Marcelo José Siqueira Coutinho de Almeida

ATMLib - Uma Biblioteca de Classes para a Construção de Simuladores de Redes ATM: Proposta e Implementação

Dissertação de Mestrado submetida à Coordenação de Pós-Gradução em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba - CAMPUS II como parte dos requisitos básicos para a obtenção do grau de Mestre em Ciências no Domínio da Computação.

Área de Concentração: Redes de Computadores.

Prof. Maria Izabel Cavalcanti Cabral, Dra.
Orientadora

Campina Grande, Paraíba, Brasil

A447a Almeida, Marcelo Jose Siqueira Coutinho de
ATMLib : uma biblioteca de classes para a construcao de simuladores de redes ATM : proposta e implementacao / Marcelo Jose Siqueira Coutinho de Almeida. - Campina Grande, 1999.
145 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Computadores 2. Simulacao Digital 3. Orientacao a Objetos 4. Dissertacao - Informatica I. Cabral, Maria Izabel Cavalcanti II. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 004.7(043)

ATMLib – UMA BIBLIOTECA DE CLASSES PARA A CONSTRUÇÃO DE SIMULADORES DE REDES ATM: PROPOSTA E IMPLEMENTAÇÃO

MARCELO JOSÉ SIQUEIRA COUTINHO DE ALMEIDA

DISSERTAÇÃO APROVADA EM 23.08.1999

PROFª MARIA IZABEL CAVALCANTI CABRAL, D.Sc
Orientadora

PROF. ELMAR UWE KURT MELCHER, Dr.
Orientador

PROF. GUIDO LEMOS DE SOUZA FILHO, Dr.
Examinador

CAMPINA GRANDE – PB

Dedicatória

À minha esposa, Helda Liana,
por ter estado comigo nesta e em outras caminhadas.
À minha filha, Marcellinha,
por ter tornado esse mundo bem mais bonito.

Agradecimentos

Este trabalho não teria sido possível sem o apoio e assistência de várias pessoas durante os vários anos de minha vida estudantil, com as quais divido a alegria e a satisfação de alcançar mais esse exito.

O maior destes créditos devo a minha família, principalmente minha mãe, minha irmã Fátima e meu irmão Christiano, que através de seus esforços pude ter acesso, dentre outras coisas, à educação e assim descobrir seu valor e seguir o caminho que me trouxe até esta etapa importante de minha vida.

Agradeço a minha Orientadora, Maria Izabel Cavalcanti Cabral, pelo constante apoio, confiança e orientação.

Agradeço ao professor Elmar Kurt Melcher, pela sua importante colaboração a este trabalho através de discussões, sugestões e esclarecimentos, principalmente no que diz respeito aos comutadores ATM.

Também agradeço a Josenildo, colega de mestrado, pelas dicas de LaTeX, as quais engradeceram em muito esse trabalho.

A todos os colegas do Grupo de Redes de Computadores.

Aos funcionários da COPIN (Aninha e Vera) e do LABCOM (Lilian, Fábio e Alberto) pela maneira prestativa que comigo lidaram, facilitando minha estadia no DSC.

À Josenilda, Romildo e Dona Inês da cantina do DSC, pela forma cordial que sempre me trataram e pela confiança em minha pessoa.

À CAPES, pelo apoio financeiro, sem o qual esse trabalho não teria sido possível.

"Nada pode ser amado sem ser conhecido antes. Quanto mais conhecemos, mais amamos. O amor é tanto mais ardente quanto o conhecimento é mais certo"

Leonardo Da Vinci.

Resumo

A necessidade crescente por sistemas de comunicação capazes de lidar de forma integrada e à altas velocidades com as mais diferentes mídias (texto, vídeo, áudio, etc.) levou a comunidade científica e industrial a optar pela tecnologia ATM como modo de transferência destas redes. No entanto, apesar de todo o esforço em torno da consolidação desta tecnologia, diversos aspectos ainda precisam ser melhorados ou solucionados. Nesse sentido, faz-se necessário a utilização de ferramentas de avaliação de desempenho flexíveis que possam auxiliar nos estudos de possíveis alterações e de novas propostas para essa tecnologia. O paradigma da orientação a objetos e a linguagem Java tem se mostrado eficientes para o desenvolvimento de ferramentas com essas características. Neste contexto, propomos e implementamos uma biblioteca de classes chamada **ATMLib** (*ATM Library*) que suporte o desenvolvimento de simuladores de redes ATM de forma eficiente e econômica, explorando as facilidades da reusabilidade propiciadas pela abordagem orientada a objetos. As classes dessa biblioteca modelam os elementos básicos de uma rede ATM, podendo ser estendidos a fim de acomodar novas funcionalidades que vierem a ser propostas. Entretanto, deve-se ressaltar que bibliotecas de software são sistemas complexos e devem ser realizados baseando-se nos princípios da Engenharia de Software. Nesse sentido, o desenvolvimento da ATMLib se deu através das fases tradicionais do ciclo de vida de um software (*Especificação de Requisitos*, *Análise*, *Projeto*, *Implementação* e *Testes*). A fim de descrever o conhecimento obtido a partir do domínio do problema e de estruturar sua solução foi utilizada a linguagem de descrição de modelos UML (*Unified Modelling Language*). O resultado destas fases foi implementado utilizado a linguagem de programação Java e, posteriormente, testado utilizando-se a ferramenta de testes JUnit. Os resultados dos testes demonstraram que a execução das classes desenvolvidas correspondem ao esperado.

Abstract

The growing demand for communication systems which are able to deal with different media over high speed links in an integrated fashion has lead the scientific and industrial community to choose ATM as the transfer mode for these networks. Despite of all efforts in consolidating this technology, many aspects still to be improved in order to use ATM. Evaluation tools, which are suficiently flexible, are need to evaluate the performance of these networks as modifications or new proposals are introduced. Object oriented paradigm and Java programming language are very useful in developing such tools. In this context, we propose and implement a library supporting the development of ATM network simulators in a fast, efficient and economical way. Classes from this library model the basic elements of ATM networks. These classes may be extended in order to agregate future new funcionalities. However, it's important to emphasize that software library are complex sistems and they must be developed based on Software Engineering principles. ATMLib was developed through traditional life-cycle phases: *Requirements Especification*, *Analysis*, *Project*, *Implementation* and *Tests*. UML (*Unified Modelling Language*) was used in order to describe the problem domain and to acquire the solution for it. Results from theses phases were implemented using the Java programming language and tested using a test tool called JUnit. Test results showed that ATMLib behaves as expected.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Este capítulo fornece a base para o desenvolvimento do trabalho apresentado nesta dissertação. Inicialmente, é dada uma visão geral sobre a importância das redes ATM dentro do contexto das Telecomunicações assim como da necessidade da utilização de ferramentas de avaliação de desempenho necessárias para sua evolução. Nas seções seguintes são apresentados os objetivos, a motivação, os trabalhos relacionados e a relevância da ATMLib. Por último, é mostrada de forma resumida a organização do restante deste trabalho.

1.1 Simulação de Redes ATM

A crescente demanda nas últimas décadas por sistemas de comunicação capazes de lidar com as mais diferentes mídias (som, imagem, vídeo, etc.) de forma integrada tem despertado a atenção da comunidade científica e industrial para a busca de soluções que possam atender a essa necessidade através de uma única estrutura de comunicação e da transmissão de informação à altas velocidades.

A solução a essa questão veio através das *Redes Digitais de Serviços Integrados*, especificamente a RDSI-FL, que utiliza o Modo de Transferência Assíncrono (ATM - *Asynchronous Transfer Mode*), uma tecnologia de comutação que se baseia em unidades de tamanho reduzido e formato fixo [8].

Nesses sistemas, todos os tipos de informação podem ser enviados por uma única infra-estrutura de comunicação, ao contrário dos tradicionais, como por exemplo as redes telefônicas trafegam voz, as redes de comutação de pacotes trafegam dados e as redes a cabo ou radio-difusão trafegam imagens.

A principal vantagem observada a partir da utilização desses sistemas é a economia devido ao compartilhamento dos recursos e à relativa simplicidade de administração devido à existência de uma rede única [16]

Entretanto, vários aspectos relevantes à tecnologia de redes ATM ainda se encontram em fase de pesquisa e padronização, o que tem estimulado inúmeros esforços por parte da comunidade científica e industrial relacionada com esses aspectos a fim de padronizar, implementar e consolidar esta tecnologia não apenas como infra-estrutura de comunicação em redes de médio e longo alcance, mas também como tecnologia principal nos ambientes das redes locais de computadores [7].

Nesse sentido, faz-se necessário o estudo do desempenho dos sistemas de redes ATM, propostos para padronização pelos órgãos responsáveis (ITU-T e Fórum ATM), assim como dos possíveis impactos causados pela sua implantação em ambientes locais a fim de substituir as tecnologias existentes.

Para auxiliar esse estudo, é fundamental que ferramentas de modelagem e avaliação de desempenho de redes ATM sejam desenvolvidas. Essas ferramentas devem ser flexíveis o suficiente para que novos conceitos ou melhorias decorrentes do processo de padronização e da evolução natural da tecnologia possam ser a ela agregados sem a necessidade de se construir novas ferramentas.

1.2 Reusabilidade de Software

Nos últimos anos tem ficado cada vez mais evidente a necessidade de reuso de software [47]. Dois motivos podem ser destacados para que essa situação ocorra: (i) a crescente demanda por sistemas de software nos mais diversos ambientes, e (ii) a crescente complexidade desses sistemas em função da necessidade por um número cada vez maior de funcionalidades por parte dos usuários.

A idéia por trás do reuso é bastante simples. Consiste em se desenvolver apenas aquilo que não existe e em reutilizar o que já está desenvolvido. Isto possibilita um tempo mais curto de desenvolvimento do produto e, consequentemente, um custo menor de produção e um tempo de entrega ao cliente mais curto.

Neste contexto, a orientação a objetos tem se mostrado uma abordagem promissora no sentido de incrementar o reuso de código por introduzir conceitos como *herança* e *interface* [2]. Bibliotecas de classes têm sido desenvolvidas e largamente utilizadas para o desenvolvimento dos mais variados tipos de aplicações, tais como interfaces com o usuário, sistemas de consultas a bases de dados, etc. Bibliotecas de classes consistem de componentes coesos e bem definidos que, propiciam aos desenvolvedores de software um ganho significativo de produtividade.

Subjacente ao processo de adoção da orientação a objetos, uma parcela da comunidade científica e industrial tem concentrado inúmeros esforços na busca de metodologias e ferramentas que favoreçam a adoção definitiva dessa abordagem como forma de desenvolvimento de software.

Nesse sentido, deve-se ressaltar a importância do trabalho de James Rumbaugh, Ivan Jacobson e Grady Booch em torno da linguagem de modelagem **UML** (*Unified Modelling Language*) e do processo de desenvolvimento Unificado [2] [3] [4]. A UML é uma linguagem de modelagem[1] que visa a unificação das notações desenvolvidas anteriormente por esses três autores. James Gosling juntamente com outros desenvolvedores da *Sun Microsystems* desenvolveu uma linguagem de programação totalmente orientada a objetos chamada **Java** [36]. Essa linguagem vem tendo grande aceitação tanto no mercado como no meio acadêmico pela sua simplicidade, portabilidade e por, dentre outros motivos, fornecer recursos para o desenvolvimento de aplicações para as mais diversas áreas (Banco de Dados, Sistemas Distribuídos, Redes de Computadores, Sistemas Concorrentes, etc.). Também deve-se ressaltar a importância do trabalho de

[1] A UML é uma linguagem de modelagem, não um método. Métodos consistem, ao menos em princípio, de uma linguagem de modelagem e de um processo. A linguagem de modelagem é a notação que os métodos usam para expressar projetos através de um conjunto de modelos. Por outro lado, o processo é, dentre outras coisas, um guia sobre a seqüência de atividades a se realizar durante um projeto.

Erich Gamma *et al.* [34], que organizou um catálogo de padrões de projeto (**Design Patterns**) a fim de permitir que soluções de projeto de sucesso possam ser reutilizadas tanto pelo próprio projetista como por outros, bastando para isso documentá-los adequadamente.

No entanto, projetar software reusável ainda permanece uma tarefa difícil, pois requer produzir componentes gerais e extensíveis. Desta forma, projetistas são solicitados a predizer futuras aplicações e incorporar seus requisitos no projeto sendo desenvolvido.

1.3 Objetivos

O objetivo deste trabalho encontra-se no âmbito da avaliação de desempenho de sistemas discretos orientados a eventos, mais especificamente na construção de ferramentas de simulação digital.

Essa dissertação propõe uma biblioteca de classes, denominada ATMLib, para auxiliar a construção de simuladores de redes ATM. Dessa forma, tendo-se a ATMLib, o desenvolvedor poderá dispor dos elementos básicos de uma rede ATM (comutadores, fontes, enlaces, etc.) para desenvolver de forma rápida seu próprio simulador.

Dessa forma, para se desenvolver um simulador o projetista precisará apenas projetar as classes de interface com o usuário e as de controle da simulação (figura 1.1).

Nessa dissertação foi utilizada a UML nas seguintes fases do ciclo de vida tradicional: *Especificação de Requisitos*, *Análise*, *Projeto*, *Implementação* e *Testes de Unidade*.

A *Especificação de Requisitos* consiste em descrever o que deve ser feito, ressaltando as características e as limitações que o software deve possuir ao final do processo de desenvolvimento, além dos recursos necessários para desenvolvê-lo.

A fase de *Análise* consiste em descrever os diferentes aspectos do sistema através de modelos de alto nível. Deve refletir o conhecimento essencial para o desenvolvimento do software descartando aquelas informações que não forem úteis.

Na fase de *Projeto* o conhecimento obtido na fase anterior deve ser transformado

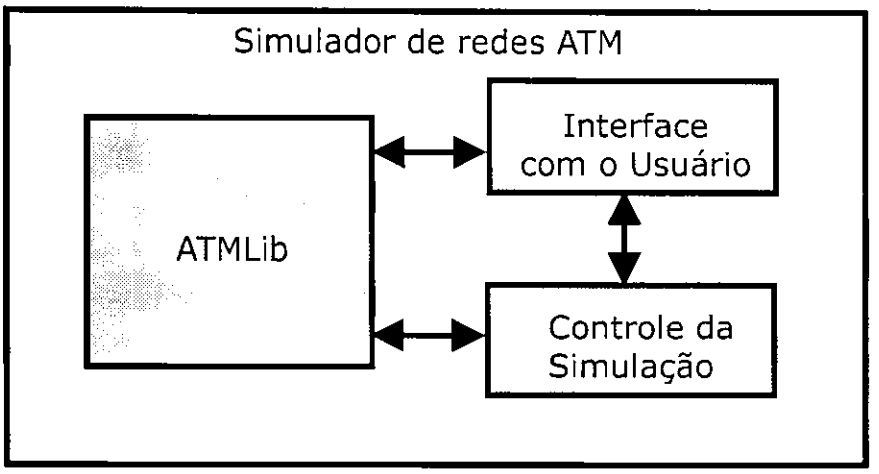

Figura 1.1: Estrutura de um Simulador desenvolvido utilizando a ATMLib

em um conjunto de representações de baixo nível passíveis de implementação pelo programador.

A fase de *Implementação* consiste em unicamente traduzir o resultado da fase anterior em um código executável utilizando uma linguagem de programação.

Na fase de *Testes de Unidade* cada uma das classes implementada é testada individualmente a fim de certificar se sua execução corresponde ao esperado.

É importante ressaltar que, conforme apresentado em [24], existem outras fases de Testes, como por exemplo *Testes de Módulo*, *Testes de Função* e *Teste Beta*. No entanto, estes não serão aplicados neste trabalho uma vez que para realizá-los faz-se necessário ter uma aplicação completa executando.

Os resultados juntamente com uma descrição mais detalhada de cada uma das fases são apresentados através de seus capítulos referentes juntamente com as eventuais discussões tanto a respeito desses resultados como das decisões tomadas.

1.4 Motivação

As redes ATM, apesar de seu rápido reconhecimento como base tecnológica para a próxima geração de comunicação global, estendendo-se sobre diversas aplicações e uma variedade de velocidades e distâncias, ainda não atingiram plenamente seu desenvolvimento. Para que isso ocorra, faz-se necessário ainda que determinadas padronizações sejam realizadas no âmbito do ITU-T e do Fórum ATM [13].

Em função desta falta de padronização, dos altos custos e da falta de um conjunto maior de aplicações multimídia (adaptadas às camadas AAl) que justifiquem sua utilização, as redes ATM ainda não ocuparam largamente o seu espaço dentro do ambiente corporativo (local). No entanto, soluções como LANE (*LAN Emulation*), MPOA (*Multicasting Protocol Over ATM*) e IPOA (*Internet Protocol Over ATM*) [21] têm sido propostas objetivando acelerar este processo.

Neste sentido, dois cenários têm se formado fortemente em torno destas redes. O primeiro diz respeito a um crescente interesse nos ambientes *desktop* (LANs) em se migrar das soluções atualmente existentes (*Ethernet*, *Token Ring*,*Token Bus*, etc.) para ATM. Uma prova disto é a disposição da indústria em buscar soluções para os problemas ainda existentes e a oferta crescente de soluções ATM no mercado (placas de rede, comutadores, etc.) para ambientes locais [7].

O segundo cenário está relacionado à padronização ainda necessária para se estabelecer o ATM em definitivo como uma solução amplamente disponível para prover a infra-estrutura de comunicação em ambientes de largo alcance (WANs). Esta infra-estrutura deverá oferecer facilidades tanto aos serviços existentes (TV, vídeo-conferência, telefonia, etc), como também para os que ainda virão a ser implementados, como por exemplo TV digital, HDTV, etc[16].

Tendo-se observado esses dois cenários, percebe-se que há nos mesmos uma grande demanda por ferramentas de modelagem e avaliação de desempenho. Essas ferramentas devem permitir tanto aos administradores de ambientes de redes estudar o impacto de se migrar das tecnologias atuais para ATM como aos projetistas e/ou responsáveis

pela criação e padronização de novos protocolos observar o seu comportamento e, consequentemente, sua viabilidade, enquanto ainda em fase de projeto. Devido ao fato de que a tecnologia ATM ainda se encontra em permanente evolução, essas ferramentas devem ser flexíveis de tal forma que inovações possam vir a ser agregadas a sua funcionalidade de forma simples e rápida.

1.5 Trabalhos Relacionados

Alguns sistemas de simulação têm sido desenvolvidos nos últimos anos a fim de permitir que estudos de avaliação de desempenho possam ser feitos sobre redes ATM, tais como o NIST [26], BONeS [27] e SimATM [28].

No entanto, faz-se necessário ressaltar que até a presente data, não foi encontrada na literatura ou mesmo na Internet pelo autor, qualquer referência de trabalhos que abordassem o desenvolvimento de uma biblioteca de classes para prover suporte ao desenvolvimento de simuladores de redes ATM.

Devido a este fato, esses sistemas de simulao constituem o único tipo de ferramenta de simulação possível de se estabelecer algum comparativo com esse trabalho. É importante ressaltar que essa comparação é unicamente de caráter subjetivo e sem a pretensão de estabelecer conclusões definitivas. Também deve-se observar que existem outros simuladores de redes ATM, sendo que esses foram os utilizados por serem os disponíveis durante o desenvolvimento desse trabalho.

Esses simuladores apesar de apresentarem recursos importantes, possuem diversas limitações que serão comentadas nas próximas sub-seções, devendo-se ressaltar que apenas aquelas relacionadas com as redes ATM serão discutidas. Essa comparação visa destacar, principalmente, a necessidade por um software que possa atender de forma abrangente a um conjunto maior de usuários. Não é objetivo, portanto, desta seção apresentar um estudo detalhado dos mesmos ou de suas limitações.

1.5.1 NIST

O NIST é um ambiente de modelagem e simulação desenvolvido pelo Instituto Nacional de Padrões e Tecnologia (*National Institute of Standards and Technology* - NIST) dos Estados Unidos, inicialmente com o propósito de estudar redes ATM, mas que em suas versões mais recentes (3.0 em diante) tem permitido também o estudo de redes de CATV (*Cable Television*)[26].

Esse ambiente possui um conjunto de elementos de redes ATM que permite ao usuário a construção de modelos de forma simples e rápida. Os elementos fornecidos são os seguintes: *Aplicação ATM*, *Equipamento Terminal de Banda Larga*, *Enlace* e *Comutador ATM*. Cada um desse elementos é caracterizado por um conjunto de parâmetros de entrada (fornecidos pelo usuário) e por um conjunto de parâmetros de saída (fornecidos pelo simulador durante e após a simulação).

Esses elementos estão disponíveis através de uma interface gráfica, onde é feita a criação e execução (animada) dos modelos, permitindo também ao usuário o acompanhamento da geração dos resultados graficamente.

A principal limitação relacionada ao NIST se refere à ausência de mecanismos que tratem a questão da transparência de tempo, o que impede sua utilização por quem deseja realizar qualquer estudo relacionado aos atrasos e às variações de atraso (*jitter delay*) de células. Dessa forma, sua utilização se restringe aos estudos relacionados com a questão da transparência de semântica, sobretudo no que diz respeito às perdas de células causadas por congestionamento.

Outras limitações observadas se referem ao suporte a sua instalação e utilização. Inicialmente, o usuário não dispõe de maiores informações à respeito de sua instalação, além do seu manual não fornece suporte completo ao software. Entretanto, é possível perceber que à cada versão lançadas a documentação do mesmo vem apresentando sensíveis melhorias. Um exemplo dessa limitação é que vários parâmetros que caracterizam o comutador não são explicados no manual disponível. O NIST suporta apenas alguns ambientes Unix, como o Solaris da *Sun Microsystems* e o Linux.

Por último, uma limitação que merece bastante atenção diz respeito a sua extensibilidade. O NIST foi desenvolvido utilizando a linguagem C e a abordagem estruturada[2], o que, apesar de seu manual descrever como acrescentar ou modificar seus módulos, não facilita a tarefa de estendê-lo a fim de ampliar o conjunto de elementos fornecidos e, consequentemente, sua funcionalidade.

1.5.2 BONES

O BONeS (*Block Oriented Network Simulator*) é um ambiente de modelagem e simulação de sistemas orientados a eventos desenvolvido pelo *Alta Group* da Cadence Design Systems. O BONeS pode ser usado para o estudo de vários tipos de sistemas, como por exemplo redes de comunicação de dados, arquitetura de computadores e sistemas distribuídos [27].

Assim como o NIST, o BONeS também dispõe de uma interface gráfica, onde é feita a modelagem da estrutura e da função do sistema. É possível construir o modelo hierarquicamente de forma gráfica, selecionando blocos da biblioteca de modelos ou usando módulos desenvolvidos pelo usuário. É possível especificar a funcionalidade dos módulos do usuário, usando uma combinação de três métodos: (i)*diagrama de blocos hierárquico*, (ii) *máquina de estados finitos* ou (iii) *usando código C ou C++*. Usando um destes métodos, o BONeS executa a simulação, obtém medidas de desempenho e exibe os resultados graficamente.

O BONeS consiste de vários módulos e produtos opcionais. Entre esses últimos encontra-se uma biblioteca para Redes ATM, composta de componentes como comutadores, PBXs, roteadores e fontes de tráfego (Ethernet e Vídeo).

A principal desvantagem do BONeS diz respeito ao seu elevado custo, principalmente quando comparado aos dois outros sistemas aqui abordados. Outro aspecto a ser considerado no BONeS é que ele apresenta um considerável grau de complexidade no aprendizado.

[2]A abordagem estruturada baseia-se em se descrever através de processos como os fluxos de dados são transformados dentro de um sistema desde a entrada até a saída. Sendo assim, o enfoque está nas funções que o sistema desempenha e não nas entidades que o compõem.

No que diz respeito à extensibilidade, o BONeS procura contemplar esta propriedade através de uma estrutura própria, baseada em diagramas de blocos, o que favorece apenas a extensão das capacidades do próprio simulador, ou seja apenas de sua própria biblioteca, dessa forma não permitindo o desenvolvimento de uma bibliteca de redes ATM genérica.

1.5.3 SimATM

O SimATM é um simulador orientado a eventos de redes ATM desenvolvido na Faculdade de Engenharia Elétrica e Computação (FEECOM) da Universidade Estadual de Campinas (UNICAMP) [28]. O SimATM foi desenvolvido a partir de um simulador para redes de computadores chamado SimNT [35] construído pelo *Optical Link Group* daquela mesma instituição.

Esse software foi desenvolvido utilizando a linguagem de programação orientada a objetos C++. Sua interface com o usuário é baseada em linhas de comandos. O elemento disponível para modelar redes é o *bloco*. Um bloco pode ser um equipamento (*equipamento do usuário* e *comutador*) ou um aplicativo.

A principal limitação observada no mesmo é a impossibilidade de se acrescentar ou modificar os componentes disponíveis. A razão principal é que não é fornecida uma biblioteca (como no caso do BONeS) ou o código fonte (como no caso do NIST) do simulador.

Também conforme apresentado em [28], observa-se que não foi dada a ênfase necessária à propriedade do reuso e, consequente, evolução do projeto. Isto não significa dizer, no entanto, que o mesmo não possa vir a ser estendido e atualizado, para acompanhar os avanços da tecnologia, mas que esta tarefa não será realizada com a mesma facilidade que se esta questão fosse observada desde o início.

O fato do SimATM ter sido implementado utilizando a linguagem C++ limita bastante sua portabilidade, restrigindo-o à plataforma sobre a qual o mesmo foi desenvolvido (*Microsoft Windows NT* e *95*).

-	**NIST**	**BONeS**	**SimATM**
Custo	Freeware	Alto	Freeware
Suporte	Fraco	Bom	Regular
Plataforma	Unix Solaris, Linux	Unix Solaris	MS-Windows
Transparência de Tempo	Não	Sim	Sim
Extensibilidade	Regular	Boa	Nenhuma
Específico às Redes ATM	Não	Não	Sim
Disponibilidade do Código Fonte	Aberto	Fechado	Fechado

Tabela 1.1: Comparação entre Ambientes de Simulação de Redes ATM

1.5.4 Comentários Gerais

Algumas questões bastante importantes devem ser observadas à respeito dos simuladores acima citados.

A primeira diz respeito ao fato que o alto custo de um produto comercial, apesar de todo o suporte que estes geralmente disponibilizam através de documentação (*help on-line*, manuais, etc.) e de um acabamento melhor em relação às interfaces com usuário, muitas vezes restringe sua aquisição, principalmente àqueles que realizam pesquisa em meio acadêmico. O fato de serem comerciais, impedem a disponibilidade do seu código fonte, o que inviabiliza quaisquer alterações nos mesmos a fim de acomodar funcionalidades que atendam às propostas de trabalho de um determinado grupo de pesquisa.

Também alguns desses simuladores, têm sido construído desde a etapa inicial ou então aproveitando um projeto anterior do grupo, o que leva às seguintes situações: (i) o código fonte e a documentação referente ao desenvolvimento não é disponível ao público em geral, mesmo que para pesquisas, como por exemplo o SimATM, e (ii) a metodologia utilizada durante o desenvolvimento do simulador não facilita sua modificação, extensão ou reuso, como por exemplo o NIST.

Por outro lado, vê-se que mesmo aqueles simuladores projetados utilizando uma abordagem mais apropriada a esses objetivos, tal como a orientação a objetos, não são abertos para serem estendidos e/ou não foram desenvolvidos com o objetivo de facilitar a reuso dos seus componentes (*classes*), dessa maneira dificultando a construção de

novas ferramentas.

Desta forma, é perceptível a inexistência de uma preocupação maior em se desenvolver uma estrutura única de desenvolvimento que, por exemplo, tal como uma biblioteca de entrada e saída de uma linguagem de programação qualquer, possa estar disponível amplamente para os projetistas de simuladores de redes ATM.

1.6 Relevância

A dissertação aqui apresentada contribui para o desenvolvimento eficiente e econômico de simuladores de redes ATM a partir de um conjunto de classes organizado e disponibilizado através de uma biblioteca de software.

Essas facilidades foram obtidas graças à utilização dos paradigmas da orientação a objetos (*Polimorfismo*, *Identidade*, *Classificação* e *Herança*) [20].

Para desenvolver novos simuladores de redes ATM, projetistas precisarão apenas quando necessário adicionar novas classes ou estender aquelas já existentes a fim de atender às novas soluções e adronizações que possam vir a surgir com o decorrer do tempo. Também será possível aos desenvolvedores de simuladores sua otimização, caso se deseje obter uma melhor performance em sua execução.

A disponibilidade de uma biblioteca de classes implementada em Java permite que desenvolvedores de simuladores de redes ATM explorem novos paradigmas de simulação, tais como Simulação Distribuída e Simulação Baseada em *Web* (*Web-Based Simulation*) [6].

Também, o desenvolvimento da ATMLib permitirá a extensão da biblioteca Java, fornecendo elementos que permitam a realização de estudos de avaliação de desempenho de redes ATM.

É importante também ressaltar a multidisciplinaridade deste trabalho, uma vez que o mesmo abrange várias áreas, tais como Simulação Digital, Engenharia de Software, Redes de Computadores e Metodologia de Programação.

1.7 Organização

Esta dissertação está dividida em sete capítulos e dois apêndices, sendo que os demais capítulos estão organizados da seguinte forma:

No capítulo dois são introduzidos conceitos de Simulação Digital. Através dele pode-se determinar quais componentes, parâmetros e medidas são importantes em um sistema de avaliação de desempenho.

No capítulo três são apresentados os conceitos básicos de redes ATM. Este estudo além de permitir uma descrição do domínio para o qual a ATMLib se destina, serve como ponto de partida para a especificação dos requisitos e modelagem dos objetos que compõem uma rede ATM.

No capítulo quatro é apresentada a especificação de requisitos da ATMLib. Neste capítulo são apresentadas e discutidas as decisões de desenvolvimento deste trabalho, como por exemplo metodologia de desenvolvimento e linguagem de programação, bem como as funcionalidades que a ATMLib deve apresentar ao final do desenvolvimento.

No capítulo cinco é apresentada a fase de análise da ATMLib. Neste capítulo são apresentados a estrutura e o comportamento da ATMLib através do diagrama de classes e dos diagramas de estado e interação, respectivamente.

No capítulo seis são apresentadas as fases de projeto, implementação e testes da ATMLib. Neste capítulo é apresentado de que forma o resultado da fase de análise é abordado a fim de implementá-lo juntamente com uma descrição da implementação e dos testes.

No capítulo sete são apresentadas as conclusões a respeito deste trabalho assim como sugestões de trabalhos que poderão ser desenvolvidos a partir dele.

No apêndice A são sumarizados os acrônimos usados neste trabalho de forma a auxiliar sua leitura.

No apêndice B são apresentadas as definições das interfaces e atributos de todas as classes da ATMLib.

"A ciência é a procura da verdade, não um jogo no qual uma pessoa tenta bater seus oponentes, prejudicar outras pessoas."
Linus Pauling.

Capítulo 2

Conceitos Básicos de Simulação Digital

Neste capítulo são apresentados os principais conceitos relacionados com a simulação digital. Desta forma, são destacados os diversos componentes e atributos que os sistemas de simulação devem possuir. Também, esse capítulo fornece um conjunto de conceitos fundamentais para o entendimento e o desenvolvimento desse trabalho.

Na seção 2.1 são apresentados os métodos de avaliação de desempenho existentes. Na seção 2.2 são apresentadas algumas definições básicas e a classificação dos modelos de simulação. Na seção 2.3 são apresentados alguns exemplos de medidas de desempenho relevantes e na seção 2.4 os elementos característicos de um simulador. Na seção 2.5 são apresentadas as disciplinas de escalonamento de filas. Na seção 2.7 é mostrado de que maneira a aleatoriedade é inserida em estudos de simulação. Por último, na seção 2.7 é apresentado um breve estudo da simulação orientada a objetos.

2.1 Introdução

O desempenho de um sistema computacional Aqui, entende-se por sistemas computacionais desde computadores pessoais até redes de computadores é um fator chave que

precisa ser levado em consideração durante seu projeto, desenvolvimento, configuração e refinamento [1]. A avaliação quantitativa da desempenho deste tipo de sistema é requerida durante todo seu ciclo de vida.

A razão principal disso se deve à evolução constante das tecnologias existentes e ao constante surgimento de novas. Sendo assim, para que melhorias no desempenho dos sistemas existentes ou ainda em fase de projeto possam ser realizadas, é preciso tanto conhecer seu desempenho qualitativa e quantitativamente, assim como as possíveis alternativas que possam levá-lo a um possível incremento.

Heidelberger e Lavenberg [1] dividem os métodos de avaliação de desempenho em três áreas principais, a saber: *medição de desempenho*, *solução analítica* e *simulação digital.*

A **medição de desempenho**[1] é realizada diretamente sobre o sistema em estudo através da coleta de informações que possam caracterizar o seu comportamento. Essa coleta de informações é feita instrumentando-se o sistema em estudo, ou seja, utilizando-se elementos adicionais (de *hardware* ou de *software*) que coletam informações à respeito do seu comportamento. A medição tem como desvantagem o fato de só poder ser realizada sobre sistemas já existentes, instrumentados e sendo executados.

A **solução analítica** é um método bastante aceito e utilizado devido principalmente a sua eficiência e ao seu custo relativamente baixo. Primeiro, ela é eficiente porque se baseia em um conjunto de equações fornecidas, usualmente, pela *Teoria das Filas.* Depois, ela é um método de relativo baixo custo porque não necessita de ferramentas (relativamente) caras para utilizá-la, como por exemplo, softwares de simulação.

É importante observar que para determinados sistemas serem tratáveis (ou seja, solucionáveis) analíticamente, é preciso que certas suposições sejam feitas à respeito de sua estrutura e comportamento para simplificar seu modelo. No entanto, alguns sistemas apresentam tal grau de complexidade que, para se fazer determinadas suposições a fim de simplifica-los e assim torná-los tratáveis, seus modelos podem se tornar muito discrepantes em relação aos sistema originais.

[1]Frequentemente encontra-se na literatura sendo referenciada pelo termo *benchmarking.*

A **simulação digital** é um método usado para realizar experimentos em um modelo a fim de obter medidas que caracterizam o comportamento de um sistema. Em outras palavras, pode-se também dizer que a simulação digital é o processo de representar o comportamento de um sistema, existente ou não, utilizando como ferramenta principal um programa, sendo executado em um computador.

A simulação digital é utilizada principalmente na solução de sistemas complexos, onde não é possível um tratamento analítico ou que não tenham sido construídos ainda, não sendo possível, portanto, se realizar medições.

Um dos principais benefícios da solução analítica e da simulação é o discernimento que se consegue da estrutura e do comportamento do sistema através do desenvolvimento do seu modelo. Isto pode ser particularmente útil durante o projeto do sistema, pois pode resultar na descoberta e, consequentemente, na correção antecipada de falhas no projeto.

2.2 Modelagem de Sistemas

Para que um sistema possa ser estudado utilizando a simulação digital, faz-se necessário que: (i) seja construído um modelo do sistema em questão e (ii) esse modelo seja traduzido em elementos possíveis de serem processados por algum programa (simulador) ou então seja codificado em alguma linguagem de programação para desenvolver um novo programa.

Além de possibilitar estudos de avaliação de desempenho, modelos também são importantes porque representam de forma inequívoca o entendimento dos projetistas que estão envolvidos nesta tarefa a respeito do sistema.

2.2.1 Sistemas e Modelos

Por sistema entende-se um conjunto de partes organizadas funcionalmente para formar um todo [11]. Nesse sentido, uma parte do sistema que pode ser tratada como um

sistema isolado é chamado de subsistema. Um sistema é, portanto, formado por um conjunto de subsistemas.

Um nó de comutação de uma rede de largo alcance (WAN) pode ser visto como um subsistema, enquanto a rede composta pelos diversos nós seria vista como o sistema.

Por outro lado, tem-se que um modelo é um conjunto de aproximações e suposições, ambas estruturais e quantitativas, sobre a como o sistema trabalha ou trabalhará. De uma maneira mais simples, pode-se dizer também que um modelo é uma abstração de um sistema, ou seja, é a observação das suas características mais importantes de acordo com o problema a ser resolvido e do ponto de vista de quem está modelando.

2.2.2 Classificação dos Modelos

Existem várias maneiras de se classificar um modelo de simulação. Em Kelton [12] esta classificação é organizada em três dimensões: *estático* vs. *dinâmico*, *contínuo* vs. *discreto* e *determinístico* vs. *estocástico.*

Estático vs. Dinâmico

Um modelo estático é aquele onde as mudanças de estado não envolvem tempo. Em outras palavras, o tempo que o sistema leva para ir de um estado para outro não é importante.

Um modelo dinâmico é aquele onde as mudanças de estado levam um certo tempo para serem realizadas. Nesse caso, o tempo é um parâmetro bastante importante a se considerar. A maior parte dos sistemas em operação existentes e de interesse de estudos de simulação são dinâmicos. Um exemplo pode ser uma sub-rede de comunicação, onde seu estado seria caracterizado pelo número de pacotes que ali se encontram. Para que seu estado se altere, é necessário que pacotes cheguem em determinados instantes do tempo - ou seja, leva-se algum tempo para que um novo pacote chegue, receba sua parcela de serviços, que leva uma determinada parcela de tempo para ser completada e, finalmente, parta do sistema em um outro instante.

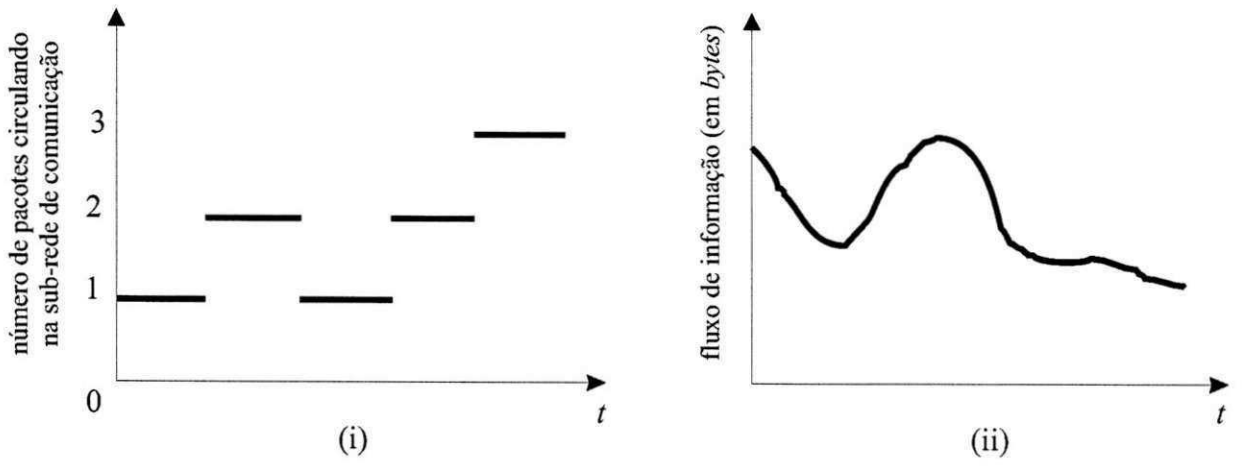

Figura 2.1: (i) - Sistema Discreto (ii) - Sistema Contínuo

Contínuo vs. Discreto

Um modelo discreto é aquele onde o estado do sistema se altera em pontos discretos do tempo. Por exemplo, observando a quantidade de pacotes circulando em uma sub-rede de comunicação, veremos que esta varia de um valor inteiro (*discreto*) para outro, conforme chegam e saem pacotes (figura 2.1.i).

Por outro lado, um modelo contínuo é aquele onde o estado do sistema muda *continuamente* com o tempo. Por exemplo, observando-se o transporte de informação em uma rede de computadores através do fluxo (*stream*) de dados - quantificados, por exemplo, através de *bytes*, vê-se que a quantidade enviada por unidade de tempo por uma determinada fonte varia de um valor para outro continuamente, sem haver mudanças bruscas, conforme pode se observar na figura 2.1.ii.

Geralmente, na prática, sistemas são compostos de algumas partes que são contínuas e de outras que são discretas. No entanto, desde que uma delas seja predominante, o mesmo possívelmente será classificado como tal [15].

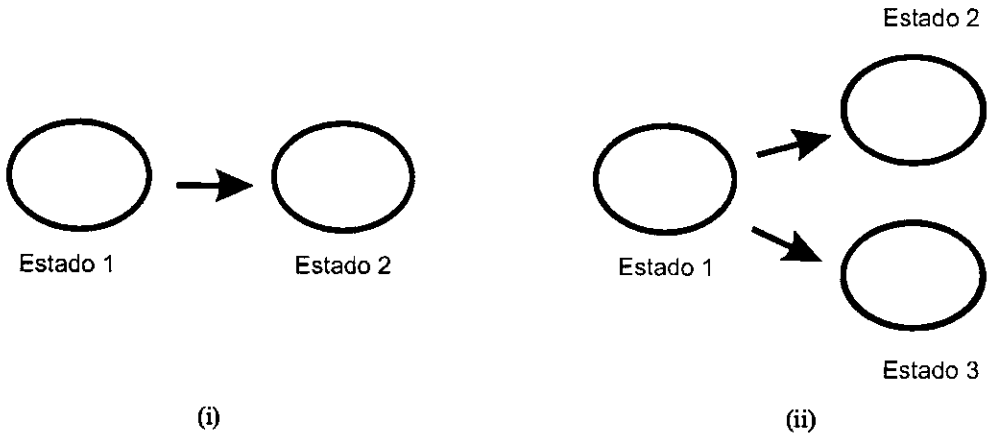

Figura 2.2: (i) - Sistema Determinístico (ii) - Sistema Estocástico

Determinístico vs. Estocástico

Um modelo determininístico é aquele onde uma entrada válida do sistema leva exatamente a uma mesma saída (figura 2.2.i). Esses modelos não apresentam variáveis aleatórias[2] [12]. Em um sistema de tráfego urbano, por exemplo, onde carros se deparam com um sinal de trânsito e há apenas uma via a seguir, quando o sinal abre, o automóvel segue sempre para o mesmo caminho (respeitando as devidas suposições!).

Um modelo estocástico é aquele onde uma entrada válida pode levar a mais de uma saída (figura 2.2.ii). Esses modelos, ao contrário dos determinísticos, apresentam variavéis aleatórias. Por exemplo, em uma agência bancária, os clientes chegam de forma aleatória no tempo e os servidores (caixas) atendem também de forma aleatória (a duração do serviço varia no decorrer do tempo). Desta forma, a cada cliente que chega, possivelmente, haverá um tempo de serviço diferente.

Modelos estocásticos são aqueles que merecem maior atenção devido à complexidade decorrente desses eventos aleatórios. Portanto, o simulador deve oferecer mecanismos que permitam modelar essa aleatoriedade.

[2]Moura et al. [11] definem uma variável aleatória como sendo "uma função que atribui um valor real a cada um dos possíveis resultados de um experimento estatístico".

É importante observar que os simuladores até então discutidos neste trabalho, assim como a ATMLib, lidam com modelos *discretos*, *dinâmicos* e *estocásticos*.

2.3 Medidas de Desempenho

Medidas de desempenho servem para expressar o comportamento de um sistema de forma quantitativa e qualitativa mediante um determinado conjunto de entradas. Essas medidas podem então variar de um sistema para outro e de acordo com o objetivo do estudo.

Por exemplo, em uma rede de comunicação de dados pode-se desejar saber o atraso médio de um pacote, o número de pacotes descartados, a fração de utilização de um roteador, etc.

No entanto, algumas dessas medidas são comuns a vários sistemas, como por exemplo [11]:

- Tempo médio de espera em fila;
- Tempo médio de atendimento em um dado servidor;
- Tempo médio de resposta do sistema;
- Número de clientes no sistema;
- Tamanho médio de uma fila;
- Tamanho máximo de uma fila;
- Fração de tempo que um servidor está livre ou ocupado;
- Status de congestionamento.

2.4 Classificação dos Simuladores

Simuladores podem ser classificados quanto aos modelos com que eles lidam (*discretos* ou *contínuos*) ou quanto à forma em que sua execução é realizada. Neste caso o simulador pode ser *orientado a eventos* ou *a processo* [12].

Um simulador é *orientado a eventos* quando sua execução está centrada nos eventos que ocorrem no sistema, em quais tarefas são acionadas nas entidades envolvidas e nos instantes de tempo em que os mesmos ocorrem. Em outras palavras, neste tipo de simulador um sistema é modelado pela definição das possíveis mudanças de estado que ocorrem no instante de cada evento e sua execução é produzida pela execução lógica associada a cada evento em uma seqüência ordenada no tempo [10]. Esse tipo de simulação estrutura-se através de um conjunto de entidades que modelam o sistema sendo estudado e de um conjunto de entidades que controlam a simulação.

Esta abordagem permite, dentre outras vantagens, a possibilidade de conhecer o estado de qualquer entidade do sistema em qualquer instante de tempo. Por outro lado, esta possibilidade torna-se bastante complexa quando o sistema sendo modelado envolve muitas entidades e muitos tipos de eventos, cada um com várias ocorrências[3].

Um simulador é *orientado a processos* quando sua execução se dá através de uma seqüencia de processos, onde cada um manipula um conjunto de eventos do mesmo tipo. Em alguns sistemas, seqüências de eventos ocorrem em padrões bem definidos, como por exemplo uma quantidade de pacotes a ser enviada através da sub-rede de comunicação. Neste caso, na simulação deste sistema, esses eventos seriam tratados conjuntamente em um mesmo processo e só se então parte para o tratamento do próximo tipo de evento. A desvantagem desses simuladores é que a seqüência em que os eventos são tratados não é a mesma do sistema real, além de acrescentar complexidade ao controle dos instantes de tempo de ocorrência dos eventos e distribuí-la entre as entidades do modelo, uma vez que não há entidades dedicadas ao controle da simulação.

[3]Conforme será visto adiante, redes ATM se inserem neste tipo de sistemas, pois devido ao tamanho de sua unidade de informação ser reduzido e alguns dos tipos de tráfego (tais como vídeo e áudio) demandarem uma grande quantidade de informação a ser enviada através da rede, uma quantidade significativa de eventos deve ser manipulada por unidade de tempo.

2.5 Elementos de um Simulador Discreto Orientado a Eventos

Um simulador deve contemplar um conjunto de elementos que permita ao usuário não apenas a construção de modelos, mas que também suporte tanto sua execução como o controle e obtenção dos valores das medidas de desempenho sendo gerados durante a simulação.

Dentre os elementos possíveis, são sumarizados abaixo alguns dos mais comumentes usados em *simuladores discretos orientados a eventos.*

2.5.1 Clientes

Um cliente (ou freguês) é uma entidade que chega a um sistema em busca de um serviço, dirige-se a um servidor para recebê-lo e, eventualmente, parte após receber ou desistir do serviço. Um cliente é caracterizado por um conjunto de atributos.

Uma entidade modelada como um cliente varia de um sistema para outro. Em uma rede de computadores, por exemplo, o cliente pode ser o pacote que busca o acesso ao meio de transmissão. Nesse caso, os atributos desse cliente poderiam ser o tempo de criaão, o tamanho e a prioridade.

2.5.2 Servidores

Servidores são entidades responsáveis por atender os clientes em suas demandas de serviço [11]. Um servidor pode possuir zero ou mais filas. No caso de haver filas, quando o cliente chega ao servidor em busca de um determinado serviço e o encontra ocupado, ele deve se dirigir à fila e aguardar por sua parcela de serviço. A ordem de atendimento às filas depende da disciplina de escalonamento que o servidor utiliza.

No exemplo anterior, o meio de transmissão em uma rede de computadores pode ser modelado como um servidor, pois os pacotes chegam até este recurso em busca de serviço (acesso ao meio de transmissão).

2.5.3 Eventos

Um evento é definido como uma ocorrência instantânea (com duração de tempo desprezível) que pode alterar o estado do sistema. Cada entidade em um sistema é composta por um conjunto de valores que a descreve em um determinado instante de tempo.

Neste tipo de simulador, faz-se necessário manter controle dos eventos que ocorrem no sistema para que nos instantes de ocorrência adequados seja possível acionar a operação correspondente.

2.5.4 Relógio Simulado

O relógio simulado é o elemento do simulador responsável pelo controle do avanço do tempo, permitindo que os eventos do sistema ocorram em sua ordem cronológica.

Em um simulador orientado a eventos, o relógio deve sempre avançar para o instante de ocorrência do próximo evento. Isto é, quando um evento ocorre no sistema, uma atividade, ou rotina, é acionada a fim de alterar o estado do sistema. Logo após isso ocorrer, o simulador deve se preocupar com o instante de ocorrência do próximo evento (chamado de *evento iminente*). Existem duas formas de adiantar o relógio para o instante de ocorrência do evento iminente.

Na primeira alternativa, o relógio é acionado em termos de eventos, não havendo dependência de uma unidade de tempo simulado (*uts*). Na outra alternativa, o relógio é incrementado em função da *uts*, sendo observados quais eventos ocorrem a cada instante.

Faz-se necessário ressaltar que não há nenhuma relação direta entre o tempo real e o tempo simulado [11]. O tempo simulado, que é descrito através de uma unidade de tempo de serviço (*uts*), representa uma certa quantidade de tempo real, como por exemplo cem nanosegundos ou duas horas. Dessa forma, o relógio é adiantado em valores múltiplos daquele definido para a *uts*. Isso permite que a simulação de uma determinada situação, que levaria vários dias para ocorrer no sistema, possa ser efetuada em segundos ou minutos. De maneira análoga, é possível ocorrer o contrário.

2.5.5 Calendário de Eventos

Conforme dito anteriormente, nesse tipo de simulador faz-se necessário criar e manter controle dos eventos ocorridos no sistema. Esse controle é feito através de uma estrutura que armazena os eventos em uma ordem cronológica.

2.5.6 Escalonador de Eventos

Em um simulador, o elemento responsável pelo acionamento adequado de uma rotina após a ocorrência de um evento é chamado de escalonador de eventos.

Com o avançar do tempo no relógio simulado, observam-se o instante de ocorrência do evento que se encontra à frente no calendário de evenbtos e o compara com o instante atual e, se for o caso, é acionada a execução das operações da entidade ao qual aquele evento aciona [10].

2.5.7 Geradores de Números e Variáveis Aleatórias

A geração de variavéis aleatórias constitui um dos elementos mais importantes de um simulador, uma vez que ele é o responsável por inserir aleatoriedade nos modelos em estudo.

Para se gerar valores para as variáveis aleatórias dois processos consecutivos devem ser realizados: *geração de números aleatórios* e *geração de valores aleatórios*

Inicialmente, é feita a *geração de números aleatórios*, que consiste em se gerar um conjunto de números equiprováveis[4]. No entanto, esses números não são realmente aleatórios, pois são gerados a partir de um número inicial chamado de *semente*. Devido a este fato são chamados de *números pseudo-aleatórios*.

No entanto, em diversas situações faz-se necessário que as variáveis aleatórias sejam caracterizadas através de outras funções de distribuição (principalmente a exponencial)

[4]Números equiprováveis são aqueles que tem a mesma probabilidade de ocorrência. Seguem uma distribuição uniforme e são gerados independentemente uns dos outros.

[11]. Sendo assim faz-se necessário converter um número aleatório em um valor aleatório de acordo com uma distribuição desejada. Para isso existem vários métodos tais como a *transformação inversa*. Este algoritmo não será apresentado aqui por questões de escopo deste trabalho, sendo que em Giozza et al. [11] são apresentados maiores detalhes sobre a geração de números e variáveis aleatórias, inclusive esse algoritmo.

Em [15] são descritas de forma detalhada as técnicas para a geração de números e de valores aleatórios.

2.5.8 Disciplinas de Atendimento

Disciplinas de atendimento são mecanismos que os servidores usam para decidir qual será o próximo cliente a ser atendido ou, no caso de haver mais de uma fila, qual fila será atendida quando o servidor estiver livre.

Abaixo são sumarizadas algumas disciplinas de escalonamento mais utilizadas em sistemas de redes de filas [15].

First Come, First Served - FCFS: O primeiro que chega à fila é o primeiro a ser atendido.

Last Come, Last Served - LCFS: O último que chega à fila, em um determinado intervalo de tempo, é o primeiro a ser atendido.

Round-and-Robin - RR: Cada um dos clientes recebe uma parcela de serviço até que se complete sua demanda total. No caso de haver mais de uma fila, cada cliente que ocupava o primeiro lugar de sua fila, recebe uma parcela de serviço até que se atinja o total desejado e parta do servidor.

Shortest Processing Time First - SPT: O cliente que demandar menor quantidade de processamento do servidor é o que será atendido.

Higher Priority First - HPR: O cliente - ou a fila, quando houver mais de uma - que tiver maior prioridade é o que será atendido.

2.6 Aleatoriedade em Simulação

Muitos dos sistemas que são estudados utilizando a técnica de simulação têm a característica da aleatoriedade [11]. Por exemplo, em um sistema bancário, os clientes chegam em instantes aleatórios e são atentidos em uma parcela de tempo também aleatória.

A razão desta aleatoriedade ocorre basicamente em função de duas razões:

- **Limitação de recursos do sistema:** Isto leva os clientes a concorrerem por recursos. Eventualmente, os recursos podem ser utilizados por um determinado cliente em um certo intervalo de tempo, levando aqueles clientes que buscam o serviço neste intervalo a desistirem ou a aguardarem em fila até que o recurso esteja disponível.
- **Ocorrência aleatória dos eventos:** As variáveis aleatórias relacionadas aos dois principais eventos de um sistema - *tempo de interchegada dos clientes* e *duração do tempo de serviço* - ocorrem de forma aleatória [11].

Para simular aleatoriedade, o simulador deve produzir valores para cada uma das variáveis aleatórias do modelo de acordo com a função de distribuição de probabilidade apropriada. Isto é feito através de dois processos consecutivos: *geração de números aleatórios* e *geração de valores aleatórios*, conforme apresentado anteriormente.

2.7 Simulação Orientada a Objetos

2.7.1 Introdução

Desde a metade da década passada, tem havido um sensível crescimento na pesquisa, desenvolvimento e aplicação de ferramentas[5] de simulação orientada a objetos (*Object Oriented Simulation* - OOS) [6].

[5]Aqui, entende-se por ferramenta tanto uma linguagem de programação quanto um ambiente de modelagem e simulação.

A razão principal para esse crescimento da SOO tem sido a crescente complexidade dos sistemas sendo modelados, como por exemplo sistemas computacionais, ambientais e de telecomunicações.

Sistemas complexos são, por natureza, maiores e mais heterogêneos que outros sistemas, além de que, frequentemente, requerem que os modelos sejam desenvolvidos e mantidos por várias pessoas. Portanto, conforme a complexidade do sistema aumenta, aumentam tambem os benefícios da utilização da SOO.

A vantagem mais tipicamente citada para se usar esta abordagem é a capacidade de modelar sistemas usando entidades que são naturais ao sistema físico. Dessa forma, por exemplo, um sistema de trânsito, que consiste de pedestres, automóveis e semáforos, pode ser diretamente representado em seu modelo por essas mesmas entidades. Isto possibilita ao modelador, dentre outras coisas, uma maior simplicidade e flexibilidade durante a concepção de seu modelo, permitindo, inclusive, decompor o sistema em subsistemas através da herança de atributos e de comportamento e daí interconectá-los a diferentes subsistemas a fim de formar novos sistemas.

Além da modelagem mais natural e, portanto, mais simples, a SOO também se propõe ao desenvolvimento mais rápido de sistemas de simulação através da utilização de modelos pré-definidos em bibliotecas. Neste caso, basta ao modelador utilizar ou, quando necessário, adicionar novos atributos ou funcionalidades a determinados elementos que compõem a biblioteca. Também é possível ao modelador criar aqueles elementos que não existem na biblioteca e, eventualmente, adicioná-lo.

2.7.2 Linguagens e Ambientes SOO

Desde o início da concepção da orientação a objetos, várias linguagens têm sido utilizadas para o desenvolvimento e solução de modelos de simulação. Para ilustrar melhor esta idéia, basta lembrar que a primeira linguagem orientada a objetos, SIMULA, foi desenvolvida objetivando a simulação digital. Desde então, muitas outras têm sido usadas para tal.

Várias linguagems de propósito geral tradicionais (não orientadas a objetos) também

Linguagem	**Linguagem Raíz**
C++	C
Classic-ADA	nenhuma
SELF	nenhuma
Smalltalk	Simula
ADA95	nenhuma
CLOS	Lisp
Eiffel	nenhuma
Modula-3	Modula-2, Mesa, Object Pascal
Objective-C	C
Java	C, C++

Tabela 2.1: Linguagens de Propósito Geral Utilizadas em Simulação

têm sido utilizadas para realizar estudos de simulação, como por exemplo FORTRAN, Pascal e C. Da mesma forma, linguagens de propósito geral para simulação estruturadas foram desenvolvidas e utilizadas largamente, como por exemplo GPSS, SIMScript e SLAM [12].

Com o desenvolvimento da abordagem orientada a objetos, surgiram linguagens voltadas para os mais diversos tipos de sistemas: gerenciamento de banco de dados, editores de textos e sistemas operacionais. Muitas dessas linguagens, no entanto, foram desenvolvidas a partir de outras já existentes. Na tabela 2.1 são apresentados alguns exemplos dessas linguagens.

Mais recentemente, surgiram as linguagens de programação de simulação orientada a objetos.

As linguagens de programação SOO como um todo podem ser classificadas de duas formas: (i)*de propósito geral* ou (ii) *específica* [6].

No primeiro caso, a linguagem pode ser utilizada para simular os mais diversos tipos de sistemas, como por exemplo linhas de montagem, redes bancárias, sistemas de trânsito, etc. Alguns exemplos são SimJava [33], Silk [39] e ModSim-III [40].

No outro caso, a linguagem é utilizada para um domínio específico de aplicação, não sendo adequada aos demais, como por exemplo G2, Taylor ED e Simple++ []. Na

Linguagem	**Linguagem Raíz**	**Técnica de Simulação**	**Domínio Típico de Aplicação**
Simple++	C++	Discreta	Fluxo de Produção Manufatura Química
Modsim-III	C++	Discreta	Simulação de Propósito Geral
Taylor ED	C++	Discreta	Fluxo de Produção Otimização do Processo de Negócios
VSE	—	Discreta	Simulação de Propósito Geral
Simjava	Java	Discreta	Simulação de Propósito Geral
Silk	Java	Discreta	Simulação de Propósito Geral
G2	C++	Discreta e Contínua	Simulação de Propósito Geral

Tabela 2.2: Linguagens de Simulação Orientadas a Objetos

tabela 2.2 são sumarizadas algumas linguagens SOO.

A vantagem das linguagens SOO em comparação com as linguagens OO de propósito geral, como por exemplo C++ e Java, é que elas favorecem na redução do tempo de desenvolvimento de modelos por prover os mecanismos de simulação (relógio, gerador de valores aleatórios, etc.) e as classes base para o seu desenvolvimento.

De forma análoga, os ambientes SOO podem ser de propósito geral e ou específicos. Existem diversos ambientes de simulação específicos a várias áreas de pesquisa. A tabela 2.3 apresenta diversos ambientes SOO para algumas áreas, assim como a linguagem em que os mesmos foram implementados.

Área de Aplicação	**Nome**	**Linguagem de Implementação**
Sistemas Computacionais	Sim286	C++
	SIGGSYS	C++
	Ptolemy	C++
	Seater	SmallTalk
	COMNET-II	ModSim-III
Sistemas Militares	IMDE	ModSim II
	OODA	ADA
	A*SIM	ADA
	IMPORT	Modula2
	SPEEDES	C++
Sistemas Ambientais	GePSi	C++
	EcoTalk	SmallTalk
	GX	SmallTalk
Sistemas de	OMOLA	C++
Manufatura	VFS-RTA	SmallTalk
Química	Gproms	Modula-2
Sistemas de	BLOCS/M	Objective-C
Manufatura	SmartSim	SmallTalk
Com Partidas Discretas	AGVTalk	SmallTalk
	CAD/MHS	SmallTalk
	OSU-CIM	SmallTalk

Tabela 2.3: Ambientes de Simulação Orientados a Objeto

"O aspecto mais triste da vida de hoje é que a ciência ganha em conhecimento mais rapidamente que a sociedade em sabedoria."
Isaac Asimov.

Capítulo 3

Redes ATM: Principais Aspectos do Domínio do Problema

Neste capítulo são apresentados os principais conceitos referentes às redes ATM a fim de permitir um maior entendimento do domínio para o qual a ATMLib se aplica.

Inicialmente, na seção 3.1, é apresentado de forma resumida o contexto em que as redes ATM surgem. O entendimento desta tecnologia e a devida compreensão de sua importância para o futuro das redes de computadores deve suceder, naturalmente, a um acompanhamento da evolução das Redes Digitais de Serviços Integrados, feito na seção 3.2. Em seguida, na seção 3.3 são apresentados os principais serviços fornecidos ou previstos pelas RDSI-FL. Na seção 3.4 é apresentado o conceito de ATM e na seção 3.5 o Modelo de Referência de Protocolos com a descrição das camadas e dos planos. Por último, na seção 3.6 são apresentadas as questões pertinentes aos Controles de Tráfego e de Congestionamento.

3.1 Introdução

As redes ATM (*Asynchronous Transfer Mode*) surgem em um cenário onde dois aspectos maiores se destacam como motivação para o seu desenvolvimento: (i) a necessidade

de uma rede única capaz de suportar diferentes fontes de tráfego adequadamente e à altas velocidades, e (ii) a necessidade por tecnologias capazes de suportar serviços que possam lidar com diferentes tipos de informação oriundas de uma mesma rede simultâneamente (sistemas multimídia).

Por outro lado, tendo sido reconhecida como uma solução técnica bastante eficiente para a comutação à altas velocidades em redes públicas de telecomunicação, nos últimos anos, o Modo de Transferência Assíncrono (*Asynchronous Transfer Mode* - ATM) tem obtido uma grande aceitação como tecnologia de transporte em ambientes locais e corporativos de alta velocidade, passando a ser assunto de inúmeras pesquisas também na área de Redes de Computadores [9].

3.2 Evolução das Redes de Telecomunicações

Desde o início dos sistemas de telecomunicações, grandes evoluções têm ocorrido em sua estrutura e funcionamento, abrangendo desde a forma de transmissão, que hoje utiliza largamente fibras ópticas, permitindo altíssimas velocidades de transporte de dados, até a forma de comutação, que passou das centrais comutadas manualmente às centrais digitais [16].

Esses sistemas, tais como telefonia, telex e comunicação de dados, apesar de possuírem algumas características em comum, foram projetados separados uns dos outros (figura 3.1). Isso fez com que cada um desses sistemas fosse desenvolvido levando em consideração apenas suas próprias características, sem levar em conta as dos demais. Assim, cada tipo de tráfego só podia ser transmitido adequadamente em sua própria estrutura de comunicação, adaptando-se mal aos outros tipos [9].

Vários aspectos negativos podem ser observados a partir desse contexto, tais como a necessidade de um usuário corporativo precisar contratar diversos serviços a, possivelmente, diferentes fornecedores para atender às diversas necessidades de comunicação de sua empresa, além da falta de otimização do uso da capacidade das infra-estruturas de comunicação. Por exemplo, enquanto a rede telefônica tem maior utilização durante o dia, diminuindo à noite a ponto de as operadoras oferecerem serviços à preços reduzidos

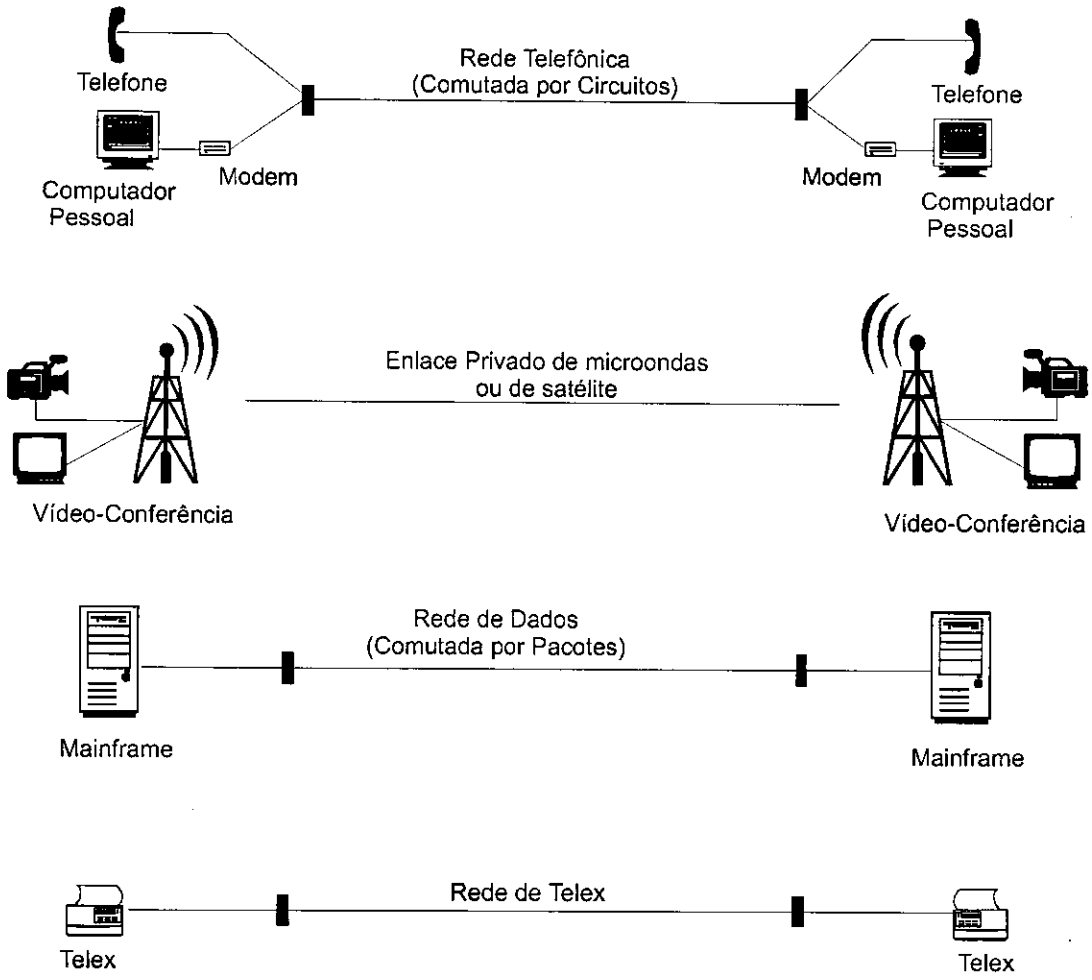

Figura 3.1: Redes de Comunicação antes da integração de serviços

neste horário, a rede de transmissão de sinais de TV é mais utilizada à noite. Desta forma, se as redes fossem integradas, nos horários em que um tipo de tráfego tivesse sua carga reduzida, a outra poderia ser mais utilizada.

O desejo de concentrar todos estes serviços em uma única rede foi o ponto de partida para que houvesse uma integração de esforços em busca de uma Rede de Serviços Integrados.

Algumas das vantagens vislumbradas com a concentração desses serviços são o uso mais eficiente dos sistemas de comunicção através do compartilhamento bem como a facilidade de gerenciamento e manutenção. Uma terceira vantagem, propiciada por estas outras duas, é a economia significativa que se pode obter com a existência dessa única rede.

Surgiram então, inicialmente, as Redes Digitais de Serviços Integrados de Faixa Estreita (RDSI-FE) e logo após, as Redes Digitais de Serviços Integrados de Faixa Larga (RDSI-FL).

3.2.1 RDSI-FE

A idéia básica por trás do conceito de Rede Digital de Serviços Integrados (RDSI-FE) é a de prover ao usuário uma interface única de transporte de dados para os diferentes tipos de fontes, além da possibilidade de acomodar outros tipos que possam vir a surgir no futuro.

O conceito de RDSI surgiu inicialmente através do CCITT (atual ITU-T) em 1972, através da recomendação G.702, com a seguinte declaração:

> " *A RDSI é uma rede digital integrada, na qual os mesmos comutadores e caminhos são usados para os diferentes serviços, como por exemplo telefonia e dados.*"

Um novo conceito mais elaborado e completo veio mais de uma decada depois (mais precisamente em 1984) como resultado das pesquisas realizadas pelo CCITT através de uma nova declaração [16]:

> *"A RDSI é uma rede, em geral evoluída da Rede Digital Integrada (RDI) de telefonia, que proporciona conectividade digital fim-a-fim para proporcionar uma variedade de serviços vocais e não-vocais, aos quais os usuários têm acesso através de um conjunto limitado de interfaces usuário-rede padronizadas."*

A RDSI, inicialmente chamada de RDSI-FE, provê serviços integrados, apesar de ainda necessitar de redes dedicadas para cada um deles, fornecendo conectividade digital para a transferência de voz, dados e imagens à baixas velocidades (figura 3.2).

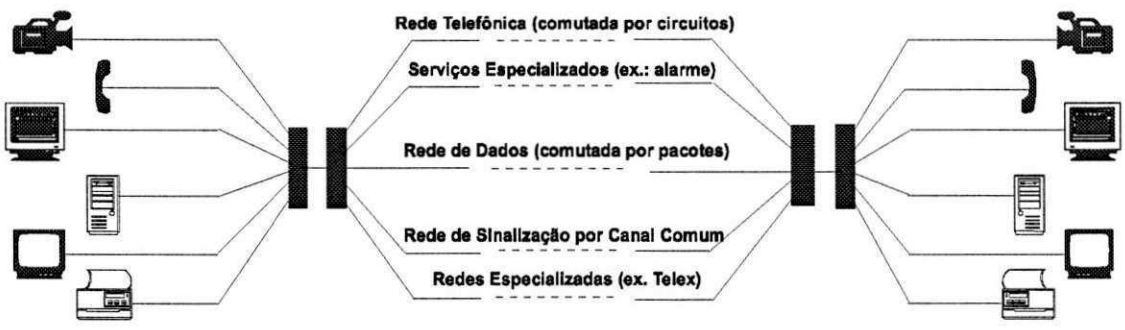

Figura 3.2: Redes Digitais de Serviços Integrados de Faixa Estreita - RDSI-FE

O modo de transferência utilizado nas RDSI-FE é o Modo de Transferência Síncrono (*STM - Synchronous Transfer Mode*).

Os padrões atuais definem dois tipos de acessos: *básico*, a uma taxa de 144 Kbps (dois canais B de 64 Kbps e um canal D de sinalização de 16 Kbps) e *primário*, com taxas correspondentes aos canais T-1 ou E-1 (1,5 ou 2 Mbps, respectivamente) [16].

3.2.2 RDSI-FL

A Rede Digital de Serviços Integrados de Faixa Larga (RDSI-FL) é a tecnologia de redes de comunicação que fornece a infra-estrutura de transporte para uma variedade de fontes de tráfego, tais como vídeo, voz e dados, em um ambiente integrado à altas

velocidades [16]. Neste tipo de redes não só o acesso é integrado, mas também toda a estrutura de transporte (figura 3.3)

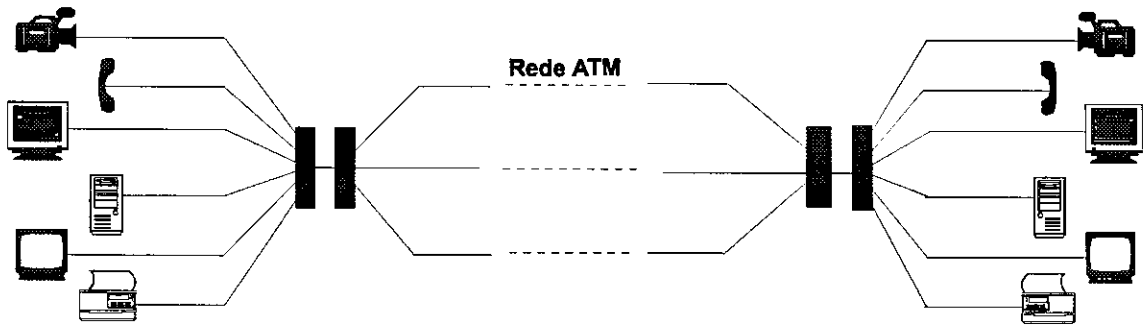

Figura 3.3: Redes Digitais de Serviços Integrados de Faixa Larga - RDSI-FL

O modo de transferência utilizado na implementação das RDSI-FL é o Modo de Transferência Assíncrono.

As altas velocidades conseguidas com esta tecnologia para a interface com o usuário (150 à 600 Mbps) permitem o surgimento e a implementação de vários serviços, tais como teleconferência, televisão de alta definição (HDTV), visualização remota de imagens, etc.

Diferente da RDSI-FE, nessa rede não apenas o acesso é único, como também há uma estrutura de transporte única. Conforme dito anteriormente, as vantagens dessA acesso unificado é a simplificação de sua gerência, além da redução significativa dos custos de implementação dos serviços que a utilizam.

A evolução para a RDSI-FL deve garantir o suporte às interfaces e serviços ora existentes. Entretanto, esta evolução deverá ocorrer ao longo do tempo, coexistindo com as redes dedicadas e com a RDSI-FE.

3.3 Serviços da RDSI-FL

O ITU-T define serviços de faixa larga como sendo aqueles que necessitam de canais com taxa de transmissão superiores àquelas do acesso primário para transmitir dados

Interativo	Conversacionais
	Transferência de Mensagens
	Consultas
Distributivo	Sem Controle do Usuário
	Com Controle do Usuário

Tabela 3.1: Classificação dos Serviços de Faixa Larga

não-convecionais, como por exemplo áudio, vídeo e imagens, de forma contínua e à altas taxas.

A classificação dos serviços, contida na recomendação I.211 do ITU-T, propõe uma divisão em duas categorias - Serviços Interativos e Serviços Distributivos, que por sua vez, subdividem-se em outras classes (tabela 3.1).

Serviços Interativos são aqueles que permitem uma transmissão bidirecional de dados entre usuário e computador e entre usuário e usuário. Pode ser conversacional, de transferência de mensagens ou de consulta.:

- **Serviços Conversacionais:** permitem que dois usuários troquem informações em tempo real sem armazenamento e retransmissão (*store-and-forward*), como por exemplo, vídeo-conferência, vídeo-telefonia, vigilância e transmissão de dados multímidia à alta velocidade.

- **Serviços de Transferência de Mensagens:** permitem uma comunicação usuário-usuário com armazenamento e retransmissão, como por exemplo, correio eletrônico de textos, voz e imagens. Estes serviços, ao contrário dos convencionais, não são realizados em tempo real.

- **Serviços de Consulta:** permitem o acesso a banco de dados remotos sob demanda. Neste caso, incluem-se aplicações tais como vídeo-texto, bibliotecas eletrônicas e vídeo sob demanda, para fins de entretenimento e de educação.

Serviços Distributivos, ao contrário dos serviços Interativos, permitem que os

dados sejam transmitidos apenas de forma unidirecional. São subdivididos em duas classes: (i) *com controle* e (ii)*sem controle do usuário.*

- **Com Controle do Usuário:** a informação é apresentada ao usuário para que o mesmo possa escolher itens separadamente e o início e a ordem em que estes serão apresentados. As aplicações deste serviço são as mídias tradicionais de recuperação de notícias (jornal, TV, revistas etc) em formato eletrônico e educação à distância.
- *Sem Controle do Usuário:* o usuário não pode controlar o início ou a ordem de apresentação das informações, como por exemplo TV por assinatura, HDTV, TV padrão, etc.

3.4 Modo de Transferência Assíncrono - ATM

O termo *modo de transferência* diz respeito aos aspectos que abrangem a transmissão, multiplexação e comutação em uma rede de telecomunicações [16].

Há dois tipos de modos de transferência: o modo de transferência síncrono (STM - *Synchronous Transfer Mode*) e o modo de transferência assíncrono (ATM - *Asynchronous Transfer Mode*).

O STM é baseado no conceito de comutação por circuitos e na multiplexação por divisão do tempo síncrona (STDM - *Synchronous Time Division Multiplexing*), onde a capacidade total do canal é alocada periodicamente a cada um dos subcanais (usuários) que os utilizam.

O ATM é baseado no conceito de comutação por pacotes (células) e na multiplexação por divisão do tempo assíncrona (ATDM - *Asynchronous Time Division Multiplexing*), onde não há alocação fixa de tempo a cada um dos subcanais. A ocupação é feita sob demanda de acordo com o tráfego em cada conexão.

Estas características conferem ao ATM determinadas vantagens, como por exemplo a flexibilidade em acomodar tráfegos variáveis ou mesmo constante, mas que utilizam apenas uma parcela da capacidade total dos canais disponíveis.

Em [16] é apresentada uma comparação mais detalhada entre o ATM e o STM.

No ambiente local/corporativo há uma demanda por redes de alta velocidade com a finalidade de se prover a interconexão de servidores às diversas redes locais, ou simplesmente à interconexão das próprias redes de uma forma quase que transparente. Neste caso, a transparência implica em um atraso de acesso a uma rede remota comparável aos atrasos de acesso obtidos em uma rede local. Para que isto seja possível, em uma rede compartilhada por um número elevado de usuários, é necessário que a taxa de transmissão seja bastante elevada [16].

O ATM baseia-se na transmissão de pequenos pacotes de tamanho fixo e formato padronizado chamados de células. Esta característica apresenta várias vantagens, tais como comutação mais simples e, portanto, mais veloz, assim como um tempo de empacotamento menor, diminuindo o retardo de transferência fim-a-fim, proporcionando um elevado nível de desempenho.

3.5 Modelo de Referência dos Protocolos da RDSI-FL

O Modelo de Referência dos Protocolos da RDSI-FL (MRP da RDSI-FL) é organizado em camadas, seguindo os princípios de comunicação em camadas da Recomendação X.200 que corresponde ao RM-OSI (*Reference Model for Open System Interconnecting*[1]) para aplicações do ITU-T [16].

O MRP consiste basicamente de três camadas (Camada Física, Camada ATM e Camada de Adaptação) e de três planos (Plano do Usuário, Plano de Controle e Plano de Gerenciamento), que dividem as funções da rede entre si - figura 3.4.

[1]Modelo de Referência para a Interconexão de Sistemas Abertos.

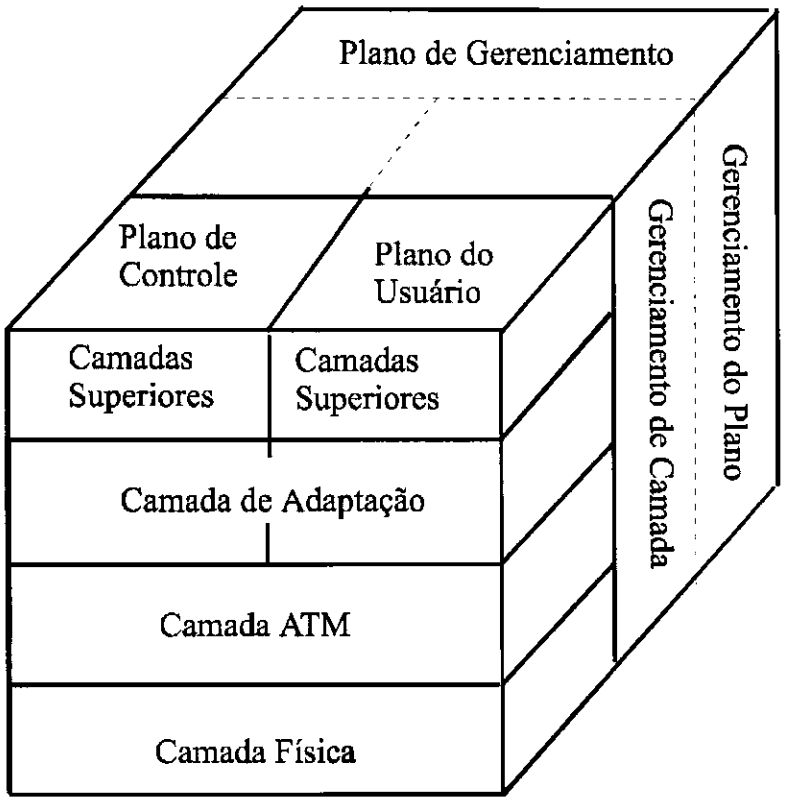

Figura 3.4: Modelo de Referência dos Protocolos da RDSI-FL

3.5.1 Camada Física

A Camada Física tem como função principal fornecer à camada ATM uma interface independente do meio de transmissão, desta forma apresentando duas subcamadas: a Subcamada de Meio Físico (*Physical Medium* - PM) e a Subcamada de Convergência de Transmissão (*Transmission Convergence* - TC).

A subcamada PM é responsável pela transmissão e recepção adequada de bits pelo meio físico, incluindo alinhamento de bits, sinalização na linha e conversão eletro-óptica.

A subcamada TC oferece um conjunto de serviços único à camada ATM, realizando as funções de desacoplamento de taxa de transmissão em relação à taxa de geração de

células, controle de erros do cabeçalho e delineamento de células.

3.5.2 Camada ATM

A camada ATM, assim como a camada física, está presente no funcionamento de todos os elementos da rede, incluindo os comutadores [9].

As funções desta camada são especificadas pela recomendação I.150 [18] e são as seguintes:

- Multiplexação e demultiplexação de células;
- Adição e remoção do cabeçalho das células;
- Chaveamento e encaminhamento de células baseado no seu cabeçalho (realizado pelos nós de comutação);
- Controle Genérico de Fluxo (*Generic Flow Control* - GFC).

3.5.3 Camada AAL

A Camada de Adaptação ATM (*ATM Adaptation Layer* - AAL) tem como função compatibilizar e oferecer os serviços desejados pelas camadas superiores, utilizando a tecnologia ATM como base e efetuando as adaptações necessárias [9].

A razão desta adaptação se deve ao fato de que a camada ATM provê um modo de transferência comum para uma variedade de serviços com características bastante diversificadas. Por exemplo, enquanto para os serviços de tráfego isócrono (CBR) é mais importante a sincronização, havendo uma certa tolerância com possíveis perdas, para os serviços de tráfegos de dados é mais importante sua integridade, havendo uma certa tolerância com os possíveis atrasos.

A AAL é a primeira camada fim-a-fim do MRP da RDSI-FL, conforme pode ser observado na figura 3.5.

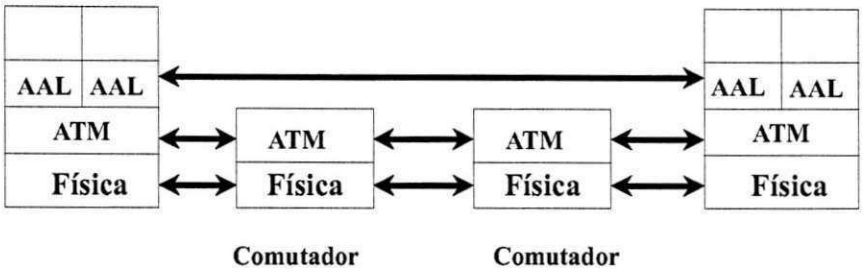

Figura 3.5: Transporte de informação através da Camada AAL

Estrutura da AAL

A AAL estrutura-se em duas subcamadas lógicas: a Subcamada de Segmentação e Remontagem (*Segmentation and Reassembly* - SAR) e a Subcamada de Convergência (*Convergence Sublayer* - CS).

A SAR responsabiliza-se pela quebra do fluxo de informações em fragmentos que podem ser acomodados no campo de informação das células na origem e pela remontagem desse fluxo a partir das células recebidas no destino. Uma mesma SAR pode ser utilizado por vários protocolos de nível superior.

A CS é responsável, dependendo do tipo de serviço, por efetuar as tarefas de multiplexação de serviços, detecção de perdas de células e recuperação da relação temporal da informação original no destino.

Classes de Serviços

A AAL utiliza a classificação dos tipos de tráfego a fim de oferecer diferentes classes de serviços e, conseqüentemente, AALs limitadas e específicas a cada uma delas. O ITU-T definiu quatro classes de serviços (Classes A à D) [9].

Classe A: É utilizada para emulação de circuitos. Aplicações que necessitam de serviços isócronos utilizam esse tipo de serviço, como a transmissão de voz e vídeo à taxas constantes (sem compressão).

Classe B: Destina-se basicamente aos tráfegos de voz e de vídeo cujas reproduções são feitas à taxas constantes, mas que podem ser codificadas com taxas variáveis através de compressão ou compactação.

Classe C: São os serviços encontrados em redes de comutação de pacotes tais como X.25, por exemplo. São serviços não-isócronos, onde a variação estatística do retardo não causa maiores problemas.

Classe D: São serviços sem conexão e com taxa variável, como por exemplo os de interconexão em redes TCP/IP.

Tipos de AAL

As AALs relacionadas às classes de serviço são basicamente quatro:

AAL 1: Efetua os procedimentos necessários para fornecer os serviços da classe A. Esses são serviços para tráfego constante (CBR), onde o sincronismo entre a origem e o destino deve ser mantido. Para que este sincronismo seja mantido é necessário que a variação do atraso seja compensada. A forma como se resolve isso é através da introdução de mecanismos de recuperação de relógio no destino.

AAL 2: Efetua os procedimentos necessários para fornecer os serviços da classe B, isto é, serviços com tráfego de taxa variável. Vídeo e áudio compactado ou comprimido são exemplos mais comuns desta classe. Atualmente ainda não existem recomendações do ITU-T sobre a operação desta AAL.

AAL 3/4: Efetua os procedimentos necessários para fornecer os serviços das classe C (com conexão) e D (sem conexão). As AALs 3 e 4 foram combinadas em uma só por apresentarem várias funções em comum;

AAL 5: Efetua os procedimentos necessários para fornecer os serviços das classes C e D, mas de forma mais simples que a AAL 3/4.

3.5.4 Plano do Usuário

O plano do usuário é responsável pela transferência de informações do usuário e do controle associado a esta transferência, tais como controle de fluxo e recuperação de erros.

3.5.5 Plano de Controle

O plano de controle é responsável pelo controle da chamada e pelas funções de controle das conexões. Ele cuida de toda a sinalização referente ao estabelecimento, supervisão e liberação de chamadas e conexões.

3.5.6 Plano de Gerenciamento

O plano de gerenciamento possui funções de gerenciamento das camadas e de gerenciamento do plano. As funções de gerenciamento do plano são aquelas que dizem respeito ao sistema como um todo e à coordenação entre os planos.

As funções de gerenciamento das camadas são aquelas que realizam a sinalização referente aos parâmetros residentes em suas entidades de protocolo, tratando dos fluxos de informação de Operação e Manutenção (*Operation And Maintenance* - OAM) específicos a cada camada.

3.6 Principais Aspectos da Camada ATM

A camada ATM desempenha papel de fundamental importância no funcionamento das redes ATM. Devido a este fato, nesta seção algumas características e funcionalidades são apresentadas em particular a fim de fornecer um maior entendimento de sua importância.

3.6.1 Formato da Célula

A Camada ATM é responsável pela definição e processamento da unidade de informação que caracteriza este tipo de rede.

A recomendação I.361 especifica o formato das células ATM e de que forma seu encaminhamento é feito na rede. Uma célula ATM possui o tamanho fixo de 53 octetos, sendo que 5 pertencem ao cabeçalho e 48 são de informações do usuário.

O formato do cabeçalho da célula irá diferir quando a transmissão estiver sendo realizada na Interface Usuário-Rede (*User-Network Interface* - UNI) - figura 3.6, ou na Interface Rede-Rede (*Network-Network Interface* - NNI) - figura 3.7. Esta diferença se resume basicamente à ausência na NNI do campo GFC e à conseqüente expansão do campo VPI.

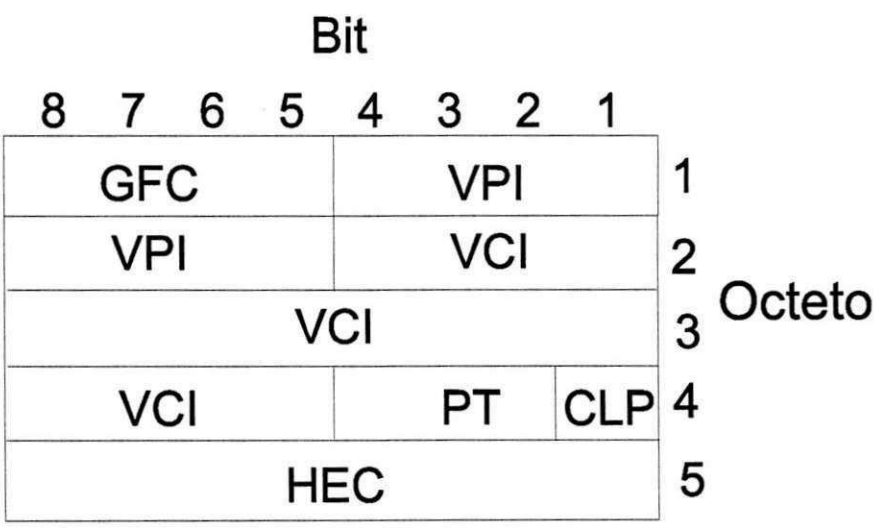

Figura 3.6: Cabeçalho de uma Célula na UNI

O campo de Controle de Fluxo Genérico (*Generic Flow Control* - GFC) é utilizado pelos mecanismos de controle de fluxo genérico, realizados apenas no acesso do usuário.

Os campos VPI (*Virtual Path Identifier*) e VCI (*Virtual Channel Identifier*) formam o rótulo da conexão utilizada pelos comutadores para encaminhar as células até o seu

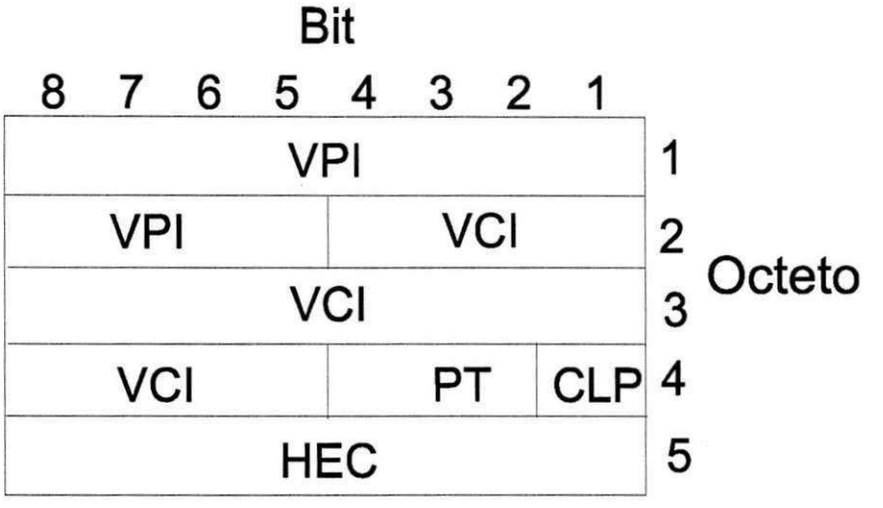

Figura 3.7: Cabeçalho de uma Célula na NNI

destino.

O campo PT (*Payload Type*) indica o tipo de informação contida no cabeçalho, ou seja, se as informações são de gerenciamento ou se são dados do usuário.

O campo CLP (*Cell Loss Priority*) indica a prioridade da célula em situações de congestionamento. Pode assumir dois valores: 0 (zero) quando a célula tem prioridade alta e 1 (um) quando tem prioridade baixa.

Por último, o campo HEC (*Header Error Control*) é utilizado para detecção e correção de de erros do cabeçalho e delineamento de células.

3.6.2 Comutação de Células

Células em uma rede ATM são transportadas através de conexões. Portanto, antes de transmitir qualquer informação do usuário é necessário que uma conexão seja estabelecida, neste caso chamada de conexão virtual comutada (*Switched Virtual Connection* - SVC), ou exista de forma permanente, chamada de conexão virtual permanente (*Permanent Virtual Connection* - PVC) [16].

Uma conexão fim-a-fim em uma rede ATM é denominada conexão de canal virtual (*Virtual Channel Connection* - VCC). Uma VCC é formada pela concatenação de conexões virtuais estabelecidas entre os vários nós de comutação entre a origem e o destino. Cada uma destas conexões é denominada de enlace de canal virtual (*Virtual Channel Link* - VCL).

Para que o comutador possa encaminhar as células corretamente até o destino é necessário que ele possua informação suficiente sobre cada uma das VCCs e, conseqüentemente, das VCLs estabelecidas através dele. Para realizar esta tarefa, o comutador utiliza de uma tabela, cujas entradas devem identificar de forma única cada conexão estabelecida. Além disso cada célula deve trazer em seu cabeçalho informações sobre por qual VCC e VCL ela foi transmitida.

Para reduzir o processamento nos nós de comutação, é comum que várias VCCs utilizem a mesma identificação e com isso sejam comutadas juntas. Desta forma, as tabelas de rotas não precisam de uma entrada para cada VCC estabelecida, mas sim para o conjunto que usa uma mesma identificação. Este conjunto de VCCs é então denominado de conexão de caminho virtual (*Virtual Path Connection* - VPC).

Desta forma, cada conexão é identificada por dois campos hierárquicos: um identificador de VPC (*Virtual Path Identifier* - VPI) e por um identificador de VCC (*Virtual Channel Identifier* - VCI). Assim, células que chegam a um comutador e pertencem a caminhos virtuais diferentes são discriminados a partir do VPI e aquelas que pertençam a um mesmo caminho, são discriminados a partir do VCI. Na figura 3.8, duas conexões de caminho virtual estão estabelecidas através do enlace, uma identificada pelo VPI *a* e a outra pelo VPI *b*. O VPI *a* é composto por duas conexões de canal virtual, identificadas pelos VCIs *1* e *2*, e o VPI *b* é composto por uma única conexão, identificada pelo VCI *1*.

Quando a comutação é feita apenas a partir do VPI, é possível reduzir o processamento do comutador que não irá precisar examinar nem mapear os VCIs deixando aos pontos terminais a responsabilidade de gerenciar os canais virtuais de acordos com suas conveniências ou necessidades [16].

Figura 3.8: Conexão de Caminho Virtual - VPC

3.6.3 Arquitetura de Serviços ATM

A tecnologia ATM é intencionada para suportar uma grande variedade de serviços e aplicações. Sendo assim, o controle do tráfego está fundamentalmente relacionado à capacidade da rede prover Qualidade de Serviço (*Quality of Service* - QoS[2]) diferenciada para as diversas aplicações da rede. Faz-se necessário, portanto, definir um conjunto de parâmetros que possam caracterizar quantitativamente e de forma não ambigua o comportamento de cada fonte de tráfego assim como a QoS que ele espera que a rede mantenha.

Esses parâmetros são negociados e acordados durante o estabelecimento de uma conexão (SVC) ou de um contrato (PVC) e servem como base para a execução das funções de controle de tráfego e de congestionamento.

A arquitetura de serviços é dividida em cinco categorias, conforme apresentado em [14]:

- Taxa de Tráfego Constante (*Constant Bit Rate* - CBR);
- Taxa de Tráfego Variável de Tempo Não Real (*non-Real Time Variable Bit Rate* - nrt-VBR);

[2]A Qualidade de Serviço (QoS) é definida na Recomendação E.800 como sendo o efeito coletivo do desempenho do serviço e que determina o grau de satisfação do usuário deste serviço. [16]

- Taxa de Tráfego Variável de Tempo Real (*Real Time Variable Bit Rate* - rt-VBR);
- Taxa de Tráfego Não Especificada (*Unspecified Bit Rate* - UBR);
- Taxa de Tráfego Disponível (*Avaliable Bit Rate* - ABR).

A seguir são descritas resumidamente cada uma dessas categorias. Maiores detalhes podems ser encontrados em [14].

Constant Bit Rate - CBR

A categoria de serviço CBR é usada por conexões que requisitam uma quantidade estática de largura de banda que deve estar disponível enquanto a conexão durar. O valor desta largura de banda é caracterizado pelo valor do parâmetro PCR (*Peak Cell Rate*), que indica a taxa máxima de envio.

A categoria CBR é intencionada para aplicações de tempo real que requerem variações de atraso bastante limitadas, como por exemplo vídeo, voz e emulação de circuito.

Real Time Variable Bit Rate - rt-VBR

Esta categoria é usada por aplicações de tempo real, ou seja, aquelas que requerem controle do atraso e da variação de atraso, mas que apresentam taxas de geração variavéis, como por exemplo vídeo MPEG[3].

Conexões rt-VBR são caracterizadas em termos de taxa de pico (*Peak Cell Rate* - PCR), taxa média a longo prazo (*Sustainable Cell Rate* - SCR) e Tamanho Máximo da Rajada (*Maximum Burts Size* - MBS).

[3]MPEG (*Motion Picture Expert Group*) é um padrão de codificação de vídeo que se baseia no conceito de *codificação diferencial*. Neste esquema, um quadro é codificado, compactado e enviado completamente em intervalos regulares até o destino. Entre esses intervalos apenas as alterações ocorridas na imagem original são enviadas [9].

Non-Real Time Variable Bit Rate - nrt-VBR

Esta categoria é intencionada para aplicações que não são de tempo real, mas que apresentam variação de tráfego e que são caracterizados em termos de PCR, SCR e MBS.

Avaliable Bit Rate - ABR

Esta categoria foi projetada para tráfegos em rajadas cuja variação da largura de banda é praticamente desconhecida, apresentando variações nos valores da taxa de geração. Para essa categoria, a rede deve fornecer um mecanismo de controle de fluxo, que informa à fonte a largura de banda atualmente disponível na rede a fim de que ela regule seu envio de células. A categoria ABR é especificada em termos de taxa de pico (*Peak Cell Rate* - PCR) e de taxa mínima de células (*Minimum Cell Rate* - MCR).

Unspecified Bit Rate - UBR

Esta categoria não apresenta garantias de *QoS*, devendo suas células serem aceitas pela rede e enviadas até seu destino desde que exista largura de banda, não sendo utilizada pelas conexões das outras categorias. Essa categoria é adequada para transmissão de pacotes IP [43], pois o IP não oferece garantias de entrega.

3.7 Controle de Tráfego e de Congestionamento

De acordo com a recomendação I.371

Ainda de acordo com o ITU-T, o controle de tráfego é dito um *controle preventivo* enquanto o controle de congestionamento é dito um *controle reativo*. Uma vez que redes ATM são redes de altas velocidades, mecanismos de controle de tráfego convencionais, tais como *janela deslizante* ou *janela saltitante* [16], não são adequados, aumentando ainda mais a necessidade de se evitar situações de congestionamento.

Os objetivos do controle de tráfego da camada ATM podem ser sumarizados como os seguintes:

- **Flexibilidade:** deve suportar um conjunto suficiente de classes de QoS para todos os serviços existentes ou que ainda possam vir a surgir.
- **Simplicidade:** o desafio é projetar controles de tráfego simples que minimizem a complexidade dos equipamentos de rede enquanto maximizam a utilização da rede.
- **Robustez:** o requisito de alcançar alta eficiência dos recursos sob qualquer circunstância de tráfego enquanto mantém funções de controle simples.

Para atingir esses objetivos, algumas funções são necessárias para controlar e gerenciar o tráfego em redes ATM, assim como as possíveis situações de congestionamento. A seguir, são descritas as duas funções mais importantes de forma resumida. Maiores detalhes podem ser vistos nos trabalhos de Prycker [8], Tanembaum [43], Monteiro [16] e do ATM Forum [14].

3.7.1 Controle de Admissão de Chamadas

Essa função representa o conjunto de ações tomadas pela rede na fase de estabelecimento da chamada (*call set-up phase*) no sentido de aceitar ou rejeitar um pedido de conexão ATM. Uma requisição de conexão para uma dada chamada é aceita somente quando recursos suficientes estão disponíveis para manter a nova conexão através da rede com a sua QoS requisitada enquanto se mantém a QoS das conexões já estabelecidas na rede. Durante esta fase um conjunto de parâmetros é negociado entre a rede e o usuário através de um contrato de tráfego.

3.7.2 Policiamento

O conhecimento da capacidade necessária à cada fonte é usado pelo mecanismo de controle de admissão de modo a decidir pela aceitação ou não de uma nova conexão com

sua QoS desejada. No entanto, esta QoS só será de fato obtida se as fontes envolvidas obedecerem aos parâmetros de tráfego especificados durante o estabelecimento da conexão. A violação dos valores negociados podem ser ou não intencionais. Os mecanismos de policiamento (UPC/NPC - *Usage/Network Parameter Control*[4]) têm a função de garantir que as funções respeitem a especificação inicial [16].

A forma como isso é feito é verificando se cada célula que chega à UNI e à NNI está de acordo com o contrato de tráfego estabelecido. Para realizar essa verificação a função de policiamento implementa um algoritmo chamado GCRA (*Generic Cell Rate Algorithm*[5] [43].

O GCRA pode ser implementado através do algoritmo de **Escalonamento Virtual** ou do **Balde Furado de Estado Contínuo** (*Continuos-State Leaky Bucket*), conforme apresentado pelo ATM Forum [14]. O GCRA depende de dois parâmetros: o incremento I e o limite de antecipação L. Devido ao fato de que neste trabalho é utilizado o BF o algoritmo de Escalonamento Virtual não será apresentado.

O *Balde Furado* é o algoritmo que tem sido mais utilizado para implementar o GCRA e é usado para definir, de maneira operacional, o relacionamento entre os parâmetros de tráfego **PCR** (*Peak Cell Rate*) e **CDVT** (*Cell Delay Variation Tolerance*) e entre **SCR** (*Sustained Cell Rate*) e **BT** (*Burst Tolerance*). O BT pode ser derivado do PCR, SCR e MBS (*Maximum Burst Size*).

No algoritmo do Balde Furado de Estado Contínuo é utilizado um BF de comprimento em unidade de tempo igual a $L + I$ e taxa de decrementação igual de uma unidade por unidade de tempo. Com a chegada da k-ésima célula no instante $t(k)$, o valor anterior do contador do BF X é atualizado, isto é, o seu valor é diminuído do tempo decorrido desde a última atualização (LCT- *Last Conformance Time*), isto é, a última vez que chegou uma célula bem comportada, e o resultado é guardado na variável auxiliar X'. Caso X' seja menor que zero, significa que o BF está vazio e X' é setado para zero. Caso contrário, se o X' for maior do que L, significa que a célula chegou antes de decorrido o intervalo aceitável (I - L) e, portanto, deve ser declarada

[4]Controle dos Parâmetros de Uso/Rede.

[5]Algoritmo de Taxa de Células Genérico.

como excessiva. Nos casos em que a célula for aceita como bem comportada é preciso atualizar o valor do contador do BF X e o instante desta atualização (LCT).

"O oposto de uma formulação correta é uma formulação falsa.
Porém, o oposto de uma verdade profunda pode ser
muito bem outra verdade profunda."
Niels Bohr

Capítulo 4

Especificação da ATMLib

Sistemas de simulação são softwares naturalmente complexos [6]. Partindo desta constatação, durante o desenvolvimento da ATMLib fez-se necessário fazer uma abordagem baseada nos princípios de Engenharia de Software, principalmente no que diz respeito à gerência do ciclo de vida do desenvolvimento da ATMLib. Neste capítulo, é apresentada a fase de especificação da ATMLib.

Na seção 4.2 são apresentadas as diversas decisões tomadas para o projeto da ATMLib. Na seção 4.3 o cenário para o qual a ATMLib se objetiva é descrito a fim de estabelecer o escopo deste trabalho. A seção 4.4 apresenta os requisitos básicos e a seção 4.5 os requisitos funcionais oferecidos pela ATMLib.

4.1 Introdução

A especificação de um software consiste em definir os objetivos gerais e específicos que guiarão seu desenvolvimento assim como os recursos necessários para sua realização. Em outras palavras, pode-se dizer que nesta fase deve se descrever bem *o que* deve ser feito e *como* fazê-lo [24].

Tendo como ponto de partida o estudo das redes ATM, conforme apresentado no capítulo anterior, e tendo observado a estrutura e as características inerentes a um

simulador discreto orientado a eventos e a uma biblioteca de classes, foram definidas os seguintes etapas a fim de especificar a ATMLib:

- **Definição das decisões de Desenvolvimento:** O desenvolvimento de um software requer um conjunto de tecnologias subjacentes que possibilitem estruturar a solução do problema. Nessa etapa são definidas a metodologia de desenvolvimento, a linguagem de programação e o ambiente de testes a ser utilizado durante o desenvolvimento deste trabalho;

- **Definição do cenário para o qual a ATMLib se aplica:** Redes ATM são redes bastante abrangentes e complexas. Uma vez que esse é um trabalho inicial e para fins acadêmicos a ser continuado através de outros trabalhos de pesquisa[1], faz-se necessário delimitar quais caracteríticas e funcionalidades destas redes serão abordadas;

- **Elaboração dos requisitos da ATMLib:** Um software possui tanto um conjunto de funcionalidades quanto um conjunto de propriedades desejáveis. Quanto mais cedo o projetista defini-los, maior será a probabilidade do projeto dar certo uma vez que será possível um melhor aproveitamento de esforços, recursos e tempo de todas as pessoas envolvidas.

É importante ressaltar que a especificação de requisitos de um software começa com a identificação e descrição dos problemas dos usuários, juntamente com uma proposta de uma provável resolução. Também é importante destacar o quão vale a pena resolver o problema em questão, uma vez que serão dispendidos esforços pessoais e recursos financeiros nesta tarefa[2].

[1] vide Capítulo 8.

[2] Essa discussão foi apresentada no capítulo 1.

4.2 Decisões de Desenvolvimento

Durante esta etapa foram definidas as questões subjacentes ao processo de desenvolvimento da ATMLib, tais como metodologia de desenvolvimento, linguagem de programação e ambientes de desenvolvimento e de testes. A razão dessa etapa é munir-se adequadamente dos recursos a serem utilizadas na abordagem do projeto.

Nas seções a seguir, são apresentadas cada uma dessas decisões juntamente com suas descrições e motivações.

4.2.1 Paradigma Adotado

Dois problemas, inicialmente, surgem na construção de qualquer sistema de software: (i) construir um sistema que atenda às necessidades do usuário, e (ii) como gerenciá-lo durante seu ciclo de vida [29]. Esses dois problemas surgem principalmente quando o projeto a ser desenvolvido é complexo e, consequentemente, demanda grupos de pessoas para projetá-lo. Sendo assim, observando-se a complexidade da ATMLib, foi escolhido o paradigma da orientação a objetos para seu desenvolvimento.

A modelagem e o projeto orientados a objetos resolvem problemas usando modelos organizados em torno de facilidades do mundo real. A construção fundamental é o objeto, que combina ambos estrutura de dados e comportamento em uma única entidade.

O paradigma da Orientação a Objetos se baseia principalmente em quatro princípios: *identidade*, *classificação*, *polimorfismo* e *herança*[20]:

Identidade: significa que os dados são quantizados em entidades discretas e distingüiveis chamadas de objetos.

Classificação: significa que objetos com a mesma estrutura de dados (*atributos*) e comportamento (*operações*) são agrupados em uma mesma *classe*. Uma classe também pode ser vista como uma abstração que descreve propriedades importantes a uma aplicação e ignora o resto.

Polimorfismo: significa que a mesma operação pode se comportar de forma diferente em classes diferentes.

Herança: é o compartilhamento de atributos e operações entre classes baseadas em um relacionamento hierárquico. Dessa forma, uma classe pode ser definida de forma abrangente e então ser refinada sucessivamente em subclasses.

Esta abordagem permite além de um gerenciamento mais simples do ciclo de vida, um meio de comunicação mais claro e eficiente entre usuário e desenvolvedor.

4.2.2 Linguagem de Modelagem Unificada - UML

Tendo-se escolhido o paradigma de desenvolvimento, o próximo passo foi a escolha da metodologia subjacente. Dentre aquelas existentes, foi escolhida uma linguagem de modelagem, a UML (*Unified Modelling Language*) [3] [4].

UML é uma linguagem de modelagem que visa unificar as notações gráficas das metodologias desenvolvidas anteriormente pelos seus autores (Grady Booch, Ivar Jacobson e James Rumbaugh). Essa linguagem vem sendo gradativamente desenvolvida por estes autores e padronizada pela OMG[3] (*Object Management Group*).

Conforme mencionado no capítulo 1, uma metodologia consiste tanto de uma *linguagem* (conjunto de estruturas que permitem descrever os diversos aspectos de um sistema) como de um *processo* (conjunto de passos que organizam essas estruturas através do desenvolvimento).

Neste trabalho, optou-se por utilizar a linguagem UML através das diversas fases de desenvolvimento, a saber: Especificação de Requisitos, Análise, Projeto, Codificação e Testes de Unidade.

Vários motivos colaboraram para essa escolha:

[3]A OMG é uma organização formada por diversas empresas com o objetivo de desenvolver e padronizar tecnologias baseadas em objetos tais como UML e CORBA. (URL: http://www.omg.com).

- A disponibilidade de vários diagramas, cada um capturando um aspecto diferente do sistema, cobrindo as diversas fases de desenvolvimento;
- A flexibilidade em se determinar quais diagramas seriam usados de acordo com as necessidades do projeto sem se prender às possiveis especificações do processo unificado (*Objectory*);
- A outra metodologia considerada - OMT [20], mostrou-se bastante limitada, principalmente no que diz respeito à fase de projeto e à ausência de um diagrama que permitisse a especificação da funcionalidade do sistema, tal como um *diagrama de use case.*
- O suporte existente em torno da UML, tal como bibliografia e comunidade de usuários, também foi um fator decisivo durante a escolha.

A seguir são descritos alguns dos diagramas fornecidos pela UML e utilizados para o desenvolvimento deste trabalho. Para maiores detalhes e outros diagramas, vide os trabalhos de Fowler [2], Rumbaugh [3] e Jacobson [4].

Diagrama de Use-Cases

Um *diagrama de use-case* descreve graficamente como ocorre uma interação típica entre um usuário e um sistema computacional. Cada uma dessas interações é chamada de *use-case*, abrangendo uma função a ser disponível ao usuário e definindo um objetivo de projeto. Dessa forma, este diagrama é bastante importante durante a fase de especificação de requisitos, onde a funcionalidade que o sistema deverá apresentar é descrita.

Para se capturar um use-case o desenvolvedor deve conversar com os prováveis usuários daquele sistema e discutir as diversas funcionalidades que eles querem do sistema. Cada uma dessas funcionalidades são separadas, recebem um nome e são descritos textualmente.

Um *diagrama de use-case* é construído a partir de um conjunto bastante simples de elementos. Um *ator* descreve um determinado tipo de usuário do sistema. É responsável

por acionar um use-case e pode ser tanto uma pessoa que desempenha algum papel em relação àquele sistema como um outro sistema. O relacionamento *uses* é usado quando uma determinada função (ou use-case) é utilizada por várias outras. O relacionamento *extends* se dá quando uma situação que não é comum ocorre, devendo haver uma função específica para tratá-la.

Diagrama de Classes

Um diagrama de classes descreve os tipos de objetos (*classes*) que compõem o sistema bem como os relacionamentos existentes entre eles [2]. Diagramas de classe também apresentam os atributos e operações de cada uma das classes e as restrições que se aplicam à forma como os objetos devem estar conectados.

Diagrama de Estados

Um diagrama de estados modela os possíveis estados que uma determinada classe pode assumir durante a execução do sistema. Cada estado é alcançado de acordo com o evento ocorrido e o estado anterior. Essa ligação entre os estados é chamada de *transição*. Cada estado tem associado a ele uma ação que é executada quando o mesmo é atingido.

Diagrama de Interação

Diagramas de interação são modelos que descrevem como grupos de objetos colaboram em algum comportamento. Essa colaboração pode ser vista como uma troca de mensagens entre as diversas classes descritas no *diagrama de classes* através de um determinado *diagrama de use-case*. Esse diagrama também é útil por permitir observar o controle de fluxo da execução de um sistema através das classes que o compõe. Pode ser de dois tipos: *Diagrama de Seqüência* ou *Diagrama de Colaboração*.

Um **diagrama de seqüência** mostra como se dá a troca de mensagens entre um conjunto de classes no decorrer do tempo em um determinado use-case. Cada classe é

apresentada através de seu tempo de vida - uma linha vertical representando seu ciclo de vida através de um determinado cénario (ou use-case).

De maneira análoga, um **diagrama de colaboração** também mostra como se dão as trocas de mensagens. No entanto, o que muda é sua notação gráfica. Enquanto em um diagrama de seqüência as trocas são mostradas através do ciclo de vida de cada objeto, junatamente com a ordem em que ocorrem, neste a seqüência das trocas é vista através de números decimais indicando a sua ordem de ocorrência.

Portanto, é possível observar que um diagrama de seqüência apresenta de forma mais natural o fluxo de execução dos objetos por mostrar tanto sua interação como sua seqüência de chamadas de métodos.

Diagrama de Pacotes

Diagramas de pacotes permitem decompor um sistema maior em um conjunto de subsistemas. Cada subsistema é descrito por um pacote de classes e as dependências existente entre elas. Cada pacote é formado por um conjunto de classes que desempenham uma função comum. Por exemplo, em um sistema de controle de vendas de passagens aéreas, as classes responsáveis pela interface com o usuário estariam em um pacote (*pacote de interface do usuário*), as classes responsáveis pelo processamento da compra em outro (*pacote processamento*) e assim por diante.

Uma relação de *dependência* existente entre dois pacotes determina que se a interface[4] de uma classe em um pacote for alterada, classes em outros pacotes necessitarão também ser alteradas. A razão disso é que uma classe envia mensagens para outra, uma classe tem outra como parte de seus atributos e uma classe menciona outra como parâmetro para uma operação. Se uma classe tem sua interface alterada, então qualquer mensagem a ela enviada pode não ser mais válida [2].

[4]Aqui o conceito de interface diz respeito às chamadas de métodos de uma classe, ou seja, sua assinatura, composta pelo modificador de visibilidade, tipo de retorno, nome do método e parâmetros. Isto é diferente da interface com o usuário, que é o conjunto de janelas exibidas a ele a fim de disponibilizar um conjunto de operações.

4.2.3 Linguagem de Programação Java

Java é uma linguagem de programação orientada a objetos desenvolvida por James Gosling [36] em 1995 na *Sun Microsystems*. Do ponto de vista do desenvolvedor de software, uma linguagem é orientada a objetos se o domínio do problema puder ser implementado como um conjunto de objetos, onde os *dados* e os *métodos* que os manipulam são vistos de forma coesa, diferente das linguagens orientadas a procedimentos, como por exemplo C e Pascal, onde *dados* e *procedimentos* são tratados de forma separada.

Esta característica permite que a implementação de um projeto desenvolvido utilizando uma metodologia orientada a objetos seja mais natural.

Além disto, Java apresenta uma série de características que a torna adequada ao desenvolvimento desse trabalho. Dentre aquelas comumente enumeradas na literatura disponível - [25], [32] e [36] - são apresentadas a seguir aquelas que levaram a uma definição em torno de Java como linguagem de implementação da ATMLib.

Simplicidade

A simplicidade em Java significa que projetos podem ser implementados rapidamente sem a necessidade de um treinamento extensivo na linguagem. A razão disto é que Java possui um conjunto bastante reduzido de instruções, além de uma sintaxe bastante similar à C e C++, o que facilita sua rápida utilização por programadores destas linguagens.

Arquitetura Neutra e Portável

Um programa escrito em Java é compilado para um formato neutro em relação à arquitetura, chamado de *bytecode*. Isto significa dizer que uma aplicação Java pode ser executada em qualquer máquina (PC, Macintosh, *Workstation*, etc.), desde que esta

possua uma Máquina Virtual Java (*Java Virtual Machine* - JVM) instalada[5] gerando os resultados do programa.

Além disso, Java garante a portabilidade da aplicação tornando explícitos os aspectos específicos da implementação da linguagem, como por exemplo o tamanho de cada tipo de dado primitivo[6], diferente de C, onde o tipo `int` pode ter 16, 32 ou 64 bits dependendo da plataforma.

Robusta

Java foi projetada para escrever código altamente confiável - ou seja, robusto [36].

A fim de atingir esse objetivo, inicialmente, foi eliminado da linguagem o uso explícito de apontadores, que em outras linguagens é uma fonte considerável de erros [32]. Java também realiza extensivamente checagem de conflitos de tipo, sendo, portanto, considerada uma *linguagem fortemente tipada.* Por último Java realiza manipulação de exceções, permitindo que o programador agrupe todo o código de verificação de erros em um só lugar, simplificando sensívelmente a tarefa de detecção e de recuperação de erros.

Desenvolvimento de Applets

Conforme apresentado em [45], Java tem também se mostrado uma linguagem bastante interessante para o desenvolvimento de simuladores discretos. Dentre as razões observadas para isso, pode-se destacar a capacidade de realizar simulações baseadas em *web*, através do uso de *applets*[7] permitindo que modelos possam ser executados

[5]Uma aplicação Java não é executada, e sim interpretada. A JVM executa duas etapas subsequentes: Na primeira é feita a **compilação** do programa fonte, produzindo um código independente de plataforma chamado *bytecode*. A segunda etapa diz respeito à **interpretação** do *bytecode*.

[6]Tipos podem ser primitivos ou por referência.

[7]Um dos primeiros usos da linguagem Java foi o desenvolvimento de pequenos programas chamados *applets*. Esses programas são embutidos em documentos HTML e então visualizados utilizando um WebBrowser. Dessa forma, é possível adicionar, dentre outras coisas, gráficos animados a estes documentos [48].

remotamente através da Internet na máquina do cliente. Dessa forma, um modelo que foi desenvolvido em uma determinada plataforma pode ser executado em qualquer uma outra que possua um *web browser* e que esteja conectado à Internet. Também é possível fazer com que essas simulações sejam animadas. Dessa forma, a execução de uma simulação pode possuir animação através de gráficos mostrando o comportamento da rede, inclusive os valores atuais das medidas de performance, como por exemplo histogramas, gráficos binários, etc.

Execução Multithreading

Java também permite que eventos concorrentes possam ser implementados diretamente através de sua capacidade de execução (*multithreading*). Isso faz com que cada evento e, consequentemente, as atividades por ele gerados possam ser executados em linhas de execução diferentes. Isso é importante, principalmente, porque simulações são aplicações onde mais de uma tarefa pode ser escalonada ao mesmo tempo, o que faz com que, eventualmente, uma delas seja executada enquanto a outra permaneça na fila de eventos (vide o capítulo 2). Desta forma, a utilização deste recurso poderia aumentar significativamente o desempenho do simulador sendo desenvolvido.

Documentação com JavaDoc

Outra característica que faz Java ser adequada para o desenvolvimento de grandes projetos é a capacidade de documentar código diretamente através do próprio software disponibilizado pela *Sun* (*javadoc*). Essa ferramenta permite a geração de páginas HTML[8] diretamente do código, através da inserção de *tags* específicas dentro dos comentários *javadoc*.

Também deve-se ressaltar que a escolha de uma linguagem de programação deve ser feita levando em consideração tanto a **familiaridade** do programador com a mesma assim como a sua **disponibilidade**. Sendo assim, estes dois aspectos influenciaram sensivelmente o autor na escolha como linguagem de programação da ATMLib.

[8]HTML: *Hiper Text Markup Language* (Linguagem de Marcação de Hipertexto).

4.3 Cenário da ATMLib

O cenário para o qual a ATMLib se insere é uma rede de longa distância (WAN), onde um ou mais usuários podem se conectar através de equipamentos terminais de banda larga - ETBL (*Broadband Terminal Equipment - BTE*) a uma rede ATM pública formada por um conjunto de comutadores que por sua vez se conecta a um ou mais equipamentos terminais, conforme pode ser visto na figura 4.1.

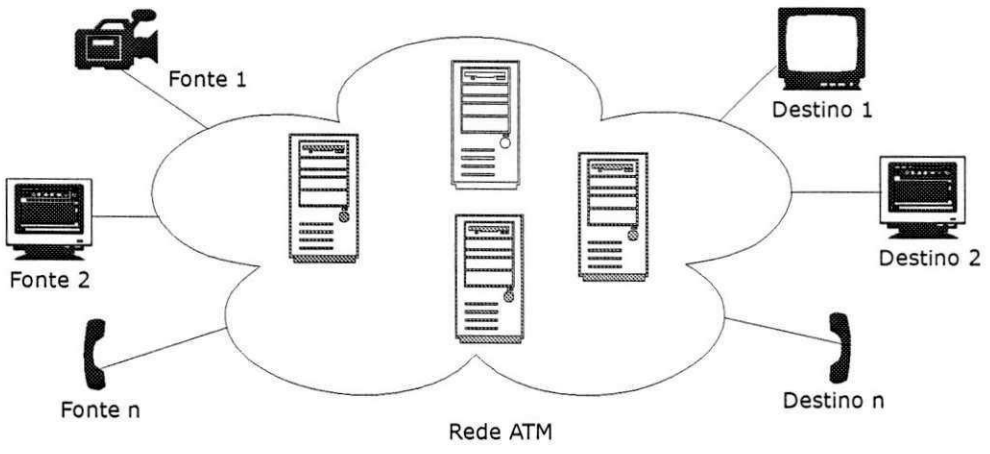

Figura 4.1: Cenário de uma rede típica a ser simulada pela ATMLib

A geração de tráfego na rede é feita através de fontes de tráfego separadas e não de aplicações multimídias. Isto significa dizer que deve haver uma única aplicação sendo executada sobre cada ETBLs. Portanto, para se simular uma aplicação multimídia, tal como uma vídeo-conferência, onde há vários processos gerando fluxos distintos de informação, o projetista deverá criar uma fonte de tráfego separada para cada um dos fluxos (vídeo, áudio, texto, etc.) e adicioná-la ao seu ETBL específico.

ETBLs modelam de forma transparente os elementos possíveis de serem conectados a uma rede ATM. Portanto, as características desse elemento são genéricas a qualquer equipamento RDSI-FL.

As conexões simuladas na ATMLib são PVCs. Isso siginifica dizer que as conexões são estabelecidas durante a configuração do modelo (ou seja, durante a entrada dos

dados que o caracterizam) e não durante sua execução. Portanto, a fase de *meta-sinalização* e *sinalização* não é tratada pela ATMLib. A razão para isso é que além de ser bastante complexa [16], a fase de sinalização propriamente dita não representa um aspecto importante para estudos de desempenho de redes ATM puras [27]. Sendo assim, a função de Controle de Admissão de Chamadas também não foi incluída.

Nas conexões da ATMLib, o fluxo do tráfego ocorre sempre em um único sentido - da origem para o destino, conforme o modelo *fonte-sorvedouro* típico de sistemas de redes de filas [11]. Nesse modelo células são geradas na origem, enviadas através da rede e recebidas no destino. Durante este percurso, são obtidas diversas informações sobre cada uma das células a fim de determinar seu desempenho. Desta forma, não há troca de informação bidirecional entre origem e destino.

Os pontos de medição (PM) de referência são os ETBLs, precisamente a camada ATM. A razão disto é que as medidas de desempenho relevantes são em relação às células. Portanto, não se faz necessário ter aplicações sendo executadas nos ETBLs destino.

A ATMLib não contempla facilidades mais avançados de redes ATM, tais como *wireless ATM*, IPOA e LAN *Emulation* por ser este ainda um trabalho embrionário, priorizando-se neste momento desenvolver seu núcleo, ou seja, as classes fundamentais para se construir modelos que permitam estudar uma rede ATM básica.

No entanto, devido a sua propriedade de estensibilidade, é possível aos usuários implementar classes que atendam a esses avanços.

4.4 Requisitos Básicos da ATMLib

4.4.1 Flexibilidade

A flexibilidade a ser oferecida pela ATMLib diz respeito a dois importantes aspectos que devem ser considerados por quem desenvolve e/ou usa uma biblioteca de software: (i) a complexidade dos aplicativos que a utilizará e (ii) a sua evolutividade.

A ATMLib deve ser flexível o suficiente para atender às necessidades de diferentes aplicações, cada uma com o seu próprio nível de complexidade. Para isto, ela deve oferecer uma API simples e abrangente.

A evolutividade da ATMLib deve ser considerada a fim de permitir que programadores que desejam acrescentar novas funcionalidades, objetivando realizar estudos que não foram previstos durante o desenvolvimento da biblioteca, possam efetuá-los sem maiores dificuldades.

Para atingir este requisito, uma alternativa natural é a utilização dos paradigmas da orientação a objetos - conforme apresentado no inicio desse capitulo, fazendo com que a ATMLib possua propriedades tais como alta coesão e independência entre as classe [23].

Bibliotecas de software que apresentam uma boa evolutividade permitem aos desenvolvedores de aplicativos uma maior produtividade no desenvolvimento destes, consequentemente, levando à economia de recursos.

4.4.2 Facilidade de Uso

A facilidade de uso é um dos requisitos mais importantes que deve ser considerado pela ATMLib. Para isso, os objetos, através de seus atributos e suas operações (*métodos*), devem ser utilizados da forma mais clara possível pelo programador durante o desenvolvimento de suas aplicações.

Além disso, a criação de objetos bem como a sua manipulação deve ser simples e de fácil utilização, mesmo para recursos mais avançados.

A fim de alcançar esse requisito, além de se buscar práticas importantes durante o desenvolvimento da biblioteca, tais como simplicidade e clareza, é essencial colocar à disposição do desenvolvedor de simuladores que irá utilizar a ATMLib um manual contendo todas as definições das classes. Também deve-se procurar documentar bem o código.

4.4.3 Portabilidade

Atualmente existe uma grande variedade de sistemas operacionais: Unix e suas variações, Windows NT-95-98, MacOs, BeOS, OS2, etc. Cada um destes utiliza um ou mais ambientes gráficos (OpenWindows, MS-Windows, etc.), que são executados sobre uma ou mais arquiteturas de hardware (*workstations*, PCs, *Macintosh*, etc.).

Toda essa variedade de recursos, apesar de ser interessante ao usuário final, pois significa um número maior de opções a sua disposição, para o desenvolvedor significa um incremento significativo de dificuldade em seu trabalho, caso deseje que sua aplicação atenda aos usuários dos vários tipos de ambientes.

A solução para esta questão deve vir através de técnicas ou ferramentas que auxiliem o desenvolvedor no sentido de minimizar seu esforço de alterar o código para cada uma das plataformas de interesse ou que possibilite que seu código seja neutro em relação à arquitetura, ambiente operacional e gráfico. Para que a ATMLib possa atender a esse requisito optou-se pela utilização da linguagem Java, cujo código produzido é independente de arquitetura, ambiente gráfico e sistema operacional [32] [36].

4.4.4 Robustez

Um programa é robusto se ele se comporta adequadamente, mesmo em circunstâncias que não foram antecipadas na especificação de requisitos, como por exemplo, quando ele encontra uma entrada de dados incorreta ou algum problema de hardware [23].

Por outro lado, observa-se que um dos maiores problemas encontrados pelos programadores é a implementação de mecanismos de tratamento de erros. Apesar de muitos erros de execução poderem ser evitados ainda em fase de projeto, através da identificação do comportamento do sistema, muitos outros não são previsíveis. Nestes casos, resta ao programador a alternativa de mecanismos oferecidos por algumas linguagens de programação, como por exemplo a manipulação de exceções [32].

Estes mecanismos permitem, por exemplo, que toda operação de entrada e saída seja acompanhada da verificação, sinalização e correção de erros.

A ATMLib deve atingir este requisito não apenas como uma questão de propiciar um funcionamento adequado, mas também para reduzir o esforço do desenvolvedor de simuladores que a utilizará na tarefa de implementar estes mecanismos, caso contrário as aplicações por ele desenvolvidas é que deveriam garantir a robustez.

4.4.5 Desempenho

O desempenho de um sistema é um requisito bastante importante, pois afeta sua usabilidade [23]. Vê-se que quando um sistema não apresenta um bom desempenho - como por exemplo, um programa que é muito lento - a produtividade do usuário é reduzida, o que o leva, eventualmente, a substituí-lo por outro sistema mais eficiente.

Este requisito é importante às aplicações de simulação, principalmente, devido ao grande número de operações que são realizadas durante a sua execução. Por isso, faz-se necessário desde o início do desenvolvimento da ATMLib buscar maneiras de se atingir este requisito.

Uma maneira é a escolha adequada dos algoritmos a serem utilizados, devendo-se optar por aqueles com maior eficiência. Pode-se ainda, a fim de auxiliar esta tarefa, utilizar perfiladores para descobrir possíveis gargalos de execução no código e assim permitir melhorias em seu desempenho. No entanto, para isso, faz-se necessário que os demais sistemas que complementam a ATMLib estejam implementados.

Uma outra maneira que pode ser observada, visto que a linguagem a utilizada no desenvolvimento é Java, é se fazer compilação *just-in-time* ou utilizar um processador PicoJava[9].

No entanto, deve-se ressaltar que sistemas de simulação de redes ATM inerentemente não apresentam níves de desempenho comparáveis com os de outros sistemas devido à vasta quantidade de eventos que são gerados e devem ser manipulados por unidade de tempo [27]. A razão desta quantidade de eventos se deve à natureza dos

[9]Processadores PicoJava são processadores cujo objetivo é acelerar a execução de programas escritos em Java.

tráfegos manipulados nestas redes, tais como vídeo, e ao tamanho reduzido da unidade de informação (*célula*).

4.4.6 Perfil do Usuário

A ATMLib destina-se a pesquisadores e planejadores de redes que desejam desenvolver ferramentas de modelagem e de simulação de modelos de Redes ATM a fim de realizar estudos de avaliação de desempenho.

4.4.7 Documentação

A documentação é considerada um item bastante relevante no desenvolvimento de um software [24]. Para isso, o codigo da ATMLib deve ser bem documentado atraves da descrição de todas as variáveis, constantes e métodos.

Devido ao recurso de criação automática de documentos HTML a partir do código fonte utilizando *Javadoc* será possível disponibilizar a API das classes através de um servidor web.

4.4.8 Disponibilidade

A ATMLib será disponível gratuitamente para fins de pesquisa através de um *site*[10]. Entretanto, será solicitado ao usuário que pretende utilizá-la o preenchimento de um formulário *on-line*. Com as informações preenchidas pelo usuário, o responsável pelo *site* - ou um aplicativo - fornecerá uma senha de acesso à página contendo a URL para fazer o *download* da biblioteca.

O preenchimento do formulário tem como objetivo procurar certificar que a biblioteca será utilizada para os fins estabelecidos (pesquisa), verificar seu grau de divulgação perante a comunidade de especialistas e obter um *feedback* dessa comunidade visando sua extensão e aprimoramento.

[10]URL: http://www.dsc.ufpb.br/marcelo.

4.5 Requisitos Funcionais da ATMLib

Uma biblioteca de suporte à simulação deve disponibilizar não apenas um conjunto de elementos que facilite a tarefa de modelagem, mas também funções que simulem e obtenham resultados que permitam uma melhor inferência a respeito do comportamento desses elementos.

No caso de simuladores específicos às redes ATM, elementos típicos destas redes, tais como aplicações ATM, ETBLs (Equipamentos Terminais de Banda Larga), comutadores, enlaces etc., devem estar disponíveis, assim como funções que permitam seu estudo de forma abrangente.

A especificação dos requisitos funcionais da ATMLib foi realizada tendo como base a literatura sobre redes ATM (vide *Referências* ao final deste trabalho). Neste sentido, um *Diagrama de Use-Case* foi elaborado a fim de expressar graficamente a interação dessas funções com seu usuário.

Conforme apresentado na figura 4.2, as funções fornecidas pela ATMLib são basicamente duas: *geração de tráfego*, que diz respeito à geração da informação pela aplicação fonte até a inserção das células na UNI de origem, e *manipulação de tráfego*, responsável pelo transporte da informação desde a UNI de origem até o ETBL destino.

Uma vez que os cálculos das medidas de desempenho são realizados desde a origem da rede até seu destino, ambas as funções citadas acima utilizam uma função em comum, que é o *cálculo das medidas de desempenho.* Devido ao fato de que cada componente possui suas próprias medidas de desempenho, faz-se necessário expandir essa função comum e definir aquelas que são adequadas a cada componente. De maneira análoga, deve-se ressaltar que essas duas funções são apresentadas em um nível maior de abstração, podendo cada uma delas ser expandida em outros níveis. Por exemplo, na função *geração de tráfego* teremos *geração de fluxo de bits*, *geração de células* e *envio de células.*

Devido à ATMLib não interagir diretamente com o usuário final, sua interação será com o *sistema de simulação*, formado por um conjunto de classes responsáveis pelo controle da simulação.

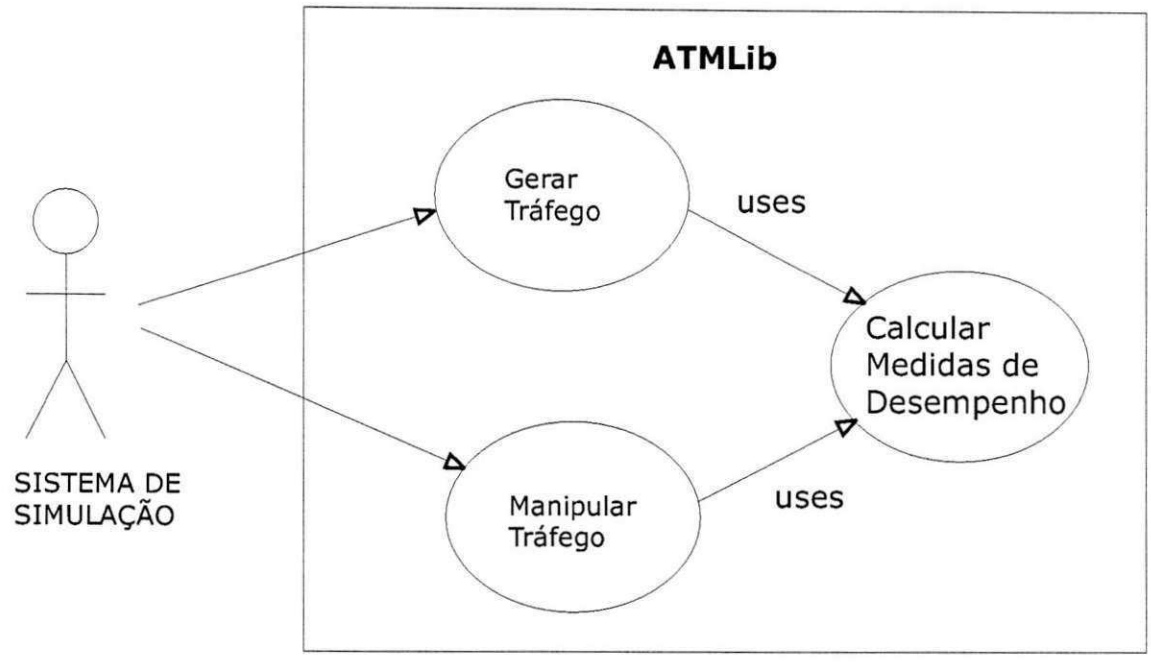

Figura 4.2: Diagrama de Use-Case da ATMLib

A seguir serão apresentadas cada uma das funções que compõem aquelas apresentadas no *diagrama de use-case*.

4.5.1 Geração de Tráfego

A *geração de tráfego* constitui uma das atividades mais importantes em uma rede ATM, pois os tráfegos gerados possuem características distintas, sendo, portanto, necessário observar o comportamento de cada uma fonte e adequá-la a um determinado tipo de tráfego, conforme especificado pelo Fórum ATM [14].

Desta maneira, a geração de tráfego na ATMLib deve apresentar essa diversidade de tráfegos a fim de prover aos desenvolvedores de simuladores facilidades naturais a este tipo de redes e, consequemente, realizar estudos mais precisos.

4.5.2 Manipulação de Tráfego

A manipulação de tráfego é acompanhada de um variado conjunto de funções. A manipulação de tráfego na rede é feita através da comutação e transmissão de células nos enlaces existentes entre os comutadores, juntamente com o cálculo das medidas de desempenho em cada um um destes componentes.

A seguir são apresentadas cada uma essas funções:

Realizar Funções de Comutação

Redes ATM são redes baseadas em circuitos virtuais. Desta forma, os comutadores da ATMLib devem ser fiéis a essa funcionalidade encaminhando células através de VPCs e de VCCs desde a origem da rede até o seu destino.

Controle de Tráfego e de Congestionamento

Mecanismos de controle de tráfego e de congestionamento devem ser fornecidos pela ATMLib a fim de verificar de que forma uma determinada rede previne e responde a uma situação de congestionamento, principalmente no que se refere à degradação dos tráfegos sendo transmitidos.

Realizar Funções de Multicast

Redes ATM possuem naturalmente a capacidade de oferecer serviços (*multicast*). No entanto, essa característica adiciona uma complexidade a mais à rede, principalmente nos comutadores.

É importante que a ATMLib disponha da facilidade de oferecer esse tipo de serviço a fim de que peculiaridades referentes ao mesmo possam ser avaliadas.

Cálculo da Fração de Perda das Células

Num ambiente de multiplexação estatística como o ATM, as células oriundas de diversas fontes competem por recursos comuns e limitados, como por exemplo o espaço nos *buffers* dos comutadores. Assim, algumas células podem ser descartadas ou perdidas. Alguns tipos de tráfegos podem tolerar um número moderado de células, enquanto que outros exigem perdas compatíveis com as disponíveis em canais dedicados [16].

Esta função deve permitir o cálculo da fração de células perdidas (CLR - *Cell Loss Ratio*) motivadas pelo descarte seletivo das funções de policiamento nos comutadores e pela limitação dos *buffers* dos ETBLs.

Cálculo do Atraso de Transferência das Células

O atraso de transferência de uma célula é definido como o tempo decorrido entre um evento *saída de célula* em um ponto de medição 1 (PM1), por exemplo ETBL origem, e o *evento entrada de célula* no ponto de medição 2 (PM2), por exemplo no ETBL destino, para uma conexão em particular.

O atraso de transferência de células entre dois pontos de medição (PM1 e PM2) é a soma do atraso de transmissão total entre a origem e o destino (valor fixo) com o atraso de processamento total nos nós de comutação existentes entre PM1 e PM2 (valor variável).

Aqui também alguns serviços são mais sensíveis do que outros, principalmente aqueles envolvendo aplicações em tempo real, como por exemplo, vídeo-conferência ou transmissão de áudio.

Cálculo da Variação de Atraso

Descreve a variabilidade do atraso de transferência de células de uma determinada conexão. Esta variação no atraso de transferência das células é causado por diversas características da rede ATM. Por exemplo, quando duas ou mais conexões são multiplexadas, células de uma determinada conexão são atrasadas (por enfileiramento)

enquanto que células de outra conexão são inseridas na saída [16]. Quando cada uma das células chegar ao destino elas terão sofrido atrasos diferentes uma quantidade de tempo diferente o que pode prejudicar sensívelmente a qualidade de serviço de determinadas aplicações, sobretudo aquelas envolvendo comunicação em áudio em tempo real, onde o sicronismo é fundamental [9].

Cálculo do Fator de Utilização do Enlace

A alocação ideal dos recursos de uma rede é um fator chave para o seu correto dimensionamento, permitindo um uso eficiente por parte dos usuários e diminuindo custos por parte dos provedores de serviços.

Por conseqüinte, faz-se necessário que recursos tais como os enlaces que conectam os comutadores possam ter sua utilização monitorada.

Cálculo do Tamanho Médio de Fila

O tamanho médio da fila é uma medida relevante porque permite avaliar o quanto um determinado *buffer* está sendo ocupado e assim efetuar o dimensionamento apropriado desse recurso, impedindo que células sejam descartadas desnecessariamente.

A ATMLib deve portanto fornecer tanto essa facilidade como a possibilidade de alterar o tamanho do *buffer* de cada comutador e ETBL.

Identificação de Congestionamento nos Comutadores

O congestionamento é a principal causa de degradação da QoS de uma rede. Por isso é fundamental que uma vez a rede esteja configurada, seja possível identificar situações de congestionamento e assim identificar suas causas e, eventualmente, redimensionar a rede.

Requisito	Não Será Satisfeito	Parcialmente Satisfeito	Totalmente Satisfeito
Flexibilidade			***
Facilidade de Uso			***
Portabilidade	+++		***
Robustez			***
Desempenho	***	+++	
Documentação			***
Disponibilidade			***

Tabela 4.1: Matriz de Requisitos da ATMLib

4.6 Considerações sobre a Especificação de Requisitos

É importante observar que o conjunto de funcionalidades apresentado não compreende a totalidade das funcionalidades possíveis para um sistema de simulação de redes ATM, mas apresenta um número significativo de operações que permite realizar vários estudos relacionados com seu desempenho.

Também deve-se observar que devido à natureza desse trabalho, alguns requisitos básicos não serão atendidos em sua totalidade. A fim de observar melhor esta questão é apresentada na tabela 4.1 uma matriz de requisitos.

Apesar de que a flexibilidade deverá ser um requisito satisfeito principalmente devido ao paradigma de desenvolvimento adotado, recursos modernos que permitiriam um incremento significativo em sua flexibilidade, como por exemplo a utilização de *Padrões de Projeto* [34], não são aplicados neste trabalho por questões de limitação dos objetivos do trabalho.

O desempenho da ATMLib poderá ser parcialmente ou totalmente satisfeito dependendo da abordagem utilizada durante a implementação. No caso de se utilizar compilação *Just-In-Time*, o código terá um desempenho significativamente melhor - comparável a qualquer linguagem compilada como C ou C++ [50]. No entanto, neste caso haverá perda da portabilidade. No caso de se fazer compilação intermediária - ou seja, gerar um código neutro - e daí interpretá-lo, o desempenho será reduzido, no

entanto mantendo a portabilidade.

De maneira análoga, a portabilidade é um requisito que pode ser totalmente satisfeito ou não ser satisfeito dependendo se o código gerado é interpretado ou compilado. Quando é compilado perde-se toda a portabilidade e quando é interpretado o código pode ser portado para os ambientes que a JVM seja disponível.

"O meio de combater uma idéia é lançar ao seu encontro uma melhor.
Nunca no mundo uma bala matou uma idéia"
Monteiro Lobato

Capítulo 5

Análise da ATMLib

Neste capítulo é apresentada a fase de desenvolvimento da ATMLib referente à modelagem de sua estrutura bem como de seu comportamento e funcionalidade. Neste sentido, um conjunto de diagramas é apresentado descrevendo individualmente cada um destes aspectos.

Na seção 5.1 é apresentado o *diagrama de classes* de uma rede ATM demonstrando a estrutura individual de cada uma das classes e as associações e restrições existentes entre as mesmas. Posteriormente, na seção 5.2 são apresentados os *diagramas de estados* de cada um dos objetos que possuem comportamento dinâmico significativo. Por último, na seção 5.3, são apresentados os *diagramas de interação* referentes aos *use-cases* apresentados no capítulo anterior.

5.1 Diagrama de Classes

Um diagrama de classes descreve os tipos de objetos (*classes*) de um sistema em estudo, assim como os relacionamentos existentes entre eles [2]. Diagramas de classes também apresentam os atributos e operações de cada classe assim como as restrições que se aplicam à forma como os objetos devem estar conectados.

Tendo-se realizado o estudo sobre redes ATM - apresentado no capítulo 3 - e tendo-se

como objetivo sua avaliação de desempenho, foi obtido seu diagrama de classes - figura 5.1. Neste diagrama, cada classe é apresentada através de seus atributos e métodos (operações), separados através de uma linha. Algumas classes possuem métodos ou atributos, como por exemplo a classe *ETBL*. Também as associações e quantificadores entre as classes são apresentados, como por exemplo um *ETBL* possui uma *fila*.

5.2 Descrição das Classes

Nesta seção são descritas individualmente cada uma classes que compõem o diagrama apresentado na figura 5.1. Nesta descrição, apresentamos a definição, os atributos e as operações de cada uma delas.

5.2.1 Fonte de Tráfego

Descrição

Esta classe representa uma aplicação sendo executada em um equipamento terminal de banda larga (ETBL) - como por exemplo uma fonte de vídeo de alta definição (HDTV) ou um servidor de áudio - e que gera tráfego de acordo com as classes de serviço especificadas pelo Fórum ATM [14]. Conforme apresentado no capítulo anterior - seção 4.6, cada fonte de tráfego gera informação para uma única classe de serviço.

Atributos

- **nome da aplicação fonte:** identifica o componente em um modelo de rede ATM. Esse atributo é útil ao usuário durante a modelagem e a realização de experimentos de simulação. Entretanto, durante a execução, entre as instâncias do sistema, esse atributo não é utilizado.
- **número de bits total:** representa a quantidade de bits que o usuário deseja que a aplicação gere em uma determinada simulação.

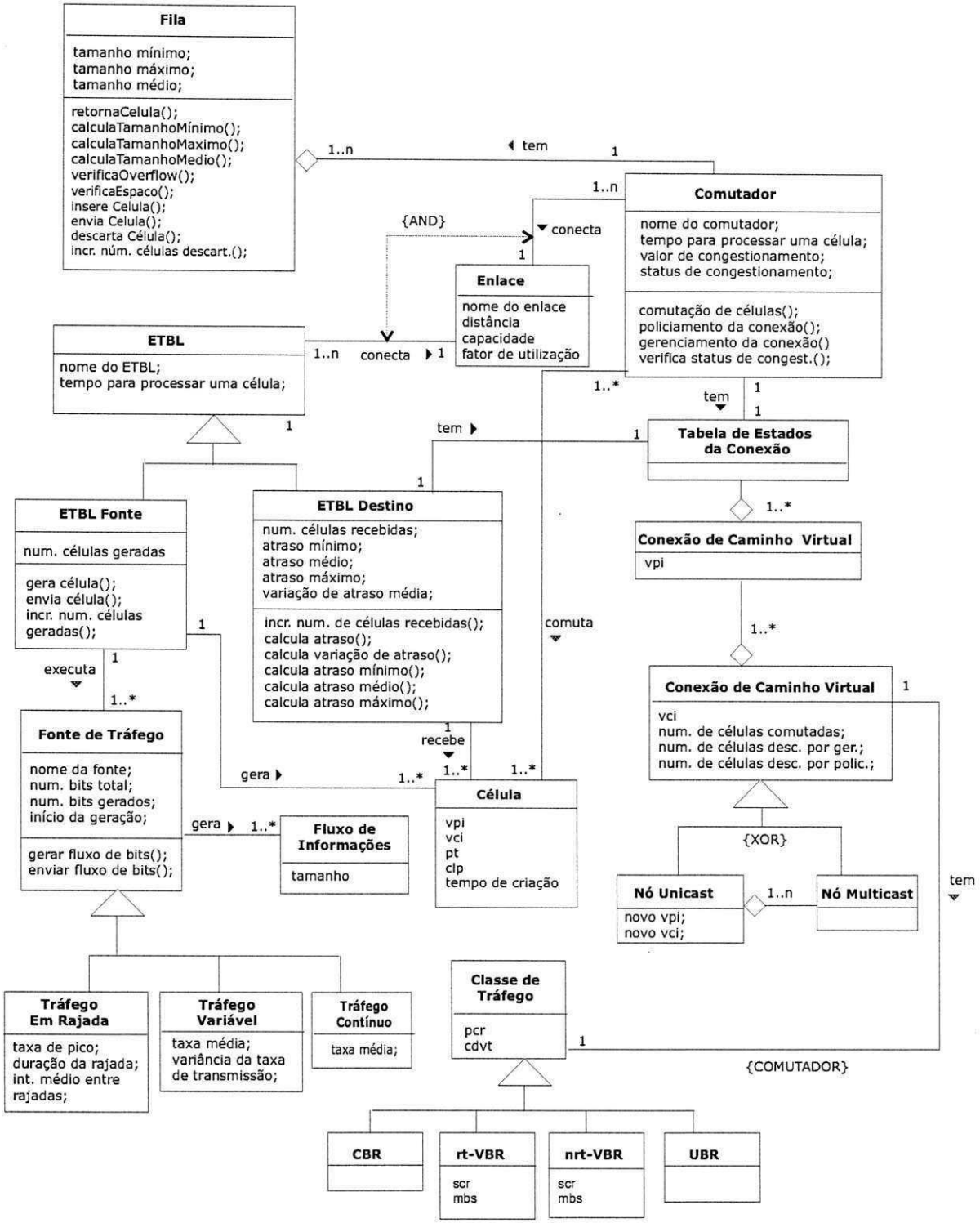

Figura 5.1: Diagrama de Classes de uma Rede ATM

- **número de bits gerados:** representa a quantidade de bits gerados pela aplicação até um determinado instante de tempo.
- **início da geração:** representa o instante em que a aplicação iniciará a geração de tráfego. Esse valor pode ser maior ou igual ao valor do início da simulação do modelo.

Operações

- **gerar fluxo de bits:** Gera um fluxo de bits que será enviado para o *ETBL* sobre o qual a fonte está sendo executada.
- **enviar fluxo de bits:** Envia um fluxo de bits para o *ETBL* sobre o qual a fonte está sendo executada.

5.2.2 Tráfego Contínuo

Descrição

Representa aquelas fontes de tráfego cuja geração e transmissão de informação é contínua, como por exemplo vídeo sem compressão, áudio sem detecção de silêncio, etc.

Atributos

- **taxa média:** representa a taxa enviada ao longo da duração em bits por segundo.

5.2.3 Tráfego em Rajada

Descrição

Representa aquelas fontes de tráfego que intercalam períodos de atividade (durante os quais transmitem à taxa de pico e de inatividade (durante os quais permanecem em silêncio). Os exemplos mais comuns desse tipo de fonte são a voz digitalizada com detecção de silêncio e a trasnmissão de texto [9].

Atributos

- **taxa de pico:** representa a quantidade máxima de bits que será enviada por segundo em uma rajada.
- **duração da rajada:** representa o tamanho do período de atividade da fonte expresso em segundos.
- **intervalo médio entre rajadas:** representa o período de inatividade da fonte expresso em segundos.

5.2.4 Tráfego Variável

Descrição

Representa aquelas fontes cuja geração apresenta variação ao longo do tempo como por exemplo a transmissão de vídeo utilizando codificação diferencial [16].

Atributos

- **taxa média:** Representa a taxa média de geração ao longo de uma chamada.
- **variância da taxa de transmissão:** Representa o valor da variância (*variação*) do tráfego durante a chamada.

5.2.5 Fluxo de Informação

Descrição

Representa o fluxo de informação gerado por uma aplicação fonte durante uma chamada.

Atributos

- **Tamanho:** representa a quantidade de bits enviada pela aplicação à camada AAL de um ETBL. Esse fluxo será posteriormente enviado à camada ATM e transformado em células.

5.2.6 Equipamento Terminal de Banda Larga

Descrição

Esta classe representa um equipamento terminal de banda larga genérico em uma rede ATM.

Atributos

- **nome do ETBL:** identifica o componente em um modelo de rede ATM. Esse atributo será útil ao usuário durante a modelagem e a realização de experimentos de simulação. Entretanto, durante a execução, entre as instâncias do sistema, esse atributo não é utilizado.
- **tempo para processar uma célula:** Representa o intervalo de tempo que um equipamento leva para processar uma unidade de informação. No caso de um *ETBL fonte*, é o tempo transcorrido desde a criação de uma célula até seu envio na porta de saída. No caso de um *ETBL destino*, é o tempo transcorrido desde a chegada da célula na porta de entrada até o momento de processamento pela camada ATM. Esse valor é constante e é informado pelo usuário durante a configuração do modelo a ser simulado.

5.2.7 Equipamento Terminal de Banda Larga - Fonte

Descrição

Representa o **ETBL** responsável pela geração de tráfego na rede e, consequentemente, sua inserção na rede.

Operações

- **gera células:** Gera uma célula a partir de um fluxo enviado por uma fonte de tráfego.
- **envia células:** Envia uma célula para o comutador diretamente conectado através de um enlace a fim de que a mesma chegue até o ETBL destino.
- **incrementa número de células geradas:** Conta o número de células que foram descartadas.

5.2.8 Equipamento Terminal de Banda Larga - Destino

Descrição

Um ETBL de destino é responsável pela recepção das células na outra extremidade da conexão.

Atributos

- **número de células recebidas:** Representa o número de células recebidas pelo ETBL.
- **número de células para calcular variação de atraso:** Representa o número de células que são utilizadas para calcular a variação de atraso.

- **atraso mínimo:** Representa o atraso médio calculado sobre todas as células recebidas.
- **atraso médio:** Representa o atraso médio calculado sobre todas as células recebidas.
- **atraso máximo:** Representa o atraso máximo calculado sobre todas as células recebidas.
- **variação de atraso média:** Representa o valor médio entre um conjunto de tempos de chegada de um determinado número células. Cada valor individual é calculado tomando como base um valor de referência. Esse valor é o tempo ideal de chegada, ou seja, o tempo que uma célula leva para ser transmitida desde a origem até o destino sem ser enfileirada.

Operações

- **incrementa número de células recebidas:** Conta o número de células recebidas por um ETBL. Disponível apenas no ETBL destino.
- **calcula atraso:** Calcula o atraso de cada célula tendo como referência o instante marcado para sua chegada.
- **calcula atraso mínimo:** Calcula o atraso mínimo entre todas as células de uma determinada conexão.
- **calcula atraso máximo de uma conexão:** Calcula o atraso máximo de cada conexão.
- **calcula atraso médio:** Calcula o atraso médio de cada conexão.
- **calcula variação de atraso:** Calcula a variação de atraso existente entre as células de uma determinada conexão.

5.2.9 Célula

Descrição

Representa a unidade de informação trocada entre os componentes de uma rede ATM. Quando uma célula é criada por um ETBL, o momento de sua criação é armazenado para que no momento de sua chegada no ETBL destino seu atraso e variação de atraso do fluxo a qual ela pertence possa ser calculado.

Atributos

- **vpi:** Representa o campo vpi (*virtual path identifier*). É utilizado para identificar um conjunto de conexões de canal virtual (VCCs).
- **vci:** Representa o campo vci (*virtual channel identifier*) e é utilizado para identificar uma determinada VCC.
- **pt:** Representa o campo pt (*payload type*) e indica se a célula é do usuário (valor 1), sinalização (valor 2) ou gerência (valor 3). Nesse trabalho apenas células do usuário serão utilizadas.
- **clp:** Representa o campo clp (*cell loss priority*) da célula e indica sua prioridade durante períodos de congestionamentos nos ETBLs e no(s) comutador(es) da rede, onde se faz necessário realizar o descarte de células que não estiverem de acordo com os parâmetros definidos na configuração da conexão.
- **instante de criação:** Representa o instante do tempo em que a célula é criada. Esse instante deve ser informado pelo ETBL e obtido através de um relógio simulado.

Os valores desses atributos não são informados pelo usuário do simulador durante a criação de um modelo, mas definidos pela aplicação durante o estabelecimento de cada conexão. Por exemplo, em uma fonte de tráfego CBR, o campo PT terá valor igual a 1 e o campo CLP igual a 0. Os campos VPI e VCI têm seus valores definidos internamente pelo simulador de forma transparente ao usuário.

5.2.10 Classe de Tráfego

Descrição

Essa classe representa os tipos de tráfegos atualmente suportados pelas redes ATM [14]. Uma classe de tráfego pode ser de um dos seguintes tipos: CBR, rt-VBR, nrt-VBR ou UBR [1].

Atributos

- **cdvt:** Representa o parâmetro CDVT (*Cell Delay Variation Tolerance*), responsável pela definição da quantidade de atraso tolerável pela fonte de tráfego.
- **pcr:** Representa o parâmetro PCR (*Peak Cell Rate*) e identifica a taxa de pico em que uma determinada fonte poderá enviar tráfego.

5.2.11 rt-VBR e nrt-VBR

Descrição

Estas classes são extensões à classe *Classe de Tráfego* e representam as classes rt-VBR e nrt-VBR, respectivamente. Ambas possuem os mesmos tipos de atributos.

Atributos

- **scr:** Representa o parâmetro *Sustained Cell Rate* e identifica a taxa média a longo prazo de células que uma determinada fonte enviará.
- **mbs:** Representa o parâmetro *Maximum Burst Size*.

[1] A classe de tráfego ABR em função de sua significativa complexidade em relação às demais foi deixada como objeto de estudo futuro. Vide Capítulo 8.

5.2.12 CBR e UBR

Descrição

Estas classes também são extensões à *Classe de Tráfego* e representam as classes CBR e UBR, respectivamente. Ambas não possuem atributos, exceto aqueles herdados de *Classe de Tráfego.*

5.2.13 Enlace

Descrição

Representa o componente de uma rede ATM responsável pelo transporte de informação entre dois componentes diretamente conectados (adjacentes), como por exemplo uma fibra óptica ou um cabo coaxial.

Atributos

- **nome do enlace:** Identifica o componente em um determinado modelo de rede ATM. Essa identificação é útil ao usuário do simulador que for desenvolvido utilizando a ATMLib. Durante sua execução, no entanto, esse atributo não é utilizado.
- **distância (em quilômetros):** Representa a distância existente entre os dois componentes que se conectam através do enlace (ETBL-Comutador ou Comutador-ETBL).
- **capacidade:** Representa a capacidade de transferência do enlace.
- **fator de utilização:** Representa a parcela de tempo em que o enlace permanece ocupado durante uma simulação.

5.2.14 Comutador

Descrição

Representa o componente de uma rede ATM responsável pela comutação de células através de vários enlaces de canais virtuais que chegam por um conjunto de portas de entrada e saem por um outro conjunto de portas de saída.

Atributos

- **nome do comutador:** identifica o componente em um modelo de rede ATM. Esse atributo é útil ao usuário durante a modelagem e a realização de experimentos de simulação. Entretanto, durante a execução, entre as instâncias do sistema, esse atributo não é utilizado.
- **tempo para processar uma célula:** Representa o tempo que o comutador leva para retirar uma célula da porta de entrada e colocar na porta de saída diretamente. Não leva em consideração o tempo de permanência em fila, pois este é um valor aleatório e é calculado durante a simulação.
- **valor de congestionamento:** Quando todas as filas atingirem um valor maior do que o atribuído, o comutador será considerado congestionado.
- **status de congestionamento:** Quando o comutador estiver congestionado esse indicador assumirá um valor representando essa condição (*verdade*) e quando não estiver, assumirá um outro (*falso*).

Operações

- **Comutação de Células:** Recebe uma célula na porta de entrada e entrega ao *gerenciador de filas*. Para isso o comutador localiza a conexão à qual pertence a célula através de seu campo VPI dentro de uma tabela de estados da conexão, verifica se é terminal, executa as funções de policiamento e, se necessário, as de gerenciamento. Por último, insere um novo cabeçalho na célula.

- **Policiamento da Conexão:** Verifica se a célula está de acordo com as características de tráfego definidas durante o estabelecimento da conexão na criação do modelo.
- **Gerenciamento da Conexão:** Atualiza as medidas de desempenho de cada uma das conexões.
- **Verifica Status de Congestionamento:** Verifica se o Comutador está congestionado.

5.2.15 Tabela de Estados da Conexão

Descrição

Representa a tabela que é utilizada pelos comutadores para realizar a comutação de células e foi modelada a partir do projeto do COMATM (Comutador ATM) conforme apresentado em [42] e [44]. Esta classe não possui atributos, mas é composta por um conjunto de conexões de caminho virtual (VPCs), sendo estas compostas por VCCs.

5.2.16 Conexão de Caminho Virtual

Descrição

Representa cada VPC estabelecida em um comutador, sendo cada uma formada por um conjunto de VCCs.

Atributos

- **vpi:** Representa o campo VPI que identifica uma conexão em uma tabela de estados de conexões.

5.2.17 Conexão de Caminho Virtual

Descrição

Representa cada uma das VCCs pertencente às VPCs estabelecidas em um determinado comutador.

Atributos

- **vci:** Representa o campo VCI de uma tabela de estados de conexões e é utilizado para identificar a que VCC dentro de uma VPC uma determinada célula pertence.
- **número de células comutadas:** Representa o número de células de uma determinada conexão que foram roteadas até o equipamento adjacente.
- **número de células descartadas por policiamento:** Representa o número de células de uma determinada conexão cujos parâmetros de tráfego (PCR e CDVT) foram violados.
- **número de células descartadas por gerenciamento:** Representa o número de células de uma determinada conexão que foram descartadas por não encontrar espaço em fila ou que estavam em fila e chegou uma célula de maior prioridade.

5.2.18 Nó Unicast

Descrição

Esta classe estende uma VCC a fim de defini-la como sendo um nó de conexão *unicast*, ou seja o endereçamento de saída é único.

- **novo VPI:** Representa o novo valor do campo VPI que a célula receberá na saída.
- **novo VCI:** Representa o novo valor do campo VCI que a célula receberá na saída.

5.2.19 Nó Multicast

Descrição

Esta classe estende uma VCC a fim de defini-la como sendo um nó de conexão *multicast*, ou seja o endereçamento de saída têm como destino um conjunto de BTEs ou Comutadores.

Esta classe permite que uma célula recebida na porta de entrada de um comutador seja multiplicada e cada uma das cópias seja enviada a destinos diferentes. Dessa forma, em qualquer nó da rede uma conexão *unicast* pode ser comutada através de várias outras, formando uma ramificação a partir daquele ponto.

Essa classe não possui atributos simples, mas é formada por um conjunto de objetos *unicast*.

5.2.20 Fila

Descrição

Representa um *buffer* de um comutador ou ETBL e seu objetivo é acomodar células cujas portas de saída se encontram ocupadas com o envio de outras células. A disciplina de atendimento desta classe é **FCFS com prioridade** - ou seja, células são atendidas conforme cheguem à fila, no entanto, aquelas que chegam e encontram outras a sua frente são atendidas primeiro desde que possuam prioridade maior.

Atributos

- **tamanho:** Representa o tamanho da fila. É definido pelo usuário durante a criação do modelo.
- **tamanho atual:** Representa o tamanho de uma fila em um determinado instante dependendo do número de células que a estiver ocupando.

- **tamanho máximo:** Representa o maior valor que o atributo *tamanho atual* obteve em uma determinada simulação.
- **tamanho médio:** Representa o valor médio que o atributo *tamanho atual* obteve em uma determinada simulação.
- **tamanho mínimo:** Representa o menor tamanho que o atributo *tamanho atual* obteve em uma determinada simulação.

Operações

- **verifica espaço:** verifica se existe espaço para se armazenar uma célula em uma fila.
- **insere célula:** insere uma célula em uma fila;
- **envia célula:** envia a célula do cabeça da fila através da porta de saída correspondente.
- **descarta célula:** descarta uma célula que acabou de chegar se não houver espaço na fila ou se não houver nenhuma célula de menor prioridade na fila. Quando a célula chega à fila, é realizada uma busca a partir da cabeça para verificar se há alguma célula com prioridade menor na fila. Se houver uma célula na posição n, esta será substituida pela célula recém-chegada. A célula que foi substituida será então inserida na posição $n + 1$ e esse processo será repetido até o fim da fila. Se alguma célula sobrar, será então descartada.
- **incrementa número de células descartadas:** toda vez que uma célula da fila gerenciada for descartada, o número de células descartadas por gerenciamento da conexão virtual (VCC) à qual esta célula pertence será incrementada.
- **retorna primeira célula da fila:** retorna a célula que está na cabeça da fila.
- **calcula tamanho médio:** calcula o *tamanho médio* de uma fila após uma simulação.

- **retorna tamanho:** retorna o valor atual do atributo *tamanho.*
- **retorna tamanho atual:** retorna o valor atual do atributo *tamanho atual.*
- **retorna tamanho máximo:** retorna o valor do atributo *tamanho máximo.*
- **retorna tamanho médio:** retorna o valor do atributo *tamanho médio.*

5.3 Diagramas de Estados

Nesta seção são descritos os comportamentos das classes até então apresentadas através de um conjunto de diagramas de classes. Algumas delas, no entanto, não apresentam um comportamento dinâmico significativo o suficiente ou até mesmo não apresentam, sendo estas apenas objetos passivos na execução do sistema - como por exemplo a classe *célula* ou a classe *fila.* Nestes casos, não se faz necessário elaborar um diagrama de estado para cada uma delas.

5.3.1 Diagrama de Estados de uma Fonte de Tráfego

Uma aplicação fonte deve gerar dados até uma quantidade estabelecida pelo usuário durante a configuração do modelo. A classe Fonte de Tráfego verifica inicialmente se esta quantidade foi atingida no estado **Verificando**. Caso tenha sido atingido, deve haver uma transição para o estado final. Caso contrário, é feita uma transição para o estado responsável pela geração de informação na rede chamado **Gerando Informação**. Depois disso, uma transição ocorre para o estado **Enviando** onde essa informação é enviada para o ETBL sobre o qual a fonte de tráfego está executando. Em seguida, ocorre uma transição para o estado **Atualizando** a fim de atualizar a quantidade de bits enviadas, retornando depois ao estado inicial **Verificando**.

O diagrama de estados de uma Fonte de Tráfego é apresentado na figura 5.2.

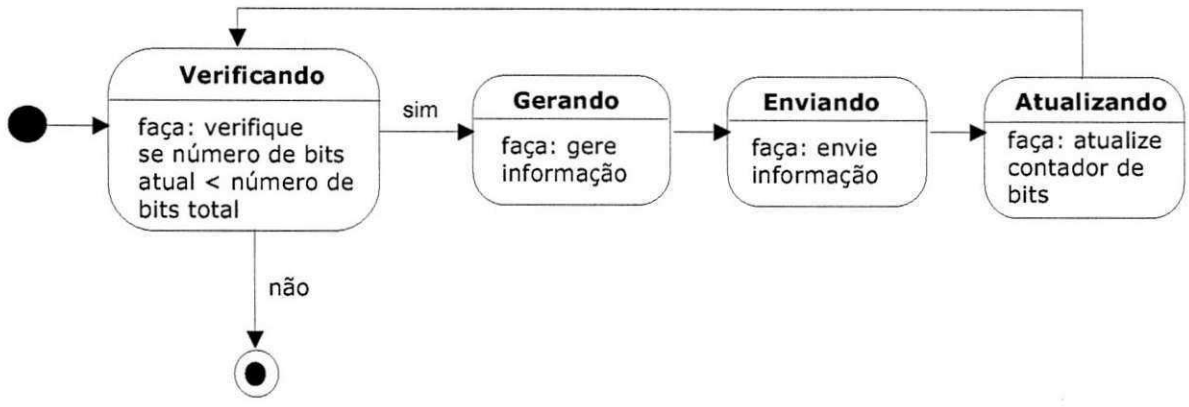

Figura 5.2: Diagrama de Estados de uma Fonte de Tráfego

5.3.2 Diagrama de Estados de um ETBL Fonte

Um ETBL Fonte recebe um fluxo de informação da aplicação que ele executa, gera as células e as encaminha ao comutador a ele conectado. Inicialmente, o ETBL aguarda fluxos de informação chegarem da aplicação fonte no estado **Esperando**. Quando um fluxo chega, ocorre uma transição para o estado **Calculando**, onde é feito o cálculo para determinar quantas células devem ser geradas a partir dele. Em seguida, no estado **Gerando**, uma célula é gerada e ocorre uma transição para o estado **Enviando** que se encarregará de enviá-la para o comutador ao qual o ETBL encontra-se conectado. Após ter enviado a célula, uma transição ocorre para o estado **Verificando** a fim de observar se o número de células relativo àquele fluxo já foi enviado. Caso tenha sido, uma transição para o estado inicial (**Esperando**) ocorre. Caso contrário, as demais células são geradas e enviadas para o enlace.

O diagrama de estados de um ETBL Fonte é apresentado na figura 5.3.

5.3.3 Diagrama de Estados de um Enlace

Um enlace deve receber células de um equipamento e enviá-las a um outro. É importante verificar o quanto esse recurso permanece ocupado a fim de definir se faz

Figura 5.3: Diagrama de Estados do ETBL Fonte

necessário aumentar sua capacidade e observar sua relação com eventuais comportamentos da rede.

Dessa forma, o comportamento da classe Enlace se inicia com o estado **Aguardando**, onde ela aguarda pela chegada de células de um determinado equipamento. Quando uma célula chega, uma transição ocorre para o estado **Calculando**, onde é feito o cálculo do fator de utilização. Em seguida, uma transição para o estado **Enviando** é feita. Neste estado, a célula recebida é enviada para o equipamento ao qual o enlace está conectado e uma transição é feita para o estado **Aguardando**.

O diagrama de estados de um Enlace é apresentado na figura 5.4.

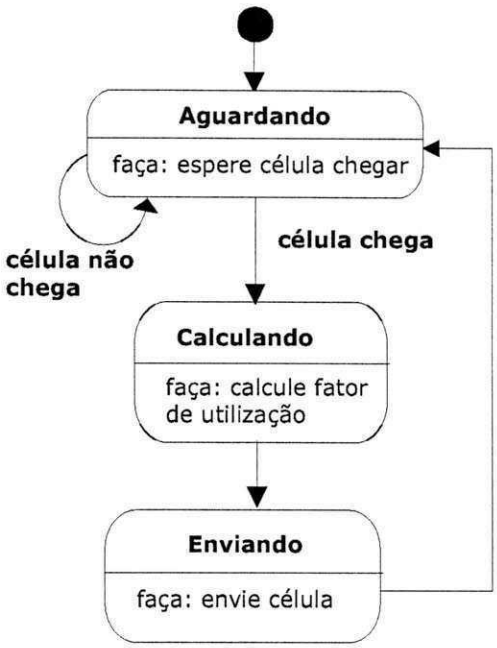

Figura 5.4: Diagrama de Estados do Enlace

5.3.4 Diagrama de Estados de um Comutador

Um comutador recebe células de um ETBL ou de outro comutador e, de acordo com as informações contidas na tabela de estados da conexão, encaminha-as até o próximo comutador ou ETBL (dependendo da configuração da rede). Se no instante do envio de uma determinada célula, houver uma outra a ser enviada pela mesma porta, a célula de menor prioridade deve ficar armazenada em uma fila (*buffer*) aguardando a vez de ser enviada. O Comutador em um determinado instante de tempo encontra-se em um dos dois estados a saber: **Comutando Células** ou **Gerenciando Fila**. A figura 5.5 mostra o diagrama de estados de um Comutador.

O primeiro estado, **Comutando Células**, está relacionado com atividades que vão desde a recepção de células na porta de entrada, até seu envio na porta de sada, conforme mostra a 5.6. O segundo estado, **Gerenciando Fila**, não faz parte exatamente do Comutador, mas da Fila que ele possui. A expansão desses estados é apresentada

na figura 5.7. A seguir são apresentados mais detalhes sobre cada um desses estados.

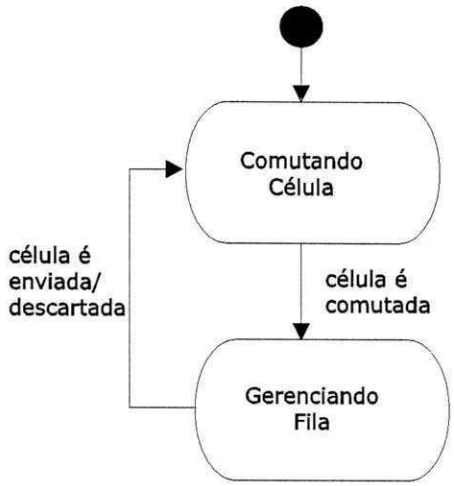

Figura 5.5: Diagrama de Estados de um Comutador

Expansão do Estado Comutando Célula

O estado **Comutando Célula** diz respeito às atividades que vão desde a recepção da célula na porta de entrada até o seu encaminhamento ao Gerenciador de Filas - figura 5.7. Para isso, o comutador fica aguardando uma célula chegar no estado **Aguardando**. Quando a célula chega, o valor do VPI é retirado para localizar a conexão à qual ela pertence. Se não localizar essa conexão, a célula é descartada e o número de células descartadas é incrementado. Caso a célula pertença a alguma conexão estabelecida é verificado se o Comutador é a Interface Usuário-Rede[2].

Se o Comutador for a Interface Usuário-Rede, é realizado o **Processamento de Policiamento**, um meta-estado onde são realizadas operações para verificar se a célula está de acordo com os parâmetros de tráfego definidos no estabelecimento da conexão. Essas operações estão de acordo com o algoritmo do Balde Furado (*Leaky Bucket*)

[2]O Processamento de Policiamento é realizado apenas no comutador de entrada na rede, ou seja, o comutador responsável pela Interface Usuário-Rede. Nos demais comutadores, é realizado apenas a comutação.

apresentado no capítulo 3. Se os parâmetros foram violados, a célula é descartada e o número de células descartadas é atualizado.

Se o Comutador não for a Interface Usuário-Rede (UNI), é verificado se a conexão à qual a célula pertence é *Unicast* ou *Multicast*. Se a conexão for *Unicast*, a célula terá seu cabeçalho atualizado conforme a Tabela de Estados da Conexão e enviada ao Gerenciador de Fila. Se a célula for *Multicast*, a célula será reproduzida e cada uma das reproduções é enviada ao Gerenciador de Fila.

Expansão do Estado Gerenciando Fila

O estado **Gerenciando Fila** é responsávelo por enviar uma célula a cada pulso do relógio (*clock*) desde que já exista alguma célula enfileirada ou chegue uma nova no comutador - figura 5.7.

Inicialmente, é verificado se chegou alguma célula no estado **Verificando Chegada**. Caso não tenha chegado uma célula, uma transição para o estado **Verificando Fila** ocorre e é observado se a fila encontra-se vazia. Caso esteja vazia, uma transição ocorre voltando ao estado anterior. Caso a fila não esteja vazia, a célula que se encontra no incio da fila é enviada. Ocorre então uma transição para o estado **Atualizando Células Enviadas** e, em seguida, outra transição para o estado **Final**.

No caso de ter chegado uma célula no estado **Verificando Chegada**, ocorre uma transição para o estado **Verificando Prioridade**. Nesse estado, verifica-se se há alguma célula na fila com prioridade menor. Se não houver, é verificado se há espaço na fila. Havendo espaço em fila, a célula é inserida e aquela que estiver no incio da fila é enviada. Não havendo esse espaço na fila, a célula é descartada e o nmero de células descartadas é incrementado. Se houver uma célula com prioridade menor, a célula recém-chegada substituirá aquela de prioridade menor, que deverá retroceder uma posição na fila, assim como todas aquelas que a seguem. Se alguma célula exceder o tamanho da fila nesse processo de rearrumação, ela é descartada e o nmero de células descartadas é incrementado.

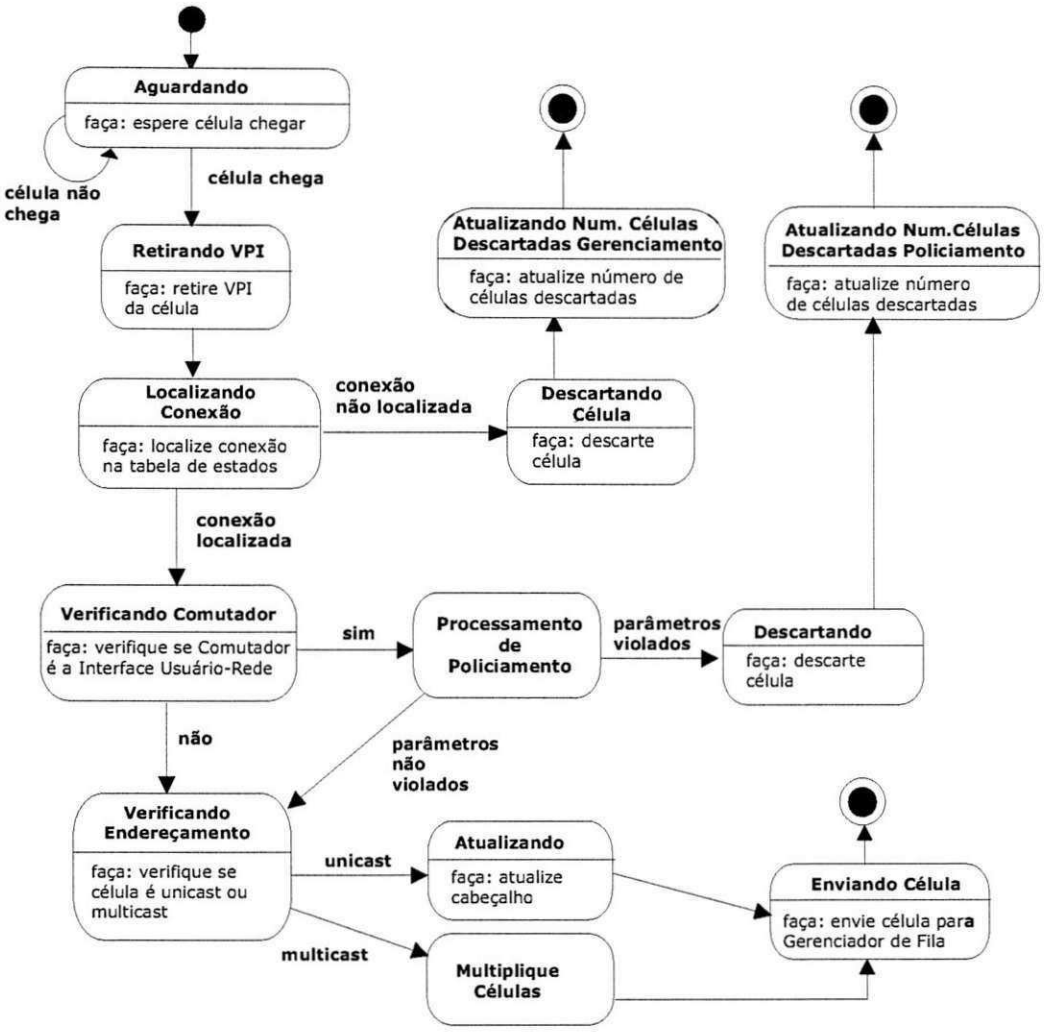

Figura 5.6: Expansão do Estado Comutando Célula

5.3.5 Diagrama de Estados de um ETBL Destino

O ETBL destino é o ponto final de um fluxo de células na ATMLib. Sua função é receber as células e calcular as medidas de desempenho relacionadas ao tempo, conforme já mencionadas na subseção 5.2.

Conforme apresentado na figura 5.8, um ETBL Destino fica inicialmente aguardando

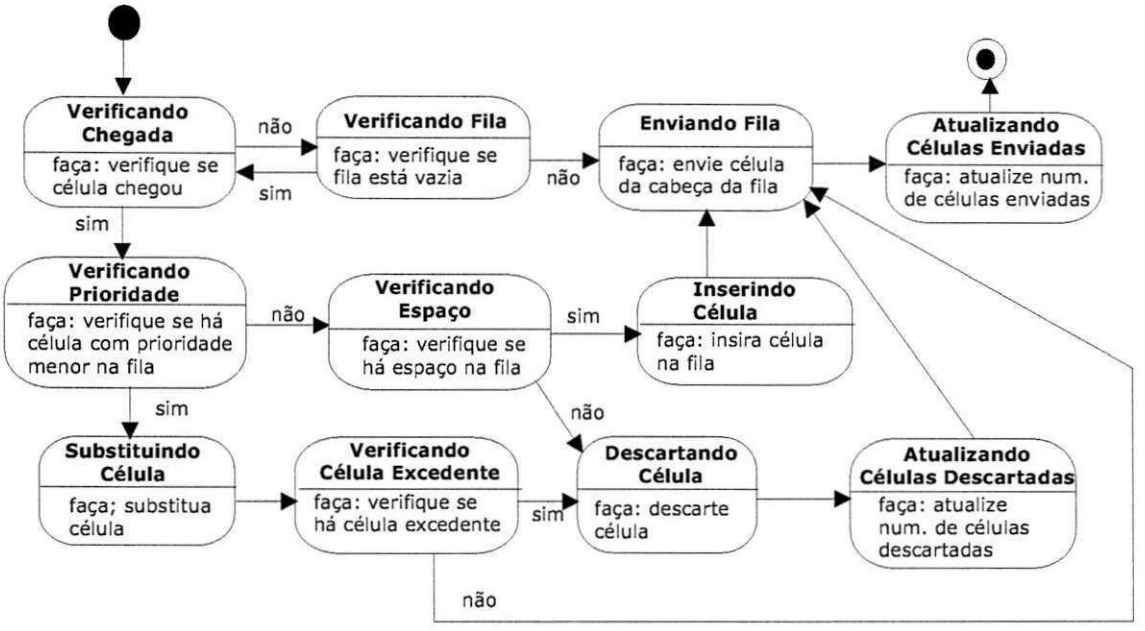

Figura 5.7: Expansão do estado Gerenciando Fila

uma célula chegar no estado **Esperando**. Quando recebe uma célula, ocorre uma transição para o estado **Calculando Atraso**. Uma vez tenha sido calculado o valor do atraso para aquela célula, uma transição para o estado **Incrementando** ocorre para que seja atualizado o nmero de células recebidas. Em seguida, ocorre uma transição para o estado **Armazenando**.

Nesse estado, os valores dos atrasos experimentados pelas diversas células que chegam ao ETBL destino devem ser armazenados para que, uma vez tenha terminado a simulação, seja possível obter medidas de desempenho tais como atraso mínimo, máximo e médio das células da conexão. Em seguida, uma transição para o estado inicial é realizada a fim de aguardar uma nova célula.

O diagrama de estados de um ETBL Destino é apresentado na figura 5.8.

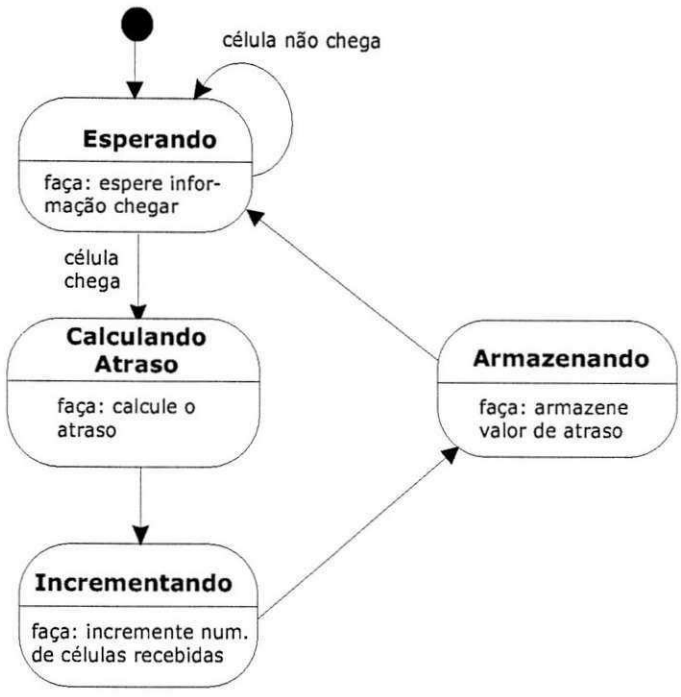

Figura 5.8: Diagrama de Estados de um ETBL Destino

5.4 Diagrama de Sequência

Diagramas de sequência mostram de que forma um conjunto de classes interage a fim de compor um determinado cenário [2]. Um cenário compreende as interações que formam um determinado *use-case*. Nesse trabalho, tem-se portanto dois cenários.

De acordo com o *diagrama de use-case* apresentado no capítulo anterior, são apresentados nessa seção dois diagramas de interação[3] cada um correspondendo aos *use-cases* já apresentados: o primeiro diz respeito à **geração de tráfego** e o segundo à **manipulação de tráfego** na rede.

Duas observações devem ser feitas sobre esses dois cenários: (*i*) os mesmos são complementares, uma vez que a geração de tráfego deve preceder sua manipulaçãoA

[3]Conforme visto no capítulo anterior, Diagramas de Seqüência podem ser de dois tipos: Diagrama de Interação e Diagrama de Colaboração.

seqüência de interação do cenário *Manipulação de Tráfego* é, portanto, uma continuação do cenário anterior - *Geração de Tráfego.* e (*ii*) existem inúmeros outros cenários, cada um correspondendo a um estado distinto de uma rede ATM. Por questão de generalidade e simplicidade, são apresentados apenas estes dois, sendo que os demais podem ser obtidos alterando-se o cenário a partir daquele ponto onde ocorreu uma interação diferente.

5.4.1 Geração de Tráfego

A *geração de tráfego* abrange as interações ocorridas entre as classes responsáveis desde a geração de informação pela aplicação do usuário na origem, passando pela geração de células até sua inserção no comutador da UNI (pública ou privada).

Neste cenário, conforme apresentado na figura 5.9, estão envolvidas cinco classes: (i) *Fonte de Tráfego*, (ii) *ETBL Fonte*, (iii) *Célula*, (iv) *Enlace* e (v) *Comutador.*

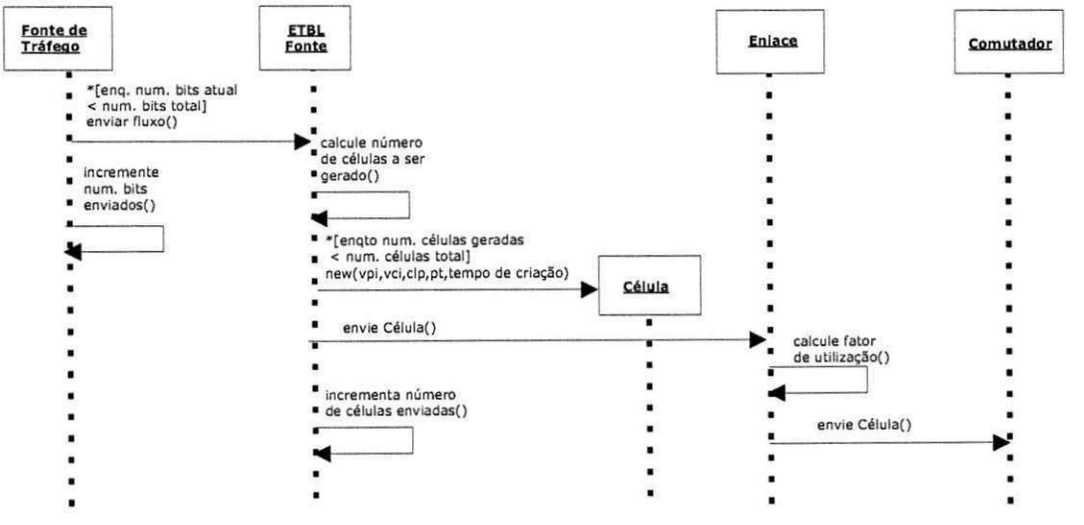

Figura 5.9: Diagrama de Sequência da Geração de Tráfego

A classe *fonte de tráfego* gera fluxos de informação (*bits*) enquanto a quantidade estabelecida pelo usuário não for alcançada. A informação é enviada para o *ETBL fonte* e a quantidade de bits enviada é incrementada.

No *ETBL fonte* a informação recebida deve ser transformada em células. Para isso, inicialmente, é feito um cálculo a fim de verificar quantas células devem ser geradas a partir desse fluxo. Tendo-se feito isso, as células devem ser geradas e enviadas para a classe *enlace*. Nesta classe o fator de utilzação é calculado e a célula enviada até o *comutador* ao qual ele está conectado.

5.4.2 Manipulação de Tráfego

A manipulação de tráfego corresponde às interações existentes entre as classes responsáveis desde a recepção das células na UNI - comutador - até sua recepção e manipulação no destino - ETBL destino.

As classes envolvidas neste cenário são as seguintes : (i) *Comutador*, (ii) *Célula*, (iii) *Tabela de Estados da Conexão*, (iv) *Fila*, (v) *Enlace* e (vi) *ETBL Destino* - figura 5.10.

A interação se inicia com a chegada de uma *célula*. O *comutador* retira seu VPI para, em seguida, localizar na *tabela de estados das conexões* aquela à qual ela pertence. Uma vez localizada, é verificado se o *comutador* é Interface Usuário-Rede (UNI). Em caso positivo, é realizado o Processamento de Policiamento e, do contrário, é realizada apenas a Comutação.

No caso de o Pocessamento de Policiamento ser realizado, é verificado se a célula está de acordo com os parâmetros estabelecidos. Se não estiver, ela será descartada e o atributo *número de células descartadas* é incrementado.

Se a célula for conformante deve-se verificar o tipo de endereçamento (*unicast* ou *multicast*). De acordo com o tipo observado, o próximo passo é a atualização do cabeçalho e o seu envio para a fila. Na *fila* serão feitas várias operações sobre a célula de acordo com o seu estado atual e com a prioridade da célula.

No caso de a fila encontrar-se vazia, a célula é imediatamente enviada para o *enlace* que o comutador utiliza para se conectar ao destino daquela célula. Em seguida, o número de células enviadas é incrementado.

Se a fila não estiver vazia, a prioridade da célula recém-chegada é retirada e comparada com a da que se encontra no início da fila. Se a prioridade da célula do início for maior, será feita uma comparação com a da posição seguinte e assim por diante até encontrar uma com menor prioridade ou o fim da fila. Se uma célula de menor prioridade for encontrada esta será substituida pela recém chegada.

Quando isso ocorre, deve-se fazer um deslocamento de uma posição em todas as células a partir da posição onde houver ocorrido a substituição. Dessa forma, a célula que estiver na posição n irá para a posição $n + 1$ e a da posição $n + 1$ irá para a $n + 2$. Se a fila estiver cheia no momento da chegada da célula, a que se encontrar na última posição não terá onde ser armazenada quando houver o deslocamento e será, portanto, descartada (nesse caso, o *número de células descartdas* é incrementado).

Tendo-se se feito o armazenamento da célula enviada pelo ETBL e, eventualmente, a re-organização da fila, o próximo passo é a retirada a célula do seu início e seu envio para o enlace que conecta-o ao destino dessa célula - *Comutador* ou *ETBL Destino*. Uma vez tenha sido enviada, deve-se incrementar o *número de células enviadas*.

Quando a célula chega ao *enlace*, o fator de utilização é calculado e a célula repassada ao equipamento que está na outra extremidade.

No caso do equipamento ser um *ETBL Destino*, incrementa-se o número total de células recebidas e, posteriormente, são calculadas medidas de desempenho relacionadas com o tempo: atraso atual, variação de atraso, atraso mínimo, atraso médio e atraso máximo.

Se o equipamento for um comutador, o conjunto de operações acima descritos é realizado.

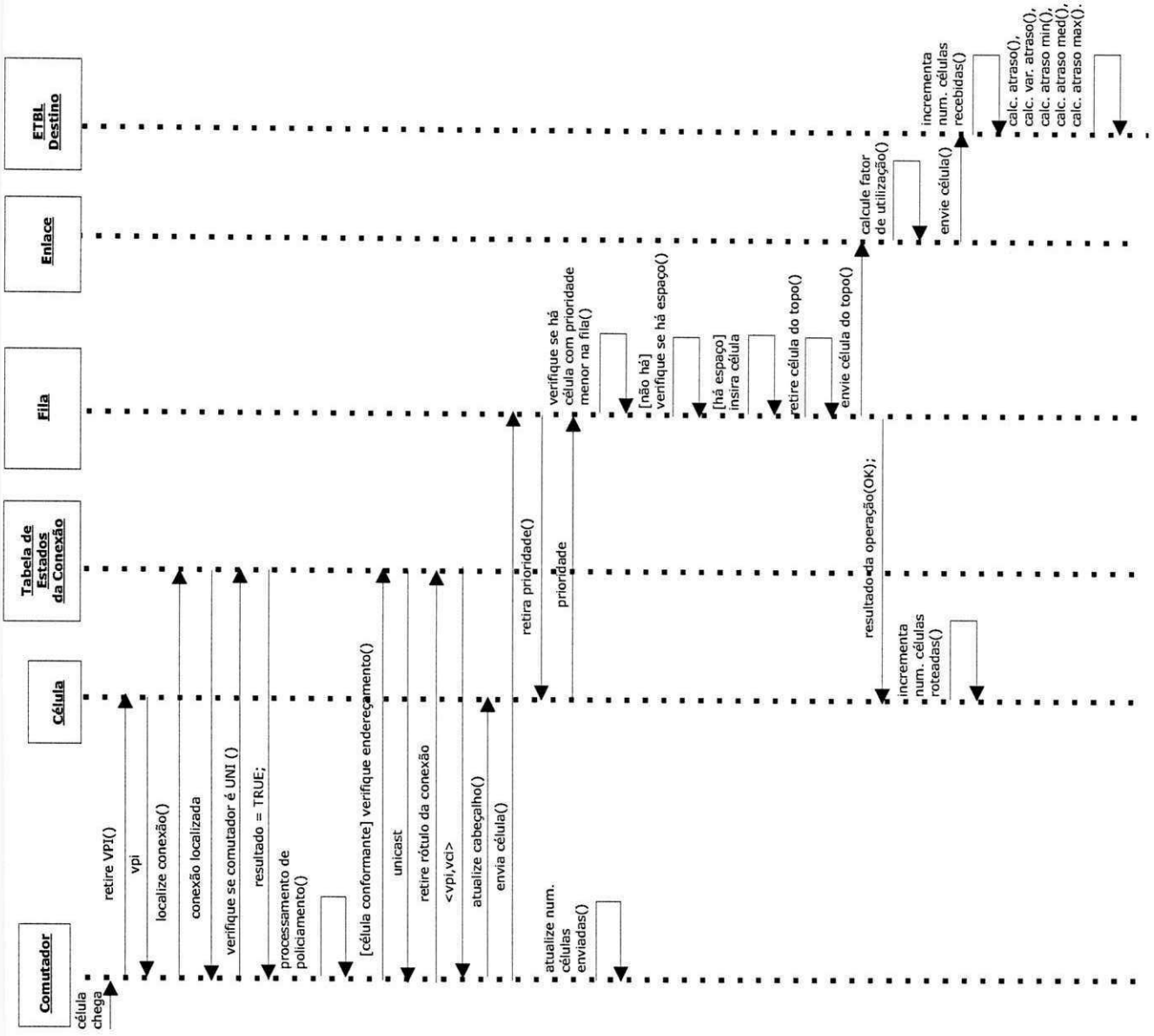

Figura 5.10: Diagrama de Sequência da Manipulação de Tráfego

"Sempre que um trabalho científico apresenta alguma informação,
ela vem acompanhada de uma margem de erro - um silencioso
mas insistente lembrete de que conhecimento algum
é completo ou perfeito.
Carl Sagan.

Capítulo 6

Projeto, Implementação e Testes da ATMLib

Neste capítulo são apresentadas as últimas fases relativas ao desenvolvimento da ATMLib. Enquanto a **Fase de Análise** foi responsável pela tradução de conceitos do domínio do problema em modelos que refletem o entendimento do desenvolvedor a respeito desse domínio, dessa forma permitindo a comunicação com o usuário, a **Fase de Projeto** consiste em traduzir estes modelos em estruturas compreensíveis pelo implementador.

As duas fases seguintes à de projeto possuem uma estreita relação entre si. Conforme o código fica pronto, testes devem ser realizados e, uma vez se verifique a ocorrência de erros, deve-se imediatamente realizar sua correção. Em função disto, estas duas fases são apresentadas conjuntamente neste capítulo.

Na seção 6.1 é apresentada a Fase de Projeto da ATMLib através da descrição de um Diagrama de Pacotes, de suas interfaces e dos principais algoritmos necessários a sua implementação. Em seguida, na seção 6.2 a Fase de Implementação é apresentada ressaltando as facilidades oferecidas pela linguagem Java nesta tarefa. A Fase de Testes é apresentada na seção 6.3 enfatizando os testes realizados e o software de testes *JUnit*.

6.1 Fase de Projeto

Após ter se analisado o domínio do problema no capítulo anterior, o próximo passo para se desenvolver a ATMLib foi decidir de que forma abordar o projeto. Isto consiste em traduzir conceitos relativos ao *domínio do problema* para o *domínio da solução* e assim guiar o desenvolvimeto até sua implementação.

O primeiro passo consiste em definir uma arquitetura que reflita estruturar o problema em subsistemas e estes compondo de outros ou das classes que os compõem. Tendo feito isso, deve-se definir de que forma cada uma das classes podem ser acionadas através da definição individual dos *métodos* e das *variáveis* em cada uma. Essa definição deve ser feita utilizando a linguagem de programação que será utilizada na implementação.

Por último, os principais algoritmos utilizados na implementação devem ser apresentados. No caso da ATMLib, os algoritmos necessários à implementação de cada classe foram derivados diretamente do diagrama de estado de cada uma delas, conforme apresentado no capítulo anterior, não sendo, portanto, necessário descrevê-los novamente uma vez que os mesmos podem ser entendidos completamente na forma em que lá se apresentam.

6.1.1 Uma Arquitetura Baseada na ATMLib

A arquitetura de um sistema forma sua organização geral e consiste de um conjunto de subsistemas que interagem e são descritos através de uma representação gráfica e conceitual. A arquitetura provê o contexto em que decisões mais detalhadas são feitas nos estágios de projeto posteriores [20].

Cada subsistema abrange aspectos do sistema que compartilham alguma propriedade comum - funcionalidade similar, mesma localização física ou execução sobre o mesmo tipo de *hardware*. Um subsistema não deve ser uma classe nem uma função mas um pacote de classes, interrelacionadas através de suas associações e que tem uma interface limitada e bem definida com outros subsistemas.

Diagrama de Pacotes

Em UML, a arquitetura de um sistema é descrita sobretudo pelo *Diagrama de Pacotes* [3] [4]. Desta forma, é apresentado a seguir o diagrama de pacotes de um simulador de redes ATM no qual a ATMLib pode ser inserida - 6.1.

Este diagrama é constituído por cinco pacotes: *ATMLib*, *Interface com o Usuário*, *java.swing*, *Controle da Simulação do Modelo* e *java.math*. Além desses pacotes, são apresentadas as relações de **dependência** existentes entre os mesmos representadas através de setas pontilhadas indicando o sentido em que elas ocorrem.

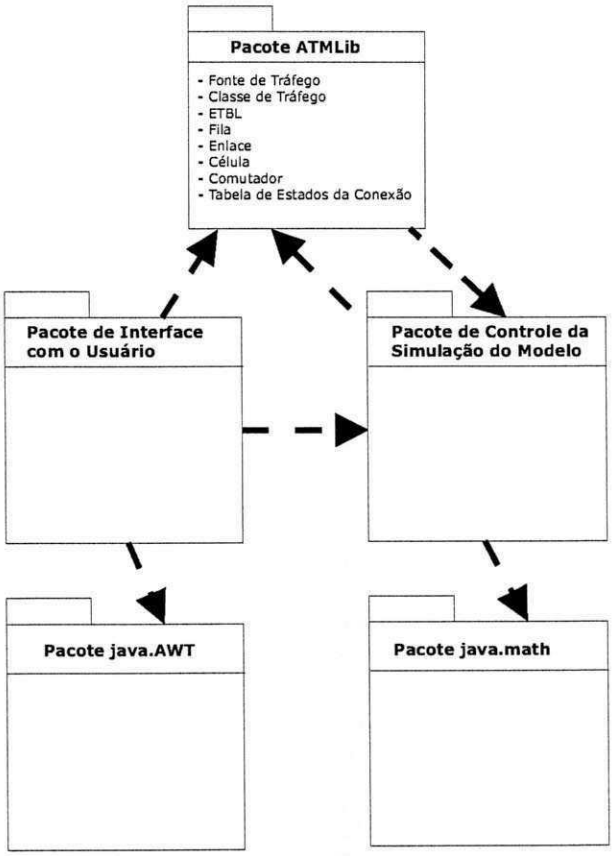

Figura 6.1: Diagrama de Pacotes da ATMLib

O pacote *Interface com o Usuário* é aquele responsável pela entrada de parâmetros durante a fase de construção dos modelos e pela exibição dos resultados após a execução dos mesmos.

O pacote *java.swing* é fornecido pela API Java [46] e deve ser utilizado a fim de fornecer as classes básicas que serão utilizadas para implementar a interface com o usuário. Este pacote é colocado aqui a título de exemplificação e sugestão de escolha de implementação. No entanto, outros recursos da linguagem, como por exemplo *java.awt*, poderiam também ser utilizados.

O pacote de *Controle da Simulação* é formado por um conjunto de classes que permitem a execução de modelos de redes ATM observando-se os princípios da Simulação Discreta Orientada a Eventos.

O pacote *java.math* também é fornecido pela API Java [46] e dispõe de classes que fornecem funções matemáticas necessárias para se simular um modelo, tais como *geração de variáveis aleatórias.*

É importante ressaltar que os demais pacotes são apresentados como forma de entender o contexto típico em que a ATMLib se insere e definir as relações de dependência que a mesma possui em relação à estes. Desta forma, os demais pacotes não serão aqui detalhados.

Dois pacotes mantém relações de dependência com a ATMLib[1]: (i) *Interface com o Usuário* e (ii) *Controle da Simulação.*

Em relação ao primeiro, a dependência se dá devido às classes da ATMLib terem todos seus parâmetros definidos pelo usuário através da interface. Portanto, uma alteração nos atributos de uma classe leva conseqüentemente à alteração na classe responsável pela entrada dos valores referentes aos atributos daquela classe. De maneira análoga, a exibição dos resultados da simulação é feita através da interface. A dependência em relação ao segundo pacote ocorre devido ao fato que o controle da simulação e, portanto, a manipulação dos eventos específicos a um determinado sistema depende da estrutura e comportamento das classes desse sistema.

[1]Pode-se dizer também que os outros pacotes *dependem* da ATMLib

Há também uma dependência da ATMLib em relação ao pacote de Controle da Simulação, uma vez que durante o momento da criação de cada célula, faz-se necessário obter da classe *relógio simulado* existente no pacote citado, o instante atual e definí-lo como sendo o instante de criação da célula. Este valor é importante para o cálculo de medidas relacionadas ao atraso, como por exemplo atraso médio, atraso máximo e variação de atraso.

6.1.2 Definição de Variáveis e Interfaces da ATMLib

A definição de variáveis e de interfaces[2] consiste em traduzir a estrutura de uma classe representada em uma notação de alto nível - baseada no *diagrama de classes* - para uma notação de baixo nível, que pode ser, inclusive, baseada na linguagem de programação que será utilizada para implementar o software em desenvolvimento. No caso deste trabalho, será utilizada a linguagem Java para realizar essa tradução.

O resultado desta tradução permite ao programador implementar o trabalho em desenvolvimento seguindo a descrição apresentada pelos projetistas responsáveis pela parte de de mais alto nível sem haver o risco de haver discrepâncias entre o projeto de alto nível (*modelos*) e o resultado final (*código*). No entanto, faz-se necessário que modeladores e programadores interajam a fim de realizar atualizações. Dessa forma, a definição de variáveis e interfaces apresentada em anexo consiste em uma versão inicial, podendo haver atualizações no decorrer da implementação. O resultado final será então a API da ATMLib que será disponibilizada juntamente com o código fonte.

[2]A interface ou assinatura de um método representa a sintaxe de sua declaração e especifica a forma como o mesmo é acionado pela própria classe ou por outras. Consiste no tipo de retorno, seu nome e, se houver, os parâmetros a ele passados. Um modificador de visibilidade define quais classes, métodos ou atributos são permitidos serem acessados por outras classes.

6.2 Fase de Implementação

A implementação de um software consiste em transformar um conjunto de diretrizes obtidas durante a fase de projeto em instruções reconhecíveis pelo computador utilizando uma linguagem de programação.

Durante essa fase, o desenvolvedor deve utilizar ferramentas que lhe auxiliem nesta tarefa. Dessa forma, conforme apresentado no capítulo 4, optou-se pela linguagem Java para a implementação deste trabalho, tendo sido escolhido o *Java Development Kit* versão 1.2 para as atividades de compilação e interpretação, juntamente com o ambiente de desenvolvimento integrado *Jbuilder* [51] (versão 2.0) da Borland a fim de auxiliar a edição das classes.

Dentre outras vantagens observadas[3], Java possibilita que um conjunto de classes com uma funcionalidade comum possam ser agrupadas através de um *pacote*. Essa possibilidade permitiu que a implementação da ATMLib fosse feita de forma direta, ou seja, suas classes foram organizadas em um pacote - chamado **ATMLib** - de acordo com o *diagrama de pacotes* apresentado no capítulo anterior. Portanto, quando um desenvolvedor for utilizá-la para implementer um simulador, ele deve inserir no início das classes que irão acionar a biblioteca a diretiva `import ATMLib.*;`.

Cada uma das classes apresentadas foi implementada em um arquivo `.java` individualmente. Após a compilação cada classe gerou um arquivo `.class`.

6.3 Fase de Testes

6.3.1 Introdução

Os testes realizados sobre o código obtido durante a implementação consistem em reduzir os defeitos (*bugs*) verificados e assegurar a qualidade do software para o usuário [24]. Existem vários tipos de testes, cada um correspondendo a uma etapa diferente.

[3]Vide capítulo 4.

Conforme apresentado no capítulo 1, o teste realizado nas classes da ATMLib é o *teste de unidade.*

Este teste unidade consiste em verificar o funcionamento de cada uma das classes individualmente sem se preocupar com as interfaces com outras classes ou com o funcionamento geral do sistema. Todo o código deve ser testado. Testando dessa forma, o desenvolvedor pode descobrir erros mais cedo além de ter um controle maior sobre os testes do que se fosse testar o sistema completo.

Devido ao fato de que a ATMLib é um subsistema dentro de um projeto maior[4] é possível apenas realizar os testes de unidade e certificar-se quanto a sua qualidade. Fica então a cargo do desenvolvedor do sistema completo, realizar os demais testes.

6.3.2 JUnit

JUnit é uma ferramenta de testes desenvolvido por Kent Beck e Erich Gamma [49] para testar classes individualmente. Essa ferramenta consiste em um conjunto de classes escritas em Java que permitem ao desenvolvedor estendê-las para implementar suas próprias classes de teste.

Quando uma classe é implementada o que se deseja testar nela é basicamente se seus métodos funcionam da forma correta e produz os valores esperados mesmo que em situações absurdas[5]. Portanto, para se testar uma classe utilizando o **JUnit** faz-se necessário implementar uma classe de testes correspondente. Por exemplo, para a classe *comutador* tem-se uma classe correspondente *TestaComutador.*

Cada classe de testes consiste de três partes: (i)*inicialização*, (ii) *exercício* e (iii) *verificação.*

A **inicialização** consiste em criar e inicializar as classes e variáveis que farão parte dos testes. Quando mais de um método de testes utilizar as mesmas classes, a inicialização podeá ser fatorada em um único método.

[4]Um simulador discreto orientado a eventos de redes ATM.

[5]Conforme apresentado no capítulo 4, esta propriedade é chamada de **Robustez**.

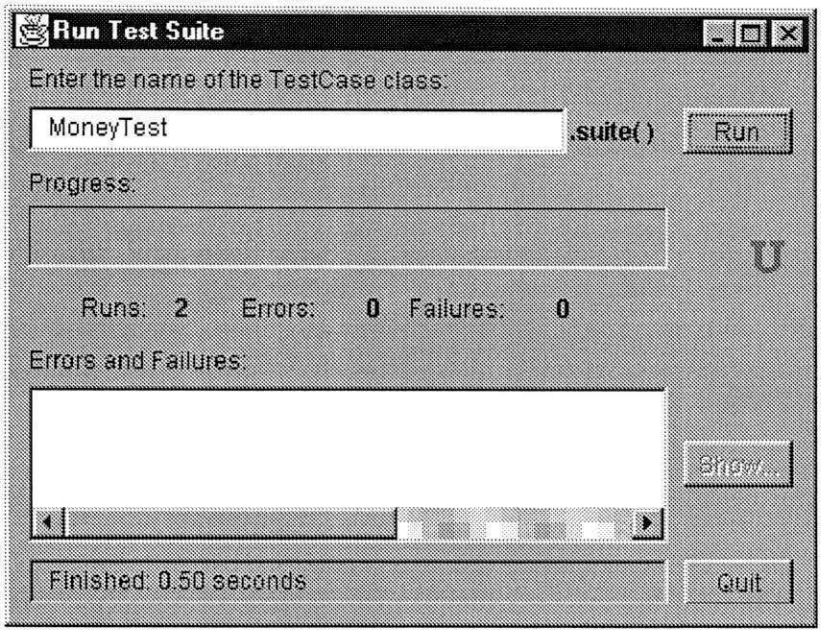

Figura 6.2: Janela do JUnit

O **exercício** consiste em adicionar os métodos de testes que irão testar indivualmente os testes das classes utilizando como parâmetro inclusive valores que estão fora da faixa normalmente esperada. Por exemplo, suponha que uma classe retorna o valor de um array mediante a passagem do índice de sua posição como parâmetro. Um dos testes poderia ser então passar um valor fora da faixa de índices permitidos (um valor negativo ou maior do que seu tamanho) e verificar se o método executa corretamente mesmo nestas situações.

A **verificação** consiste em verificar através de métodos fornecidos pelo **JUnit** - chamados de assert e assertEquals - se os valores retornados correspondem ao esperado. Por exemplo, em um método responsável pela inserção de um elemento em um *array* cujo resultado é um valor *boolean* (true corresponde ao resultado positivo e false caso contrário), um dos possíveis testes é tentar inserir elementos em um array vazio e confirmar se o resultado é true e, da mesma forma, em um *array* cheio e confirmar se o resultado foi false. Se o resultado for diferente a classe apresenta erros.

Há duas formas de testar uma classe: através de linha de comando ou de uma interface gráfica.

No caso da linha de comando, deve-se simplesmente acionar a classe através do interpretador Java. Por exemplo, no caso da classe *TestaComutador* seria digitado no prompt de comandos de uma janela DOS o comando java TestaComutador. No caso de haver mais de uma classe a ser testada deve se repetir isso para cada classe.

No caso da interface gráfica, após o usuário digitar no prompt o comando java test.ui.TestRunner uma pequena Janela é exibida (figura 6.2) e o usuário irá digita em uma janela de comandos o nome da classe que deseja testar e clicar no botão **run**. Se a execução não apresentar erros, uma barra na janela fica verde. Por outro lado, se for verificado algum erro a barra fica vermelha e é informado o número de erros e a descrição de quais foram. Também é possível observar a pilha de execução do teste e assim descobrir onde o erro ocorreu. Para isso deve-se clicar no botão **show**. Na figura 6.3 pode-se observar uma janela apresentando o *logging* da pilha de execução.

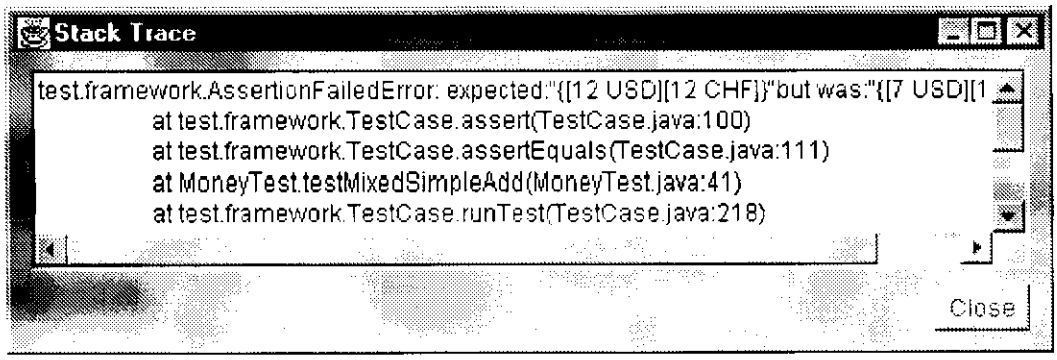

Figura 6.3: Janela com log de Execução

6.3.3 Testes Realizados

Todas as classes da ATMLib foram testadas utilizando o **JUnit**. Em cada classe foram testados diretamente os métodos disponíveis aos usuários - (*métodos públicos*).

Os métodos que são utilizados por outros para executar operações auxiliares foram testados indiretamente na medida em que aqueles que os utilizam eram testados.

Os testes realizados em cada classe foram feitos passando parâmetros para os métodos a fim de verificar se os resultados estavam dentro do esperado. Foram utilizados tanto valores especificados dentro da faixa aceitável de valores como valores considerados absurdos, como por exemplo retirar uma célula de uma fila vazia ou inserir em uma fila cheia. Conforme e existência de erros era verificada, correções eram feitas e os testes repetidos. Esse ciclo se repetiu até o desaparecimento dos erros observados.

6.4 Considerações sobre a Fase de Testes

Uma biblioteca de classes é apenas uma parte de um sistema de software maior. Por isso, foi possível apenas realizar os testes de unidade. Desta forma, o **JUnit** foi a ferramenta utilizada para este fim.

Os resultados dos testes demonstraram que a execução das classes da ATMLib correspondem ao comportamento esperado, garantindo assim a confiabilidade dos modelos obtidos.

Apesar de não ter sido possível realizar os demais testes, podemos afirmar que até onde os objetivos foram inicialmente estabelecidos este trabalho atende ao esperado cabendo àqueles que forem continuá-lo realizá-los.

"Toda a nossa ciência, comparada com a realidade, é primitiva e infantil - e, no entanto, é a coisa mais preciosa que temos."
Albert Einstein.

Capítulo 7

Conclusões e Trabalhos Futuros

Nesta seção é apresentada a finalização desse trabalho e de que forma o mesmo pode vir a ser continuado.

Na seção 7.1 são apresentadas algumas considerações acerca deste trabalho. Em seguida, na seção 7.2 são apresentadas as conclusões sobre as diversas decisões tomadas durante o desenvolvimento. Por fim, na seção 7.3 são apresentadas algumas sugestões de trabalhos futuros como forma de continuação deste trabalho.

7.1 Considerações Gerais

Nesta trabalho, foi apresentado o desenvolvimento de uma biblioteca de classes de suporte à construção de simuladores discretos de redes ATM chamada ATMLib.

Este desenvolvimento se deu desde a fase inicial, onde foi descrita a necessidade atualmente existente por bibliotecas dessa natureza, até a fase de testes, onde as classes puderam ser testadas individualmente. A fim de alcançar os requisitos estabelecidos, esse desenvolvimento baseou-se primordialmente nos paradigmas da orientação a objetos, tendo como metodologia subjacente o processo unificado (UML), utilizando a linguagem de programação Java.

uma vez que conceitos como *herança*, *agregação* e *encapsulamento* são facilidades inerentes a essa abordagem.

A metodologia UML mostrou-se bastante eficiente, principalmente no que diz respeito aos diagramas de modelagem do sistema, cada um possuindo um conjunto de notações de fácil compreensão pelo cliente e pelo implementador e de fácil aprendizado pelo projetista. Também vale ressaltar que sua notação é bastante diversificada, permitindo expressar através dos modelos uma grande quantidade de conceitos do mundo real, tais como *herança* e *agregação*.

Java mostrou-se uma linguagem de programação bastante eficiente por Ter simplificado significativamente a implementação deste trabalho. A razão disto é que por ser puramente orientada a objetos [36], a codificação das classes foi bastante natural e direta, uma vez que Java suporta facilidades utilizados durante a modelagem, tais como *herança* e *agregação*.

Também, pelo fato de ser neutra em relação à arquitetura, Java permitiu que a ATMLib seja portável para um conjunto significativo de plataformas.

7.3 Trabalhos Futuros

Extensões a este estudo, a título de trabalhos futuros, podem ser feitas segundo as seguintes perpesctivas:

7.3.1 Desenvolvimento de uma Biblioteca de Classes para o Controle da Simulação

Essa biblioteca deverá ser adicionada à ATMLib, conforme mencionado anteriormente, a fim de desenvolver um simulador de redes ATM. Também poder-se-ia agregá-la a uma outra biblioteca de classes de qualquer tipo de rede (Fast/Gigabit Ethernet, FDDI, etc.) ou mesmo a qualquer outro sistema de redes de filas, tais como linhas de montagem, sistemas de tráfego, etc.

Faz-se necessário novamente ressaltar que a ATMLib é ponto de partida para o desenvolvimento de sistemas mais abrangentes de modelagem e avaliação de desempenho de redes ATM. Esses ambiente permitirão, dentre outras vantagens, a flexibilidade de poder adicionar novos componentes ou estender aqueles existentes a fim de acrescentar novas funcionalidades que possam vir a surgir com a evolução natural da tecnologia ATM. Essa flexibilidade deve-se, principalmente, à utilização do paradigma da orientação a objetos durante o processo de desenvolvimento.

7.2 Conclusões

Conforme apresentado no capítulo 1, não foram encontrados na literatura trabalhos que abordassem o desenvolvimento de bibliotecas de classes que modelassem componentes de redes ATM. Dentre as alternativas atualmente existentes para se estudar o desempenho destas redes, observa-se que os mesmos foram desenvolvidos de forma bastante restrita, atendendo um escopo de seus projetos. Assim, é possível estabelecer as seguintes diferenças desse trabalho com aqueles apresentados no referido capítulo:

- A ATMLib foi desenvolvida observando a estrutura e o comportamento de uma rede ATM, deixando o controle da simulação a cargo de um outro conjunto de classes (a ser desenvolvida). Esta disassociação entre o sistema sendo modelado e o controle de simulação possibilita uma maior flexibilidade no que diz respeito a sua utilização, implementação e manutenção.
- A ATMLib foi desenvolvida utilizando a orientação a objetos como abordagem de desenvolvimento. Essa abordagem permite uma maior flexibilidade no que diz respeito à acomodação de mudanças e, conseqüentemente, a evolução do software.

A escolha da orientação a objetos como paradigma de desenvolvimento desse trabalho foi bastante natural uma vez que a *flexibilidade* e *reusabilidade* são requisitos fundamentais para bibliotecas de software. A orientação a objetos demonstrou-se totalmente adequada ao desenvolvimento deste trabalho e à obtenção desses requisitos

7.3.2 Expansão da Biblioteca

Conforme mencionado no capítulo 4, a ATMLib se propõe a um determinado cenário de redes ficando algumas facilidades a serem desenvolvidas como extensão deste trabalho.

Nesse sentido, uma facilidade de grande importância que deve ser adicionada a este trabalho é a classe de tráfego ABR. Essa classe possui um conjunto de características que leva a um significativo incremento de complexidade a sua modelagem e, consequentemente, ao desenvolvimento de um sistema de simulação de redes ATM. Sendo assim, funções tais como monitoração e realimentação dos parâmetros de controle de tráfego de acordo com o tráfego atual da rede [16] observando-se os diversos algoritmos propostos (NIST-ER, EPRCA, etc) [52] requer um estudo detalhado à parte, o qual poderà ser desenvolvido a posteriori.

Desta forma, é sugerida a expansão das classes *ETBL* e *Comutador*, a fim de que elas contemplem as funcionalidades oferecidas por esta classe de tráfego.

Também é importante um acréscimo de complexidade aos ETBLs, tanto de origem como de destino, a fim de permitir que mais de uma aplicação seja executada sobre eles.

7.3.3 Criação de Componentes Java Beans

Java Bean é um componente de software reusável que pode ser manipulado visualmente em uma ferramenta de desenvolvimento gráfica. A fim de permitir o desenvolvimento desses componentes, a linguagem Java fornece uma API específica chamada *java.beans*[1].

Desta forma, a disponibilidade de *beans* voltados para redes ATM permitirão que ambientes de modelagem e avaliação de desempenho dessas redes possam disponibilizar ao usuário de forma simples e flexível um conjunto de elementos de redes através de ícones em sua interface. Uma vantagem em disponibilizar os elementos dessa forma é a facilidade em modelar um sistema, que pode ser feita simplesmente clicando com o *mouse* nos ícones referentes a cada elemento e informando seus parâmetros.

[1]Disponível a partir da versão 1.1

7.3.4 Implementação da ATMLib usando C++

Conforme mencionado anteriormente, a portabilidade fornecida pela linguagem Java tem como principal desvantagem a redução do seu desempenho. Apesar de alguns autores afirmarem que o código Java compilado direcionado a uma determinada arquitetura (*compilação JIT*) atinge níveis comparáveis ou superiores de desempenho [50], a implementação deste trabalho utilizando a linguagem C++ poderia ser feita de forma a realizar estudos comparativos de desempenho entre as duas abordagens.

Essa alternativa é de relativa simplicidade uma vez que a sintaxe das duas linguagens é bastante similar e todo o processo de desenvolvimento deste trabalho foi realizado independente de linguagem de programação.

"Eu não me envergonho de corrigir os meus erros
e mudar minhas opiniões porque não me
envergonho de raciocinar e aprender."
Alexandre Herculano

Apêndice A

Acrônimos

AAL (ATM Adaptation Layer): Camada de Adaptação ao ATM;

API (Application Programming Interface): Interface de Programação da Aplicação;

ATMLib (ATM Library): Biblioteca ATM;

ATM (Asynchronous Transfer Mode): Modo de Transferência Assíncrono;

BF: Balde Furado;

BONeS (Block Oriented Network Simulator): Simulador de Redes Orientadas a Bloco;

BT (Burst Tolerance): Tolerância à Rajadas;

BTE (Broadband Terminal Equipment): Equipamento Terminal de Banda Larga;

CBR (Constant Bit Rate): Taxa de Tráfego Constante;

CDVT (Cell Delay Variation Tolerance): Tolerância à Variação de Atraso de Células;

CLP (Cell Loss Priority): Prioridade de Descarte de Célula;

CS (Convergence Sub-Layer): Sub-Camada de Convergência;

ETBL: Equipamento Terminal de Banda Larga;

GFC (Generic Flow Control): Controle de Fluxo Genérico;

GCRA (Generic Cell Rate Algorithm): Algoritmo de Taxa de Células Genérica;

HEC (Head Error Control): Controle de Erro de Cabeçalho;

IPOA (IP Over ATM): IP Sobre ATM;

LAN (Local Area Network): Rede Local;

LCT (Last Conformance Time): Último Tempo Conformante;

MRP : Modelo de Referência de Protocolos;

NNI (Network-Network Interface): Interface Rede-Rede;

NIST (National Institute of Standards and Technology): Instituto Nacional de Padrões e Tecnologia;

NPC (Network Parameter Control): Controle de Parâmetros de Rede;

nrt-VBR (Variable Bit Rate): Taxa de Tráfego Variável de Tempo Não-Real;

OMG (Object Management Group): Grupo de Gerenciamento de Objetos.

OMT (Object Modelling Technique): Técnica de Modelagem de Objetos;

OOS (Object Oriented Simulation): Simulação Orientada a Objetos;

PHY (Physical Medium): Meio Físico;

PVC (Permanent Virtual Connection): Conexão Virtual Permanente;

RDSI-FE: Rede Digital de Serviços Integrados de Faixa Estreita;

RDSI-FL: Rede Digital de Serviços Integrados de Faixa Larga;

rt-VBR (Variable Bit Rate): Taxa de Tráfego Variável de Tempo Real;

SAR (Segmentation And Reassembly): Segmentação e Remontagem;

SVC (Switched Virtual Connection): Conexão Virtual Comutada;

STM (Synchronous Transfer Mode): Modo de Transferência Síncrono;

UML (Unified Modelling Language): Linguagem de Modelagem Unificada;

UNI (User-Network Interface): Interface Usuário-Rede;

UPC (Usage Parameter Control): Controle de Parâmetros de Uso;

VPI (Virtual Path Identifier): Identificador de Caminho Virtual;

VCI (Virtual Channel Identifier): Identificador de Conexão Virtual;

VCC (Virtual Channel Connection): Conexão de Canal Virtual;

VCL (Virtual Channel Link): Enlace de Canal Virtual;

VPC (Virtual Path Connection): Conexão de Caminho Virtual;

WAN (Wide Area Network): Rede de Grande Distância;

Apêndice B

Interfaces

B.1 Fonte de Tráfego

- **Atributos**

  1. protected String nomeFonte;
  2. protected long numBitsTotal;
  3. protected long numBitsAtual;
  4. protected double inicioGeracao;

- **Métodos**

  1. public Fluxo gerarBits();
  2. public void enviarBits();
  3. private void incrementarNumBitsGerados();

B.2 Fluxo de Tráfego

- **Atributos**

1. private int tamanho;

- **Métodos**

1. public void alteraTamanho(int novoTamanho);
2. public int retornaTamanho();

B.3 Tráfego em Rajada

- **Atributos**

1. private long taxaDePico;
2. private long duracaoMediaRajada;
3. private long intervaloMedio;

- **Métodos**

1. public void alteraTaxaPico(double novaTaxa);
2. public void alteraDuracaoMediaRajada(double novaDuracao);
3. public void alteraIntervaloMedio();

B.4 Tráfego Variável

- **Atributos**

1. private long taxaMedia;
2. private long variancia;

- **Métodos**

1. public void alteraTaxaMedia(long novaTaxa);
2. public void alteraVariancia(long novaVariancia);

B.5 Tráfego Contínuo

- **Atributos**

  1. private long taxaMedia;

- **Métodos**

  1. public void alteraTaxaMedia(long novaTaxa);

B.6 Classe de Tráfego

- **Atributos**

  1. protected int pcr;
  2. protected float cdvt;

- **Métodos**

  1. public void alteraPCR(int novoPCR);
  2. public void alteraCDVT(float novoCDVT);
  3. public double retornaPCR();
  4. public double retornaCDVT();

B.7 rt-VBR e nrt-VBR

- **Atributos**

  1. private int scr;
  2. private float mbs;

- **Métodos**

  1. public void alteraSCR(int novoSCR);
  2. public void alteraMBS(float novoMBS);
  3. public int retornaSCR();
  4. public float retornaMBS();

B.8 ETBL

- **Atributos**

  1. protected String nomeETBL;
  2. protected float tempoProcessarCelula;

- **Métodos**

  1. public String retornaNomeETBL();
  2. public void alteraNomeETBL(String novoNome);
  3. public void alteraTempoProcessarCelula(float novoTempo);
  4. public float retornaTempoProcessarCelula();

B.9 ETBL Origem

- **Atributos**

  1. private long numCelulasGeradas;

- **Métodos**

  1. private Celula geraCelula();

2. private void enviaCelula();
3. private void incrementaNumCelulasGeradas();
4. public void alteranumCelulaGeradas(long novoNumCelulasGeradas);
5. public long retornaNumCelulasGeradas();

B.10 ETBL Destino

- **Atributos**

1. protected long numeroDeCelulasRecebidas;
2. protected double atraso;
3. protected double atrasoMax;
4. protected double atrasoMin;
5. protected double atrasoMed;
6. protected double varAtraso;

- **Métodos**

1. private void incrementaNumCelulasRecebidas();
2. private void calculaAtraso();
3. private void calculaVariacaoDeAtraso();
4. private void calculaAtrasoMin();
5. private void calculaAtrasoMax();
6. private void calculaAtrasoMed();
7. private void calculaAtrasoMed();
8. public double retornaAtrasoMin();
9. public double retornaAtrasoMax();
10. public double retornaAtrasoMed();
11. public long retornaNumCelulasRecebidas();

B.11 Célula

- **Atributos**

  1. private int vpi;
  2. private inr vci;
  3. private int pt;
  4. private int clp;
  5. private float tempoCriacao;

- **Métodos**

  1. public int retornaVPI();
  2. public void alteraVPI(int novoVPI);
  3. public int retornaVCI();
  4. public void alteraVPI(int novoVCI);
  5. public int retornaPT();
  6. public int retornaCLP();
  7. public void alteraCLP();
  8. public float retornaTempoCriacao();

B.12 Enlace

- **Atributos**

  1. protected String nomeEnlace;
  2. protected double distancia;
  3. protected double capacidade;

4. protected double fatorUtilizacao;

- **Métodos**

1. private void calculaFatorUtilazacao();
2. public float retornaDistancia();
3. public void alteraDistancia(double novaDistancia);
4. public float retornaCapacidade();
5. public void alteraCapacidade(double novaCapacidade);
6. public double retornaFatorUtilazacao();
7. public void alteraFatorUtilzacao(double novoFator);

B.13 Fila

- **Atributos**

1. private int tamanho;
2. private int tamMin;
3. private int tamMax;
4. private float tamMed;

- **Métodos**

1. public Celula retornaCelula();
2. public boolean insereCelula();
3. public void descartaCelula();
4. private void calculaTamanho();
5. private void calculaTamMax();
6. private void calculaTamMin();

7. private void calculaTamMedio();
8. public float retornaTamMin();
9. public int retornaTamanho();
10. public int retornaTamMax();
11. public float retornaTamMedio();

B.14 Comutador

- **Atributos**

1. private String nomeComutador;
2. private float tempoProcCelula;
3. private int valorCong;
4. private boolean statusCong = false;

- **Métodos**

1. public void comutaCelulas(Celula celula);
2. private void policiamento(Celula celula);
3. private void gerenciamento();
4. public boolean retornaStatus();
5. public void alteraStatus();
6. public String retornaNome();
7. public void alteraNome(String novoNome);
8. public void alteraTempoProcessarCelula(float novoTempo);
9. public float retornaTempoProcessarCelula();

B.15 Conexão de Caminho Virtual

- **Atributos**

  1. protected int vci;
  2. protected long numCelulasRoteadas;
  3. protected long numCelulasDescPoliciamento;
  4. protected long numCelulasDescGerenciamento;

- **Métodos**

  1. public int retornaVCI();
  2. public long retornaNumCelulasRoteadas();
  3. public long retornaCelulasDescPoliciamento();
  4. public long retornaCelulasDescGerenciamento();

B.16 Conexão Unicast

- **Atributos**

  1. private int vpi;
  2. private int vci;

- **Métodos**

  1. public int retornaVPI();
  2. public int retornaVCI();

B.17 Conexão Multicast

- **Atributos**

  1. private Vector Conexões;

- **Métodos**

  1. public int retornaVPI();
  2. public int retornaVCI();

Referências

[1] P. Heidelberger and L. Lavenberg, *Computer Performance Evaluation Methodology*, IEEE Transactions on Computers, Vol. C-33, pp. 1195–1220, December, 1984.

[2] M. Fowller and K. Scott, *UML Distilled - Applying the Standard Object Modeling Language*, Addison-Wesley, January, 1997.

[3] Ivar Jacobson, James Rumbaugh and Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[4] Ivar Jacobson, James Rumbaugh and Grady Booch, *The Unified Software Development Process*, Addison-Wesley, 1999.

[5] Averril M. Law and Michael G.McComas, *Simulation Software for Communications Networks: The State of Art*, IEEE Communications, Vol. C-33, pp. 44–50, March, 1994.

[6] Chell A. Roberts and Yasser M.Dessouky, *An Overview of Object-Oriented Simulation*, SIMULATION, Number 70, pp. 359–368, June, 1998.

[7] Ian F. Akyildiz and Keith L. Bernhardt, *ATM Local Area Networks: A Survey of Requirements, Architetures and Standards*, IEEE Communications Magazine, Number 70, pp. 72–80, July 1997.

[8] Martin de Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN*, Prentice Hall, Third Edittion, 1995.

[9] Luiz F. Gomes Soares, Guido Lemos Souza e Sérgio Colcher, *Redes de Computadores: Das LANs, MANs e WANs às Redes ATM*, Editora Campus, Rio de Janeiro, 1995.

[10] Luiz F. Gomes Soares, *Modelagem e Simulação Discreta de Sistemas*, VII Escola de Computação, São Paulo, 1990.

[11] William Ferreira Giozza, José Antão Moura Beltrão, Jacques Philippe Sauvé e José Fábio Marinho de Araújo, *Redes Locais de Computadores: Protocolos de Alto Nível e Avaliação de Desempenho*, McGraw-Hill/Embratel, São Paulo - SP, 1986.

[12] W. David Kelton, Randall P. Sadowski and Deborah A. Sadowski, *Simulation with Arena*, WCB/McGraw-Hill, 1998.

[13] The ATM Forum, *ATM: User Network Interface Specification*, Prentice Hall, 1993.

[14] The ATM Forum, *Traffic Managemnt Specification*, Version 4.0, April, 1996.

[15] Jerry Banks, John S.Carson II and Barry L. Nelson, *Discrete-Event System Simulation*, Prentice Hall, Second edition, 1996.

[16] José Augusto Suruagy Monteiro, *Rede Digital de Serviços Integrados de Faixa Larga (RDSI-FL)*, Departamento de Informática - UFPE, IX Escola de Computação, Recife - PE, 1994.

[17] ITU-T, *Recommendation I.350: General Aspects of Quality of Service and Network Performance in Digital Networks, including ISDN*, March, 1993.

[18] CCITT, *Revised Recommendation I.150: B-ISDN ATM Funcional Characteristics*, 1992.

[19] CCITT, *Traffic Control and Congestion Control*, 1993.

[20] James Rumbaugh, Michael Blaha and William Premerlani and Frederick Eddy and William Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

[21] John Amoss and Daniel Minoli, *IP Applications with ATM*, McGraw-Hill, 1998.

[22] Martin D. Carrol and Margareth A. Ellis, *Designing and Coding Reusable C++*, McGraw-Hill, 1998.

[23] Carlo Ghezzi, Mehdi Jazayeri and Dino Mandrioli , *Fundamentals of Software Engineering*, Prentice-Hall, 1991.

[24] Jacques Phillipe Sauvé, *O Processo de Desenvolvimento de Software na Light Infocom Tecnologia S/A*, Light Infocom S.A., http://www.dsc.ufpb.br/ pet/ 1997.

[25] Laura Lemay and Charles L. Perkins, *Teach Yourself Java in 21 Days*, Sams.net, 1996.

[26] Nada Golmie, Mark Corber, Yves Saintllan, Alfred Koenig and David Su, *The NIST ATM/HFC Network Simulator - Operation and Programming*, Third Edition, National Institute of Standards and Technology, Gathersburg - MD, March, 1998.

[27] Alta Group, *BONeS Designer - ATM Library Reference*, Release 1.0, Cadence Design Systems, Inc., Foster City - CA, December, 1994.

[28] Antônio Marcos Alberti, *SimATM: Um Ambiente para a Simulação de Redes ATM*, Dissertação de Mestrado, Universidade Estadual de Campinas - UNICAMP, Campinas - SP, 1998.

[29] Júlio Cézar Oliveira Gomes, *Development of a Solar-Terrestrial Analysis and Reference System (STARS) via Object Oriented Methodology*, Relatório de Estágio, Department of Applied Computer Science Ehime University - Japan, 1997.

[30] José Eduardo Carvalho Bussman, *BART - Uma Biblioteca Orientada a Objetos de Apoio à Recuperação Textual*, Dissertação de Mestrado, Universidade Federal da Paraíba - UFPB, Campina Grande - PB, 1995.

[31] Mark Grand and Jonathan Knudsen, *Java Fundamental Classes References*, O'Reilly and Associates Inc., 1997.

[32] David Flanagan, *Java in a Nutshell*, O'Reilly and Associates Inc., 1998.

[33] Fred Howell and Ross McNab, *SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling*, http://www.dcs.ed.ac.uk/home/fwh/emin/docs/websim/, Department of Computer Science - The University of Edinburgh, Scotland - UK, 1997.

[34] Erich gamma, Richard Helm, Ralph Johson and John Vlissides, *Design patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[35] Jackson Klein, Leonardo de Souza Mendes, Moisés Renato Nunes Ribeiro e Hélio Waldman, *SimNT: Manual do Usuário*, Versão 3.0, Departamento de Comunicações, Faculdade DE Eng. Elétrica e Computação, Universidade de Campinas, 1997.

[36] James Gosling and Henry McGilton, *The Java Language Environment: A White Paper*, Sun Microsystems, 1995.

[37] Livio Lambarelli, *ATM Service Categories: The Benefits to the User*, White Paper, http://www.atmforum.com, April, 1997.

[38] Fred Howell and Ross McNab, *A Guide to the Simjava Package*, Department of Computer Science - The University of Edinburgh, Scotland - UK, http://www.dcs.ed.ac.uk/home/hase/simjava, 1999.

[39] *Silk: A Java-Based Process Simulation Language*, Proceedings of the 1997 Winter Simulation Conference, Atlanta, GA, pp 475-482, 1997.

[40] *MODSIM-II: An Object-Oriented Simulation Language for Sequencial and Parallel Processors*, Proceedings of the 1989 Winter Simulation Conference, Piscataway, NJ, pp 172-189, 1989.

[41] Fred Howell and Ross McNab, *Simjava 1.2: Design Notes for Developers*, http://www.dcs.ed.ac.uk/home/hase/simjava, 1999.

[42] Hamilton S. Silva, José A . Suruagy Monteiro e William F. Giozza, *Concepção da Tabela de Estados das Conexões para o Comutador ATM*, Relatório Técnico, Versão 1.0, UFPB/UFPE/USP, 1995.

[43] Andrew S. Tanenbaum, *Redes de Computadores*, Tradução da Terceira Edição, Editora Campus, 1997.

[44] Hamilton S. Silva, José A . G. Lima, José A . Suruagy Monteiro e William F. Giozza, *Arquitetura Básica e Descrição Funcional do Comutador ATM*, Relatório Técnico, Versão 1.0, UFPB/UFPE/USP, Maio, 1995.

[45] R. McNab and F. W. Howell, *Using Java for Discrete Event Simulation* The University of Edinburgh, UK, 1996.

[46] Cay S. Horstmann and Gary Cornell, *Core Java - The Fundamentals*, Volume I, Sun Microsystems Press, 1999.

[47] Niklas Landin and Axel Niklasson, *Development of Object Oriented Frameworks*, Department of Communication Systems, Lund University, Lund, Sweden, 1999.

[48] Michael T. Goodrich and Roberto Tamassia, *Data Structures and Algorithms in Java*, John Wiley and Sons Inc., 1998.

[49] Kent Bech and Erich Gamma, *Test Infected: Programmers Love Writing Tests*, http://c2.com/cgi/wiki?JavaUnit, 1999.

[50] Carmine Mangione, *Performance Tests show Java as fast as C++*, http://www.javaworld.com/jw-02-1998/jw-02-jperf.html, 1999.

[51] Borland International, Inc., *Jbuilder 2 - Developer's Guide*, Manual do Usuário, 1998.

[52] Paulo Sérgio Sausen, *Simulação e Análise de Mecanismos de Controle de Fluxo para Serviço ABR em Redes ATM*, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Agosto, 1998.