^{(2)}$ com uma taxa funcional f:
  - f = μ_1, se $\mathcal{A}^{(1)}$ esta no estado a;
  - f = 0 , se $\mathcal{A}^{(1)}$ esta no estado b;
  - f = μ_2, se $\mathcal{A}^{(1)}$ esta no estado c;

Quando o evento e4 acontece com a taxa λ_4 existe uma probabilidade π no autômato $\mathcal{A}^{(1)}$ de ocorrer a transição do estado c para o estado b, e uma probabilidade 1-π de ocorrer a transição do estado c para o estado a.

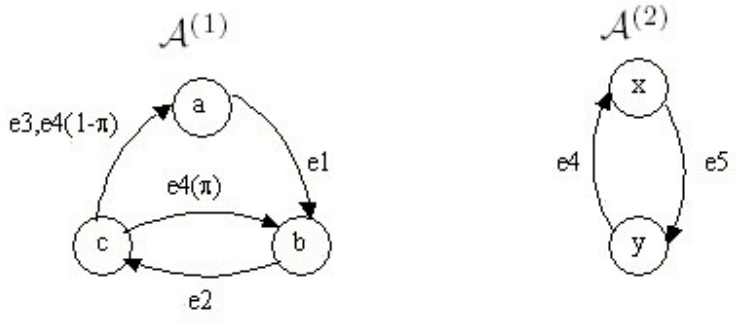


Figura 2.5: Rede de autômatos estocásticos [1]

Redes de autômatos estocásticos é um formalismo de grande poder de representação e simples de ser construído, por não ser um único sistema, mas sim um conjunto de subsistemas que podem ou não possuir dependências entre si.

Capítulo 3

Metodologia

A técnica aqui apresentada é baseada no método de conversão de UML para QN[4]. Neste método é utilizada uma estrutura intermediária chamada *execution graph* e um processo chamado *Software Performance Engineering*[15] [1]para resolvê-la e gerar valores numéricos para a QN. Na técnica de conversão de UML para SAN aqui apresentada a estrutura EG será montada da mesma forma, que no método de conversão de UML para QN, porém sua finalidade é diferente. O EG será utilizado para a criação de autômatos que compõem a SAN resultante.

A conversão de UML para SAN se dará por dez passos. Alguns deles serão total ou parcialmente reutilizados da conversão de UML para QN que são os casos da Obtenção do Perfil dos Usuários, Obtenção das Linhas de Execução e Adição dos Tempos de Processamento ao EG. Os demais 7 passos adaptam o EG e o UML para a conversão específica para o formalismo SAN.

Na próxima sessão, serão apresentadas as restrições com relação à modelagem UML e com relação à SAN gerada por esta técnica. Em seguida, será apresentada uma modelagem hipotética obtida em [11], a qual será utilizada para exemplificar a técnica ao longo de seus passos.

3.1 Restrições

Para a aplicação da conversão de UML para SAN a modelagem UML deverá seguir algumas regras e restrições descritas a seguir.

[1]O processo *Software Performance Engineering* não será utilizado nesta técnica. Este é citado aqui apenas como referência.

Na modelagem comportamental, a primeira restrição que é com relação a generalização de atores. O tratamento da generalização de atores não será demonstrada nesta técnica, e portanto não deve ser utilizadas para que se possa avaliar o desempenho com a técnica apresentado a seguir. A generalização pode ser substituída pela inserção das relações entre o ator especialista e os casos de uso ligados ao ator generalista.

Na modelagem lógica, os diagramas de seqüência deverão possuir apenas chamadas síncronas, pois chamadas assíncronas podem gerar EGs desconexos [4].

Ainda na construção dos diagramas de seqüência, algumas regras deverão ser seguidas no projeto destes diagramas. Este é o caso dos casos de uso incluídos, estendidos e especialistas para que estes gerem EG com coerentes.

As seqüências dos casos de uso incluídos não deverão ser criados como diagramas de seqüência em separado, mas deverão ser inclusos nos diagramas de seqüência dos casos de uso base. Assim, estes serão traduzidos em EG como parte constante dos casos de uso base inseridos em cada contexto dos mesmos.

Os diagramas de seqüência dos casos de uso estendidos deverão conter todo o cenário onde este se inserem, ou seja, deverão estar contidos nas interações dos diagramas de seqüência dos casos de uso base nos devidos pontos de extensão. Se este caso de uso for estendido por mais de um caso de uso, cada diagrama de cada caso de uso que o estende, deve conter seus fluxos representados nos devidos diagramas de seqüência.

No caso da generalização de casos de uso os diagramas de seqüência dos casos de uso especialistas, assim como na extensão, deverão conter interações dos diagramas de seqüência do casos de uso geral. Assim, o caso de uso geral não possuirá um diagrama de seqüência, pois seus fluxos já estarão representados nos diagramas dos especialistas.

Na modelagem física ou de implantação existe uma restrição com relação aos nós do diagrama de implantação. Os nós, como máquinas clientes, na realidade não são apenas um recurso, mas um conjunto de recursos que, individualmente, podem gerar gargalos. Na técnica exposta estes serão representados apenas com dois estados, ocupado ou livre. Considera-se a existência de apenas um recurso físico (conjunto processador e memória) para cada nó do diagrama de implantação, desconsiderando máquina multi processadas. Para representar um nó multi-processado, o projetista do sistema deverá considerar o tempo compartilhado entre os processadores no cálculo dos tempos gastos por cada chamada neste tipo de recurso.

Com relação aos modelos SAN gerados por esta técnica podem haver limitações na capacidade de processamento devido à quantidade de estados globais geradas. A resolução SAN gerada por esta técnica está restrita aos recursos de *hardware* (processador e memória)

disponíveis. Não está no escopo deste trabalho um estudo em relação a esta limitação, porém sabe-se da existência desta limitação nas ferramentas de análise de modelos SAN. A conseqüência disso poderá ser notada em arquiteturas lógicas e físicas muito complexas, pois estas podem gerar autômatos com muitos estados, gerando com isso muitos estados globais.

3.2 Exemplo

Para ilustrar os passos da metodologia será utilizado um exemplo de um sistema bancário. Este sistema é um exemplo acadêmico simples, que será utilizado para exemplificar a técnica. Este exemplo foi retirado da página da disciplina de Mo-delagem Conceitual de Sistemas [11]. O diagrama de casos de uso é composto por 3 atores e 8 casos de uso. Na Figura 3.1 está a representação gráfica do mesmo e a seguir está a descrição de cada caso de uso.

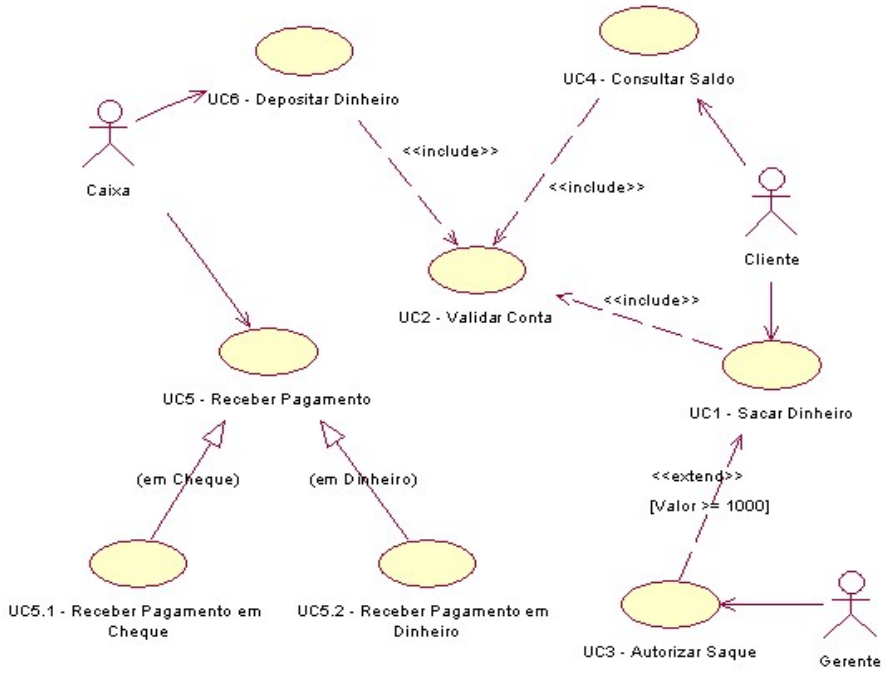


Figura 3.1: Diagrama de Casos de Uso Sistema Bancário

UC1

Caso de uso: Sacar dinheiro

Atores: Cliente

Pré-Condições: o Cliente possui cartão do banco e senha cadastrada.
Pós-Condições: lançada a transação na conta do Cliente, atualizado o saldo da conta corrente e liberado o dinheiro.
Seqüência Básica:

1. Este caso de uso começa quando o Cliente realiza a leitura do cartão do banco, no caixa eletrônico.
2. O Cliente informa a sua senha.
3. *Include* Validar Conta.
4. O Cliente informa o valor do saque; Extend (quantia elevada) Autorizar Saque Resposta do sistema.
5. O sistema autoriza o saque e lança o débito na conta corrente do Cliente.
6. O sistema libera o dinheiro.

Seqüência Alternativa:
5a: Fundos Insuficientes:

1. O sistema não autoriza o valor solicitado para saque pelo Cliente.
2. A operação é cancelada.

UC2
Caso de Uso: Validar conta
Atores:
Pré-Condições:
Pós-Condições:
Seqüência Básica:

1. O sistema valida a conta corrente e senha do Cliente, autorizando a operação.

Seqüência Alternativa: 1a. Cliente Inválido

1. O sistema não reconhece a conta corrente e senha do Cliente como válida.

2. A operação é cancelada.

UC3

Caso de Uso: Autorizar saque
Atores: Gerente
Pré-Condições:
Pós-Condições:
Seqüência Básica:

1. O Gerente consulta informações da conta corrente de um cliente para deli-berar sobre a liberação de saque em valor elevado.
2. O sistema apresenta as informações sobre o cliente e suas movimentações bancárias.
3. O Gerente autoriza o saque no valor solicitado.

Seqüência Alternativa:
3a: Saque não autorizado

1. O Gerente não autoriza o saque no valor solicitado.
2. O Sistema cancela a operação.

UC4

Caso de uso: Consultar Saldo
Atores: Cliente
Pré-Condições: o Cliente possui cartão do banco e senha cadastrada.
Pós-Condições:
Seqüência Básica:

1. Este caso de uso começa quando o Cliente realiza a leitura do cartão do banco no caixa eletrônico.
2. O Cliente informa a sua senha.
3. *Include* Validar Conta.

4. O Sistema Informa o Saldo.

UC5

Caso de uso: Receber Pagamento
Atores: Caixa
Pré-Condições: o Caixa é identificado e autenticado.
Pós-Condições: o pagamento recebido é registrado no sistema associado ao Caixa.
Seqüência Básica:
Seção Principal

1. Este caso de uso começa quando o Caixa registra o documento de cobrança bancária a ser pago.
2. O sistema valida a aceitação do documento de cobrança a ser pago.
3. O Caixa informa a opção desejada:
   3.1 Se for pagamento em dinheiro, ver subseção Receber pagamento em di-nheiro.
   3.2 Se for pagamento em cheque, ver subseção Receber pagamento em cheque.
4. O sistema registra o pagamento.
5. O sistema imprime o comprovante.

Subseção: Receber pagamento em cheque

1. O Caixa recebe o cheque e o registra no sistema.
2. O sistema valida os dados do cheque.

Subseção: Receber pagamento em dinheiro

1. O Caixa registra o valor em dinheiro recebido.
2. O sistema informa o troco a ser repassado ao pagante.

UC6

Caso de uso: Depositar Dinheiro

Atores: Caixa
Pré-Condições:
Pós-Condições:
Seqüência Básica:

1. Este caso de uso inicia quando um Caixa irá fazer um depósito em dinheiro em uma conta de um cliente.
2. O Caixa informa a conta alvo.
3. *Include* Validar Conta.
4. O Caixa informa a quantia a ser depositada.
5. o sistema deposita e imprime o comprovante.

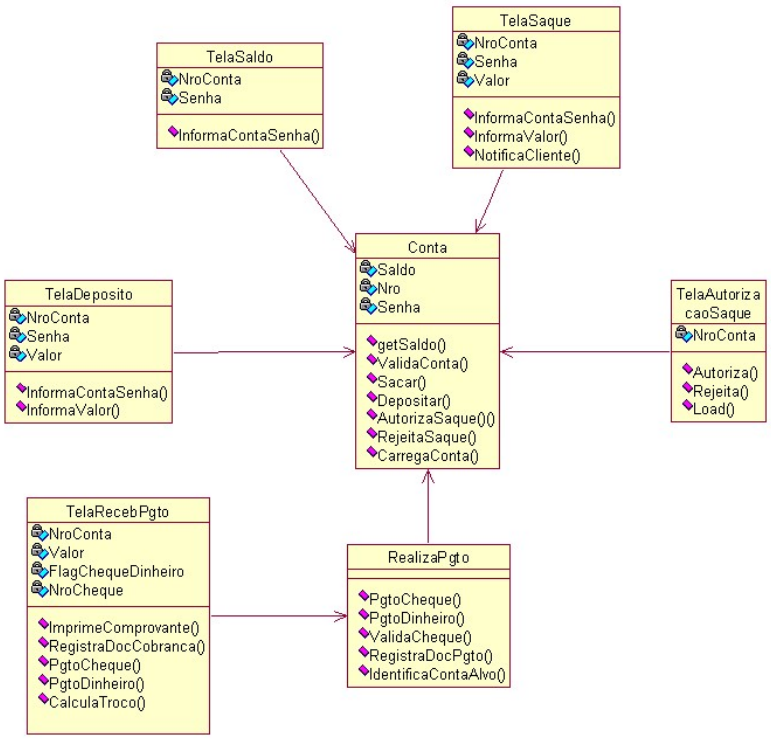


Figura 3.2: Diagrama de casos de uso sistema bancário

O projeto das classes foi resumido ao diagrama de classes demonstrado na Figura 3.2[2]. As Classes de Conta e RealizaPgto são responsáveis por todo o processamento de servidor. As classes iniciadas por telas são as classes que representam a interface com o usuário. Este modelo foi criado com estruturas simples não utilizando nenhum padrão de projeto, a fim de produzir diagramas de seqüência pequenos e por conseqüência produzir um EG não muito extenso para a condução dos passos da conversão de UML para SAN apresentados nas próximas seções.

A complexidade da arquitetura lógica (modelo de classes) pode determinar produzir um EG com muitos nodos o que pode gerar uma SAN com muitos estados e inviabilizar a resolução da SAN. A quantidade máxima de estados globais resolvíveis é dependente do *hardware* no qual se está rodando o *software* de re-solução de SAN, como já mencionado na seção de restrições.

Os diagramas de seqüência gerados para implementar os casos de uso descritos no diagrama de casos de uso da Figura 3.1 são demonstrados nas figuras a seguir.

Para a implementação do caso de uso UC1 - Sacar Dinheiro foram criados três diagramas de seqüência: a básica com a seqüência de sucesso; a alternativa onde trata o caso de saldo não suficiente; e a alternativa 2 com a falha, vinda de caso e uso incluído, na validação da conta. Estes estão representados nas Figuras 3.3, 3.4 e 3.5 respectivamente. Note que, conforme descrito nas restrições, as seqüências do caso de uso UC2 incluído foram adicionados nos diagramas de seqüência do caso de uso UC1.

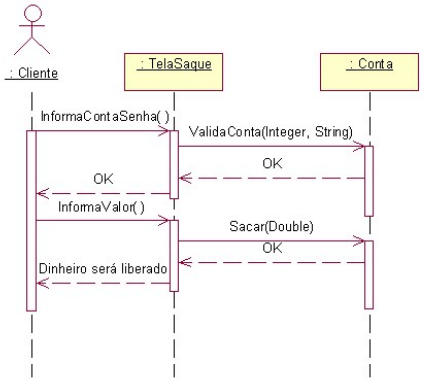


Figura 3.3: Diagrama de Seqüência básica do Caso de Uso UC1 - Sacar Dinheiro

[2] o diagrama de classes não será utilizado na conversão de UML para SAN. Este apresentado aqui apenas para melhor ilustrar o exemplo do sistema bancário.

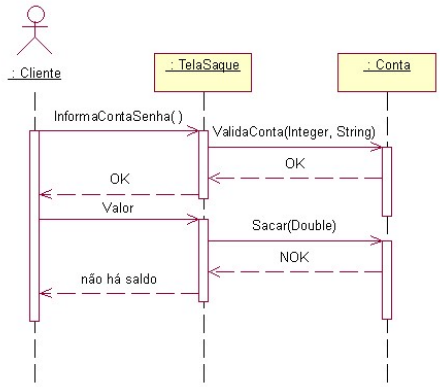


Figura 3.4: Diagrama de Seqüência Alternativa 1 do Caso de Uso UC1 - Sacar Dinheiro

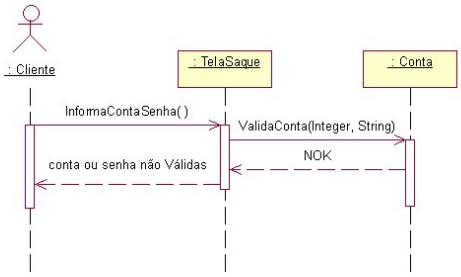


Figura 3.5: Diagrama de Seqüência alternativa 2 do Caso de Uso UC1 - Sacar Dinheiro

Para implementar o caso de uso UC3 - Autorizar Saque foram gerados dois diagramas de seqüência demonstrados nas Figuras 3.6 e 3.7. O processo do banco para a autorização de saques com valor maior que 1000 assume, que sempre haverá um gerente disponível no terminal de saque para autorizar, de forma online, o saque.

Para o caso de uso UC4 - Consultar Saldo foram gerados dois diagramas de seqüência demonstrados nas Figuras 3.8 e 3.9. O diagrama de seqüência alternativo foi resultado da inclusão do caso de uso UC2 - Validar Conta, assim como no caso de uso UC1 - Sacar Dinheiro.

Para os casos de uso UC5 - Receber Pagamento, UC5.1 - Receber Pagamento em Cheque e UC5.2 - Receber Pagamento em Dinheiro foram criados dois diagramas de seqüência, um para cada especialista (US5.1 e UC5.2). Estes são encontrados nas Figuras 3.10 e 3.11 respectivamente. A seqüência do caso de uso geral UC5 foi inserida nos diagramas de seqüência dos especialistas, conforme descrito nas restrições.

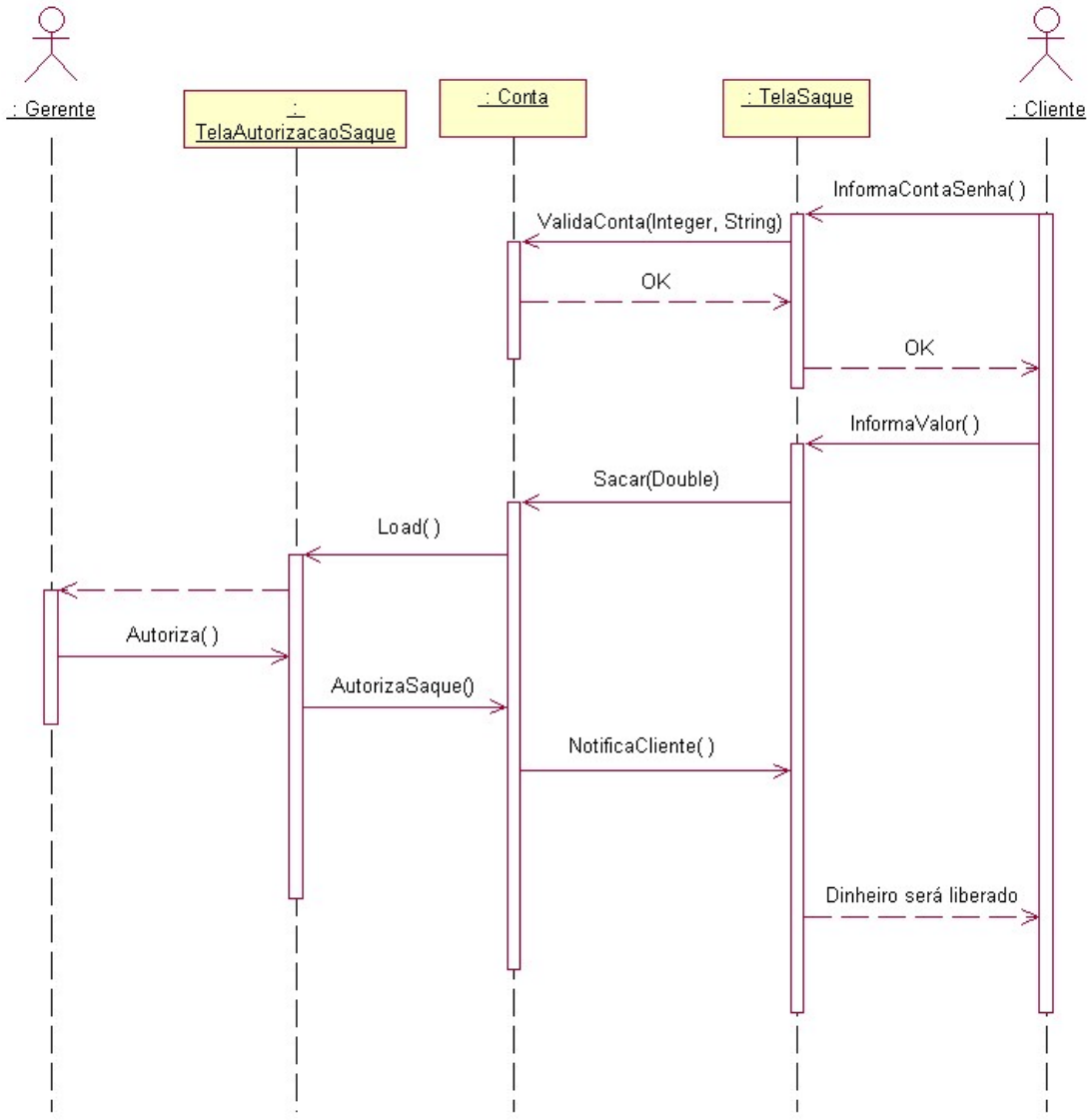


Figura 3.6: Diagrama de Seqüência básica Caso de Uso UC3 - Autorizar Saque

Para o caso de uso UC6 - Depositar Dinheiro foram gerados dois diagramas de seqüência demonstrados nas Figuras 3.12 e 3.13. O diagrama de seqüência alternativa foi resultado da inclusão do caso de uso UC2 - Validar Conta, assim como no caso de uso UC1 - Sacar Dinheiro e UC4 - Consultar Saldo.

O sistema será implantado numa arquitetura física bem simples, composta por um terminal de atendimento e um servidor. Estes nós são interligados por uma rede. O diagrama

de implantação com esta arquitetura é demonstrado na Figura 3.14.

Neste exemplo, existem dois componentes: InterfaceUsuario, o qual contem todas as classes que começam por Tela e o componente ControladorBanco que contêm as classes Conta e RealizaPgto. O componente InterfaceUsuario será implantado no nó Terminal, já o ControladorBanco será implantado no nó Servidor.

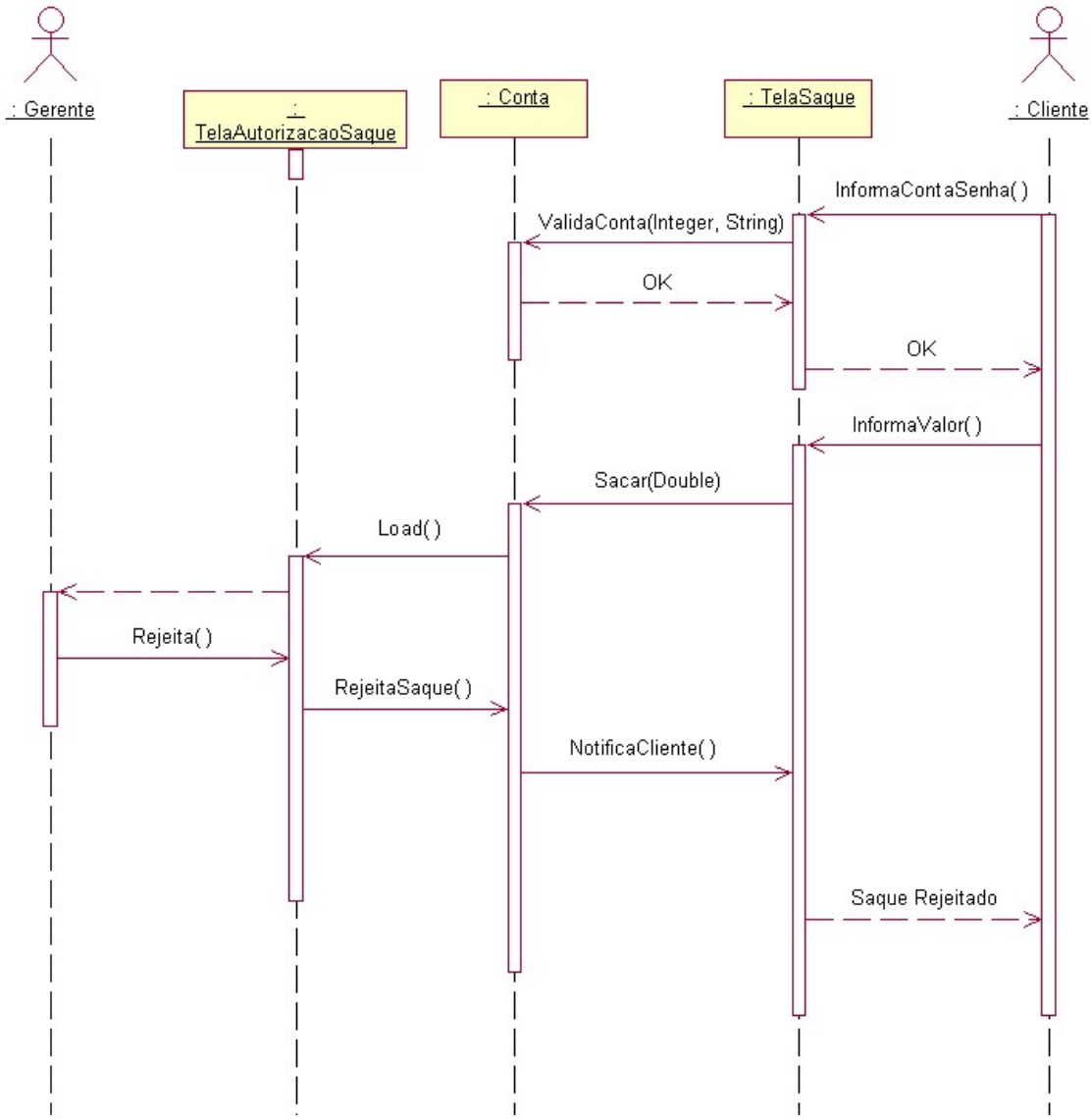


Figura 3.7: Diagrama de Seqüência básica Caso de Uso UC3 - Autorizar Saque

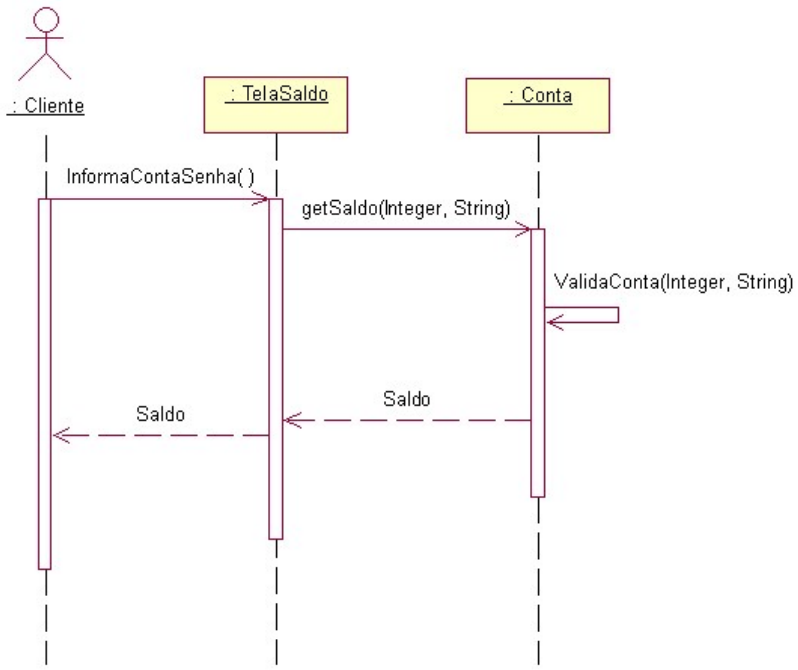


Figura 3.8: Diagrama de Seqüência básica Caso de Uso UC4 - Consultar Saldo

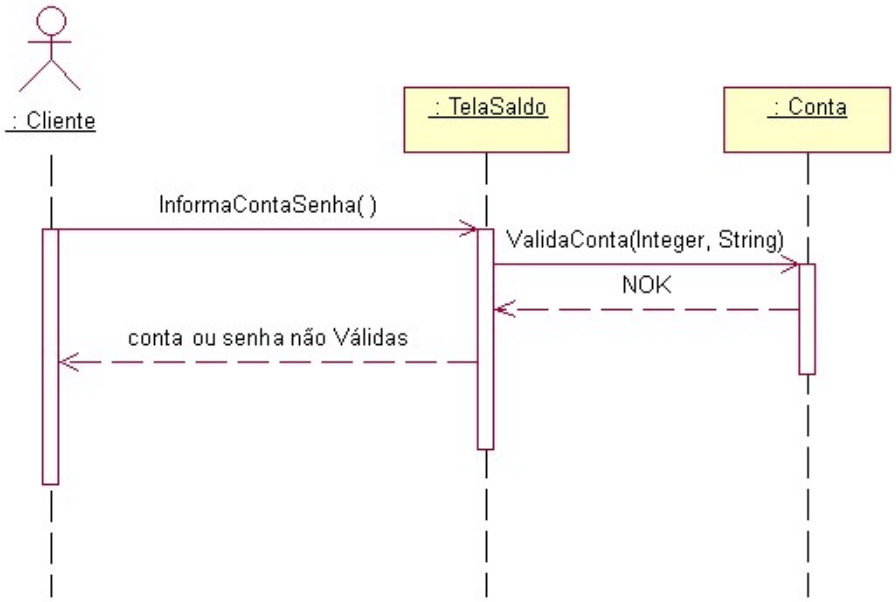


Figura 3.9: Diagrama de Seqüência alternativa Caso de Uso UC4 - Consultar Saldo

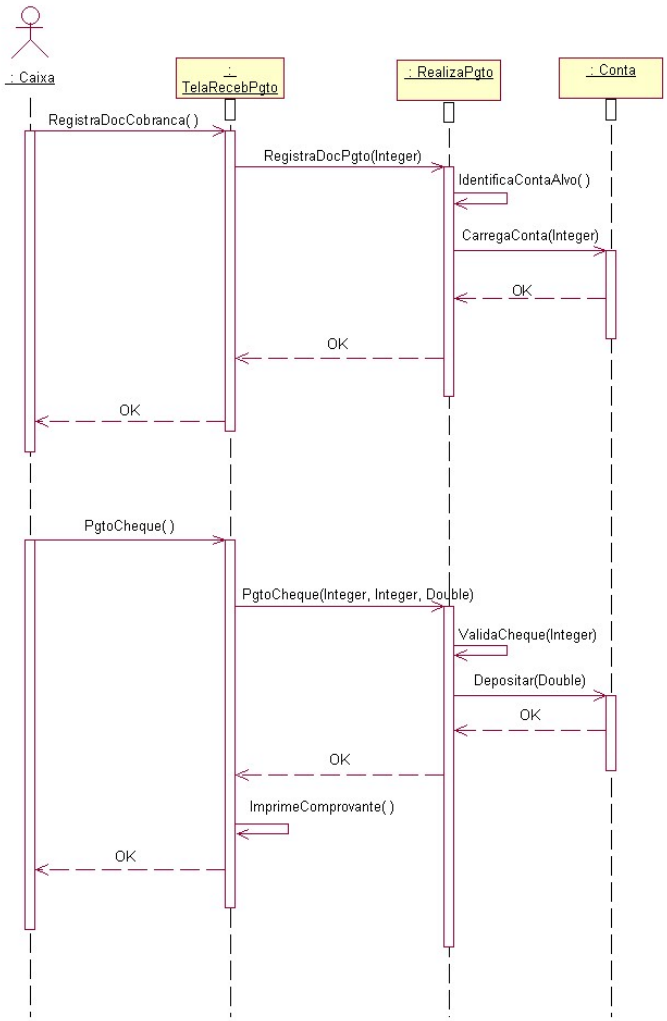


Figura 3.10: Diagrama de Seqüência básica Caso de Uso UC5.1 - Receber Pagamento em Cheque

3.3 Obtenção do Perfil do Usuário

Na avaliação do desempenho de um *software*, o comportamento do usuário é um fator de extrema importância, pois com este é possível focar em gargalos do sistema, onde realmente os usuários, do mesmo, irão utilizá-lo com mais freqüência.

Em UML, o comportamento dos usuários do sistema está demonstrado nas relações entre atores e casos de uso, no diagrama de casos de uso. Neste, estão descritas todas as pos-

síveis funcionalidades, representadas pelos casos de uso, que os usuários, representados por atores, podem executar. Porém, não estão especificadas quais as funcionalidades com maiores chances de serem acessadas e por conseqüência maiores chances de provocarem problemas de desempenho no *software*.

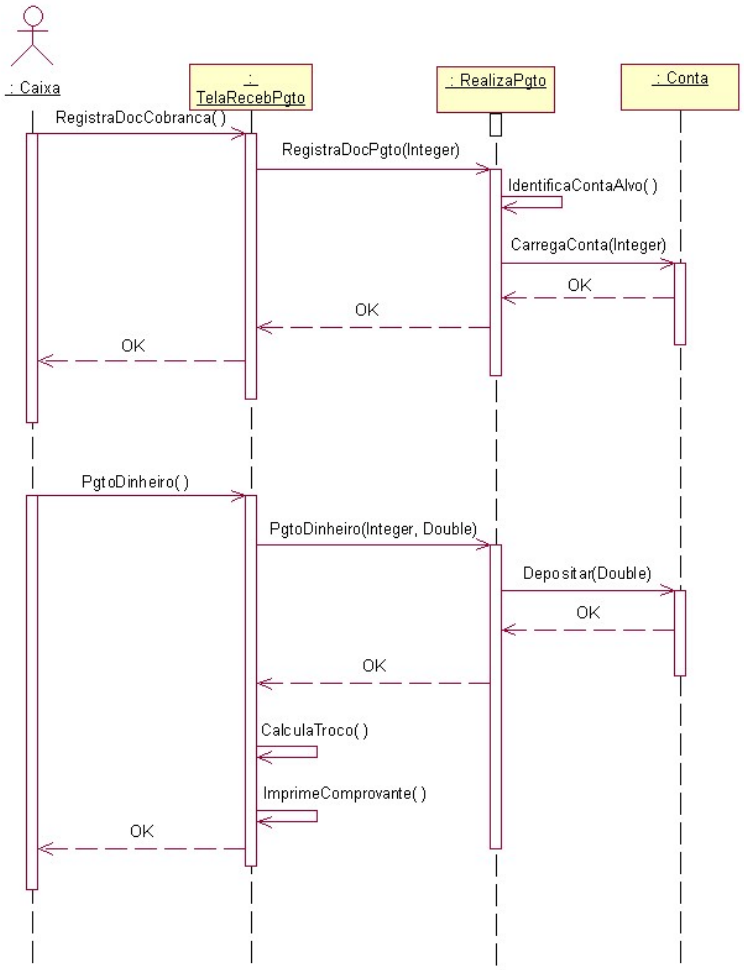


Figura 3.11: Diagrama de Seqüência básica Caso de Uso UC5.2 - Receber Pagamento em Dinheiro

Para determinar as chances de acesso a um módulo, assim como no método de conversão de UML para QN [4], faz-se necessária a adição de probabilidades ao diagrama de casos de uso para descrever o comportamento mais provável dos usuários diante do sistema.

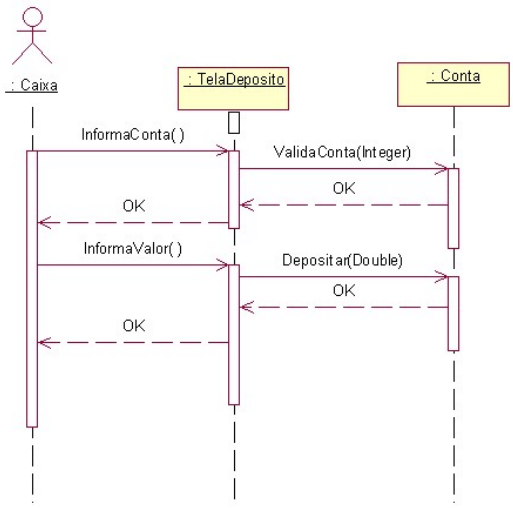


Figura 3.12: Diagrama de Seqüência básica Caso de Uso UC6 - Depositar Dinheiro

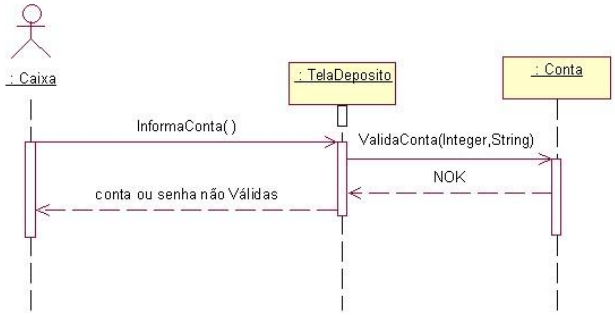


Figura 3.13: Diagrama de Seqüência básica Caso de Uso UC6 - Depositar Dinheiro

Figura 3.14: Diagrama de Implantação do Sistema Bancário

Na metodologia apresentada por Cortellessa e Mirandola eram descritos dois conjuntos de probabilidades: o de probabilidades de usuários serem determinados atores; e o de atores acessarem casos de uso. Com estes conjuntos de probabilidades pode-se determinar a

probabilidade consolidada de cada caso de uso ser executado. Para isso, basta somar-se as multiplicações das probabilidades associadas a cada ator, que a acessa pelas probabilidades deste atores acessarem o caso de uso, conforme foi demonstrado no método de conversão de UML para QN [4].

Este método, porém, não se faz menção ao tratamento das relações entre casos de uso (inclusão, generalização e extensão) e nem a generalização de atores. Com isso, a metodologia acaba restringindo certos casos muito comuns. A fim de eli-minar algumas destas restrições, e assim aumentar a abrangência da técnica aqui apresentada, será demonstrado, além da conversão de UML para SAN, o tratamento da inclusão, da extensão e da generalização de casos de uso. A generalização de atores não será tratada ficando assim como um trabalho futuro, que será listado nas considerações finais.

A relação inclusão entre casos de uso se trata de uma relação, onde os fluxos do caso de uso incluso são adicionados aos fluxos do caso de uso que o inclui [13], utilizada para fins de reutilização de fluxos de casos de uso. Neste tipo de relação, não existe qualquer condição, para que as seqüências inclusas sejam executadas, o que faz com que os fluxos incluídos sejam executados, tantas vezes quanto o caso de uso base executar. Portanto, não se faz necessária a adição de uma nova probabilidade para relações de inclusão, já que estas são executadas juntamente com o caso de uso base.

No caso da extensão, a funcionalidade estendida só será executada caso uma condição, associada a mesma, seja verdadeira [13]. Com isso, entende-se que haverá uma probabilidade desta condição ser verdadeira. Conseqüentemente, existirá, também, uma probabilidade complementar, desta condição ser falsa indicando assim, a não execução da funcionalidade contida no caso de uso estendido. Assim, diferentemente da inclusão, na relação de extensão faz-se necessária a adição de probabilidades de execução diferente do caso de uso base. Logo, existirá mais um conjunto de probabilidades que deve ser adicionado ao diagrama de casos de uso: as probabilidades das extensões serem executadas.

Um caso de uso estendido pode também possuir outras ligações com outros casos de uso ou mesmo com atores. Neste caso, a probabilidade consolidada vista no método de conversão de UML para QN e aqui também utilizada não é afetada pela probabilidade da condição. Isso se deve, por que a condição está associada a apenas uma relação de extensão, ou seja, determinará quando os diagrama da seqüência do casos de uso estendido serão executados no caso de uso base. Assim, afeta apenas a execução das seqüências do caso de uso base.

A última relação entre casos de uso, a generalização ou especialização, é composta por condições de execução para cada caso de uso especialista [13]. Estas condições são com-

plementares, ou seja, uma funcionalidade de um especialista sempre será executada quando o caso de uso geral for acessado por um ator. Isso remete a adição de probabilidades de execução de cada caso de uso especialista e ainda, as somas das probabilidades de todos os especialistas deverão ser igual a 1, ou seja devem estar normalizadas. Obtendo assim um quarto, e último, conjunto de probabilidades relacionadas as condições das generalizações, que devem ser definidos pelo analista.

No exemplo do sistema bancário, o diagrama de casos de uso com as probabilidades está representado na Figura 3.15. Note que as regras de somas descritas no método de conversão de UML para QN estão sendo seguidas com exceção da probabilidade 0,15 associada à condição da extensão do caso de uso UC3. Existe uma probabilidade implícita e igual a 0,85 da condição [Valor>=1000] ser falsa e assim a funcionalidade da extensão não ser executada no caso de uso base. Esta probabilidade será utilizada na adição das probabilidades aos nodos de *branch* que será vista nas próximas seções.

3.4 Obtenção das Linhas de execução

Este é o segundo passo do método de conversão de UML para QN apresentado por Cortellessa e Mirandola [4]. Neste, é criada a estrutura do EG[15], o qual representa todos os cenários do sistema, obtidos dos diagramas de seqüência. No método de conversão de UML para QN, este EG é utilizado segundo a metodologia SPE [14]. Porém, na técnica de conversão de UML para SAN, aqui apresentada, esta metodologia não será utilizada, mas o EG servirá de base para a criação de autômatos pertencentes a SAN.

O algoritmo apresentado por Cortellessa e Mirandola não será alterado para incluir os casos de inclusão, extensão e generalização. Para isso, estes casos de-verão ser modelados, conforme descrito na seção de restrições. Este algoritmo está descrito no apêndice A. Neste algoritmo, cada interação de cada diagrama de seqüência é identificada como uma tupla (l,A1,A2) onde l é o método chamado ou a resposta, A1 é a classe ou objeto de origem da chamada e A2 é a classe ou objeto alvo.

As seqüências de casos de uso incluídos deverão ser incluídas nos diagramas de seqüência dos casos de uso base. Conseqüentemente, serão traduzidos em nodos do EG, junto com os diagramas de seqüência dos casos de uso base.

Conforme descrito na seção de restrições, os diagramas de seqüência dos casos de uso estendidos deverão incorporar todo o cenário, onde estes se inserem, ou seja, todas as interações do casos de uso base serão inclusas nos diagramas de seqüência do caso de uso estendido.

Com isso, os diagramas de seqüência do casos de uso estendido conterão chamadas em comum com os diagramas de seqüência do nodo base. Como cada interação só é traduzida um vez, conforme o algoritmo de Cortellessa e Mirandola, isso produzirá um caminho em comum na tradução dos diagramas de seqüência do caso de uso base, com o caso de uso estendido, com um nodo *branch*, ou de seleção, no ponto de extensão. Note que, ao final da extensão as interações também serão comuns fazendo com que o último nodo básico gerado pela extensão seja ligado, novamente, ao nodo imediatamente posterior ao ponto de extensão do caso de uso base.

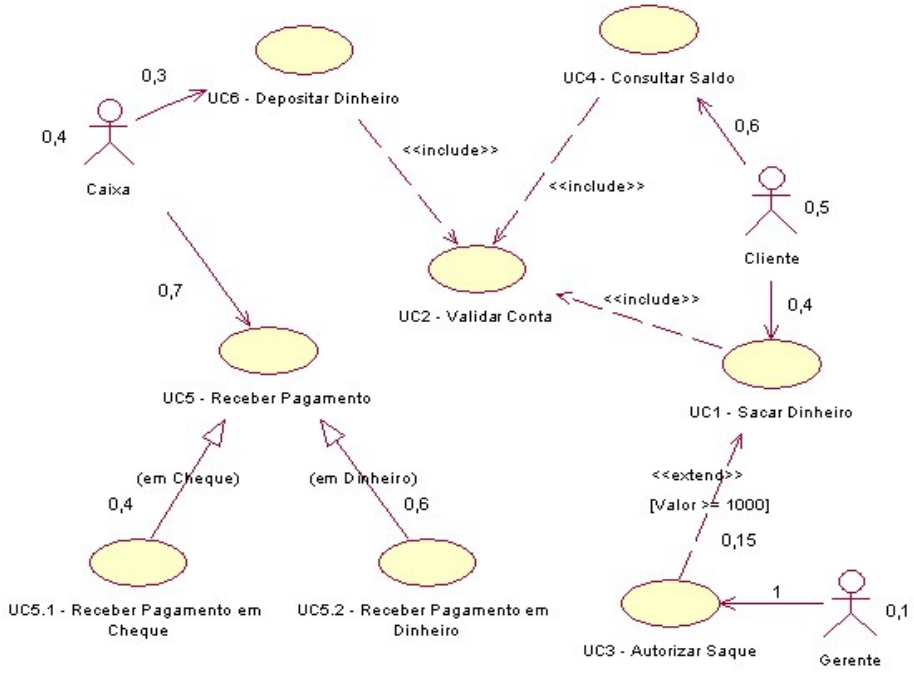


Figura 3.15: Diagrama de Casos de Uso Sistema Bancário com Probabilidades

O EG equivalente aos diagrams do caso de uso UC1 e aos diagramas do caso de uso estendido UC3 são demonstrados na Figura 3.16 [3]. Note que, o as interações dos diagramas de seqüência que são comuns e foram adicionados nos diagramas de seqüência do caso de uso estendido foram traduzidos em nodos básicos formando um caminho em comum que foi destacado na Figura 3.16.

[3]Para demonstrar a estrutura do EG gerada pelo algoritmo serão utilizadas as abreviaturas para atores, classes, métodos e respostas indicadas nas tabelas B.1, B.2, B.3 no apêndice B

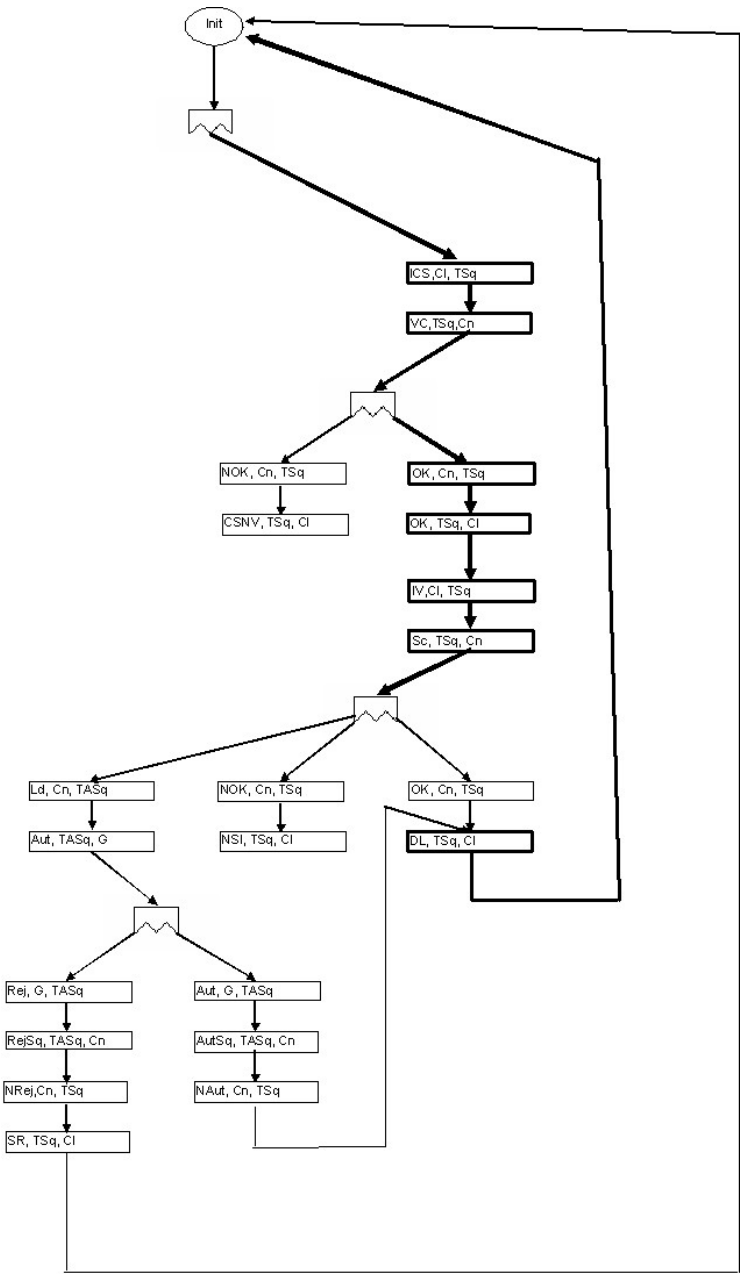


Figura 3.16: Estrutura do EG gerado Os diagramas dos casos de uso UC1 e UC3 demonstrando o fluxo comum.

Na generalização, assim como a extensão, os diagramas de seqüência deverão conter as interações dos diagramas de seqüência do caso de uso geral. Isso também produzirá caminhos comuns entre os casos de uso especialistas e o geral com nodos *branch* no EG, onde haverão opções de caminho para cada especialista. Na Figura 3.17 é demonstrado a estrutura do EG

gerado para os diagramas de seqüência dos casos de uso UC5.1 e UC5.2.

No exemplo do sistema bancário a estrutura do EG gerado por este algoritmo será o apresentado na Figura 3.18.

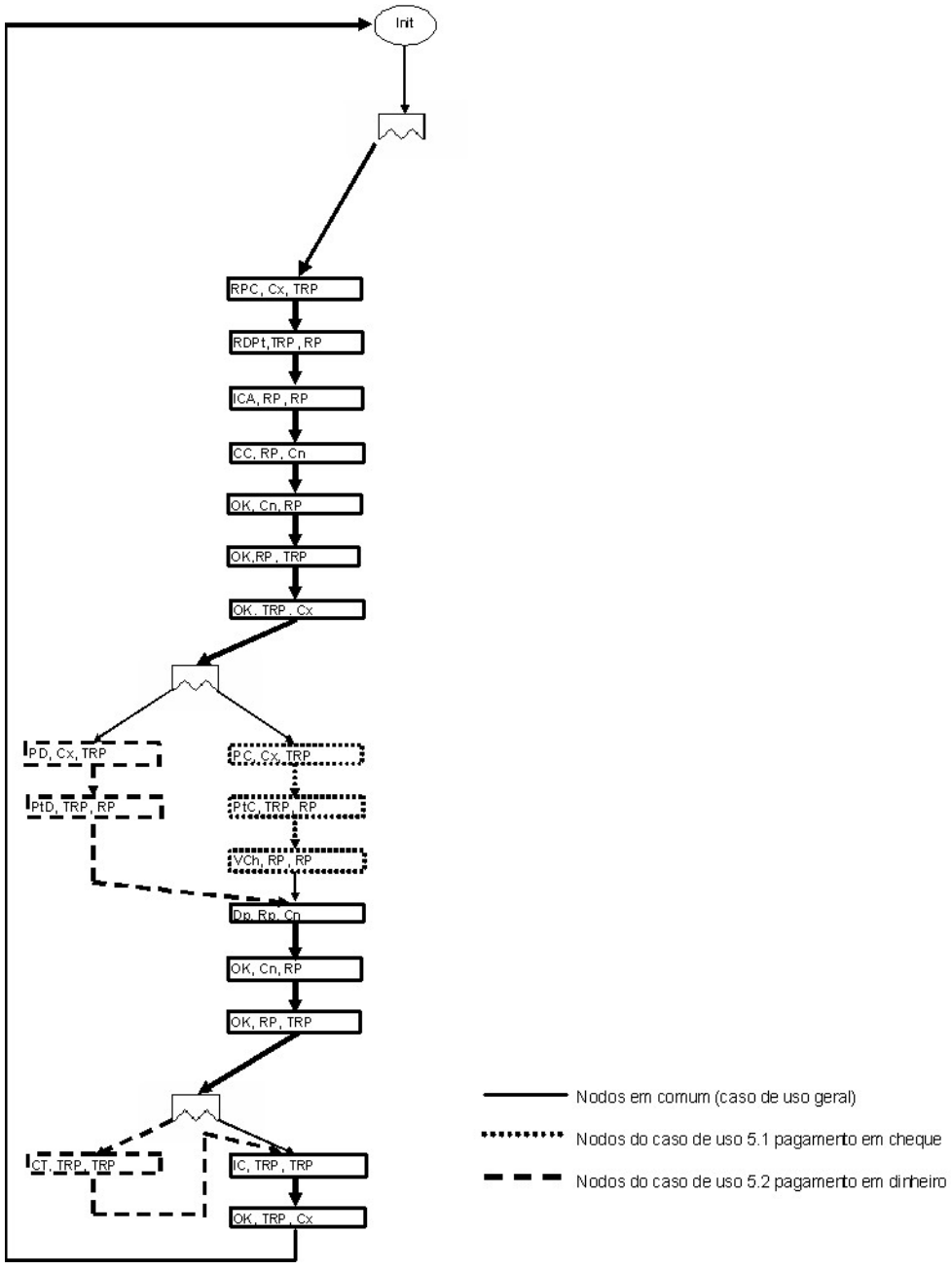


Figura 3.17: Estrutura do EG gerado Os diagramas dos casos de uso UC1 e UC3 demonstrando o fluxo comum.

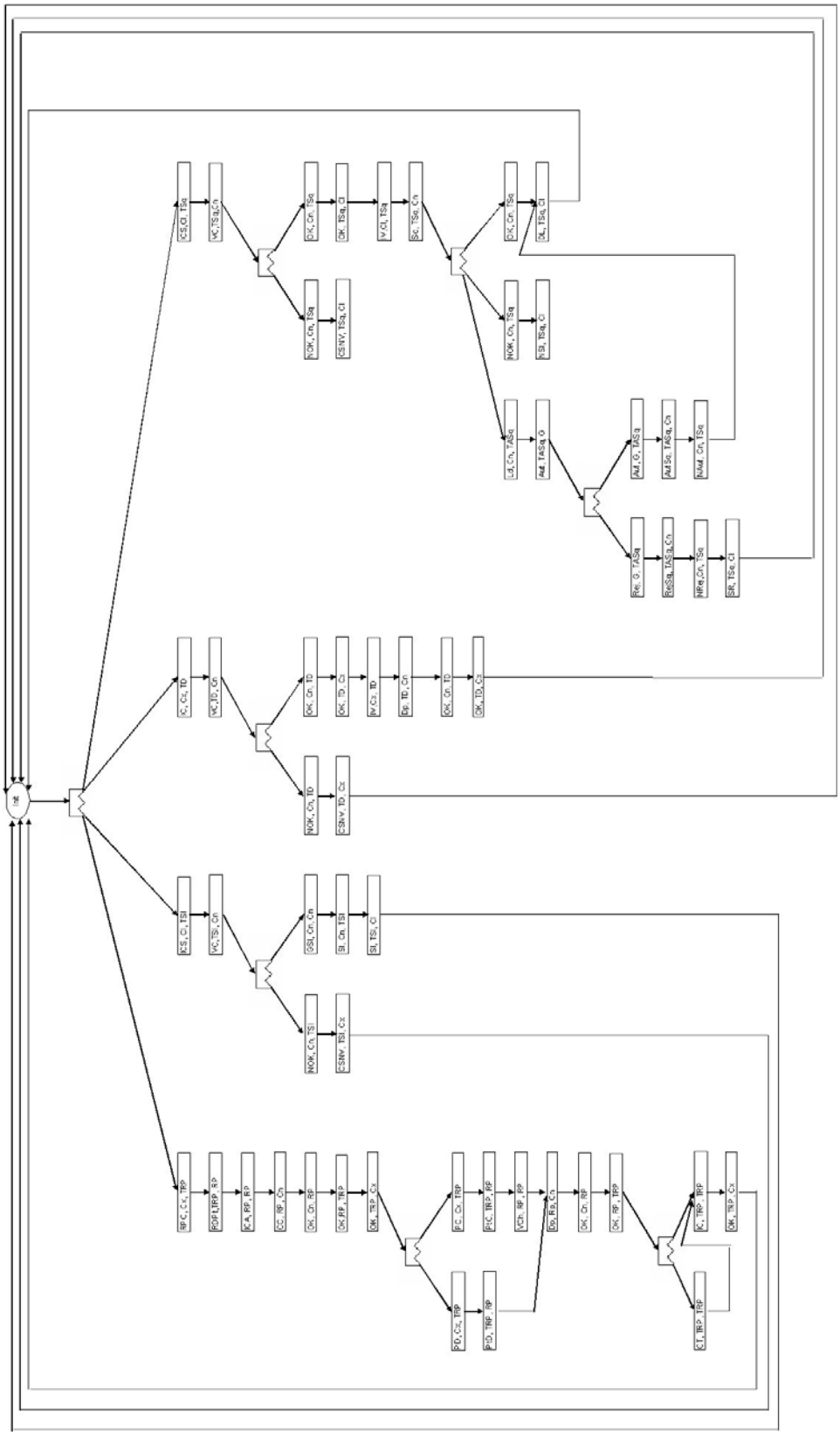


Figura 3.18: Estrutura do EG gerado para o exemplo do Sistema Bancário

3.5 Adição das Probabilidades nos Nodos *Branch*

As probabilidades inseridas no diagrama de caso de uso, no primeiro passo desta metodologia, serão utilizadas para determinar os caminhos mais prováveis no *software*. Como já mencionado, todos os caminhos do sistema estão representados no EG montado no passo anterior.

No EG, os diferentes caminhos podem ser facilmente identificados nos nodos de *branch*. Serão adicionadas, aos ramos destes nodos, probabilidades referentes a cada caminho que estes representam. Porém, assim como no método de Cortellessa e Mirandola, as probabilidades de cada caso de uso não são suficientes [4] para obter-se as probabilidades de cada caminho do EG. Necessita-se de probabilidades de cada diagrama de seqüencia de um caso de uso ser executado, que devem ser informadas pelo analista. Se estas probabilidade forem multiplicadas pelas probabilidades consolidadas dos casos de uso, geram a probabilidade consolidada de cada diagrama de seqüência ser executado[4], exceto nos diagramas de seqüência de casos de uso estendidos e de generalizações, ou quais serão demonstrados a seguir.

No caso dos diagramas de extensão e generalização as probabilidades conso-lidadas de cada caso de uso será defina pela multiplicação da probabilidade consolidada do caso de uso base, da extensão ou generalização, pela probabilidade da condição de extensão ou especialização ser verdadeira. Este resultado deve ser multiplicado pela probabilidade de cada diagrama de seqüência ser executado. Ainda, como nas restrições, os diagramas de seqüência da extensão contêm todas as chamadas, entende-se que o caso de uso base desta extensão será executado apenas quando a extensão não ocorrer. Com isso, as probabilidades dos diagramas de seqüência do caso de uso base devem ser revistas multiplicando as probabilidades de seus diagramas de sequência pela probabilidade da condição da extensão ser falsa [4].

Após definidas todas as probabilidades consolidadas de cada diagrama de seqüência, pode-se, então, iniciar a adição das probabilidades aos nodos de *branch* do EG. A adição deve começar do nodo Init seguindo para o próximo nodo *branch* e deste para os próximos, em seus ramos até que todos os nodos *branch* contenham probablidades. Para cada ramo do nodo *branch* deverão ser analisados que diagramas de seqüência formam este ramo, e deve-se somar as probabilidades dos diagramas de seqüência que o compõem.

Segue-se estes passos até que todos os ramos de todos os nodos *branch* estejam preenchidos. Após isso, deve-se aplicar uma normalização em cada nodo *branch* para que o contenham, então, probabilidades, ou seja, o somatório dos ramos que partem do nodo *branch* seja igual

[4] O calculo da probabilidade da condição da extensão ser falsa consiste da seguinte subtração: 1 - probabilidade da condição ser verdadeira

a 1.

No exemplo do sistema bancário, com as probabilidades de cada diagrama de seqüência e o cálculo da probabilidade consolidada indicados na Tabela 3.1, a estrutura do EG com as probabilidades consolidadas não normalizadas é demonstrada na Figura 3.19. Na figura, cada nodo Branch deve ser normalizado, para isso basta dividir o número de cada ramo do *branch* pelo somatório dos números de todos os ramos deste mesmo nodo *branch* obtendo assim probabilidades.

Tabela 3.1: Probabilidades dos diagramas de seqüência

UC	Diagrama	Prob Diag de Seq	Cálculo da Prob
UC1	Básica	0,80	0,5 * 0,4 * 0,80 * 0,85 = 0,136
UC1	Alternativa 1	0,05	0,5 * 0,4 * 0,05 * 0,85 = 0,0085
UC1	Alternativa 2	0,15	0,5 * 0,4 * 0,15 * 0,85 = 0,0255
UC3	Básica	0,90	0,1* 1 * 0,90 + 0,5 * 0,4 * 0,15 * 0,90 = 0,117
UC3	Alternativa	0,10	0,1* 1 * 0,10 + 0,5 * 0,4 * 0,15 * 0,10 = 0,013
UC4	Básica	0,90	0,5 * 0,6 * 0,90 = 0,27
UC4	Alternativa	0,10	0,5 * 0,6 * 0,10 = 0,03
UC5.1	Básica	1	0,4 * 0,7 * 0,4 * 1 = 0,112
UC5.2	Básica	1	0,4 * 0,7 * 0,6 * 1 = 0,168
UC6	Básica	0,90	0,4 * 0,3 * 0,90 = 0,108
UC6	Alternativa	0,10	0,4 * 0,3 * 0,10 = 0,012

3.6 Identificação dos recursos do sistema

Neste passo, serão identificados os recursos computacionais do sistema(Processadores, rede, etc). Estes serão uma parte muito importante na avaliação do desempenho do *software*, já que podem produzir tempo de espera devido a uma super alocação, ou podem indicar uma sub utilização de recursos, ou ainda, podem indicar um desbalanceamento dos recursos de *hardware*.

Em UML, os recursos estão representados no diagrama de implantação em termos de nós e ligações entre eles. Para transformar estes recursos em autômatos e representá-los em SAN é necessário identificar os possíveis estados dos mesmos. Tipicamente, um nó possui um processador e uma memória [3] e seus estados podem ser ocupado ou livre, onde no primeiro

não aceita novos processos, no segundo estado aceita o processamento de novos processos. Assumindo-se que a ocupação do processador representa a ocupação do nó, então tem-se apenas dois estados por nó.

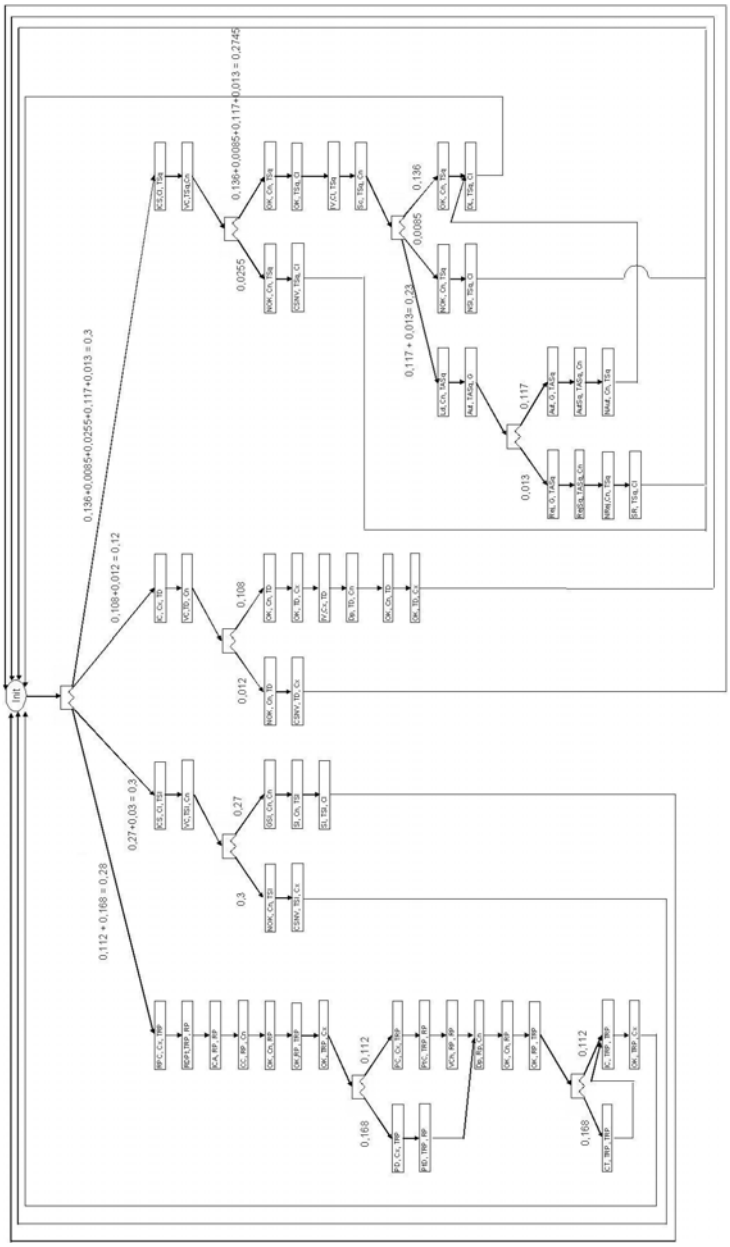


Figura 3.19: Estrutura do EG com probabilidades não normalizadas.

Com isso, tem-se os estados dos autômatos que representam os recursos, faltam, portanto, as transições de estados possíveis e eventos e taxas. As transições para estes dois estados

são de livre para ocupado e de ocupado para livre. Outras transições como livre para livre e ocupado para ocupado não fazem sentido aqui, pois uma vez ocupado o recurso, este pode apenas ser liberado e vice-versa.

Os eventos e taxas de ocupação e desocupação dos recursos, não dependem deles próprios, mas do *software* que é executado nestes recursos, já que os diferentes algoritmos ou tarefas levam diferentes tempos para serem executados. Os eventos e as taxas serão demarcados junto à definição dos autômatos que representam o *software*, que será visto nas próximas seções.

No exemplo do banco, onde existem um terminal e um servidor, serão gerados dois autômatos com dois estados cada um. Na Figura 3.20 são demonstrados estes autômatos.

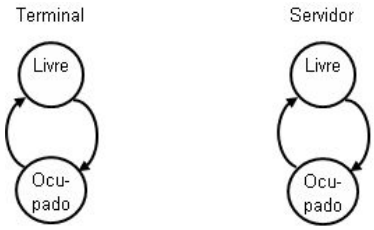


Figura 3.20: Recursos do Sistema em termos de autômatos.

Após converter todos os recursos (nós) do diagrama de implantação em autômatos de dois estados está pronta a conversão. As ligações entre estes recursos (rede) não serão representadas como autômatos, mas como eventos sincronizantes entre os recursos e as linhas de execução e sua latência será incluída em estados nos autômatos que representarão o *software*, como será visto nas próximas seções.

3.7 Adição dos Tempos de Processamento ao EG

Após criada a estrutura do EG e identificados quais os recursos estão disponíveis ao sistema, irá-se adicionar dos valores de tempo de execução nos nodos básicos do EG. Os valores de tempo são os tempos médios de execução do método representados pelo nodo básico do EG no recurso disponível ao sistema, ao qual o componente a que pertence a classe alvo está implantado. Esta informação é obtida através do diagrama de implantação do modelo UML. Este tempo diz respeito ao processamento executado por algoritmos dentro deste método, desconsiderando-se os tempos gastos por algoritmos em outros métodos chamados por este,

já que os tempos de processamento de métodos por ele chamados serão considerados nos nodos básicos seguintes a este.

A coleta dos tempos de processamento deve-se dar da seguinte forma: para cada nodo básico deve existir um vetor, onde cada posição deste vetor representa o tempo de processamento em um recurso do sistema. O tamanho deste vetor é igual ao número de recursos do sistema (nós), identificados no quarto passo desta técnica, mais duas posições. As duas posições extras do vetor representam o tempo de pensamento do usuário (*thinking time*) e a latência da rede.

Este vetor deve ser preenchido sempre com valores positivos e diferentes de zero quando a chamada de método em questão utilizar o recurso, ao qual a posição do vetor representa, e valores nulos ou iguais a zero para identificar os recursos que não serão utilizados na chamada representada pelo nodo básico. A exceção a esta regras são os nodos básicos onde a chamada é disparada pelo usuário. Nestes casos, irão existir, além dos tempos dos recursos, o tempo de pensamento do usuário (*thinking time*) que é o tempo de reação do usuário ao sistema. Como já mencionado este tempo deve ser indicado em uma das duas posições extras, inseridas no vetor, a outra posição é referente ao atraso introduzido pela latência da rede que será tratada nas próximas seções.

Após criados e preenchidos todos os vetores para todos os nodos básicos do EG, deve-se, então, observar o nodo Init. Diferentemente dos nodos básicos, este nodo conterá não um tempo de processamento em um ou mais recursos, mas a taxa de chegada de usuários. Neste nodo, não será necessária a criação de um vetor, mas apenas indicar o tempo médio com que os usuários chegam ao sistema.

No exemplo do sistema bancário, supõem-se que cheguem usuários a cada mi-nuto e que um usuário leva em média 5 segundos para reagir a uma mensagem na tela. Com isso, e a análise de tempo de processamento de cada chamada do projetista, foi gerado o EG com os tempos em milisegundos demonstrados na Figura 3.21. Para melhor visualização, foram demonstrados nesta figura apenas os nodos básicos do casos de uso UC6. Os demais casos de uso seguirão a mesma abordagem.

3.8 Criação de Pacotes de Usuário

Em SAN, diferentemente de QN, não existe o conceito de clientes trafegando através de uma rede. Assim, a representação da quantidade de usuários no sistema em SAN não é tão claro e direta. Em SAN, este conceito pode ser representado de diversas formas: por um autômato

que representa sua quantidade e taxas funcionais, pela replicação de autômatos (um para cada cliente), agrupando clientes e calculando as novas taxas em um único autômato, entre outras.

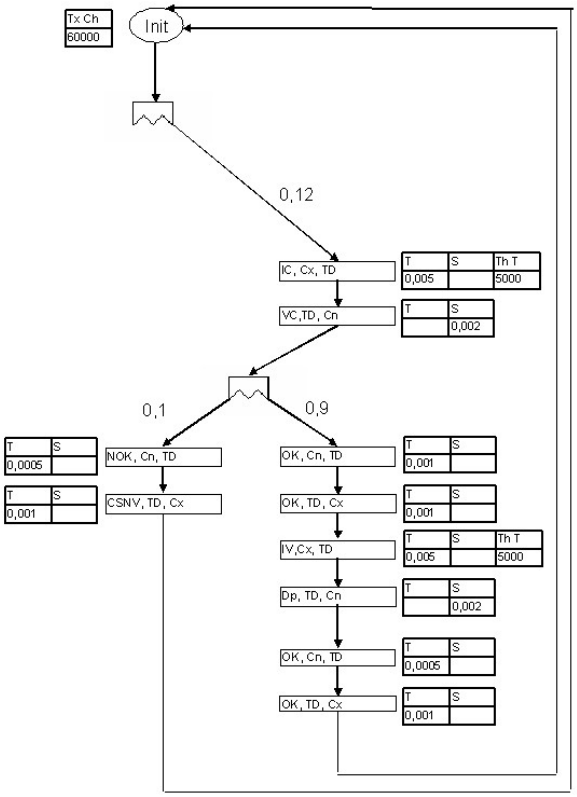


Figura 3.21: Estrutura do EG com probabilidades normalizadas e o tempo de processamento para os nodos básicos dos diagramas de seqüência do caso de uso UC6.

Nesta técnica, onde o EG se transformará em um autômato, se for utilizada a forma de um autômato que representa a quantidade de usuários no sistema, teria-se um limitação no número máximo de usuários. Além disso, não representaria a disputa por recursos do sistema, pois haveria apenas um autômato gerado a partir do EG, o qual seria executado em seqüência evitando esta disputa.

Uma segunda alternativa viável ao problema seria com replicações dos autômatos gerados pelo EG a nível de usuário. Isso causaria uma disputa a nível de usuário individualmente pelos recursos, porém causaria uma limitação na quantidade de usuários (autômatos), pois o autômato gerado pelo EG possui muitos estados e causando a explosão de espaço de estados, o que pode inviabilizar a resolução do SAN pela grande quantidade de estados globais que

seriam gerados.

A terceira alternativa seria agrupar usuários em um único autômato gerado pelo EG e recalcular as taxas de transição de estados de acordo com os usuários. Entretanto, isso geraria o mesmo problema da primeira forma apresentada, sem disputa por recursos.

Porém, se forem avaliadas a segunda e a terceira forma, pode-se montar uma nova forma de representar os usuários onde alguns autômatos seriam replicados e cada um destes representaria um determinado conjunto ou pacote de usuários. Assim a estrutura do EG será replicada em um número definido de pacotes. E cada pacote conterá um número de usuário que acessarão o sistema ao mesmo tempo. Esta forma de representação produz uma concorrência pelo recursos e uma representação de usuário sem a explosão de espaço de estados, tornando-se assim a melhor opção de representação para esta técnica.

Para replicação dos EG e agrupar os autômatos gerados, deve ser considerado que o tempo de execução indicado no passo Adição dos Tempos de Processamento ao EG de um nodo básico é o tempo para um usuário executar cada método, ignorando atrasos inseridos pela rede. O que remete a um cálculo bruto, onde se em um determinado momento o sistema tiver 100 usuários, o sistema ocupará o recurso por 100 vezes o tempo de processamento em cada nodo básico, já que os atrasos não são considerados neste tempo.

Com isso, se 100 usuários forem o número máximo de usuário concorrentes no sistema, poderá-se dividi-los em 5 pacotes de 20 usuários cada e assim replicar 5 EGs iguais com tempos dos nodos básicos multiplicados por 20. Dessa forma, estariam formados 5 pacotes representando o processamento de 20 usuários.

Quanto ao tempo de reação do usuário, entende-se que estes usuários estão utilizando o recurso ao mesmo tempo, e portanto cada um deles em média leva o mesmo tempo de pensamento individualmente. Com isso, reagem, em média, com o mesmo tempo e o tempo de pensamento é independente do número de usuários representados no EG.

A taxa de chegada de usuários associada ao nodo Init também sofrerá alterações devido a mudança na quantidade de usuário que chega ao mesmo tempo. No caso de pacotes com 20 usuários deve-se recalculá-la para representar a taxa de chegada de 20 usuários ao mesmo tempo.

No exemplo do sistema bancário, suponha que em média chegam 2,5 usuário ao mesmo tempo com um máximo de 5 usuários. Isto geraria dois pacotes de 2,5 usuários e mais uma réplica do EG com os tempos de processamento multiplicados por 2,5. Os tempo de processamento do EG original, também, serão multiplicados por 2,5. Na Figura 3.22 pode ser observado que os tempos de processamento dos nodos básicos dos diagramas de seqüência

do caso de uso UC6 foram multiplicados por 2,5 formando um ramo de um dos dois EGs destes exemplo. Note que os tempos gastos pelos usuários não foram multiplicados.

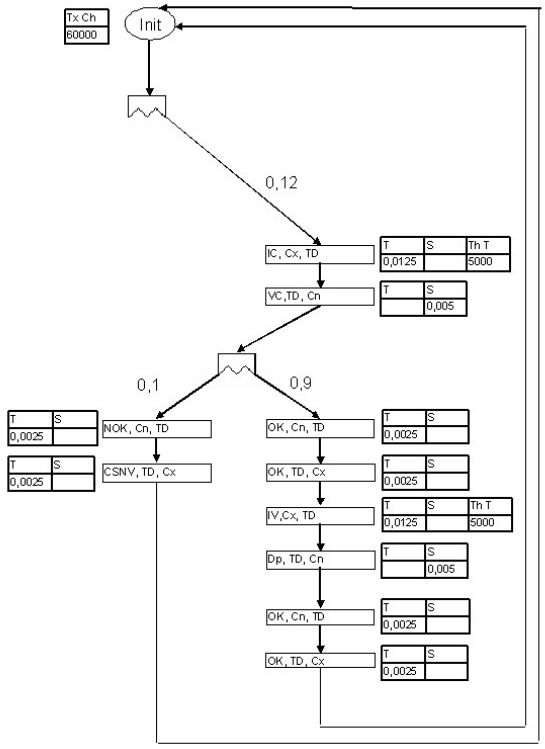


Figura 3.22: Um Ramo de um EG com a multiplicação dos tempo por 2,5.

3.9 Adição da Latência da Rede

Com os EGs prontos e representando grupos de usuários, pode-se então adicionar o atraso introduzido pela latência da rede e de recurso que tragam atraso as chamadas de métodos. A latência será representada por um tempo médio de atraso inserido pela rede na forma de um elemento no vetor de processamento dos nodos básicos do EG.

Pela latência representar um atraso, assim como o tempo de pensamento do usuário, este tempo não deve ser multiplicado pelo número de usuários de cada pacotes, pois entende-se que a largura de banda é suficiente para suportar o número máximo de usuários e portanto, o atraso não é proporcional ao número de usuários, e sim, uma média de tempo em que os dados ou chamadas levam para atravessar a rede. Em outras palavras, não estará representada disputa pela rede.

Para adicionar a latência nos tempos de processamento dos EGs deve-se observar o diagrama de implantação do UML, onde os componentes e nós se relacionam. A latência deve ser introduzida quando houver uma chamada a um método de um componente localizado em um nó diferente do componente chamador. Neste caso, o tempo da latência da rede ou do recurso de comunicação deve ser introduzido na posição da latência no vetor do nodo básico, que contém a chamada ao método remoto.

Assim, este processo deve ser realizado em todas as réplicas do EGs. Ao fim o EG conterá todos os tempos de processamento e atrasos necessários para representar o *software*.

No sistema bancário, existe uma rede entre o nó Terminal e o Servidor. Supondo que esta rede produz em média um atraso de 50 milisegundos nas chamadas do componente InterfaceUsuario para o componente ControladorBanco e em suas res-postas. Portanto, devem ser adicionados 50 milisegundos para cada vetor dos nodos básicos onde há uma chamada feita por uma classe do componente InterfaceUsuario a um método em uma classe no componente ControladorBanco ou vice-versa. Na Figura 3.23, são demonstrados os nodos básicos do EG para o caso de uso UC6 com a latência.

3.10 Simplificação do EG

Como mencionado anteriormente, diferentemente da metodologia de Cortellessa e Mirandola [4], o EG será utilizado para a criação de um autômato e não para a execução da metodologia SPE[15].

Para a criação de um autômato com menos estados e com a mesma representação, evitando assim a explosão de espaço de estados, faz-se necessária a simplificação dos EGs. Esta simplificação se dá através da diminuição do número total de nodos básicos de cada EG, os quais serão transformados em estados na conversão dos EGs para autômatos, vista a seguir.

Esta simplificação é feita de forma a agrupar os nodos básicos adjacentes com a utilização dos mesmos recursos, e sem um tempo de pensamento do usuário, pois não serão adicionados atrasos entre estes nodos e todos sincronizarão com os mesmo autômatos de recursos. Note que nodos com latência de rede podem ser agrupados com nodos sem latência, pois a latência é introduzida antes do início do processamento da chamada e as chamadas no mesmo recurso não introduzem atrasos referentes à rede.

Esta simplificação é feita de forma a agrupar os nodos básicos adjacentes com a utilização dos mesmos recursos, e sem um tempo de pensamento do usuário, pois não serão adicionados atrasos entre estes nodos e todos sincronizarão com os mesmo autômatos de recursos. Note

que nodos com latência de rede podem ser agrupados com nodos sem latência, pois a latência é introduzida antes do início do processamento da chamada e as chamadas no mesmo recurso não introduzem atrasos referentes à rede.

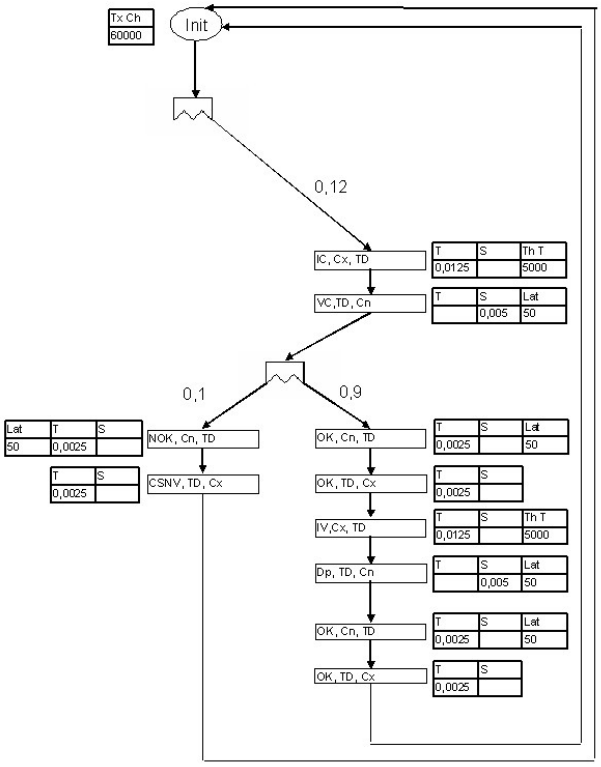


Figura 3.23: Estrutura do EG com Latência.

Os nodos adjacentes com a utilização de um mesmo recurso são agrupados em um único nodo básico, que é nomeado com todas as tuplas que o compõem, e os tempos de processamentos de cada recurso são somados formando um novo vetor de recursos.

Na Figura 3.24 é demonstrado o resultado da simplificação aplicada ao exemplo do sistema bancário nos nodos básicos dos diagramas de seqüência do caso de uso UC6.

Ao final desta operação, os EGs estarão com menos nodos básicos e gerarão, assim, autômatos com menos estados, mas com a mesma representação de disputa e ocupação de recursos. Esta manobra é, simplesmente, para diminuição do número de estados globais da SAN gerada por essa técnica, a qual tente a viabilizar um número maior de casos que podem ser resolvidos.

3.11 Conversão do EG em Autômatos

Terminada a simplificação, os estados dos autômatos estocásticos ficam visíveis no EG, como sendo os nodos básicos restantes e o nodo Init e as transições como as ligações entre os nodos básicos. Porém, existem alguns nodos básicos que terão um tratamento especial, e portanto poderão gerar mais de um estado. Estes são os nodos com tempo de pensamento e atraso introduzido pela rede, além dos demais nodos: nodos de laço, nodos de branch e os nodos de fork e join.[5]

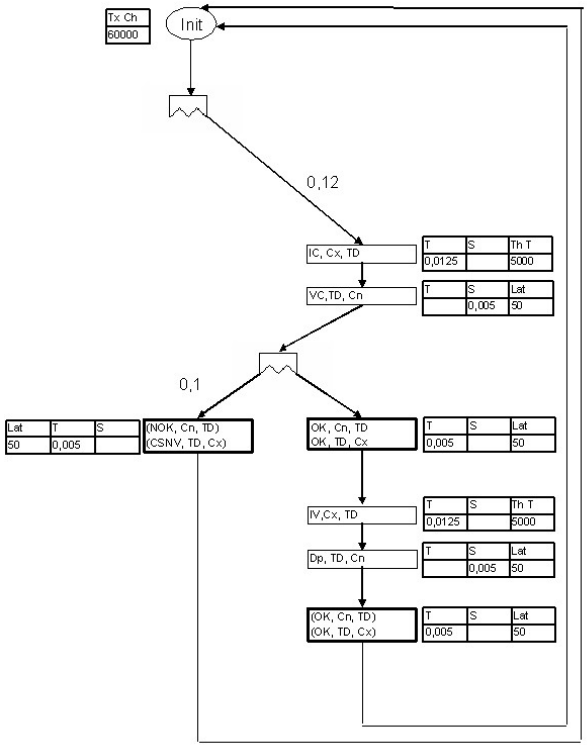


Figura 3.24: EG Simplificado.

Cada ligação do EG será transformada em uma transição de um autômato, e conterá um evento que significa o final do processamento de uma chamada e início de outro.

Quanto as taxas, se for considerado que uma transição de um nodo básico para outro depende do término de seu processamento, pode-se dizer que a taxa de saída de um nodo básico é proporcional ao tempo de processamento deste nodo. Se consideramos que, quanto

[5] A partir dessa conversão os autômatos que foram gerados a partir do EG serão chamados de Autômatos do EG.

maior o tempo de processamento, menor será a taxa de saída deste estado, pode-se afirmar que a taxa de saída é inversamente proporcional ao tempo de processamento. Com isso, tem-se que:

$$\text{TaxaSaída} = \frac{1}{TempoProcessamento}$$

Esta taxa pode ser facilmente calculada para nodos básicos, onde não há atrasos e nenhum tempo de pensamento ou reação do usuário.

Os nodos básicos com um tempo de pensamento do usuário serão divididos em dois estados. O primeiro estado representará o estado de usuário pensando. A taxa do evento de saída deste estado será dependente do tempo de pensamento, ou seja a taxa de cada evento de saída será o inverso do tempo de pensamento do usuário. O segundo estado representará o processamento da chamada feita pelo usuário. As taxas de saída deste estado serão, portanto, o inverso do tempo gasto com o recurso utilizado. Na Figura 3.25, pode ser observada a transformação de um dos nodos básicos do caso de uso UC6 que está associado a um tempo de reação do usuário.

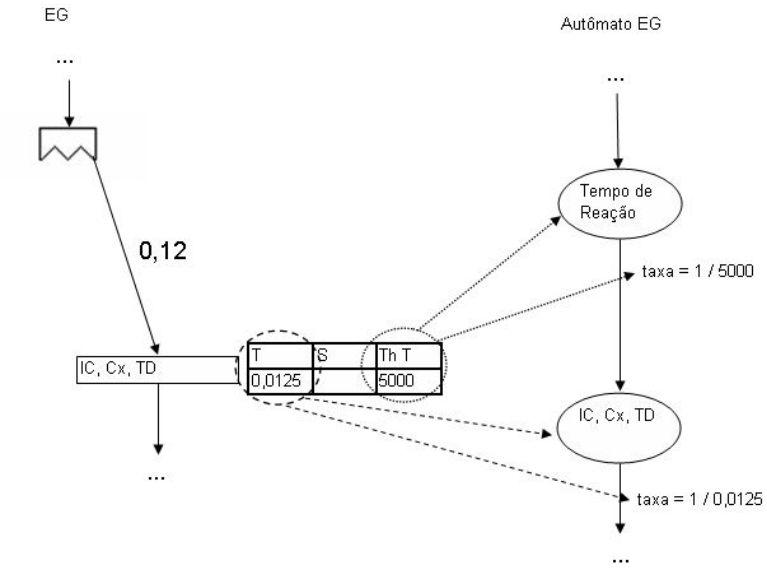


Figura 3.25: Conversão de nodos básicos com Tempo de Reação do Usuário

Os nodos básicos com valor de atraso provocado pela latência da rede são tratados da mesma forma, que os nodos com tempo de reação do usuário. O nodo é convertido em dois estados, o primeiro representando o atraso introduzido, onde os evento de saída terão uma taxa que será o inverso do atraso. O segundo estado representa a execução do método

chamado e a taxa dos eventos de saída será o inverso do tempo gasto, com o recurso que este utiliza.

Os nodos de laço podem ser simplesmente suprimidos e as conexões, que chegam a ele, serem adicionadas ao nodo imediatamente posterior a ele.

Os nodos *branch* serão suprimidos e seus ramos serão ligados ao estado que representa o nodo básico ou Init imediatamente anterior a este *branch*. No estado do nodo básico ou Init anterior continuará havendo apenas um evento de saída e a este devem ser aplicadas as probabilidades de cada ramo do *branch*, para decidir a transição a ser efetuada. Na figura 3.26 é demonstrado como o nodo *branch* é suprimido e suas probabilidades associadas ao evento de transição do nodo básico anterior.

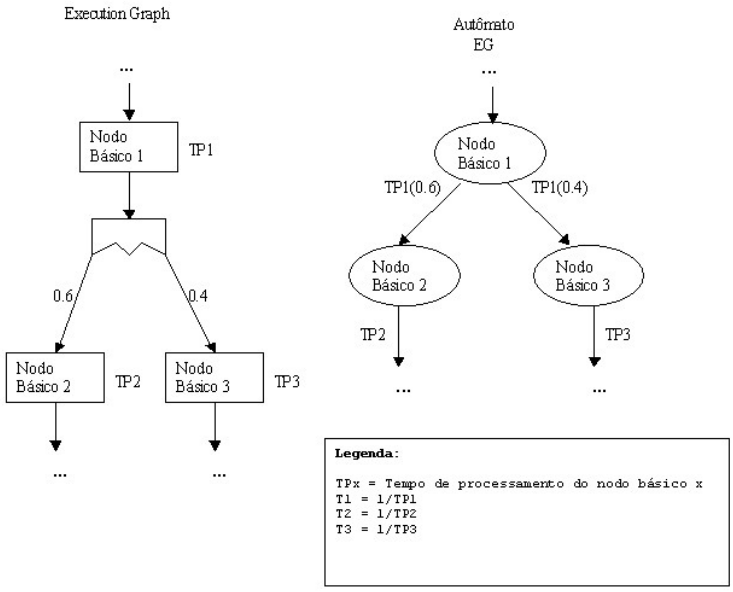


Figura 3.26: Conversão de Branch

Os nodos de fork e join, assim como cada ramo do fork, serão transformados em autômatos separados. Cada autômato será composto por um nodo de fork seguido das seqüências de nodos do ramo do fork terminado por um estado de join. Na Figura 3.27 estes autômatos estão representados como FJ1 e FJ2.

O autômato gerado pelo EG será composto pelos estados que representam o nodo imediatamente anterior ao fork e posterior ao join ligados diretamente. Os autômatos de Fork/Join serão iniciados pelo evento de saída do nodo anterior ao fork, ou seja, este evento sincronizará o autômato do EG e os autômatos deste Fork/Join. A sincronização do join dos autômatos

ocorre na taxa do estado que representa o nodo básico posterior ao join. Esta taxa é dependente dos autômatos de Fork/Join, quando todos estiverem no estado join o evento passa a ter uma taxa diferente de zero. Este evento também sincronizará o autômato EG com os autômatos de Fork/Join retornando ao estado de fork e assim preparando para uma nova execução do Fork. Na Figura 3.27 é demonstrada esta conversão. Mais detalhes sobre os eventos e taxas serão vistos a seguir.

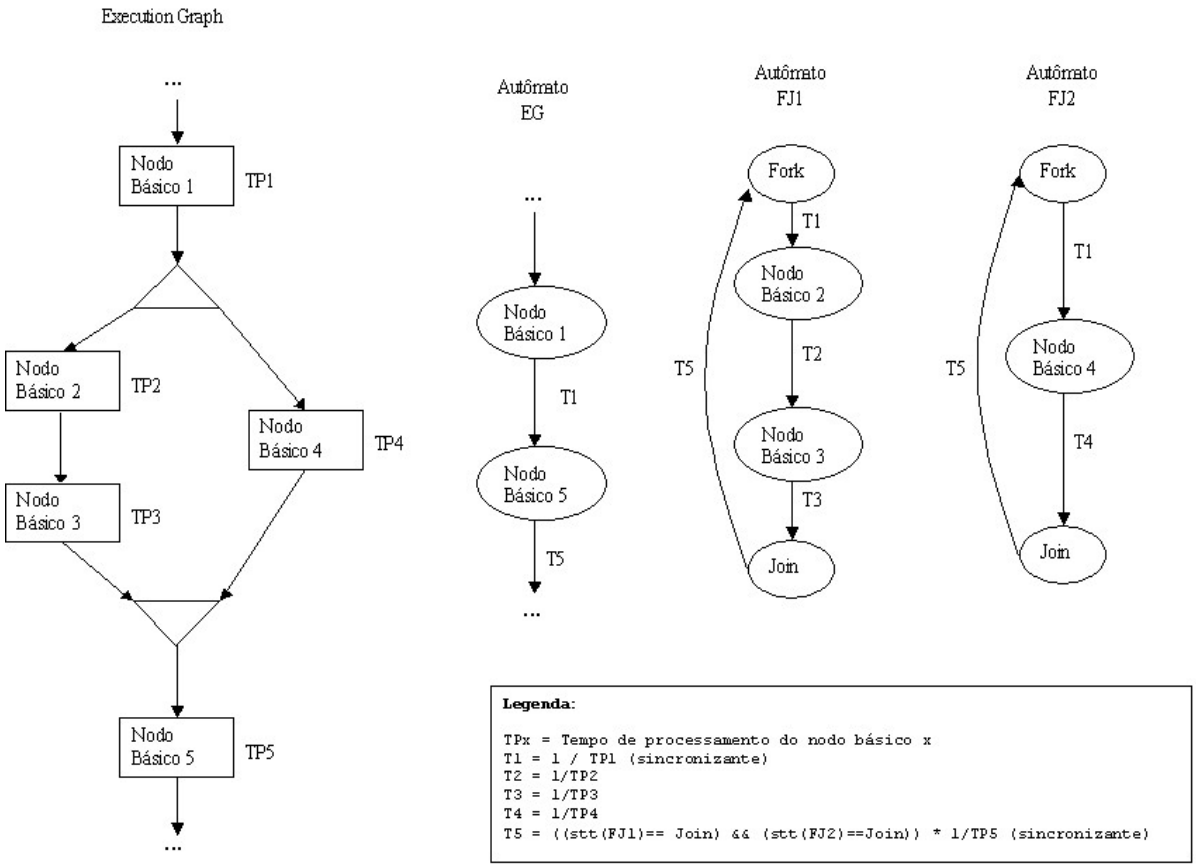


Figura 3.27: Conversão de Fork e Join

No exemplo EG dos diagramas de seqüência do caso de uso UC6 do sistema bancário, a conversão é demonstrada na Figura 3.28.

3.12 Consolidação e Resolução da SAN

Com os autômatos do EG e os autômatos dos recursos gerados têm-se, agora, que unificá-los em uma SAN. Para isso, deve-se acertar as sincronizações entre os processamentos contidos

nos estado que representam os nodos básicos com os recursos. As informações referentes a qual recurso é utilizado em cada nodo está no EG nos vetores de cada nodo básico.

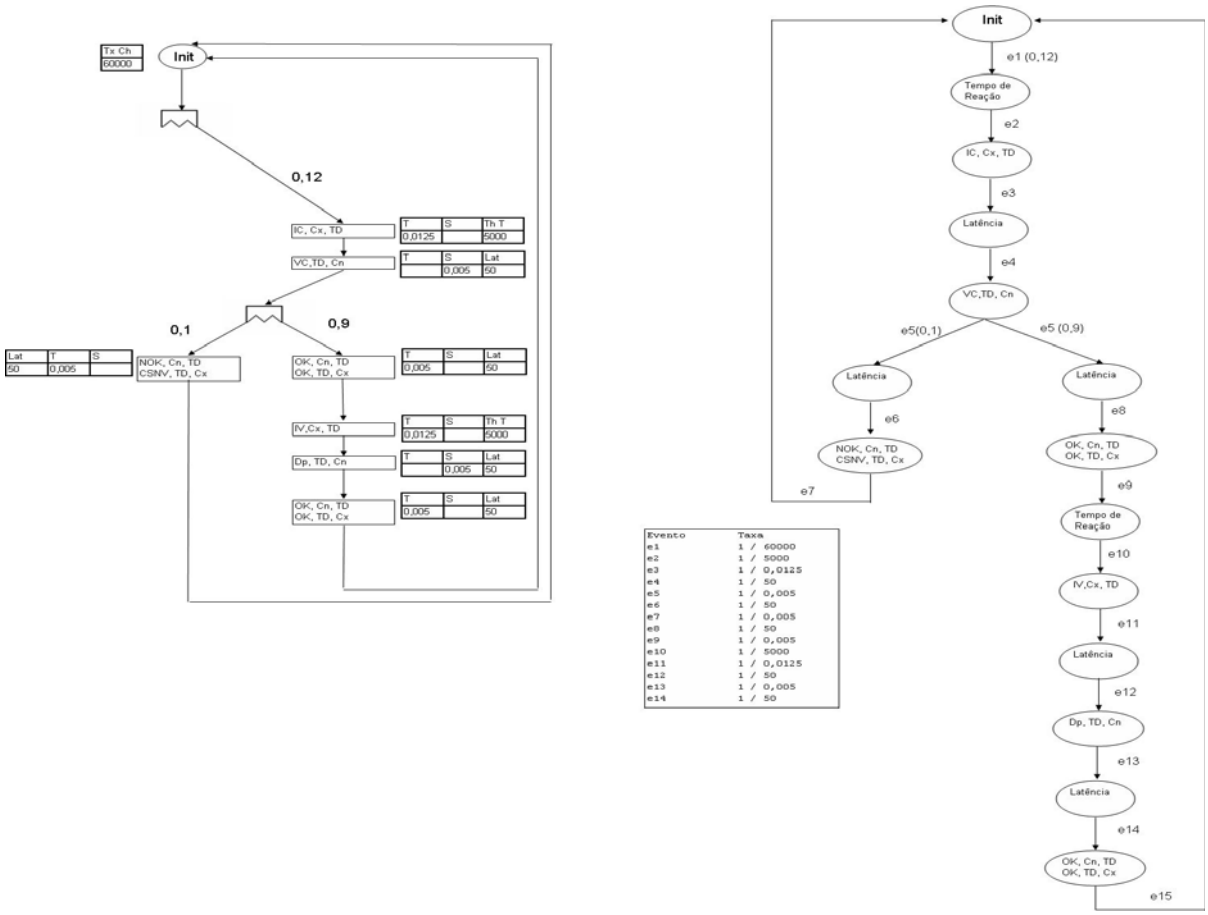


Figura 3.28: Conversão do EG em Autômato

Como os atrasos provocados pela rede e pelo usuário foram separados em estados distintos dos estados, que ocupam recursos computacionais, pode-se identificar, claramente, quais os estados que ocuparão recursos do sistema e quais os liberarão. A chegada aos estados que ocupam recursos indicará, além do início do processamento, a ocupação do recurso, onde as chamadas contidas neste estado serão processadas. Logo, os eventos de chegada a um estado que ocupará um recurso do sistema deverão sincronizar o autômato do EG com o autômato do recurso na transição do estado livre para o ocupado. Da mesma forma, os eventos de saída destes estado deverão sincronizar o autômato do EG com o autômato do recurso ocupado na transição de ocupado para livre, liberando assim, o recurso para ser utilizado por outras chamadas.

Os estados de atraso, (latência da rede ou tempo de pensamento do usuário) bem como, o estado que representa o nodo init não ocuparão recursos e portanto os eventos de chegada não sincronizarão com a utilização de um recurso nem os de saída, sincronizarão com a liberação de um recurso. No exemplo do sistema bancário, para o autômato gerado do EG dos diagramas casos de seqüência do caso de uso UC6, a sincronização entre o autômato e os recurso gera a SAN demonstrada na Figura 3.29.

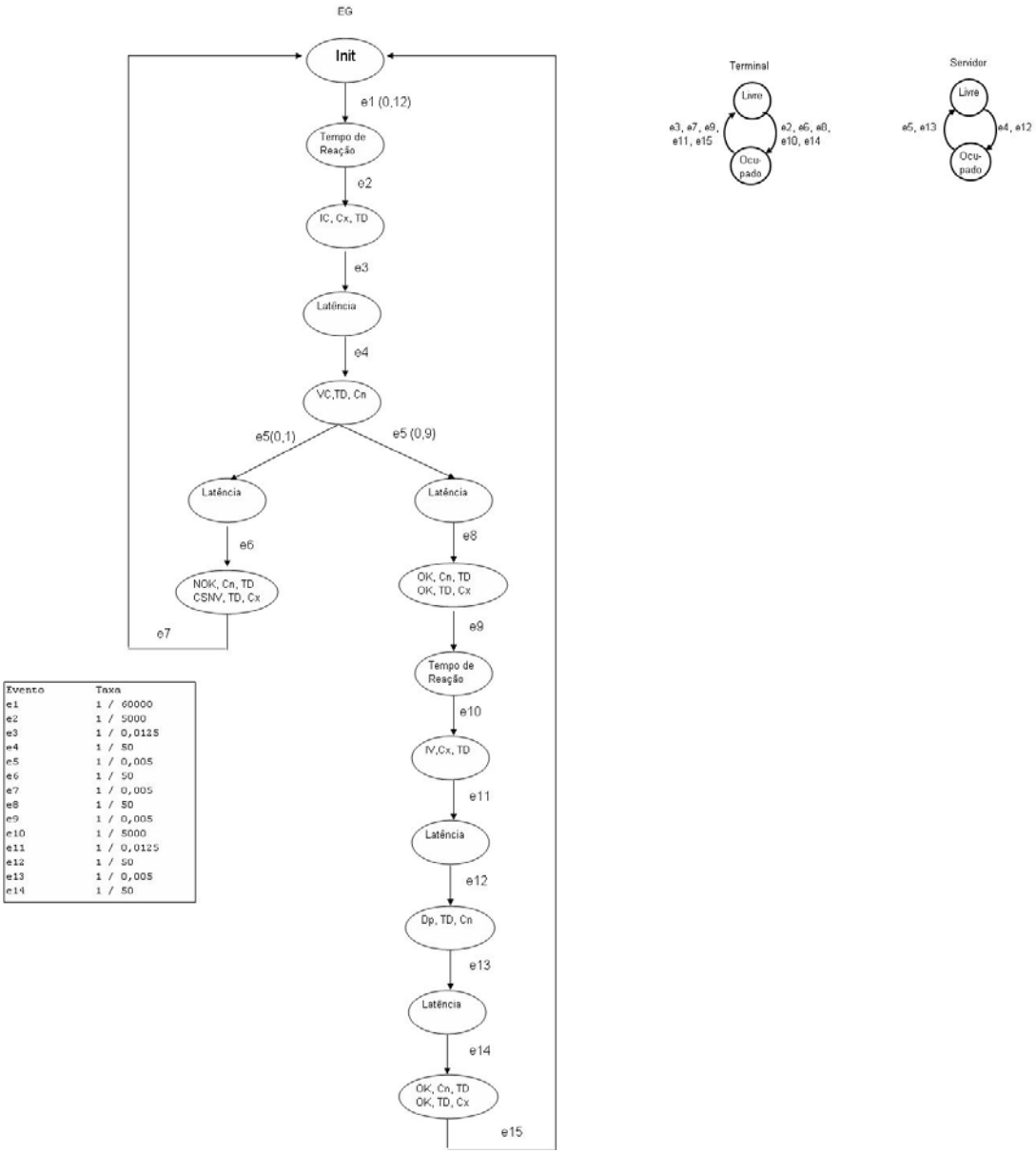


Figura 3.29: Autômato do EG Sincronizado com Recursos

Ao completar a sincronização para todos os estados com processamento, a SAN estará completa. Para resolvê-la, pode-se utilizar ferramentas de resolução de SAN, como por exemplo o PEPS [10]. Desta SAN podem ser obtidos resultados como tempo médio de processamento, utilização média de recursos, etc.

Capítulo 4

Estudo de Caso

Para verificar a técnica apresentada no Capítulo 3, será realizado um estudo de caso. Para este estudo, será utilizada uma modelagem em UML de uma aplicação já implementada e implantada[1]. Desta serão obtidos os tempos de processamento médios e tempos de latência médios. Com estes dados será medida a utilização média de cada recurso do sistema, através da SAN gerada pela técnica. Este será aplicado com diferente configurações de pacotes para verificar o impacto da concorrência pelos recursos na utilização dos mesmos.

4.1 Modelo

A modelagem do *software* será apresentada apenas em termos de diagramas de casos de uso, seqüência e implantação. Os demais diagramas não serão apresentados aqui.

O diagrama de casos de uso deste *software* é composta por um ator e três casos de uso: UC01 - *View Information*, UC02 - *View Details* e UC05 - *Help* conforme demonstrado na Figura 4.1. Entre estes casos de uso, dois relacionam-se diretamente com o ator (UC01 e UC05)e um é estendido (UC02). Por ser o único ator, o SystemUser tem probabilidade de ator igual a 1. A probabilidade de um SystemUser acessar o caso de uso UC01 é de 95% e o SystemUser tem 5% de chance de consultar o help em um acesso. No caso de uso UC01 o Ator tem 80% de chance de necessitar de mais informações e, assim, acessar o caso de uso UC02. O diagrama de casos de uso com as probabilidades é demonstrado na Figura 4.1.

[1] A intenção inicial deste estudo era de avaliar os resultados da aplicação da técnica com resultados reais desta aplicação, porém por restrições de tempo esta comparação ficará para trabalhos futuros.

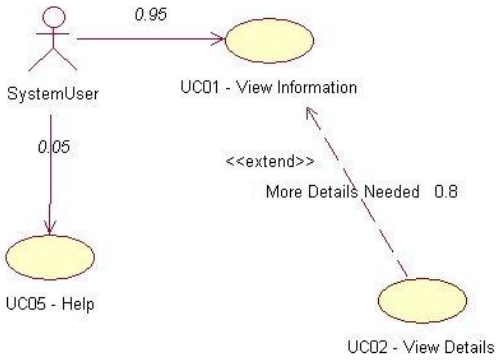


Figura 4.1: Diagrama de Casos de Uso do Estudo de Caso

Os diagramas de seqüência derivados dos fluxos dos casos de uso são de-monstrados nas Figuras 4.2, 4.3 e 4.4. Note que o diagrama de seqüência dos casos de uso estendido possui todas as chamadas do caso de uso base. Observe, também, que este sistema possui um grande número de chamadas de método. Muitas destas são feitas a elementos no mesmo componente do chamados, como será visto adiante.

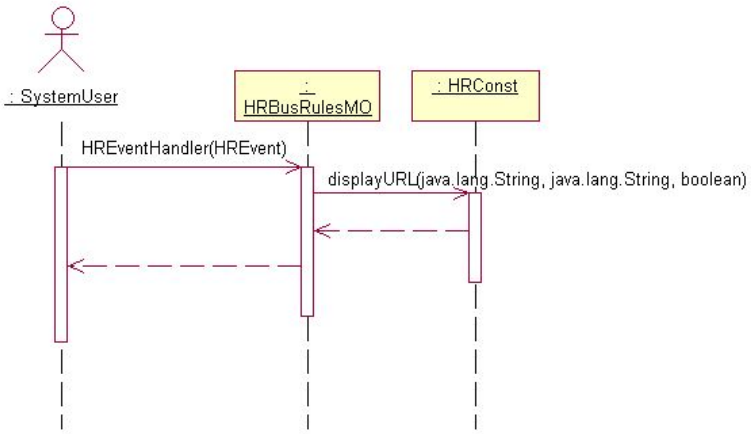


Figura 4.2: Diagrama de Seqüência do Caso de Uso UC05

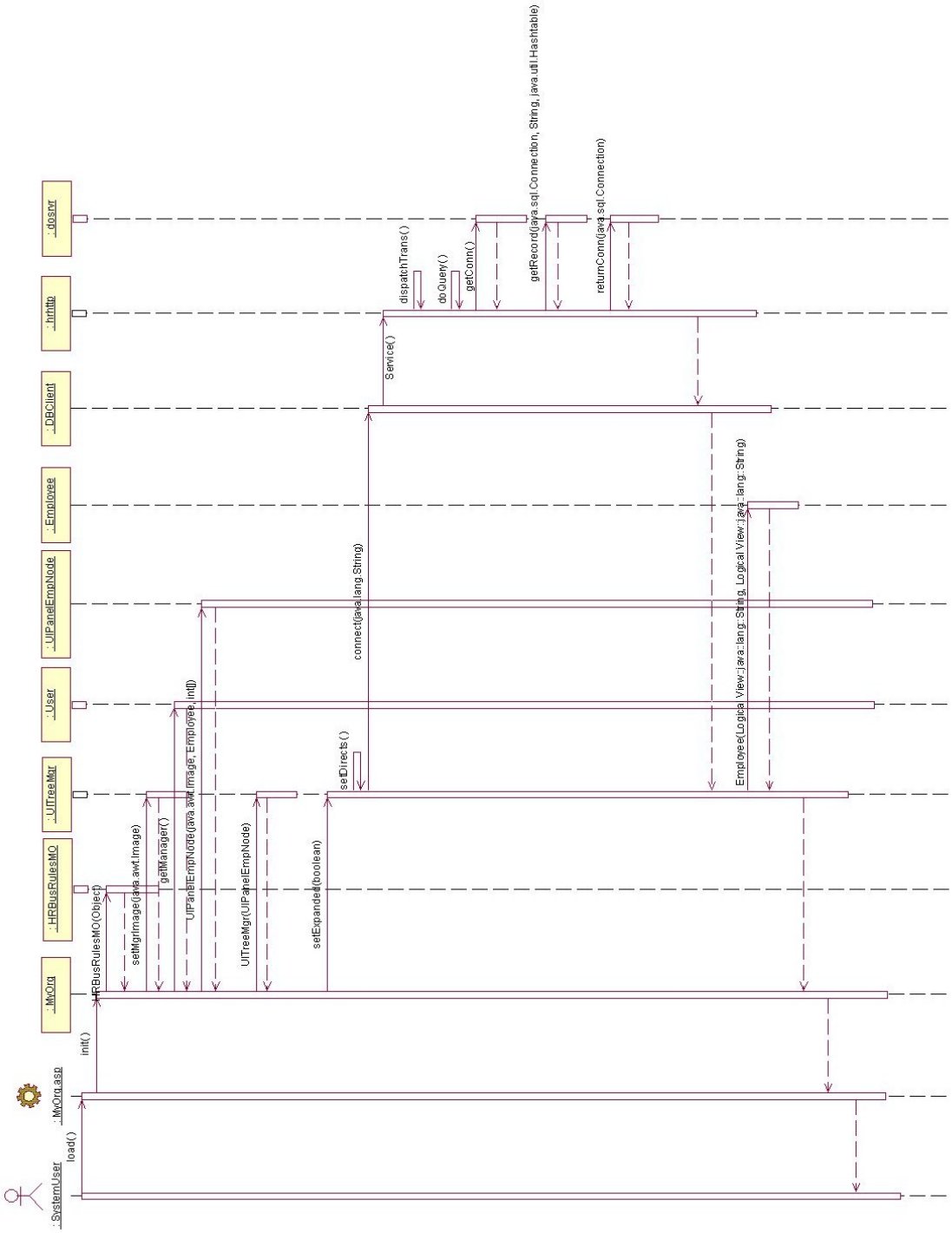


Figura 4.3: Diagrama de Seqüência do Caso de Uso UC01

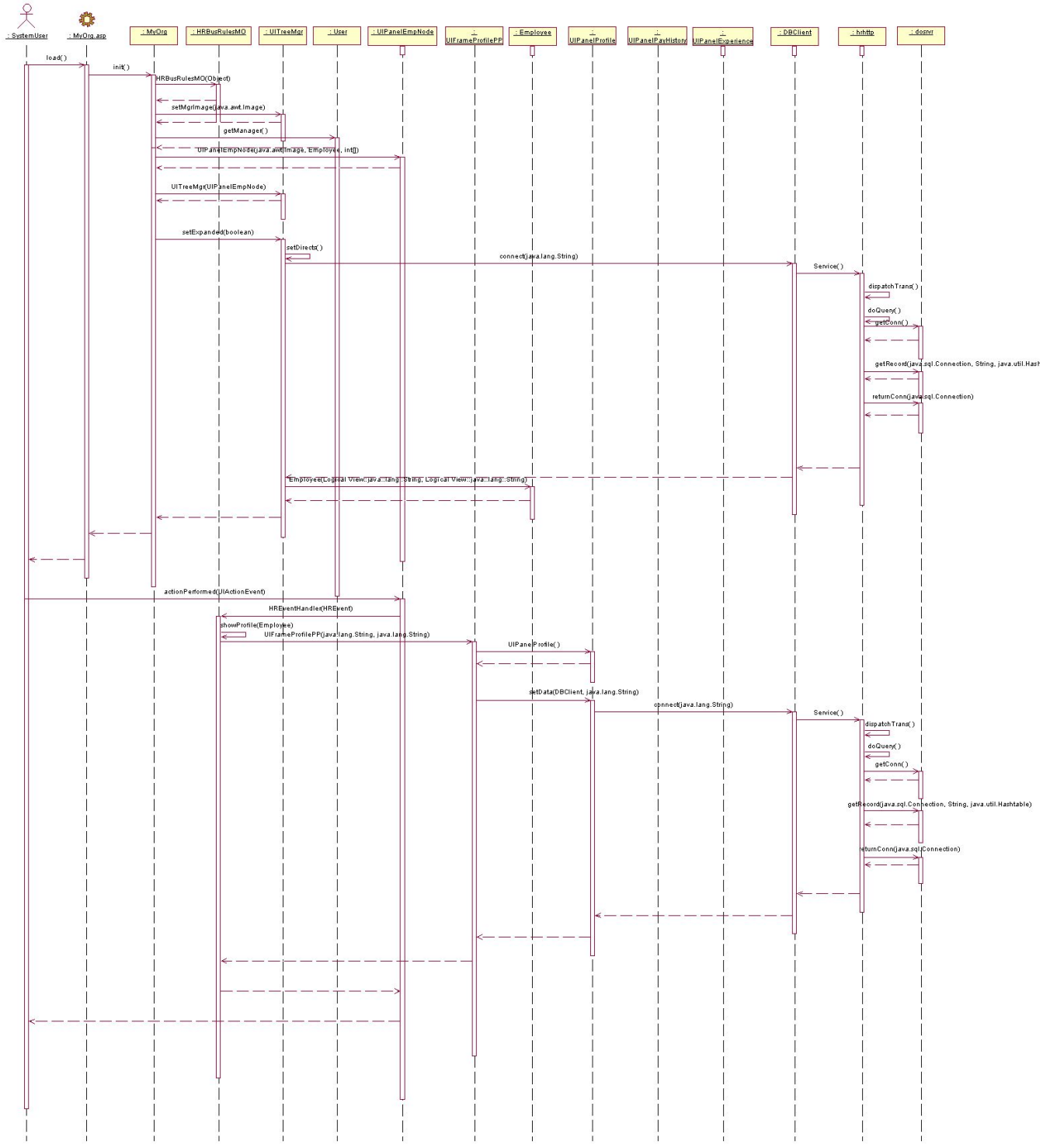


Figura 4.4: Diagrama de Seqüência do Caso de Uso UC02

Este *software* está implantado em uma plataforma bem simples, composta por uma máquina cliente e um servidor interligados por uma rede. O diagrama de implantação é demonstrado na Figura 4.5.

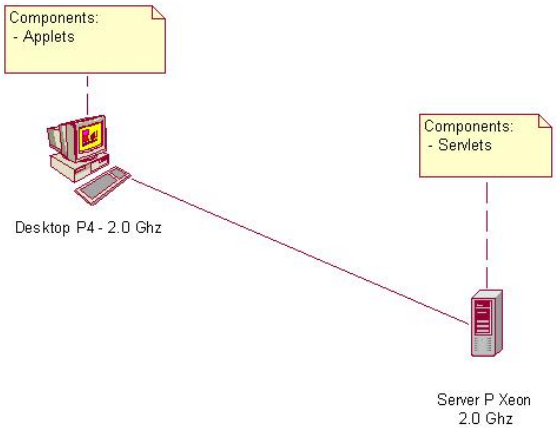


Figura 4.5: Diagrama de Implantação do Estudo de Caso

Neste sistema existe dois componentes: cliente e servidor. O componente servidor compreende as classes hrhttp e dosrvr. As demais classes pertencem ao componente cliente. O componente cliente está implantado na máquina cliente e o servidor na máquina servidor.

4.2 Conversão para SAN

Para aplicar a técnica descrita no Capítulo 3, são necessários, ainda, outros dados a serem informados. Estes dados são: as probabilidades de cada diagrama de seqüência dentro de cada caso de uso, o tempo médio de cada chamada, a latência média da Rede, o tempo médio de reação do usuário, tempo de médio de chegada de usuário, a quantidade de pacotes e a quantidade de usuários em cada pacote.

Como existe apenas um diagrama de seqüência para cada caso de uso, as pro-babilidades de cada diagrama de seqüência ser executado, quando o caso de uso for acessado é de 100%, ou seja igual a 1.

O tempo médio de cada chamada foi medido no sistema na implantação atual. Estes tempos já simplificados e multiplicados pela quantidade de usuários de cada pacote é demonstrada na SAN gerada pela técnica vista na Figura 4.6. Neste, estão representados 3 pacotes com 25 usuário em cada um , como será demonstrado a seguir.

O tempo médio de latência da rede, assim como os tempos de processamento, foi medido no ambiente onde o sistema está implantado. O resultado obtido foi de 200 milisegundos em média de atraso introduzido pela rede.

Na verificação a ser realizada considera-se um tempo médio de reação do usuário igual a 3 segundos. Nesta mesma verificação será considerado, que os usuários chegam ao sistema em média de 1 em 1 segundo, após os usuários correntes terem deixado o sistema.

A quantidade de usuário utilizados na verificação será de 75 usuários. Estes serão divididos em 5 configurações de pacotes diferentes:

- 1 pacote com 75 usuários;
- 2 pacotes com 37,5 usuário em cada um;
- 3 pacotes com 25 usuário em cada um;
- 4 pacotes com 38,75 usuário em cada um;
- 5 pacotes com 15 usuário em cada um.

Com estas informações, e após aplicar os passos descritos no Capítulo 3, obtém-se, para o cenário com 3 pacotes, a SAN demonstrada na Figura 4.6. Note que o passo da simplificação reduziu, drasticamente, o número final de estados na SAN em relação à quantidade de chamadas nos diagramas de seqüência. Isso se deve pelo grande número de chamadas realizadas entre classes em componentes na mesma máquina, ou, como no caso, no mesmo componente, como já mencionado.

Para resolver esta SAN será utilizada a ferramenta PEPS [10]. Nesta ferramenta a SAN é descrita através de um arquivo texto. Este arquivo é demonstrado no anexo C. Neste, está representando, assim com a SAN da Figura 4.6, o cenário com três pacotes contendo 25 usuários cada.

4.3 Resultados

Para medir a utilização dos recursos do sistema na SAN gerada pela técnica, basta verificar a probabilidade de cada recurso estar no estado ocupado. Na ferramenta PEPS adicionam-se funções na seção de resultados, uma para cada recurso. Neste caso, foram adicionadas as funções clntUtilizacao e srvrUtilizacao demonstradas no Apêndice C. Note que a SAN contém além das funções de utilização, as funções que resultam na ociosidade dos recursos.

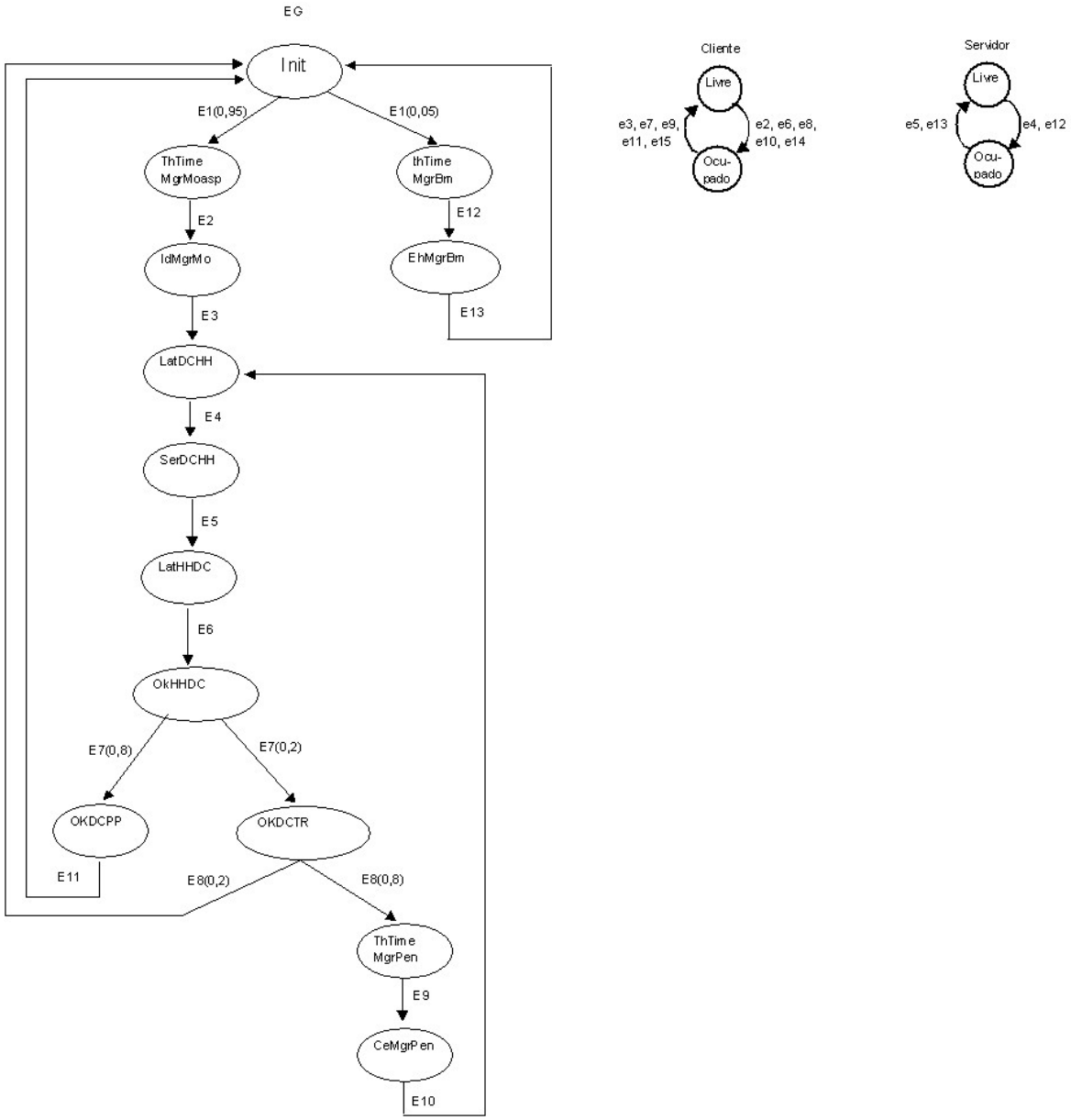


Figura 4.6: SAN gerada para o Estudo de Caso

A análise da utilização dos recursos depende muito da concorrência por estes recursos. Conforme mencionado anteriormente, o número de pacotes define a concorrência pelos recursos. Nesta verificação, o mesmo cenário descrito na seção anterior será resolvido nas 5 configurações diferentes descritas na seção anterior.

Os resultados obtidos para as utilizações do cliente e servidor em cada um dos cinco testes da SAN são demonstrados na Tabela 4.1.

Tabela 4.1: Resultados do Estudo de Caso

	1 Pacote	2 Pacotes	3 Pacotes	4 Pacotes	5 Pacotes
Utilização do Cliente	45,89%	41,03%	37,60%	35,19%	33,36
Utilização do Servidor	47,90%	45,72%	43,51%	41,37%	39,34

Com estes resultados pode-se notar que, com o aumento no número de pacotes os percentuais de utilização parecem em via de conversão a um número que melhor representa a utilização dos recursos. Observando as diferenças entre as porcen-tagens, estas diminuem, conforme aumenta-se os números de pacotes. Isto pode indicar a convergência ao número mais próximo da real utilização dos recursos. Levando a interpretar que, quanto maior o número de pacotes, mais preciso é o resultado.

Outra interpretação válida, é que estes números vem decrescendo devido ao aumento do atraso introduzido pela concorrência. Sendo a chegada de usuários dependente da saída de usuário, isto pode causar a diminuição da quantidade de usuários, que acessaram o sistema em um mesmo período de tempo. Isso causaria uma redução na utilização dos recursos, pois menos usuários os utilizam num mesmo espaço de tempo.

Portanto, a baixa quantidade de pacotes pode não representar a realidade, pois não pressupõem atrasos por disputa de recursos. Por outro lado, a grande quantidade de pacotes pode introduzir atraso em demasia aumentado o erro embutido nos resultados da utilização dos recursos.

Capítulo 5

Conclusões

Com este trabalho, foi possível demonstrar a existência de uma relação de implicação entre alguns modelos em UML e o formalismo SAN, e ainda, que esta implicação pode ser utilizada para obter medidas de desempenho do *software* modelado em UML.

A principal contribuição deste é a demonstração da técnica de conversão de UML para SAN. Esta foi obtida a partir de duas técnicas existentes: *Software Preformance Engineering* (SPE)[15] e método de conversão de UML para Redes de Filas de Espera (QN)[4]. Do SPE foi herdada a estrutura *Execution Graphs* (EG), a qual também é utilizada por Cortellessa e Mirandola na conversão de UML para QN[4]. Do método de conversão de UML para QN foram utilizados os passos de preparação dos diagramas UML e o algoritmo de conversão de diagramas de seqüência para EG. A partir desses, foi possível modificar passos existentes e adicionar novos, para gerar uma SAN ao invés de uma QN. Entre os passos mo-dificados podem ser citados como a obtenção do perfil do usuário e a identificação dos recursos de sistema, e entre passos adicionados estão a conversão de EG em Autômatos e Simplificação do EG.

Além da técnica de conversão de UML para SAN, este trabalho demonstrou a forma de tratamento de inclusões, extensões e generalizações nos diagramas de casos de uso. Com isso, o método de Cortellessa e Mirandola[4] e a técnica aqui apresentada passaram a viabilizar a conversão de uma maior quantidade de modelos em UML, tornando-os mais abrangentes.

Foi demonstrada a conversão de UML para SAN e eliminadas algumas restrições existentes em métodos similares, porém ainda existem restrições em relação ao UML requerido para gerar a SAN. Estas restrições podem ser tratadas em futuros trabalhos sobre metodologias e técnicas de conversão de UML para modelo de avaliação de desempenho. Entre, estes trabalhos, está a de-monstração do tratamento de generalização de atores, conforme mencionado ao longo da técnica aqui apresentada. Existem ainda, trabalhos futuros nos diagramas de

seqüência, para demonstrar como tratar chamadas assíncronas.

No modelo SAN gerado existem alguns estudos futuros a serem considerados. A técnica atual pressupõem que os recursos possuem apenas um processador. Isso é expresso na visão, de que existem apenas dois estados por recurso: livre e ocupado. A técnica deve ser melhorada para melhor representar recursos multi-processados. Com isso, será possível verificar, não só a utilização de um recurso, mas também a de cada processador que o compõem.

Outra melhoria a ser efetuada na SAN gerada é um melhor controle da chegada de usuário, pois, conforme comentado no estudo de caso, se houver um grande atraso devido a recursos indisponíveis a taxa de chegada de usuários será afetada diretamente, já que a chegada está diretamente relacionada ao fim do processamento dos usuários correntes. Assim, a técnica pode gerar um percentual de erro grande nos resultados.

Além das reduções de restrições com relação ao UML e melhorias na SAN gerada, existe um trabalho de validação desta técnica.

Um último estudo futuro visionado, a partir deste, é a conversão da UML 2.0 para SAN, já que neste foi utilizado o padrão UML 1.5.

Por fim, este trabalho promove um nova frente de estudos na avaliação de de desempenho de *softwares* orientados a objeto.

Referências Bibliográficas

[1] BENOIT, Anne; BRENNER, Leonardo; FERNANDES, Paulo; PLATEAU, Brigitte. The PEPS software tool. In: 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, 2003, Urbana, IL. Urbana, IL: Springer Berlin / Heidelberg, 2003. p. 98-115.

[2] BOOCH, Grandy. Object-oriented design. Redwood City, CA: Benjamin/Cummings, 1991, 580 p.

[3] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML Guia do Usuário. Rio de Janeiro, RJ: Editora Campus, 2000, 472 p.

[4] CORTELLESSA, Vittorio; MIRANDOLA, Rafaela. Deriving a Queueing Network based Performance Model from UML Diagrams. In: Workshop on Software and Performance, 2000, Ottawa, ON. Nova York, NY: ACM Press, 2000, pp. 58-70.

[5] GU, Gordon Ping, PETRIU, Dorina C. . Early evaluation of software performance based on the UML performance profile. In: IBM Centre for Advanced Studies Conference, 2003,Toronto, ON. Toronto, ON: IBM Press, 2003, pp. 66-79.

[6] JACOBSON, Ivar; CHRISTERSON, Magnus; JONSSON, Patrik; ÖVERGAARD, Gunnar. Object-oriented software engineering : a use case driven approach. Nova York, NY: ACM Press , 1992, 552 p.

[7] KÄHKIPURO, Pekka. UML Based performance modeling of object-oriented distributed systems. In: Software Performance Prediction extracted from Designs,1999, Edimburgo, UK. Edimburgo, UK: Heriot-Watt University, 1999. p. 1-8.

[8] LAZPWSKA,Edward D. Quantitative system performance : computer system analysis using queueing network models. Upper Saddle River, NJ: Predice-Hall, 1984, 417 p.

[9] Object Management Group. The current UML specification, Version 1.5. Disponível em: <http://www.uml.org/> Acesso em: Julho, 2004.

[10] Performance Evaluation Group - PEG. Performance Evaluation of Parallel Systems - PEPS. Disponível em: <http://www.inf.pucrs.br/ peg/peps.html>. Acesso em: Novembro, 2004.

[11] RIBEIRO, Marcelo B. "Exemplo de Modalagem UML". Disponível em: <http://www.inf.pucrs.br/ blois/materiais/modelagem/material.zip>. Acesso em: Outubro, 2004.

[12] RUMBAUGH, James; BLAHA, Michael; PREMARLANI, William; EDDY, Frederick; LORENSEN, William; ÖVERGAARD, Gunnar. Object-oriented : modeling and design. Upper Saddle River, NJ: Prentice-Hall, 1991, 500 p.

[13] RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady . The Unified Modeling Language Reference Manual. Reading, MA: Addison-Wesley, 1999, 550 p.

[14] SMITH, Connie U., WILLIAMS, Lloyd G. Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives. IEEE Transations on Software Engineering, New York, NY, v. 19, n. 7, pp. 720-741, jul. 1993.

[15] SMITH, Connie U. Designing High-Performance Distributed Applications Using Software Performance Engineering: A Tutorial. In: Computer Measurement Group Conference, 1996, San Diego, CA. Turnersville, NJ: Computer Measurement Group, 1996, p. 498-507.

Apêndice A

Algoritmos

Algorithm 1 Conversão Diagramas de Seqüência em EG

```
1: Adicione Nodo Init ao EG
2: Adicione uma transição pendente ao nodo Init
3: for all Sequence Diagrams do
4:    while existem transições não convertidas neste diagrama de seqüência do
5:       considere a próxima interação (l,A1,A2)
6:       converte interação(l,A1,A2)
7:    end while
8:    if existe uma transição pendente then
9:       ligue a transição pendente ao nodo Init
10:   end if
11: end for
```

Algorithm 2 Converte interação(l, A1, A2)

```
1: if interação (l,A1,A2) já foi convertida then
2:   if existe uma transição pendente then
3:     ligue-a ao nodo (l,A1,A2)
4:     if foi criado um ciclo que não inclui o nodo Init then
5:       adicione ao EG um novo nodo de Ciclo
6:     end if
7:   end if
8: else
9:   if (l,A1,A2) é uma chamada simples then
10:    if existe uma transição pendente then
11:      crie um novo nodo básico (l,A1,A2)
12:      ligue a transição pendente ao novo nodo básico
13:    else
14:      if existe um último nodo visitado then
15:        crie um novo nodo de branch após o último nodo visitado
16:      else
17:        crie um novo nodo de branch após o nodo Init
18:      end if
19:      insira o caminho já existente como um ramo do branch
20:      crie um novo nodo básico (l,A1,A2)
21:      insira um novo ramo no branch ligando-o ao novo nodo
22:    end if
23:    Crie uma nova transição pendente após o nodo (l,A1,A2)
24:  else
25:    //(l,A1,A2) é uma chamada múltipla com número de processos igual a c
26:    if existe uma transição pendente then
27:      crie um nodo Fork com multiplicidade c
28:      ligue a transição pendente ao nodo fork
29:    else
30:      if existe um último nodo visitado then
31:        crie um novo nodo de branch após o último nodo visitado
32:      else
33:        crie um novo nodo de branch após o nodo Init
34:      end if
35:      insira o caminho já existente como um ramo do branch
36:      crie um nodo fork com multiplicidade c
37:      insira um novo ramo no branch ligando-o ao novo nodo
38:    end if
39:    for all processos em (l,A1,A2) do
40:      traduza seus nodos em nodos básicos ligados entre si
41:    end for
42:    crie um nodo join
43:    ligue todos os processos ao nodo join
44:    crie uma nova transição pendente após o nodo join
45:    passe para a interação imediatamente após as interações múltiplas
46:  end if
47: end if
```

Apêndice B

Tabelas de Abreviaturas para EG

Tabela B.1: Abreviaturas de Atores

Ator	Abreviatura
Cliente	Cl
Caixa	Cx
Gerente	G

Tabela B.2: Abreviaturas de Classes

Classe	Abreviatura
Conta	Cn
RealizaPtgo	RP
TelaSaque	TSq
TelaAutorizacaoSaque	TASq
TelaSaldo	TSl
TelaRecebPgto	TRP
TelaDeposito	TD

Tabela B.3: Abreviaturas de Métodos e Respostas

Método ou Resposta	Abreviatura
Autoriza	Aut
CalculaTroco	CT
CarregaConta	CC
conta ou senha não Válidas	CSNV
Depositar	Dp
Dinheiro será liberado	DL
getSaldo	gSl
IdentificaContaAlvo	ICA
ImprimeComprovante	IC
Informa Autorizacao é necessária	IA
InfomaContaSenha	ICS
InformaValor	IV
Load	Ld
não há saldo	NSl
NOK	NOK
NotificaAutorizacao	NotAut
NotificaRejeicao	NotRej
OK	OK
PgtoCheque	PC
PgtoDinheiro	PD
RegistraDocCobranca	RDC
RegistraDocPgto	RDPt
Rejeita	Rej
Sacar	Sc
Saldo	Sl
Saque Rejeitado	SaqRej
ValidaCheque	VCh
ValidaConta	VC

Apêndice C

SAN Resultante do Estudo de Caso

identifiers
T1 = 1/1000; // taxa de chegada de usuários
T2 = 1/3000; // user thinking Time
T3 = 1/300;
T4 = 1/200; // latência
T5 = 1/24;
T6 = 1/200;// latência
T7 = 1/1;
T8 = 1/6;
T9 = 1/3000; // user thinking Time
T10 = 1/12;
T11 = 1/5;
T12 = 1/3000; // user thinking Time
T13 = 1/6;

events
loc E11 (T1) EG1
syn E21 (T2) EG1, Clnt
syn E31 (T3) EG1, Clnt
syn E41 (T4) EG1, Srvr
syn E51 (T5) EG1, Srvr
syn E61 (T6) EG1, Clnt
loc E71 (T7) EG1
syn E81 (T8) EG1, Clnt

syn E91 (T9) EG1, Clnt
syn E101 (T10) EG1, Clnt
syn E111 (T11) EG1, Clnt
syn E121 (T12) EG1, Clnt
syn E131 (T13) EG1, Clnt

loc E12 (T1) EG2
syn E22 (T2) EG2, Clnt
syn E32 (T3) EG2, Clnt
syn E42 (T4) EG2, Srvr
syn E52 (T5) EG2, Srvr
syn E62 (T6) EG2, Clnt
loc E72 (T7) EG2
syn E82 (T8) EG2, Clnt
syn E92 (T9) EG2, Clnt
syn E102 (T10) EG2, Clnt
syn E112 (T11) EG2, Clnt
syn E122 (T12) EG2, Clnt
syn E132 (T13) EG2, Clnt

loc E13 (T1) EG3
syn E23 (T2) EG3, Clnt
syn E33 (T3) EG3, Clnt
syn E43 (T4) EG3, Srvr
syn E53 (T5) EG3, Srvr
syn E63 (T6) EG3, Clnt
loc E73 (T7) EG3
syn E83 (T8) EG3, Clnt
syn E93 (T9) EG3, Clnt
syn E103 (T10) EG3, Clnt
syn E113 (T11) EG3, Clnt
syn E123 (T12) EG3, Clnt
syn E133 (T13) EG3, Clnt

reachability = 1;

network CS1 (continuous)
aut EG1
stt init
to(thTimeMgrMoAsp) E11(0.95)
to(thTimeMgrBm) E11(0.05)
stt thTimeMgrMoAsp
to(ldMgrMo) E21
stt ldMgrMo
to(latDcHh) E31
stt latDcHh
to(serDcHh) E41
stt serDcHh
to(letHhDc) E51
stt letHhDc
to(okHhDc) E61
stt okHhDc
to(okDcTr) E71(0.2)
to(okDcTp) E71(0.8)
stt okDcTr
to(init) E81(0.2)
to(thTimeMgrPen) E81(0.8)
stt ceMgrPen
to(latDcHh) E101
stt thTimeMgrPen
to(ceMgrPen) E91
stt okDcTp
to(init) E111
stt thTimeMgrBm
to(ehMgrBm) E121
stt ehMgrBm
to(init) E131

aut EG2
stt init
to(thTimeMgrMoAsp) E12(0.95)
to(thTimeMgrBm) E12(0.05)
stt thTimeMgrMoAsp
to(ldMgrMo) E22
stt ldMgrMo
to(latDcHh) E32
stt latDcHh
to(serDcHh) E42
stt serDcHh
to(letHhDc) E52
stt letHhDc
to(okHhDc) E62
stt okHhDc
to(okDcTr) E72(0.2)
to(okDcTp) E72(0.8)
stt okDcTr
to(init) E82(0.2)
to(thTimeMgrPen) E82(0.8)
stt ceMgrPen
to(latDcHh) E102
stt thTimeMgrPen
to(ceMgrPen) E92
stt okDcTp
to(init) E112
stt thTimeMgrBm
to(ehMgrBm) E122
stt ehMgrBm
to(init) E132

aut EG3
stt init
to(thTimeMgrMoAsp) E13(0.95)

to(thTimeMgrBm) E13(0.05)
stt thTimeMgrMoAsp
to(ldMgrMo) E23
stt ldMgrMo
to(latDcHh) E33
stt latDcHh
to(serDcHh) E43
stt serDcHh
to(letHhDc) E53
stt letHhDc
to(okHhDc) E63
stt okHhDc
to(okDcTr) E73(0.2)
to(okDcTp) E73(0.8)
stt okDcTr
to(init) E83(0.2)
to(thTimeMgrPen) E83(0.8)
stt ceMgrPen
to(latDcHh) E103
stt thTimeMgrPen
to(ceMgrPen) E93
stt okDcTp
to(init) E113
stt thTimeMgrBm
to(ehMgrBm) E123
stt ehMgrBm
to(init) E133

aut Clnt
stt livre
to(ocupado) E21 E61 E91 E121 E22 E62 E92 E122 E23 E63 E93 E123
stt ocupado
to(livre) E31 E81 E101 E111 E131 E32 E82 E102 E112 E132 E33 E83 E103 E113 E133

```
aut Srvr
stt livre
to(ocupado) E41 E42 E43
stt ocupado
to(livre) E51 E52 E53

results
clntUtilizacao = st Clnt == ocupado;
srvrUtilizacao = st Srvr == ocupado;
```