MARCELO OTTE

A UTILIZAÇÃO DA FERRAMENTA SMV NA VERIFICAÇÃO FORMAL DE SISTEMAS DE COMUNICAÇÃO DE DADOS

FLORIANÓPOLIS
1999

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

A UTILIZAÇÃO DE FERRAMENTAS DE VERIFICAÇÃO FORMAL EM SISTEMAS DE COMUNICAÇÃO DE DADOS

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica

MARCELO OTTE

Florianópolis, maio de 1999

A UTILIZAÇÃO DA FERRAMENTA SMV NA VERIFICAÇÃO FORMAL DE SISTEMAS DE COMUNICAÇÃO DE DADOS

MARCELO OTTE

'Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em Controle, Automação e Informática Industrial, e aprovada em sua forma final pelo Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. José Eduardo Cury, Dr.
Orientador

Edson Roberto De Pieri, Dr.
Coordenador do Curso de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. José Eduardo Cury, Dr.
Presidente

Prof. Jean Marie Farines, Dr.

Murilo da S. Camargo, Dr.

Vitório B. Mazzola, Dr.

"O crédito pertence àquele que lutou com denodo; que não desistiu nunca; que sente grande entusiasmo e empenho; que se devota a uma boa causa; e que no melhor dos casos, obtém ao fim o triunfo da grande realização; e que, no pior, quando fracassa, ao menos fracassa ousando galhardamente. Assim, o seu lugar nunca será entre aquelas almas tímidas, que não conhecem nem a vitória nem a derrota"

(THEODORE ROOSEVELT)

AGRADECIMENTO

A todos aqueles que me ajudaram na realização de mais um desafio, em especial:

Aos meus pais, que sempre me apoiaram.

A José Eduardo Cury, que me orientou com várias idéias.

A todos os profissionais do Laboratório de Controle e Micro-Informática – LCMI, sob coordenação de Jean Marie Farines, que me ensinaram.

A Universidade Federal de Santa Catarina - UFSC, que investiu no meu desenvolvimento.

Aos professores da UFSC, que partilharam seus conhecimentos.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

A UTILIZAÇÃO DE FERRAMENTAS DE VERIFICAÇÃO FORMAL EM SISTEMAS DE COMUNICAÇÃO DE DADOS

Marcelo Otte

Maio/1999

Orientador: Prof. Dr. José Eduardo Cury.
Área de Concentração: Controle, Automação e Informática Industrial.
Palavras-chave: Verificação formal, Checagem de Modelos, comunicação de dados, ATM.
Número de Páginas: 154

A utilização de ferramentas de verificação formal em sistemas de comunicação de dados é um tema que contem diversos tópicos de pesquisa que estão constantemente recebendo o ingresso de novos estudos. Apesar da essência do conceito de verificação formal já estar fundamentada, novas metodologias, técnicas, estratégias e ferramentas estão sendo desenvolvidas a cada dia, tornando a verificação formal cada vez mais próxima e adequada à utilização por projetistas que desenvolvem sistemas, tais como os de comunicação de dados.

O mito de que as ferramentas de verificação formal estão restritas ao meio acadêmico, onde apenas matemáticos estariam aptos a utilizá-las, pode ser desmistificado com trabalhos que mostram como uma pessoa que conhece um sistema pode tirar proveito da verificação formal para identificar erros. Este trabalho nasceu em torno deste objetivo, mostrando características das metodologias e ferramentas de verificação, e como elas podem ser aplicadas num sistema de comunicação de dados.

O trabalho foi dividido em capítulos inter-relacionados. No capitulo 1 trabalham-se os principais conceitos relacionados ao tema. No capítulo 2 apresentam-se as vantagens e desvantagens da verificação formal sobre a simulação. São mostrados alguns métodos

aplicáveis na verificação formal, tecendo algumas considerações e comparações entre eles. Mostra-se o poder da Checagem de Modelos e suas aplicações, introduzindo o capítulo 3. Neste capítulo trabalham-se detalhes da Checagem de Modelos, mostrando como é feita a verificação através desta metodologia e apresentando seus maiores problemas, como a explosão de estados, e as principais soluções, além de mostrar como se utilizam as especificações através de Lógicas Temporais. Apresentam-se também as principais ferramentas de Checagem de Modelo do mercado, o SPIN e o SMV, introduzindo o capítulo 4. Neste são mostradas algumas características especificas da ferramenta SMV, que será utilizada na análise de um sistema padronizado pelo ATM Forum. Durante todo o capítulo a ênfase é dada sobre o SMV, porém, ao invés de abrir um capítulo para a solução SPIN, esta ferramenta é comparada com o SMV ao longo da exposição. No capítulo 5 apresentam-se as principais características a serem consideradas na verificação de protocolos de comunicação de dados. Através da utilização do modelo OSI como filosofia buscou-se abranger qualquer protocolo. Entretanto houve a necessidade de se mostrar algumas características próprias do sistema a ser discutido, o ATM, que é apresentado no capitulo 6. Durante todo este capitulo mostraram-se características do ATM, que é um sistema de comunicação de dados relativamente simples que tira proveito das baixas taxas de erros dos modernos sistemas de transmissão. Dentro do universo de diferentes aplicações suportadas pelo ATM focou-se num aspecto específico, o gerenciamento de tráfego, que é trabalhado dentro do capítulo 7, utilizando-se da ferramenta SMV para verificação de projetos. O trabalho conclui no capitulo 8 com as recomendações e conclusões propriamente ditas.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for degree of Master in Electrical Engineering.

THE USE OF FORMAL VERIFICATION TOOLS ON DATA COMMUNICATION SYSTEMS

Marcelo Otte

May/1999

Advisor: Prof. Dr. José Eduardo Cury.
Area of Concentration: Control, Automation and Industrial Computer Science.
Keywords: Formal Verification, Model Checking, data communication, ATM.
Number of Pages: 154

The use of formal verification tools on data communication systems is a theme that includes serveral topics of research, which on their part are continuously receiving the input of new studies. Although the fundamentals of formal verification are already set, new methodologies, techniques, strategies and tools are being developed day after day, bringing formal verification always closer and rendering its use more adequate for designers who work on the development of systems, such as those on data communication.

The myth that formal verification tools are restricted to academic circles, where only mathematicians would be able to use them, can be demystified by works which demonstrate how a person who has a good knowledge of a system may benefit from formal verification in order to detect errors. This work arose around this purpose, presenting the attributes of the methodologies and verication tools and in which manner they can be performed on a data communication system.

This work has been divided into interrelated chapters. In Chapter 1 - Introduction, the main concepts of the theme are laid out. In Chapter 2 - Advantages and disadvantages of formal verification over simulation are presented. Several applicable formal verification methods are shown, while some considerations and comparisons are drawn between them.

The power of Model Checking and its applications are demonstrated, leading to Chapter 3. In this chapter detail about Model Checking is delt with, demonstrating how the verification is made through this methodology and presenting the major problems attached thereto, as is the case of state explosion, and the main solutions, as well as how the specifications are used through the Temporal Logics. The main Model Checking tools found in the market, the SPIN and the SMV, are presented too, leading to Chapter 4. In this chapter, several specific characteristics of the SMV tool are shown, which is to be used for the analysis of a system standardized through the ATM Forum. Through the whole course of this chapter a special emphasis is placed on SMV, but no chapter has been opened on the SPIN solution, this tool having been compared to the SMV along the exposition, instead. Once the methods and the tools have been presented, the system to be verified has to be introduced. Thus, in Chapter 5 the main characteristics to be taken into account in the verification of a protocol are presented. This chapter strives to cover any protocol, especially because it uses the OSI as philosophy. However the need arose to show some characteristics that are particular to the system to be discussed, namely the ATM, which is seen in Chapter 6. During this chapter, applications of ATM are delt with in detail, seeking as a basis the ideas explained in the preceding chapter. The ATM is a relatively simple data communication system, which benefits from the low error rates of modern transmission systems, however there are several different characteristics for several ATM supported applications, what makes it necessary, within the scope of this work, to point towards a specific aspect, the traffic management, which is delt with in Chapter 7, using the SMV tool in the verification of projects. The work closes in Chapter 8, with the recommendations and conclusions proper.

Sumário

Índice de Figuras

Índice de Tabelas

Capítulo 1 - Introdução

O constante avanço tecnológico, que está proporcionando à humanidade um progresso em taxas nunca antes vistas, pode ser percebido em quase todas as áreas de conhecimento e atuação. Da área social, passando pela biológica, e finalmente chegando as áreas de engenharia e desenvolvimento, a tecnologia vem permitindo atingir objetivos nunca sonhados por gerações passadas. Mas é no desenvolvimento de sistemas e dispositivos que o avanço tecnológico pode ser notado de forma mais intensa e disseminada. Empresas desenvolvem sistemas de ultima geração para realização de processos cada vez mais rápidos, complexos e abrangentes que os sistemas anteriores, e se encontram constantemente pressionadas pelo mercado competitivo. Tempos de projeto e desenvolvimento precisam ser encurtados. Sistemas que duravam 10 anos, hoje precisam ser substituídos a cada 5 ou 3 anos devido a sua obsolescência.

Muito do avanço tecnológico em áreas como a biologia, medicina, construção, transportes, militar, e outras, se deve à evolução de dispositivos dotados de sistemas cada vez mais eficientes, complexos e baratos. Os sistemas computacionais, com softwares que dão vida a sistemas de hardware com milhões de transistores, são uma das coisas mais complexas jamais construídas pela humanidade. Com o barateamento dos computadores e a sua larga aceitação, cada vez mais se exige que tarefas mais complicadas sejam realizadas.

Infelizmente está cada vez mais difícil garantir que os complexos sistemas atuais estejam livres de erros, pois os custos dos testes e validações tradicionais cresce exponencialmente com o aumento do sistema. Assim, muitos sistemas atuais estão sendo lançados no mercado sem uma validação completa [Hol91].

Para a maioria dos sistemas atuais os erros podem ser corrigidos causando o mínimo prejuízo para o usuário. Entretanto, quando se pensa em sistemas nos quais um erro pode comprometer a vida humana, a validação se mostra além de tudo uma obrigação. Sistemas que controlam dispositivos de exploração espacial, aeronáutica, e outros, muitas vezes não permitem que um erro seja detectado e consertado a posteriori, levando em muitos casos à morte [JH97].

Além do prejuízo para o usuário, um sistema com erros não detectados custa muito caro para as empresas envolvidas no processo de desenvolvimento e comercialização do produto,

principalmente para aquelas que desenvolvem sistemas que se utilizam de circuitos digitais, ciências da computação e protocolos de comunicação de dados [CGL96].

Muitos sistemas de validação, como as simulações, não conseguem trabalhar com o nível de complexidade dos sistemas atuais com a velocidade e confiança exigidos pelos projetistas. Entretanto algumas técnicas estão sendo aprimoradas para realizar tal tarefa. Destas, as mais promissoras são as técnicas conhecidas como métodos formais, que utilizam lógica matemática para provar que sistemas estão livres de erros [Chr98].

Existem diversas ferramentas com diversas peculiaridades para verificação formal. Considerando que cada ferramenta pode ter uma linguagem para descrever o sistema em análise, uma linguagem para descrever os comportamentos desejados do sistema, um algoritmo próprio para trabalhar os dados, e um conjunto de informações sobre a análise, pondera-se que existam inúmeras variáveis que devem ser levadas em conta pelo usuário na escolha de uma ferramenta específica.

As ferramentas de verificação formal precisam ser alimentadas com as características do sistema e especificações em análise. Por mais amigável, bem concebida e completa que seja a ferramenta, cabe ao usuário entender o seu funcionamento para saber como descrever um sistema em análise. Como modelar um sistema? Quais são as características que são fundamentais e aquelas que podem ser desconsideradas?

Na modelagem de sistemas de comunicação de dados existem diversas questões a serem consideradas. De acordo com a metodologia, ferramenta, e estratégias utilizadas para focar o problema, a verificação pode ser uma tarefa mais fácil, mais conclusiva, completa, ou não.

Buscou-se através deste trabalho, em duas etapas, entender melhor a utilização de ferramentas de verificação em um sistema específico. Na primeira etapa buscou-se fazer uma análise das principais metodologias e ferramentas, analisando suas características principais, mostrando vantagens e desvantagens de cada uma. Numa segunda etapa, utilizando a ferramenta "Symbolic Model Verifier" (SMV), demonstraram-se algumas análises e considerações sobre como verificar sistemas de comunicação de dados,. De uma forma concisa, a questão problema pode ser expressa como:

Como as ferramentas automáticas de verificação formal, tais como o "Symbolic Model Verifier" (SMV), são aplicadas na análise de modelos de sistemas a eventos discretos que descrevem o funcionamento de sistemas de comunicação de dados?

Valorada e validada a questão problema, estabelecem-se respostas prováveis e provisórias, denominadas hipóteses:

- Existem tantos sistemas com características distintas que determinadas ferramentas de validação são mais eficazes e eficientes em certos tipos de sistemas em detrimento de outras.

- O usuário precisa ter um conhecimento profundo do sistema a ser submetido a uma verificação por qualquer ferramenta. Características modeladas incorretamente poderão mascarar erros futuros.

- O usuário precisa conhecer bem a ferramenta a ser utilizada antes de afirmar se um sistema apresenta-se livre de erros ou não.

- Um determinado tipo de sistema, como por exemplo um sistema de comunicação de dados, apresenta características comuns a todos os sistemas deste tipo.

- Estratégias de modularidade e abstração podem simplificar o modelo dos sistemas analisados, permitindo a analise e verificação de sistemas maiores.

- A metodologia e linguagens, utilizadas em cada ferramenta para descrever o comportamento dos sistemas, e as especificações a serem testadas, influenciam em características da verificação, tais como velocidade, automatização, expressividade, cobertura, segurança.

Definidas as hipóteses estabelece-se que o objetivo geral deste trabalho é o de mostrar que, adotando metodologias e estratégias de modelagem e verificação adequadas, pode-se utilizar ferramentas de verificação formal, como o SMV, para identificar, projetar e comprovar comportamentos de sistemas reais, como os sistemas de comunicação de dados. Deste objetivo derivam-se os seguintes objetivos específicos:

- Descrever as metodologias de verificação de sistemas, apresentando suas características básicas e escolhendo aquela mais adequada ao objetivo geral;
- Dentre da metodologia de verificação escolhida apresentar as ferramentas mais utilizadas e escolher aquela que será utilizada na verificação dos modelos;
- Apresentar uma metodologia para modelagem de sistemas de comunicação de dados;
- Modelar um sistema de comunicação de dados;
- Definir quais são as especificações desejadas para um sistema de comunicação de dados;
- Utilizar uma ferramenta para verificar se as especificações desejadas são válidas para o sistema modelado.

Assim, utilizando as teorias e métodos gerais que sustentam as ferramentas de validação de sistemas a eventos discretos, e as técnicas para análise de sistemas de comunicação de dados, aplicou-se a ferramenta SMV para verificação de comportamentos de um modelo construído baseado no sistema de comunicação de dados ATM.

O trabalho foi estruturado em capítulos, cuja denominação e breve comentário são apresentados a seguir:

- **Capítulo 1 – Introdução** – Este capítulo introduz os principais conceitos e apresenta características do trabalho, tais como: tema, hipóteses, objetivos e metodologia.

- **Capítulo 2 – Metodologias de Verificação** – Define sistemas, apresenta metodologias de verificação e as vantagens de se utilizar a verificação formal sobre a não formal, discorre sobre os principais métodos de verificação formal e não formal, e mostra o que está se delineando para o futuro das metodologias de verificação.

- **Capítulo 3 – A Checagem de Modelos em detalhe** – Um dos métodos de verificação formal mostrados no Capítulo 2, a Checagem de Modelos, apresenta uma série de características que o tornaram largamente utilizado nas ferramentas atuais. Este capítulo mostra os detalhes desta metodologia.

- **Capítulo 4 – Ferramentas de Verificação Formal** – Mostra características das ferramentas mais conhecidas para as metodologias formais mostradas no capítulo 2 e no capítulo 3.

- **Capítulo 5 – Validando Protocolos de Comunicação** – Define o conceito de protocolo, apresenta as características e serviços prestados e estabelece os principais pontos e estratégias para verificação de um sistema de comunicação de dados.

- **Capítulo 6 – ATM – Modo de Transferência Assíncrono** – Apresenta as principais características deste modo de transmissão de dados, e mostra como é feita a gerência de tráfego do ATM, que dará forma aos modelos verificados no capítulo 7.

- **Capítulo 7 – Verificando características de uma rede ATM** – Trabalham-se as características da gerência de tráfego do ATM, mostradas no capítulo 6, através da aplicação de estratégias de verificação utilizando a ferramenta SMV.

Capítulo 2 – Metodologias de Verificação

Na verificação de sistemas utilizam-se modelos e metodologias adequados aos objetivos pretendidos. Neste capítulo mostrar-se-á o porquê da utilização de modelos e qual o mais adequado para a análise a ser desenvolvida nesta dissertação. Mostrar-se-ão também quais as principais metodologias de verificação e as vantagens e desvantagens de cada, escolhendo também a mais adequada ao objetivo do trabalho.

2.1 – Modelando os sistemas

Na natureza existem diversos tipos de sistemas com diferentes características. Alguns são simples, outros são complexos, muitos não foram satisfatoriamente entendidos, outros ainda não foram identificados. Para Aurélio [Fer88], um sistema é: *"conjunto de elementos, materiais ou ideais, entre os quais se possa encontrar ou definir alguma relação"*. O ser humano em sua incansável busca pelo saber está constantemente identificando relações entre os elementos que o cercam, utilizando diversas ferramentas para tal fim. Para analisar, entender, e sintetizar um sistema utiliza-se do artifício da modelagem, que é uma ferramenta muito poderosa e eficiente, embora deva ser constantemente discutida.

Durante muitos anos as relações entre elementos naturais eram o único foco dos modelos para sistemas. Com a industrialização e a informatização o homem passou a construir sistemas cada vez mais complexos. Assim, a definição anterior de sistema feita por Aurélio, onde o homem exerce uma função mais passiva, analisando o que já existe e definindo relações, passa a ser melhor definida como a *"disposição das partes ou dos elementos de um todo, coordenados entre si, e que funcionam com estrutura organizada"* [Fer88], onde o homem passa a agente ativo e de transformação.

2.1.1 – Classificando sistemas

Os sistemas podem ser classificados conforme a figura 2.1 [Cur94].

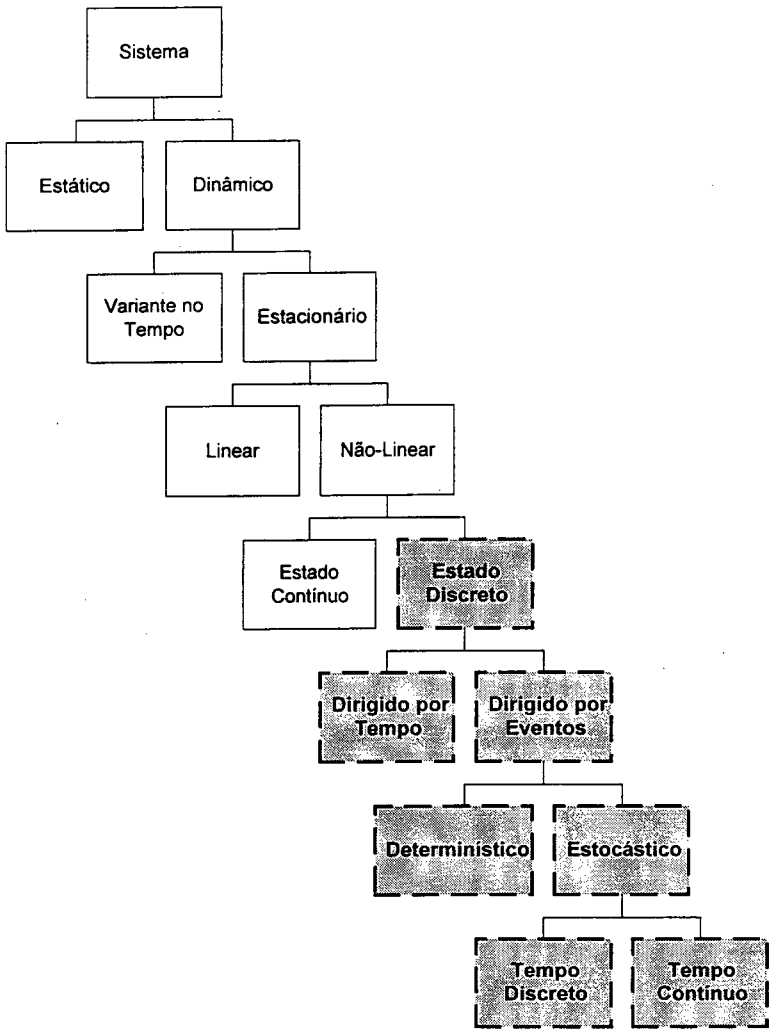


Figura 2.1 – Classificação dos Sistemas

Na figura 2.1, os sistemas que são compreendidos pela área destacada são denominados Sistemas a Eventos Discretos (SED), e são largamente construídos pelo homem, principalmente desde o nascimento dos computadores. Alguns exemplos que pertencem a esta classe de sistemas são: redes de comunicação, sistemas de produção, sistemas de tráfego, sistemas logísticos, sistemas operacionais.

Os SEDs apresentam duas características principais [Cur94]: apresentam grandezas discretas e evoluem por eventos instantâneos. A figura 2.1 mostra que tais sistemas apresentam estados discretos contrapondo-se aos sistemas de estados contínuos.

2.1.2 – Modelando os sistemas

Para os sistemas denominados contínuos as grandezas variam num espaço de infinitos estados. Grande parte dos fenômenos naturais são contínuos e são eficientemente modelados por equações diferenciais ordinárias e parciais, constituindo-se num paradigma para a análise e controle destes sistemas [Cur94].

Para os sistemas a eventos discretos existem várias formas de modelagem. Entretanto são dois os fatores básicos que determinam como será modelado um sistema: as características do próprio sistema e os objetivos pretendidos com o modelo.

Os modelos que se preocupam em analisar as possíveis seqüências de estados, ou eventos gerados; informar se um determinado estado pode ser alcançado; ou se é possível a ocorrência de uma seqüência de eventos, são denominados modelos lógicos. Eles podem explicitar ou não o tempo dos eventos/transições de estados.

Os modelos que podem responder questões sobre desempenho e performance do sistema quanto a tempos, médias, e outras questões relacionadas aos tempos envolvidos, são conhecidos como modelos de performance, e precisam ter o tempo explicitado.

Dentro de cada grupo de modelos (lógico ou performance) existem diferentes metodologias de modelagem, e muitas pesquisas estão sendo desenvolvidas na área de sistemas a eventos discretos.

> "De todo modo, nenhum dos modelos serve atualmente como paradigma. Os SEDs formam uma área de pesquisa de intensa atividade e desafios" [Cur94]

2.1.3 – Autômatos de estados finitos – (Finite state automata).

Uma das modelagens mais utilizadas para SED é a de autômatos de estado. As principais ferramentas para verificação formal, que serão mostradas no capítulo 4, utilizam esta estratégia ([Hol91] e [CGL96]).

Independente da variação do modelo de autômato utilizado para modelar um SED, ele contém um conjunto de estados possíveis e um conjunto de transições entre estados [CGL96].

Um autômato de estados finitos modela um SED através de quatro elementos principais: um alfabeto, que é um conjunto que representa os eventos, um conjunto finito de estados do SED, um mapa que define os eventos factíveis para cada estado, e uma função de transição de estados [Cur94]. Pode-se acrescentar ao modelo um conjunto de saídas e uma função que mapeie os estados e eventos nestas saídas, constituindo um modelo com seis elementos. Um modelo que será discutido posteriormente, conhecido como estrutura de Kripke, utiliza um conjunto de estados, uma função de transição e uma marcação dos estados definindo suas características [CGL96].

Modelos baseados em autômatos são dualmente baseados em Linguagens: pode-se descrever o comportamento do sistema através da linguagem formal gerada pelos autômatos. A linguagem privilegia a descrição comportamental externa, focando a análise nas seqüências de eventos que o processo pode gerar, enquanto que os autômatos descrevem a estrutura interna [Cur94]. Nos capítulos 3 e 4 ver-se-á que as técnicas de verificação formal, que incorporam o método denominado de Checagem de Modelos (o mais difundido atualmente), utilizam os autômatos para descrição dos SEDs. Os autômatos permitem diversas manipulações, como por exemplo minimização, combinação e determinização, possibilitando a análise e verificação de sistemas interrelacionados [CGL96], [Hol91], [Mcm92].

A modelagem por Linguagens é uma ferramenta muito poderosa, que, além de permitir análises e sínteses de sistemas, também é a base para diversas estratégias de controle. Utilizando técnicas de abstração, hierarquização, observações parciais, e outras é possível implantar controles de SEDs com sucesso [Won94], [CFC94].

Várias descrições de SEDs utilizam Diagramas de Transição de Estados (DTE) como representação gráfica de autômatos, embora tabelas de transição também sejam utilizadas. No capítulo 3 estruturas de Kripke serão representadas graficamente através de DTEs.

2.2 - As metodologias de verificação

2.2.1 – Por que utilizar verificação formal?

No capítulo 1 apresentaram-se as razões para a verificação de um sistema. De uma forma sucinta tem-se:

- Permite com um baixo custo testar diversas variações de sistemas, ajudando no projeto de diferentes dispositivos de controle, interface e outros;
- Diminui a probabilidade de se detectarem erros posteriormente, evitando o alto custo de correção destes, o que poderia inviabilizar o produto;
- Diminui o risco de vida de seres vivos que dependem do sistema testado;
- Ajuda na síntese de processos partindo de especificações de auto-nível.

CLARKE e WING [CW94] em seu artigo *"Formal Methods: State of the Art and Future Directions"*, afirmam que o uso de métodos formais não garante a priori a correção. Mesmo considerando uma análise extensiva e completa sobre um modelo muitos pontos de erros podem passar desapercebidos, principalmente por causa da própria modelagem e/ou especificação. Mesmo assim, o uso de métodos formais pode aumentar o entendimento de um sistema, mostrando inconsistências, ambigüidades, e soluções incompletas, que dificilmente seriam detectadas sem o uso destes métodos.

2.2.2 – Por que não se utiliza verificação formal rotineiramente?

Apesar do conceito de verificação formal ser anterior à década de 80 e de existirem diversos métodos e ferramentas disponíveis, a verificação formal não é uma prática difundida. Algumas razões são apontadas pelos diversos autores. CLARKE e WING [CW94] afirmam que no passado o uso dos métodos de verificação formal na prática parecia algo impossível, principalmente por causa das notações obscuras, da dificuldade de utilizar as ferramentas da época, e da inadequação destas à prática. Apesar deste estado de coisas ter existido no passado, e muitas melhorias nos métodos e ferramentas terem mostrado outro panorama para as verificações formais, provavelmente ficou o preconceito quanto a qualquer solução oriunda deste ramo de conhecimento [CW94]. Uma das razões, e também conseqüência, deste preconceito são os diversos conceitos equivocados propagados pelas próprias revistas científicas, inclusive o próprio IEEE, apontado por BOWEN [BH94]. Ele apresenta sete mitos relacionados aos métodos formais, complementando outros sete mitos apresentados anteriormente por HALL [Hal90]. Desta lista de mitos pode-se ver claramente que o público alvo dos métodos de verificação ainda não compreendeu os conceitos relacionados a esta poderosa ferramenta de projeto.

Alguns dos 14 mitos citados apontam algumas razões para a não utilização dos métodos formais. Um mito citado pelo autor afirma que a aplicação de métodos formais exige matemáticos profundamente treinados, confirmando a má impressão causada pelos primeiros esforços nesta área, com notações confusas e fortemente ancoradas em lógica matemática. Mostrar-se-á, ao longo deste e dos próximos capítulos que os avanços das técnicas de verificação formal, apesar de baseados em lógica matemática, são acessíveis a qualquer projetista. O importante, como será mostrado, é conhecer bem o sistema a ser verificado.

A principal razão pelo insucesso dos métodos formais, de acordo com DILL [Dil98], é a razão econômica. Para ele o tempo de projeto de um sistema é crucial, e muitas vezes o uso de métodos formais pode atrasar o projeto. O autor aponta que uma semana de atraso de um projeto de um microprocessador eqüivale a uma perda de dez milhões de dólares. Tal afirmação é considerada por BOWEN [BH94] como um dos mitos sobre verificação formal. O autor afirma que não existe um número suficiente de aplicações para formar uma cultura de verificação formal que possa fornecer informação suficiente sobre quanto tempo é economizado e quando tempo é investido num projeto.

Outro mito apresentado por BOWEN [BH94] é o que afirma que Métodos formais só se aplicam a software. Apesar de se encontrar em diversas ferramentas focadas em software, encontram-se diversos exemplos e relatórios de utilização de ferramentas em protocolos, hardware, etc.

Dos mitos apresentados pode-se juntar três que tem a mesma natureza num que diria: Os métodos formais substituem com eficácia e eficiência todos os outros métodos. Tal afirmação é falsa, pois os métodos formais não garantem a priori a correção [CW96]. Além disto, os mesmos autores mostram que a tendência é ter-se no futuro a união de vantagens de diversos métodos. LUCENT [Luc97] afirma que, apesar do poder de técnicas como a Checagem de Modelos, elas não irão eliminar a simulação, pois em sistemas com um número extremamente alto de elementos a serem considerados, tanto a simulação, com vetores de entrada previamente determinados, como as técnicas de exaustão são incapazes de "varrer" todos os cenários possíveis. Assim utilizam-se, por exemplo, técnicas de simulação que geram vetores randômicos, que também não garantem a priori a correção, mas são mais eficazes na detecção de erros em sistemas com grande número de elementos do que os outros métodos citados. Além disto, técnicas formais não podem checar circuitos dependentes de tempo, como osciladores. Estes circuitos caem no comportamento analógico que não são manipulados pelas representações matemáticas das técnicas formais.

2.2.3 – Diferenças entre Métodos Formais e Não-Formais

Pode-se classificar as ferramentas de verificação de sistemas em duas grandes famílias: as ferramentas não-formais e as formais. Em "White Paper: What is Formal Verification?" [Chr98] afirma-se que a diferença entre os métodos formais e os não-formais está basicamente na utilização de métodos matemáticos que apresentem soluções completas, ou seja, quando utiliza-se um método formal, e obtêm-se uma solução, pode-se garantir que todas as possibilidades do modelo do sistema foram consideradas, diferentemente do caso dos métodos não-formais, como a simulação, onde cenários com erros podem passar desapercebidos na verificação de um sistema através de vetores de entrada

A simulação é a família de técnicas de verificação não-formais mais difundidas. Durante décadas a simulação era o único meio para verificar a integridade de um projeto antes da industrialização [Luc97]. Para verificar um sistema utilizando simulação, o usuário deve aplicar vetores de teste na entrada do sistema em análise e comparar os vetores de saída resultantes com valores esperados [Nor96]. Estes testes supõem que, se para determinado conjunto de vetores de entrada o sistema se comporta dentro de determinados padrões, então o sistema está livre de erro. Entretanto tal afirmativa não é verdadeira, pois alguns aspectos do projeto do sistema podem ter passados desapercebidos [Chr98]. Além disto, muitos dos sistemas atuais são tão grandes que os projetistas não conseguem mais criar todos os vetores necessários para verificar o sistema adequadamente através da simulação.

Outro problema com a simulação é o tempo envolvido, que vem crescendo muito mais rápido do que o crescimento do tamanho dos sistemas. Um exemplo são os tempos de simulação dos novos chips, que demoram dias ou até semanas. Por isso, no mercado competitivo as pressões de tempo forçam os projetistas a utilizar apenas um subconjunto dos vetores para verificar cada revisão. Isto reduz o tempo de simulação mas pode deixar passar erros sem serem detectados.

As ferramentas de verificação formal, através da utilização de técnicas de dedução racional emprestadas da matemática, comparam a lógica do sistema em análise diretamente contra a lógica da especificação, fornecendo geralmente uma análise completa do sistema. Assim, elimina-se o risco que os projetistas teriam de escolher um subconjunto incompleto de vetores de entrada. Além disto, as ferramentas de verificação formal são muito mais rápidas

do que a simulação, completando a verificação numa fração do tempo que um simulador demoraria para analisar um conjunto de vetores adequado [Luc97]

2.2.4 – Os tipos de métodos de verificação não formal

Na família de ferramentas não formais tem-se: [Chr98]

- **Métodos analíticos** – Expressões matemáticas fornecem informações sobre variáveis do sistema.
- **Métodos de simulação** – O modelo é avaliado através da análise dos dados de saída em função de dados de entrada, inferindo o valor de variáveis de interesse.

2.2.5 – Os métodos de verificação formal

Apesar de se encontrarem diversas denominações de métodos de verificação formal, apenas dois métodos são citados unanimemente pelos autores: os Provadores de Teoremas e os Checadores de Modelos (Model Checkers). Além destes dois, alguns métodos geralmente apresentam características muito semelhantes, podendo ser considerados variações. É o caso do Checador de Provas (Proof Checker), no qual o documento "*Formal Verification Tools*" [JH97] afirma ser muito confundido com os Provadores de Teoremas. Apesar de aparentemente semelhantes o autor afirma que estes são mais automáticos do que aqueles, apesar de aqueles permitirem um maior controle sobre a prova.

2.2.5.1 – Checadores de Modelos

Para a quase totalidade dos autores, a Checagem de Modelos é um método de verificação formal desenvolvido para tornar mais confiável e automática a verificação de sistemas. As especificações desejadas são expressas em lógica temporal e o sistema é modelado através de conjuntos de transição de estados [CGL96]. As ferramentas que utilizam esta metodologia buscam atingir a prova por exaustão. Para um dado sistema elas geram todos

os possíveis estados de um sistema e testam cada estado individualmente. Os algoritmos de Checagem de Modelos fornecem uma resposta afirmando se o sistema é válido ou fornecendo um contra-exemplo mostrando como a especificação não é atendida [CGL96]. Porém nem sempre as ferramentas conseguem analisar sistemas grandes e complexos devido à explosão de estados.

Ferramentas comerciais utilizando técnicas de Checagem de Modelos estão apenas começando a entrar no mercado, apesar de já existirem diversas ferramentas produzidas por universidades tais como o SMV e o Murphi. Estas ferramentas são Checadores de Modelos para uso geral, mas exigem que o usuário expresse as propriedades em linguagens especializadas, como o CTL ou LTL [CGL96], [Mcm98].

As ferramentas que utilizam as técnicas de Checagem de Modelos necessitam de uma grande quantidade de memória para trabalhar sistemas concorrentes, pois o número de estados tende a crescer exponencialmente com o aumento da complexidade do sistema. Sistemas simples podem produzir milhões de estados. Diversos autores tratam o problema de explosão de estados, como [Mcm92], [Yan93], [BCM90], e outros. Esse problema será detalhado no capítulo 3, e algumas técnicas e estratégias para reduzi-lo serão mostrados nos capítulos 5 e 7.

Checadores de Modelos podem testar três tipos básicos de erros. Entretanto, não importando qual tipo de erro esteja sendo procurado, eles precisam de algum controle humano ou utilizar objetivos definidos pelo usuário.

- **Erros quanto à especificação** – Checadores de Modelos permitem que determinadas características sejam construídas na forma de especificações, que serão confrontadas com o sistema sob verificação. Pode-se identificar estados de erro, seqüências inválidas de estados, violação de critérios de justiça, entre outros.

- **Congelamentos (deadlocks)** – Estados dos quais o sistema não consegue escapar, isto é, uma vez atingido este tipo de estado o sistema, num tempo futuro, nunca atingirá outro estado. A ferramenta deve encontrar todos os estados em que o sistema não evolui, apontando quais os caminhos/eventos que levaram a tais situações. Os estados finais do sistema tem todas as características citadas para um congelamento. Assim, a ferramenta também deve ser capaz de diferencia-los baseado em informações fornecidas pelo usuário.

- **Ciclos não progressivos**- São ciclos dos quais o sistema não consegue sair e nos quais nenhum trabalho é feito. Para checar estes erros, a ferramenta deve procurar por ciclos onde o sistema pode ficar em loop, e, baseado em especificações do usuário de quais são os ciclos desejáveis, apontar aqueles que não o são.

2.2.5.2 –Provadores de Teoremas / Checadores de Prova

Os Provadores de Teoremas ainda estão sob desenvolvimento acadêmico. As ferramentas para este método são baseadas em linguagens matemáticas especializadas, sendo mais aplicável no desenvolvimento de sistemas de alto nível [Chr98]. CLARKE e WING [CW96] afirmam que a Prova de Teoremas é uma técnica em que tanto o sistema como as especificações são expressas como fórmulas matemáticas. A lógica é dada por um sistema formal que define um conjunto de axiomas e um conjunto de regras de inferência.

Os Checadores de Provas, que são uma variação dos Provadores de Teoremas, dependem do usuário para serem conduzidos, e, embora removam grande parte das tarefas tediosas, não podem substituir a intuição humana para realização das provas. Assim, ao invés de serem geradores de provas (se fossem totalmente automatizados), eles simplesmente checam a prova que o usuário fornece. É importante notar que quando se utiliza essas ferramentas o usuário deve primeiro saber como provar a declaração no papel antes de prová-la com a ferramenta [Chr98].

Os Provadores de Teoremas necessitam de menos controle sobre a prova do que os Checadores de Prova. Se a prova não vai bem então o usuário não consegue definir o que o Provador de Teorema deve fazer. Mesmo os mais avançados Provadores de Teoremas necessitam de algum controle humano. Entretanto isto não desvaloriza os Checadores de Provas e os Provadores de Teoremas, pois eles garantem a certeza necessária para sistemas críticos. Qualquer proposição que pode ser mostrada num Checador de Prova é garantida. Além disto, Checadores de Prova trabalham bem em sistemas extremamente abstratos, mesmo no nível de abstração de teoremas matemáticos [JH97].

2.2.5.3 – Outras denominações

Existem outras denominações para ferramentas de validação formal, principalmente aquelas desenvolvidas para sistemas mais específicos. Por exemplo, os Checadores de Código se aplicam ao desenvolvimento de software, verificando se um pedaço de código atende a uma dada especificação [JH97]. Já os Checadores de Equivalência verificam se uma descrição de um sistema é funcionalmente equivalente a uma descrição padrão, permitindo tanto a simulação como a síntese de sistemas.. Do primeiro modo um sistema e descrito e comparado com uma descrição desejada. Uma vez validado, este sistema pode ter suas características adotadas como padrão e permitir o modo de síntese [Chr98].

2.2.6 – Integrando os tipos de métodos formais

As metodologias principais de verificação formal, a Checagem de Modelos e a Prova de Teoremas, ao invés de concorrerem entre si, podem se complementar, somando as vantagens e reduzindo as desvantagens. CLARKE e WING [CW96] apontam que é promissora esta combinação, sendo que o método de Prova de Teoremas pode ser utilizado para compor um sistema de estados finitos através de uma abstração de um sistema, enquanto que a Checagem de Modelos realiza a verificação.

As ferramentas de verificação formal denominadas Checadores de Modelos trabalham com a semântica do sistema, enquanto que os Provadores de Teoremas trabalham com a sua sintaxe, ou seja, trabalhando sobre o espaço de estados alcançáveis por exaustão a Checagem de Modelos está validando as propriedades de cada estado, enquanto que trabalhando sobre os conjuntos de axiomas e conjuntos de regras de inferência, o Provador de Teoremas está trabalhando sobre a sintaxe, e por dedução realizando a prova de uma declaração [JH97]. Decorrente da natureza da análise, o Provador de Teoremas pode trabalhar com um espaço de estados infinitos, ao contrário da Checagem de Modelos [CW96].

Enquanto os Provadores de Teoremas trabalham com lógica simbólica, os Checadores de Modelo trabalham com seqüências de transição concretas de estados. Assim, quando aquele método reporta um erro ele identifica qual passo lógico da prova é inválido, enquanto este modelo retorna um ou todos os caminhos que conduzem o sistema a um estado inválido.

2.3 – Conclusão

Para verificar um sistema de comunicações é necessário fazer abstrações, como será mostrado nos capítulos 5 e 7. Entretanto, quando o sistema já foi projetado, como é o caso do sistema utilizado neste trabalho (capítulos 6 e 7), é possível construir um modelo descrevendo e atribuindo qualidade aos estados, mapeando os conjuntos de transições e especificando seqüências de eventos e/ou estados. Assim, parece mais adequado o uso da metodologia de Checagem de Modelos para verificação deste tipo de sistema do que a utilização da metodologia de Prova de Teoremas. Se o objetivo fosse criar um sistema a partir de especificações de alto nível, verificando suas propriedades, provavelmente a utilização deste outro método seria mais adequado.

No próximo capítulo detalhar-se-ão alguns conceitos da metodologia escolhida, a Checagem de Modelos.

Capitulo 3 – A Checagem de Modelos em detalhe

A Checagem de Modelos é uma metodologia poderosa que permite a verificação de sistemas de forma automática, sendo a base das ferramentas de verificação mais difundidas, conforme será mostrado no capítulo 4. Ela apresenta características importantes que serão mostradas neste capítulo. Ver-se-á que as especificações são feitas em lógica temporal, e mostrar-se-á o que são e quais são as lógicas empregadas para Checagem de Modelos, bem como suas vantagens e desvantagens. Também será mostrado como é feita a verificação explícita de um sistema composto de um conjunto de estados. Ver-se-á que esta verificação apresenta problemas de limitação do tamanho do sistema, o que levou ao desenvolvimento de técnicas de redução do espaço de estados e técnicas de manipulação mais eficientes, como a verificação simbólica.

3.1 – O nascimento da Checagem de Modelos

Nos anos 80 foi desenvolvida uma técnica para verificação de modelos que exigia menos tempo e intervenção humana do que as técnicas tradicionais de simulação e das técnicas formais como os Provadores de Teoremas e Checadores de Provas. Era conhecida como Checagem de Modelos baseada em Lógica Temporal, que modelava os sistemas como um sistema de transição de estados e expressava as especificações do comportamento desejado para o sistema em lógica temporal proposicional [CGL96], [Mcm92].

Baseado em informações sobre o comportamento do sistema e com informações sobre quais são os comportamentos desejados, a técnica verifica se o sistema atende as especificações fornecidas. A idéia básica é a geração dos estados acessíveis do sistema fazendo uma varredura em cada estado por exaustão para identificar aqueles que violam alguma especificação ou levam o sistema a um congelamento, a um ciclo não progressivo, ou a outros comportamentos indesejáveis.

O Sistema a ser verificado pela Checagem de Modelos geralmente é modelado usando algum tipo de linguagem de especificação: Redes de Petri, Álgebra de processos, Autômatos, SDL, VHDL, etc. No capítulo 2 foi escolhido o modelo de Autômatos para utilização nas verificações realizadas neste trabalho.

3.2 – Expressando as especificações

Na técnica de Checagem de Modelos as especificações que determinam o comportamento desejado para um sistema são escritas em lógica temporal, que permite descrever a ordenação no tempo dos eventos de um sistema sem ter que declarar o tempo explicitamente. A lógica temporal nasceu de estudos de filósofos sobre como o tempo é declarado numa linguagem natural [Bit96].

A lógica temporal é um formalismo para descrever seqüências de transição entre os estados de sistemas que interagem com o meio, isto é, sistemas reativos [CGL96].

Uma fórmula pode especificar que eventualmente algum estado é alcançado ou que nunca será alcançado.

> "A principal característica de uma Lógica Temporal é o fato de uma determinada fórmula lógica poder apresentar valores distintos em instantes diferentes do tempo. Esta característica é formalizada através da introdução, na sintaxe da linguagem, de diversos Operadores Temporais." [Bit96].

Existe uma grande variedade de lógicas temporais. Elas diferem nos operadores que oferecem e na semântica destes operadores.

3.2.1 - Classificando as lógicas temporais

Existem diversos critérios de classificação da lógica temporal. Pode ser classificada em proposicional ou primeira ordem, global ou composicional, ramificada ou linear, pontual ou intervalar, passado ou futuro [Eme90].

A lógica proposicional é baseada em fórmulas construídas com proposições atômicas (que expressam características atômicas sobre os estados), conectivos booleanos (que interligam as proposições construindo-se as características de cada estado), e os operadores temporais. Esta lógica temporal corresponde ao mais abstrato nível de raciocínio lógico.

Quando se refina as proposições atômicas construindo expressões com variáveis, constantes, funções e quantificadores obtêm-se a lógica temporal de primeira ordem [Eme90].

O critério global versus composicional refere-se à característica de abrangência da lógica: ela pode ser utilizada apenas para análise de um sistema concorrente isoladamente, e então formar uma visão global, ou pode compor lógicas abrangendo mais sistemas permitindo a composição de vários sistemas e análises.

Lógicas temporais podem ser classificadas de acordo com a natureza do tempo. Para a lógica temporal linear para cada momento de tempo só há uma possibilidade de futuro. Na lógica ramificada em cada momento pode-se avançar para diferentes futuros possíveis.

O mais comum é que os operadores temporais da lógica sejam verdadeiros ou falsos em determinados pontos no tempo. Entretanto algumas lógicas trabalham sobre intervalos de tempo, com o objetivo de simplificar a formulação de propriedades de correção [Eme90].

A lógica temporal foi criada podendo descrever eventos tanto no passado como no futuro, entretanto o interesse da maioria das análises de sistemas se encontra no futuro, assim muitos tipos de lógica temporal não possuem operadores de passado [Eme90].

Das diversas classificações citadas existem diversos representantes, sendo que o critério de classificação mais utilizado é concernente à natureza do tempo: ramificada ou linear.

3.2.2 – Lógicas Temporais Lineares

Existem basicamente duas lógicas Temporais Lineares: A lógica temporal proposicional linear (PLTL – Propositional Linear Temporal Logic) e a lógica temporal linear de primeira-ordem (FOLTL – First-order linear temporal logic). A FOLTL é uma variação da PLTL introduzindo funções, quantificadores, variáveis, e constantes à PLTL [Eme90]. Vários autores utilizam apenas a sigla LTL representando a PLTL

3.2.3 – Lógicas Temporais ramificadas

A representação da estrutura lógica temporal ramificada parece com uma árvore infinita [Eme90], e pode ser definida através da estrutura temporal M=(S, R, L), onde:

- **S** é o conjunto de estados
- **R** $\subseteq$ **SxS** é a relação de transição. Para todos estado s pertencente a S existe um outro estado s' também pertencente a S tal que (s,s') pertença a R)
- **L: S** $\rightarrow$***P*(AP)** é a função que rotula cada estado com um conjunto de proposições atomicas verdadeiras para o estado

Sendo AP o conjunto das proposições atômicas, a sintaxe das fórmulas de estado é dada por [CGL96]:

- Se p$\in$ AP então p é uma fórmula de estado
- Se f e g são fórmulas de estado então $\neg$f, f$\wedge$g e f$\vee$g são fórmulas de estado.
- Se f é uma fórmula de percurso então E(f) e A(f) são fórmulas de estado.

A sintaxe das fórmulas de percurso é dada por:

- Se f é uma fórmula de estado então f também é uma fórmula de percurso
- Se f e g são fórmulas de percurso então $\neg$f, f$\wedge$g, f$\vee$g, Xf, fUg, e fRg são fórmulas de percurso.

Um caminho em M é uma seqüência infinita de estados, π=so, s1, ...si.... (para todo i $\geq$ 0 (si, si+1) pertence a R).

π^i denota o sufixo de π começando em s_i.

A notação M,s|=f significa que f é válida no estado s da estrutura M. A notação M, π |= f significa que f é válida ao longo do caminho π da estrutura M.

A representação M=(S, R, L) recebeu a denominação de estrutura de Kripke em homenagem a um dos primeiros matemáticos que deu a um modelo uma interpretação teórica baseado em lógica modal [Mcm92].

Uma das lógicas temporais ramificadas mais simples é o CTL (Computational Tree Logic), que originariamente descrevia propriedades de árvores computacionais. É composta de quantificadores de caminhos e operadores temporais.

São dois os quantificadores de caminho:

A – para todos os caminhos
E – para alguns caminhos

Estes quantificadores são utilizados num estado particular para especificar que todos os caminhos ou alguns caminhos que começam neste estado tem alguma propriedade.

Os operadores temporais descrevem propriedades do caminho através da árvore, sendo cinco os operadores[CGL96]:

- **F (No futuro – In the future):** A propriedade será verdadeira em algum ponto do caminho;
- **G (Globalmente – Globally):** Especifica que a propriedade é verdadeira em cada estado do caminho;
- **X (Próxima vez – Next time):** A propriedade é verdadeira no segundo estado do caminho;
- **U (Até - Until):** Existe um ponto no caminho onde a segunda propriedade é verdadeira, e em todos os estados precedentes a este ponto a primeira é verdadeira;
- **R (Liberação - releases):** A segunda propriedade deve ser verificada enquanto no caminho a primeira for verdadeira;

Alguns autores [Mcm92], [Eme90] não consideram o operador R, o que se justifica por ele ser um dual do operador U.

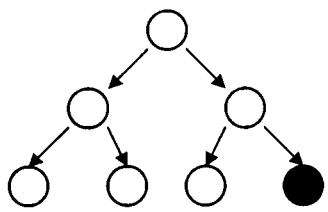
Na lógica CTL os operadores temporais ocorrem somente em pares consistindo de A ou E seguido por F, G, U, X ou R. Não há operadores de tempo passado. A obrigatoriedade de utilização de operadores sempre precedidos por quantificadores de caminho limitam o poder de expressividade desta lógica [Eme90].

Considerando que o R é um dual de U pode-se afirmar que para o CTL são 8 os operadores básicos: AX, EX, AG, EG, AF, EF, AU, EU. Estes 8 operadores podem ser expressos como combinações de EX, EG e EU [CLG96].

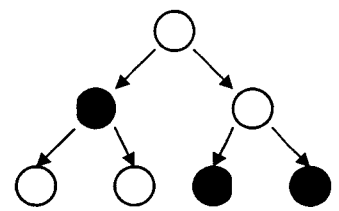
- $\mathbf{AX}f = \neg\mathbf{EX}(\neg f)$

- **AG**f = ¬**EF**(¬f)
- **AF**f = ¬**EG**(¬f)
- **EF**f = **E**[verdadeiro **U** f]
- **A**[f **U** g] = ¬**E**[¬g **U** ¬f ^¬g] ^¬**EG**¬g
- **A**[f **R** g] = ¬**E**[¬f **U**¬g]
- **E**[f **R** g] = ¬**A**[¬f **U**¬g]

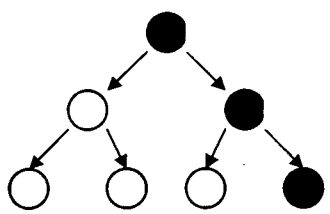
Os operadores mais comuns estão representados na figura 3.1



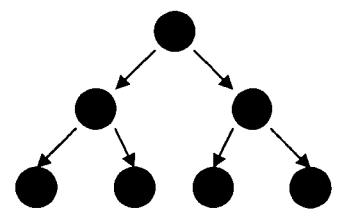
M, so |= EFg



M, so |= AFg



M, so |= EGg



M, so |= AGg

Figura 3.1 – Operadores básicos em CTL

3.2.4 – Aumentando a expressividade: O CTL*

Independente da lógica CTL* ser considerada uma extensão do CTL [Eme90] ou do CTL ser uma restrição do CTL* [CLG96], o importante a ser considerado é que o CTL* permite que operadores temporais sejam combinados através de conectivos booleanos sem a necessidade de serem precedidos por quantificadores de caminho.

As fórmulas da lógica CTL* compreendem fórmulas do estado, que são verdadeiras num estado específico, e fórmulas de caminho, que são verdadeiras ao longo de um caminho específico, incorporando as expressividades das lógicas CTL e LTL. Por exemplo:

- A(XGp) é uma fórmula em LTL, onde p é uma fórmula de caminho
- EX(AGp) é uma fórmula em CTL, onde p é uma fórmula de caminho
- A(XGp)^EX(AGp) é uma formula em CTL*, onde p é uma fórmula de caminho

3.2.5 – Considerações sobre a expressividade

Na maioria dos casos utilizam-se ou CTL ou LTL como lógicas para expressar comportamentos desejados para um sistema. Apesar dos Checadores de Modelos já existirem há quase duas décadas ainda existe muitas controvérsias sobre qual é a melhor lógica para utilização: a LTL ou a CTL? De acordo com VISSER [Vis98], a lógica LTL é considerada mais adequada para expressar as especificações desejadas, enquanto que a CTL exige menor complexidade do Checador de Modelos.

Mesmo sendo mais expressiva que a CTL, a lógica LTL é considerada de insuficiente expressividade [Wol83] e algumas melhoras foram obtidas com variações da linguagem, introduzindo operadores gramaticais e outros artifícios para aumentar a expressividade [Eme90]. Apesar destas melhorias o CTL* continua apresentando melhor expressividade do que o LTL.

Existem ferramentas de Checagem de Modelos utilizando a LTL e/ou CTL para especificação. Algumas ferramentas estão sendo desenvolvidas para utilizar a CTL*.

> "Para algumas lógicas, que possuem poder de expressibilidade adequados para captar importantes propriedades de correção, pode-se desenvolver algoritmos muito eficientes para Checagem de Modelos. Outras lógicas não podem ser utilizadas para Checagem de Modelos tão eficientemente."[Eme90].

O objetivo da Checagem de Modelos é responder a pergunta: Dada uma especificação em alguma lógica temporal e um sistema modelado, o sistema atende a especificação?

Para atingir ao objetivo, a Checagem de Modelos deve, tanto para o LTL como para o CTL, sobre uma estrutura de Kripke M =(S, R, L) definida no item 3.2.3, marcar os estados que atendem a especificação dada p, isto é M, s |= p, onde p é uma especificação expressa dentro da linguagem LTL ou CTL. Mas qual é a real diferença na implementação de Checagens de Modelos utilizando CTL ou LTL?

EMERSON [Eme90] mostra que para o LTL o trabalho de Checagem de Modelos é PSPACE-completo, enquanto que para o CTL o trabalho é de tempo polinomial determinístico, ou seja P.

Assim verifica-se que, apesar da desvantagem de expressividade do CTL sobre o LTL, o CTL apresenta-se computacionalmente mais eficiente do que o LTL, uma vez que este é PSPACE-completo que é computacionalmente mais complexo do que PSPACE-hard, que é mais complexo do que NP-complete, por sua vez mais complexo do que NP-hard, do que NP e finalmente do que P [Ort96], [HS92].

3.3 – A Checagem de Modelos de estados explícitos

3.3.1 – A Checagem de Modelos de estados explícitos utilizando CTL

Adotando a lógica CTL como base da linguagem de especificação do comportamento desejado para o sistema a ser verificado, pode-se entender o funcionamento da Checagem de Modelos através da explicitação dos estados.

Dada uma estrutura de Kripke M=(S, R, L), que representa um sistema de estados finitos concorrente, e uma lógica temporal f, que expressa uma especificação desejada, o objetivo é encontrar o conjunto de todos os estados de S que satisfaçam f, ou seja:

$$\{s \in S \mid M, s \models f\}$$

Nem todos os estados precisam atender as especificações para que o sistema se comporte como especificado. É suficiente que todos os estados iniciais estejam dentro do conjunto de estados que atendem as especificações.

As especificações na Lógica Temporal devem ser verificadas em todos os estados. Um algoritmo de verificação de modelos que utiliza a lógica temporal CTL deve seguir os seguintes passos (elaborados baseado em [CGL96]):

1 – Utilizar transformações lógicas para segmentar a especificação feita em CTL em 6 tipos de sub-fórmulas básicas;
2 - Atribuir a constante n o comprimento da fórmula em CTL da especificação;
3 – Definir uma variável x com o índice 1;
4 – Para cada sub-fórmula de comprimento x segmentada no item 1 verificar quais são os estados do modelo que a atendem;
5 – Rotular cada estado encontrado no item anterior com a sub-fórmula analisada;
6 – Se x é menor do que n então aumente x em um e volte ao passo 4;
7 – Verificar quais são os estados que tem como rótulo a especificação completa;
8 – Conhecendo os estados iniciais verifica-se se estes estão todos contidos nos estados levantados pelo item 7. Se sim então o sistema atende a especificação. Caso contrário, não atende.

Realizando transformações na fórmula original de uma especificação CTL é possível compo-la de apenas sub-fórmulas $\neg f_1$, $f_1 \vee f_2$, EXf_1, $E[f_1 \cup f_2]$ e EGf_1. Em uma das transformações, por exemplo, o conectivo booleano de adição $\wedge$ pode ser escrito como $\neg$ e $\vee$. Assim, para fórmulas $\neg f_1$ marcam-se os estados que não tem f_1. Para $f_1 \vee f_2$ marcam se aqueles que possuem propriedade f_1 ou f_2. Para EXf_1 marcam-se todos os estados que tem como sucessor f_1 [CGL96].

Para manipular $E[f_1 \cup f_2]$ acham-se todos os estados marcados com f_2, então, indo no sentido inverso das relações de transição dos estados acham-se todos os estados que podem ser alcançados por caminhos que passam por estados marcados com f_1 [CGL96].

Para determinar quais estados devem ser marcados com EGf_1 é necessário construir uma estrutura de Kripke somente com estados que atendem a especificação f_1 e adequar L e R de acordo. Deve-se então decompor o gráfico de estados em componentes não triviais fortemente conectados. Estes são sub-gráficos com o maior tamanho possível onde cada estado é alcançável partindo de qualquer outro estado através de um caminho totalmente contido no sub-gráfico, e este deve conter mais de um estado ou um estado com um laço em si mesmo.

Partindo dos estados contidos em componentes não triviais fortemente conectados procura-se, numa operação reversa ao sentido das transições, todos os estados que podem ser alcançados através de caminhos que contém f_1. Estes então são os estados que atendem a EGf_1 [CGL96].

3.3.2 – Exemplo de aplicação da Checagem de Modelos utilizando o CTL

Na figura 3.2 mostra-se um modelo de um sistema de um forno de microondas apresentado por CLARKE [CLG96]. Apresenta-se nesta figura a estrutura de Kripke com alguns acréscimos para facilitar o entendimento, como as negações das proposições que são falsas nos estados, e rótulos de eventos nas transições de estados.

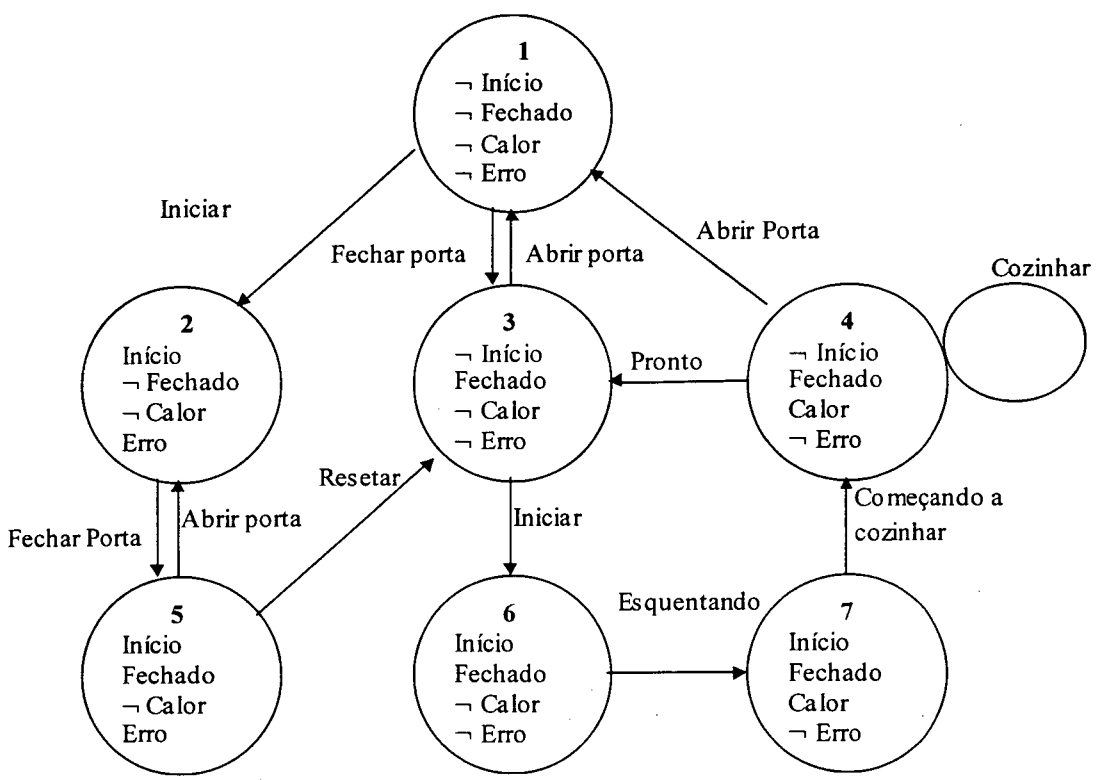


Figura 3.2 – Um sistema para aplicação de checagem de modelos de estados explícitos

Utilizando a técnica de model cheking explícito procura-se verificar se a especificação AG (início → AF calor) é válida para o sistema.

A especificação afirma que para todos os caminhos (A) em todos os estados (G) é verdadeira a afirmação de que se "início" for verdade num estado então em todos os caminhos no futuro (F) se chegará a um estado com "calor" verdadeiro.

Aplicando os 8 passos do algoritmo proposto anteriormente tem-se:

1 - Manipulando-se esta fórmula para os 8 operadores CTL básicos chega-se a equivalência AG (início $\rightarrow$ AF calor) $\equiv \neg$EF (início$\wedge$EG$\neg$calor). Assim, as sub-fórmulas são:

Comprimento 1 – "início", "$\neg$calor"

Comprimento 2 – "EG$\neg$calor"

Comprimento 3 – "início$\wedge$EG$\neg$calor"

Comprimento 4 – "$\neg$EF (início$\wedge$EG$\neg$calor)"

2 - n = 4

3 – x = 1

4 e 5 – Para "início" tem-se verdadeiros os estados {2, 5, 6, 7}

Para "$\neg$calor" tem-se verdadeiros os estado {1, 2, 3, 5, 6}

6 – x = 2

4 e 5 – Para encontrar os estados "EG$\neg$calor" encontra-se a estrutura de Kripke somente com estados onde $\neg$calor é verdadeiro, conforme a figura 3.3

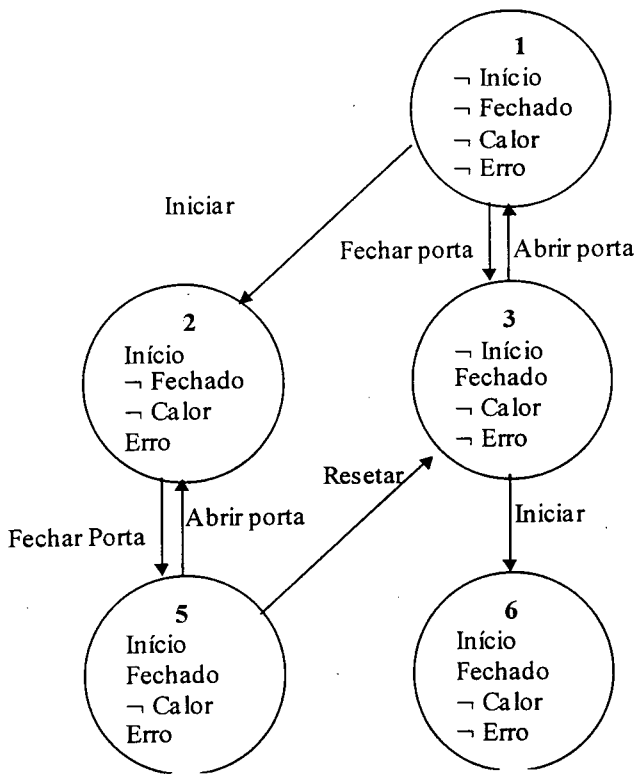


Figura 3.3 – Subsistema para aplicação de checagem de modelos de estados explícitos

Verifica-se que é necessário tirar o estado 6 para ter-se um componente não trivial fortemente conectado composto pelos estados {1, 2, 3, 5}.

Como além dos estados {1, 2, 3 e 5} não existe nenhum outro estado, na estrutura reduzida da figura 3.3, que permita alcançar o componente não trivial fortemente conectado, então define-se que para "EG¬calor" tem-se verdadeiros os estados { 1, 2, 3 e 5}

6 – x=3

4 e 5 - Para "início∧EG¬calor" tem-se verdadeiros os estados verdadeiros em "início" e os verdadeiros em "EG¬calor" = {2, 5}

6 – x=4

4 e 5 – Para encontrar "¬EF (início∧EG¬calor)" encontram-se todos os estados que eventualmente no futuro alcançam os estados 2 e 5, que são os estados onde a especificação "início∧EG¬calor" é verdadeira. Verifica-se pela figura 3.2 que a partir de qualquer estado do modelo é possível alcançar o estado 2 e o estado 5. Como se esta interessado em achar os

estados que não eventualmente no futuro alcançam 2 e 5 então tem-se "¬EF (início∧EG¬ calor)" = { }.

7 – Nenhum estado atende a especificação

8 – Como o estado inicial 1 não atende a especificação, então o modelo do sistema do forno de microondas não atende a especificação.

Numa rápida inspeção visual da figura 3.2 é possível concluir que partindo do estado inicial 1 nem sempre se chegará a um estado com "início" verdadeiro do qual para todos os caminhos no futuro se chegará a um estado com "calor". Nota-se que partindo do estado 1 é possível ficar somente dentro do laço 1, 2, 5, 3, 1, 2, ..., ou dentro do laço 1, 2, 5, 2, 5 ..., ou dentro do laço 1, 2, 5, 3, 1, 3, 1 .... indefinidamente. Em termos do funcionamento do forno estas seqüências eqüivaleriam a uma série de operações possíveis porém erradas. No primeiro laço eqüivaleria a mandar o forno funcionar com a porta aberta, o forno acusaria um problema, o operador notaria a porta aberta e a fecharia, resetaria o forno, que desligaria o comando de iniciar, e retiraria a condição de erro. Então o operador abriria a porta ao invés de mandar funcionar e daí por diante. O terceiro laço mostra que o operador pode ficar abrindo e fechando a porta depois de retirar a condição de erro. E no segundo laço o operador poderia comandar o início com a porta aberta e ficar abrindo e fechando a porta na esperança do forno retirar a condição de erro.

3.3.3 – Obrigações de justiça

Além das especificações fornecidas em lógica temporal sobre o comportamento de um sistema, muitas vezes interessa também incluir obrigações de justiça no modelo verificado.

As obrigações de justiça podem ser reunidas num conjunto denominado F, que será acrescido a estrutura de Kripke.

A estrutura de Kripke incluindo obrigações de justiça é definida como M = (S, L, R, F), onde $F = \{ P_1, P_2, ..., P_k\}$ é o conjunto de obrigações de justiça. $F \subseteq P(S)$

Um caminho é dito justo se, para um grupo de obrigações, cada uma é infinitamente e válida ao longo do caminho.

Uma obrigação de justiça pode ser um conjunto arbitrário de estados, geralmente descrito por uma fórmula da lógica utilizada para Checagem de Modelos.

Por exemplo, Π=so, s1, ... é justo se e somente se para cada P $\in$ F inf(Π) $\cap$ P $\neq$ 0; onde inf(Π) ={s/s=si, i $\rightarrow \infty$}.

S, L e R são os mesmos definidos no item 3.2.3. F é um conjunto de estados interelacionados, compondo o que pode-se dizer de caminhos justos [CLG96].

M, s $\models_f$ f indica que a fórmula f é verdadeira no estado s da estrutura justa de Kripke M.

M, $\Pi \models_f$ f a fórmula é justa ao longo da caminho Π em M.

3.3.4 - Obrigações de justiça na checagem explícita de modelos CTL

Sendo M = (S, L, R, F)

F = {P1, P2, ..., Pn} o conjunto de obrigações de justiça

O componente fortemente conectado C do grafo M é justo se e somente se para cada Pi $\in$ F existe um estado ti $\in$ (C $\cap$ Pi)

Como exemplo checa-se o forno

AG(start $\rightarrow$ AF heat)

Deve-se considerar somente caminhos que o usuário opera o microondas corretamente. Isto significa que infinitamente e freqüentemente start$\wedge$close$\wedge\neg$error deve valer. Assim, F = { start$\wedge$close$\wedge\neg$error}

S(start) = {2, 5, 6, 7}

S($\neg$heat) = {1, 2, 3, 5, 6}

Quando calcula-se o conjunto de componentes fortemente conectados vemos que {1, 2, 3, 5} não é justo pois não contem nenhum estado que satisfaça start$\wedge$close$\wedge\neg$error, estão:

S(EG$\neg$heat) = 0

S(EF(start$\wedge$EG$\neg$heat)) = 0

S($\neg$(EF(start$\wedge$EG$\neg$heat))) = { 1, 2, 3, 4, 5, 6, 7}

Portanto o programa satisfaz a fórmula considerando as obrigações de justiça.

3.3.5 – Automatizando o processo de Checagem de Modelos de estados explícitos

Teoricamente para qualquer lógica temporal proposicional a Checagem de Modelos atinge seu objetivo, uma vez que é possível uma busca exaustiva através dos caminhos de uma estrutura finita [Eme90]. Assim, é possível construir um software que permita automatizar o processo de Checagem de Modelos utilizando a representação explicita da estrutura de Kripke na forma de um gráfico de transição. Os primeiros softwares utilizavam esta técnica. Porém o problema da explosão de estados, que ocorre quando se modela um sistema com dois ou mais processos concorrentes com um grande número de estados, inviabilizou o uso de softwares baseados nesta técnica mesmo com o avanço espantoso da capacidade de processamento e memória dos computadores. Para permitir que um maior números de estados simultâneos fossem manipulados nasceu a técnica de representações simbólicas, onde funções booleanas representam os estados, permitindo uma manipulação mais rápida e compacta. Nesta técnica as funções boolenas são representadas por diagramas de decisão binário [Mcm92], [CLG96], [Eme90].

3.4 – Verificação de Modelos simbólica

Em 1987, Kenneth L. McMillan, introduziu a utilização de representações simbólicas para os gráficos de transição de estados, tornando possível a análise de sistemas maiores. A representação de um sistema é realizada através de diagramas de decisão binário ordenados, que oferecem uma forma canônica para as fórmulas booleanas, mais compacta que as formas normais de conjuntivos e disjuntivos, e permitem uma manipulação mais eficiente [CLG96], [Mcm92].

3.4.1 – Diagramas Orientados de decisão binários

Os Diagramas Orientados de decisão binários são formas de representação canônica de fórmulas booleanas. (Ordered Binary Decision Diagrams – OBDD). Eles são construídos através de algoritmos que transformam uma função booleana em uma árvore de decisão binária que é então trabalhada eliminando-se toda redundância. Desta forma os OBDDs ficam mais compactos e podem ser manipulados mais eficientemente [CGL96].

Cada estado é codificado por uma designação de valores booleanos para o conjunto de variáveis de estado associadas ao sistema. Assim, a relação de transição pode ser expressa como uma formula booleana em termos de dois conjuntos de variáveis, um conjunto codificando o velho estado e o outro codificando o novo. Esta formula é então representada por um diagrama de decisão binário [Mcm92].

3.4.2 - Representando estruturas de Kripke com OBDDs

Estruturas de Kripke podem ser representadas por diagramas de decisão binário utilizando o seguinte princípio [CGL96]:

Se R é uma relação n-ária sobre {0, 1} então R pode ser representada pelo OBDD

fr(x1, ..., xn) = 1 se e somente se R(x1, ..., xn)

Sendo R uma relação sobre o domínio finito D

D tem 2^m elementos para m>1

Codifica-se os elementos de D usando a bijeção $\phi:\{0, 1\}^m \rightarrow$ D que mapeia cada vetor booleano de comprimento m em um elemento de D

Usando a codificação ϕ constroi-se:

$\hat{R}(x\bar{1},\ldots,x\bar{n}) = R(\phi(x\bar{1}),\ldots,\phi(x\bar{n}))$ $\qquad \hat{R}(mxn)$

$x\bar{i}$ é um vetor de m variáveis booleanas que codifica a variável xi de D.

R pode agora ser representado como o OBDD determinado pela função característica $f\hat{r}$ de $\hat{R}$

Pode-se estender tal técnica para diferentes domínios, assim, considerando a estrutura de Kripke M = (S, R, L):

S $\rightarrow$ codifica-se os estados. Assumindo que existam exatamente 2^m estados $\rightarrow \phi:\{0, 1\}^m \rightarrow$ S (Vê-se que a função característica representando S é o OBDD para 1)

R $\rightarrow$ equivalente aos estados, porém são necessários dois conjuntos de variáveis booleanas, estados iniciais x estados finais (Geralmente escreve-se o segundo conjunto de variáveis $\hat{R}(\bar{x},\bar{x}') \rightarrow R$ é representado por $\hat{fr}$

L: states $\rightarrow$ subconjuntos de proposições atômicas $\leftrightarrow$ proposições atômicas $\rightarrow$ subconjuntos de estados

A proposição atômica é mapeada no conjunto de estados que a satisfazem: {s/a $\in$ L(s)} = La

Codificando ϕ como antes representa-se cada proposição atômica separadamente

3.4.3 – Limites além da verificação simbólica

As representações simbólicas aumentam sensivelmente o tamanho dos sistemas que podem ser verificados, porém muitos sistemas ainda são extremamente grandes para serem manuseados. A definição de sistemas extremamente grandes está diretamente relacionada com o aparato computacional disponível para a verificação, e da técnica de verificação e especificação utilizadas, como ver-se-á no capítulo 7. Entretanto, mesmo utilizando-se computadores de alta capacidade de processamento e memória, se a modelagem de um sistema não utilizar técnicas para reduzir o tamanho do espaço de estado, a explosão pode ser atingida facilmente, por exemplo, com a composição síncrona de dois espaços de estados (módulos) relativamente grandes.

Para não comprometer a eficiência dos algoritmos pode-se adotar linguagens de especificação menos expressivas como o CTL, que podem dificultar a identificação de erros. Paralelamente existem diversas técnicas que permitem trabalhar com sistemas com maior número de estados, como a da abstração, indução, simetria e estrutura modular, diminuindo o espaço de estados resultante. Entretanto tais técnicas podem esconder erros importantes para o projeto.

Das técnicas para redução do espaço de estados resultante a que fornece melhores resultados é a de estrutura modular. Ela se aplica principalmente em circuitos complexos e protocolos. As especificações para estes sistemas podem ser decompostas em partes fundamentais, que serão checadas em separado. Se for possível mostrar que o sistema é satisfeito em cada parte então isto implica que o sistema total satisfaz as especificações. Um

exemplo em que pode-se aplicar esta técnica é o sistema de comunicação, onde tem-se bem dissociados três processos de estados finitos: O Transmissor, o meio de transmissão e o Receptor. A especificação do sistema deve ser decomposta em três propriedades locais, uma para cada parte, e deve ser testada em separado. Com tal procedimento aumenta-se o tamanho dos sistemas que podem ser verificados [Hol91].

Na técnica de abstração utiliza-se uma mapeamento do modelo completo do sistema para um sistema mais abstrato, com um menor número de estados e transições. Já na técnica da simetria explora-se a existência de sub-sistemas replicados. Ter simetria em sistemas implica que existe um grupo de permutação que preserva a funcionalidade do gráfico de transição de estados e permite reduzir o número de estados. Na técnica de indução trabalha-se estabelecendo considerações sobre famílias de sistemas a estados finitos, isto é, estabelece-se que sistemas da mesma família apresentam características semelhantes, então pode-se, analisando um sistema, deduzir o comportamento de outros sistemas da mesma família.

3.5 – Conclusão

Utilizando técnicas de verificação simbólica, linguagens de entrada e outras características, as ferramentas de Checagem de Modelos apresentam vários conceitos semelhantes, e também diferenças significativas. No capítulo 4 serão citadas as principais ferramentas de verificação de conhecimento público, concentrando o foco nas similaridades e diferenças entre duas ferramentas de Checagem de Modelos, o SMV e o SPIN.

Capítulo 4 – Ferramentas de verificação formal

Nos capítulos 2 e 3 mostraram-se as principais características das metodologias de verificação formal. Entretanto pouco se falou dos instrumentos que permitem tais verificações, mais especificamente as ferramentas. Neste capítulo mostrar-se-ão os principais elementos das ferramentas de verificação formal, indicando quais são as principais ferramentas para cada metodologia mostrada nos capítulos 2 e 3. Ver-se-ão aspectos econômicos, construtivos, operacionais, alguns sucessos e detalhes, enfatizando as duas ferramentas de Checagem de Modelos mais conhecidas: o SMV e o SPIN. Mostrar-se-á também um exemplo de utilização da ferramenta SMV num sistema simples, mostrando características da ferramenta que será utilizada no capítulo 7 para verificar características do sistema de comunicação de dados escolhido, o ATM.

4.1 - Os elementos da Verificação Formal

Chrysalis [Chr98], apresenta cinco elementos como os principais a serem considerados nas ferramentas de Verificação Formal:

- **Linguagem** – usada para definir as propriedades do projeto. Para ser considerada formal uma linguagem deve ter um conjunto restrito de regras de transformação que governem a conversão de modelos escritos na linguagem para um conjunto específico de construtores matemáticos. Em relação ao sistema propriamente dito, suas propriedades devem ser especificadas numa linguagem. Não há padrões. Assim, os principais fatores na escolha de uma ferramenta pelo usuário, considerando a linguagem, são a facilidade e expressividade.

- **Base de dados** – O banco de dados do sistema armazena o sistema de equações que descreve o comportamento do sistema. As informações do banco de dados são obtidas através do processo de síntese da linguagem.

- **Compilação** – Traduz as características do sistema para a base de dados.

- **Bibliotecas** – Se a ferramenta de verificação formal é para ser utilizada com sistemas específicos, como desenvolvimento de circuitos ou softwares, então ela deve possuir bibliotecas de suporte para a tradução da linguagem de entrada para o banco de dados. As bibliotecas que a ferramenta utiliza devem ser verificadas para ficarem isentas de erros.

- **Algoritmos** – Os algoritmos são transparentes para o usuário. Se a ferramenta faz o trabalho corretamente num tempo razoável, utilizando uma razoável quantidade de memória, então o algoritmo que é utilizado não é importante. O ponto importante entretanto é definir o que é um trabalho correto. O usuário tem que avaliar se o algoritmo que esta sendo utilizado é extensivo suficientemente para fornecer a resposta correta. Provas que sempre retornam positivas são tão boas como nenhuma prova.

Os elementos apontados são focados nas percepções e necessidades do usuário. Outros autores fazem referencia a características desejáveis para as ferramentas de verificação formal, mas muitos apresentam principalmente características internas inerentes ao método/ferramenta, que são, ou deveriam ser, transparentes para o usuário. CLARKE e WING [CW96], por exemplo, citam características e conceitos desejáveis que devem ser incorporados nas ferramentas. As características são claramente vistas pelo ponto de vista do usuário, enquanto que os conceitos são a base que proporcionará melhor atingir as características demandadas pelos usuários, sendo para isto transparente.

Analisando os principais focos e preocupações dos diversos pesquisadores e desenvolvedores de soluções para verificação formal, verifica-se que as características e conceitos podem ser agrupadas em aspectos econômicos, operacionais e construtivos.

4.1.1 – Aspectos econômicos

É consenso dos criadores e pesquisadores de soluções que o fator econômico tem muita influência no sucesso dos métodos formais. Foi apresentado no capítulo 2 que um dos mitos quanto aos métodos formais é o de que o uso destes acrescentam atrasos no projeto, refletindo

em maiores custos. Por outro lado, erros não percebidos na fase do projeto podem exigir muitos recursos nas etapas posteriores para serem corrigidos. Assim, é necessário que o usuário tenha consciência dos custos economizados através do investimento em técnicas de verificação formal. Por mais que autores como CLARKE e WING [CW96] sugiram que as ferramentas devam apresentar um retorno do investimento rápido, muitas vezes esta ponderação é dificultada pela forma subjetiva como o usuário trata a questão [BH94].

4.1.2 - Aspectos Operacionais

Aspectos como facilidade de uso, eficiência, facilidade de aprendizado, integração com linguagens de modelos específicas e outras, são intimamente relacionados com a(s) metodologia(s) e técnica(s) aplicada(s) na ferramenta. Entretanto, os aspectos operacionais não deveriam ser dependentes dos aspectos construtivos, mas sim o inverso. É necessário entender bem a necessidade dos usuários e utilizar e desenvolver técnicas e combinações para atender estas necessidades.

Muitas das ferramentas de verificação formais ainda estão na fases embrionárias e de amadurecimento. Muitos dos produtos que estão sendo oferecidos são o resultado direto de pesquisas e desenvolvimentos em ambientes voltados para o produto, que ainda não tiveram tempo de se adequar às características do mercado.

4.1.3 - Aspectos Construtivos

Para atingir os aspectos operacionais e econômicos desejáveis numa ferramenta de verificação formal, é necessário pesquisar constantemente soluções mais eficazes e eficientes, tais como a composição de métodos buscando somar vantagens, o aprimoramento de técnicas de abstração, a otimização dos algoritmos, incorporação de linguagens padrão, e outros [CW96].

4.2 – Algumas das principais ferramentas de verificação formal

Algumas das ferramentas de verificação formal mais utilizadas de acordo com o tipo de tecnologia de verificação são [Her96]:

- **Provadores de Teoremas/Checadores de Provas** - Larch, Z/EVES, PVS
- **Checadores de Modelos** – SMV, SPIN

Focando na metodologia de Checagem de Modelos, que foi escolhida no capítulo 3 para a verificação dos sistemas apresentados nesta dissertação, mostram-se no item 4.3 as principais características das ferramentas mais utilizadas.

4.3 – Características das ferramentas de Checagem de Modelos

Serão demonstrados detalhes de duas ferramentas de Checagem de Modelos muito difundidas: o SMV e o SPIN.

Propõem-se os seguintes tópicos para análise e descrição:

- Informações Básicas:
  - Propósito;
  - Desenvolvedores;
  - Histórico;
  - Sucessos;
  - Necessidades de Hardware e Software.
- Informações Detalhadas:
  - Linguagens do Modelo;
  - Linguagem da Especificação;
  - Características da Análise.

4.3.1 – SMV

- **Propósito** - É uma ferramenta para verificação de sistemas a estados finitos contra especificações em lógica temporal CTL e/ou LTL. O SMV não foi criado pensando numa aplicação específica, como protocolos ou software, podendo ser utilizado em qualquer sistema a estados finitos.

- **Desenvolvedores** – Carnegie Mellon University – CMU e Cadence Berkeley Laboratories.

- **Histórico** - No início da década de 80, CLARKE, Edmund M e EMERSON, E. Allen, da Carnegie Mellon University, começaram o desenvolvimento de uma ferramenta de verificação formal conhecida como Simbolic Model Verifier (SMV) [CGL96]. Em meados de 1996, MCMILLAN, K. L, desenvolveu uma versão melhorada da ferramenta baseada no SMV original da CMU. A versão incorpora os recursos originais e acrescenta diversas facilidades tais como: especificações em LTL; uma melhoria na linguagem de descrição dos comportamentos, tornando a linguagem de descrição dos comportamentos do sistema mais poderosa; e outras [Mcm98b].

- **Sucessos** – Ver Item 4.4.

- **Necessidades de Hardware e Software** - Necessita de enormes quantidades de memória para as representações OBDD. Em alguns casos ultrapassa-se 200 Megabytes [JH97].

- **Linguagens do Modelo** - Utiliza uma linguagem própria baseada em autômatos de estados finitos para modelar o sistema. A linguagem de especificação pode suportar sistemas assíncronos e síncronos, estruturação hierárquica e definições do usuário sobre a inicialização e codificação [Mcm98b]. Tem a capacidade de utilização de módulos que podem ser instanciados múltiplas vezes permitindo a formação de cascatas. As transições entre estados de um modelo podem ser determinísticas ou não determinísticas, possibilitando a construção de modelos no nível de abstração

adequado. A linguagem de entrada do SMV apresenta semelhanças com a linguagem C, tanto na possibilidade de modularização, reuso, controles de fluxo, quanto na concisão do código. A versão do SMV da Cadence desenvolvida por MCMILLAN, apesar de apresentar uma série de melhorias na estruturação e facilidade de descrição dos comportamentos dos modelos em relação a ferramenta original, apresenta os mesmos pontos básicos principais:

Definição de módulos – Cada módulo possui um nome e uma possível lista de variáveis que serão trocadas entre o módulo em questão e aquele que o instanciou. O primeiro módulo a ser executado é o módulo "main", analogamente ao C.

Declaração de variáveis – Em cada módulo as variáveis devem ser declaradas definindo os possíveis estados. Alguns tipos de variáveis, como as booleanas, já possuem um intervalo definido como {0, 1}. No caso de números inteiros ou matrizes é necessário indicar os estados possíveis ou um intervalo possível. É importante considerar que intervalos muito amplos tendem a acrescentar mais estados no modelo, podendo contribuir para a explosão de estados. Tanto nas variáveis simples como nas matrizes os intervalos são indicados por X..Y, que é equivalente a {X, X+1, X+2, ..., Y-2, Y-1, Y}.

Inicialização das variáveis – As variáveis declaradas devem ser inicializadas para que a ferramenta simule o real comportamento do sistema modelado. Algumas variáveis não são inicializadas pois elas dependem diretamente de outras variáveis. Uma tentativa de inicialização e posterior redefinição é barrada pela ferramenta.

Evolução das variáveis – Cada variável evolui a cada instante para um valor igual ou diferente daquele em que se encontra em função da variação dos valores de uma ou mais variáveis associadas. As variáveis evoluem dentro dos intervalos definidos pela declaração, e qualquer descrição de evolução fora dos intervalos declarados leva a variável a um estado indeterminado que pode

mascarar um erro de uma verificação e/ou pode indicar um modelo mal construído.

- **Linguagem da Especificação** - O SMV utiliza lógica temporal proposicional para definir as especificações do comportamento desejado. O SMV original da CMU permite apenas a utilização de CTL, enquanto que o SMV da Cadence permite tanto o CTL como o LTL. Especificações podem ser escritas dentro de cada módulo ou dentro do módulo principal (main). Como o módulo principal instancia os demais, ele pode referenciar variáveis dentro dos módulos, permitindo a construção de especificações que fazem referencia a todas as variáveis de todos os módulos.

- **Características da Análise** - A ferramenta SMV representa internamente o conjunto de estados e de transições utilizando Diagramas de Decisão Binários, o que garante que a representação das formulas Booleanas fica muito compacta. Trabalhando sobre as fórmulas booleanas o algoritmo verifica todos os estados por um processo de exaustão, identificando um possível estado que torna pelo menos uma especificação falsa. Quando tal estado é identificado o algoritmo traça uma seqüência de estados que levaram ao estado incorreto. Assim, nem todas as especificações são verificadas e nem todos os estados de erro são mostrados. Felizmente a ferramenta permite que se verifique uma especificação específica. Entretanto, no caso dos estados inválidos, o algoritmo mostrará apenas o primeiro estado que levou ao erro.

4.3.2 – SPIN

- **Propósito** - O SPIN foi projetado para testar especificações de sistemas concorrentes, especificamente protocolos de comunicação. Por exemplo, pode testar o comportamento de protocolos procurando ciclos não progressivos, ou testar a especificação de dois processos para garantir exclusão mútua de recursos críticos, como um partilhamento de memória [JH97], [Bel98]. Apesar da ênfase dada a protocolos de comunicação, o SPIN pode ser utilizado para rastrear erros lógicos de projeto de quaisquer sistemas distribuídos, tais como sistemas operacionais,

sistemas de comutação, algoritmos concorrentes, protocolos de sinalização de ferrovias, etc [Bel98].

- **Desenvolvedores** - SPIN: AT&T Bell Labs.

- **Histórico** - O SPIN é um pacote de software desenvolvido pelo grupo de métodos formais e verificação da Bell Labs que se tornou o pacote mais difundido em uso atualmente. Existem milhares de usuários em 34 países do mundo [Bel98].

- **Sucessos** – Ver Item 4.4.

- **Necessidades de Hardware e Software** - Roda em todas as versões do UNIX e sob o Windows NT e 95. Algumas funções necessitam de muita memória, acima de 100 Megabytes de RAM [JH97].

- **Linguagens do Modelo** - A linguagem de entrada utilizada é o PROMELA (Protocol ou Process Meta Language), que é uma linguagem não determinística. Todos os sistemas devem ser modelados por esta linguagem, que é muito parecida com o C. Ela contém três tipos de objetos: processos, variáveis e canais [Bel98]. As variáveis em PROMELA podem ser locais ou globais e contém diferentes capacidades de tamanho: byte (8), int(32), short(16), bit(2). Além disto é possível a definição de matrizes, como por exemplo "Byte state [n]", que define uma matriz unidimensional com n elementos. Como no C os elementos das matrizes podem ser referenciados utilizando os índices: State[0] = state [3] + 5 * state[3*2/n] [Bel98].

  Os canais são filas partilhadas entre processos, que devem ter o tipo de mensagem a ser transmitido (é possível a utilização de matrizes) e o tamanho da fila. Quando um canal está cheio qualquer nova mensagem é perdida. Se um canal está vazio qualquer processo que busca uma mensagem ficará retido até a mensagem ser enviada por outro processo. Pode-se definir canais síncronos ou assíncronos [JH97].

  Para verificação dos sistemas, o SPIN utiliza uma estratégia de linguagem de entrada que disponibiliza a introdução de rótulos como o "end", para verificar congelamentos, e "progress" para verificar ciclos não progressivos. Estes rótulos

permitirão que o algoritmo distinga entre terminações e ciclos normais do processo dos congelamentos e ciclos não progressivos. Os processos são escritos em PROMELA e são nomeados podendo ser instanciados por outros processos, análogo ao C. O comportamento dos diversos processos é definido ao longo da linguagem PROMELA, mas eles são executados através do processo principal, análogo ao main do C, que é denominado "init". Além da execução de cada processo podem ser geradas cópias dos mesmos até o limite definido pelo hardware e software operacionais [Hol91]. Um processo termina quando atinge o fim do corpo de declarações, mas não antes de todos os processos instanciados por ele terem terminado primeiro. Os processos podem ser assíncronos ou serem sincronizados. Utilizando uma definição de canal com tamanho zero define-se um ponto onde as mensagens apenas podem passar e não são armazenadas. Interações de mensagens através destes pontos são sincronizadas, por definição. Essa comunicação através do canal é binária: Apenas dois processos, um transmissor e um receptor podem ser sincronizados desse modo [Hol91]. O controle de fluxo das declarações em PROMELA é feito através do uso de condições e laços, semelhante ao C.

- **Linguagem da Especificação** - O SPIN utiliza a Lógica Temporal Linear para definição do comportamento desejado. Em PROMELA o critério de correção é baseado em propriedades que são definidas como impossíveis. A notação em PROMELA para obrigações temporais é: Never { "corpo da obrigação temporal} [Bel98]. A estratégia de marcação de processos com rótulos "end" e "process" permite a detecção de congelamentos, terminações normais do sistema, e terminações devido a protocolos incompletos.

  Existem dois meios de verificar uma obrigação temporal no SPIN: Para uma seqüência de execução uma obrigação temporal é atendida somente quando ela pode terminar, isto é, a obrigação pode ser violada se o final do conjunto de declarações do corpo da obrigação em PROMELA é alcançável. A outra forma é: para uma seqüência de execução cíclica a obrigação é atendida somente quando um ciclo de aceitação explicito existe. Os rótulos de aceitação dentro de obrigações temporais são definidas pelo usuário. Na ausência de rótulos de aceitação nenhum comportamento circular pode atender uma obrigação temporal. Para checar uma

obrigação temporal cíclica os rótulos de aceitação devem ocorrer somente dentro da obrigação e não em outro lugar do código [JH97].

- **Características da Análise** - Um modelo de um sistema escrito em PROMELA será executado como uma seqüência de declarações, que incluem controle de seqüência, condicionais, associações, envio ou recepção de mensagens. Processos concorrentes são iniciados sem qualquer consideração sobre suas velocidades, e podem se comunicar com outros processos através de mensagens através de canais e/ou alterando valores de variáveis partilhadas e globais [JH97]. Uma declaração deve ser executável para que o processo que modela o sistema rode. Associações e afirmativas são sempre executáveis, entretanto condicionais são executáveis se e somente se elas são verdadeiras.

4.4 – Sucessos de utilização de ferramentas de verificação formal

Tem-se os seguintes exemplos de sucessos nas utilizações de ferramentas de verificação formal [Her96]:

4.4.1 - Circuitos

- **SMV**
  - Um barramento de conferencia da HP;
  - PCI bus bridges;
  - Uma unidade de interface X86;
  - Um cache on-line para o processador PowerPC.
- **SPIN** - Sistema de comutação AT&T 5ESS.
- **Z/EVES** - circuito CMOS simples.
- **PVS** - Microprocessador AAMP5.

4.4.2 - Protocolos

- **SPIN**
  - Um sistema de controle de processos para robôs industriais;
  - Um protocolo de comunicação entre tarefas para o Sistema operacional RUBIS;
  - Um protocolo para reduzir colisões em meios como a Ethernet.

- **SMV**
  - Protocolos PCI local bus;
  - Futurebus+;
  - Protocolos de Autenticação.

4.4.3 – Sistemas de Banco de dados

O Z/EVES pode ser utilizado para mostrar se um sistema de banco de dados tem um comportamento adequado.

4.4.4 - Algoritmos

- **PVS**
  - Algoritmos do IEEE;
  - Uma análise de um software para determinação de GPS em Naves Espaciais;
  - Correção de código de programas.

4.5 – Exemplo de utilização de uma ferramenta de verificação formal

Para demonstrar a utilização de uma ferramenta de verificação formal escolheu-se o SMV da Cadence. Esta ferramenta será utilizada para verificar algumas características do sistema de comunicação de dados escolhido, o ATM, conforme será mostrado no capítulo 7. Entretanto, antes de trabalhar num sistema mais complexo aplicar-se-á a ferramenta num sistema mais didático.

4.5.1 – O almoço dos filósofos

Considere dois filósofos num almoço contemplativo. Cada um pode estar em três estados: meditando, comendo ou bebendo. A máquina de estados que representa um dos filósofos é mostrada na figura 4.1. (Este exemplo foi extraído de [Zil92].)

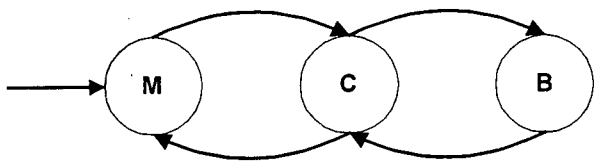


Figura 4.1 – Máquina de estado de um filósofo

Os dois filósofos almoçando juntos compõem um sistema de nove estados mostrado na figura 4.2.

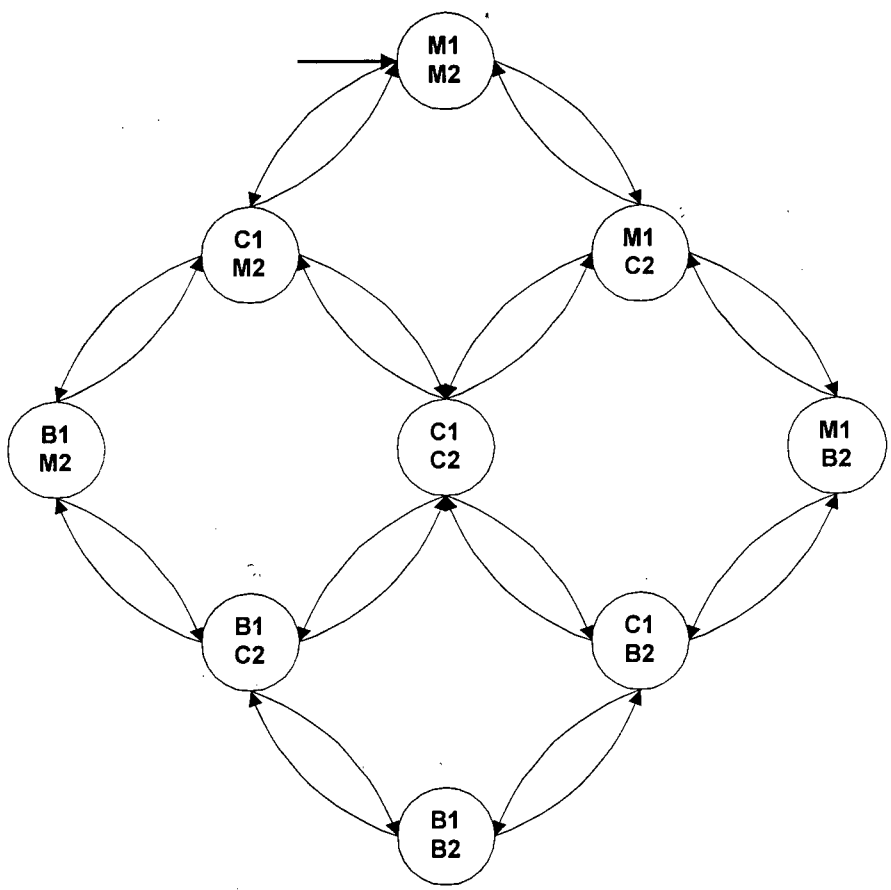


Figura 4.2 – Máquina de estado composta dos dois filósofos

Como o sistema composto é didaticamente simples todos os estados possíveis puderam ser representados, ficando óbvios alguns estados que serão discutidos. Entretanto na prática os

sistemas são enormes, sendo impossível traçar diagramas de transição de estados e realizar verificações manuais confiáveis.

4.5.2 – Os recursos compartilhados e a especificação para o sistema

Os dois filósofos dispõem cada um de uma almofada, para meditação, dois pauzinhos e uma porção de comida. Para beber existe apenas um copo de saque para compartilharem entre si. Na figura 4.2 pode-se ver que existe um estado mapeado onde os dois estão bebendo, entretanto este é um estado de erro, que se atingido pode prejudicar todo o funcionamento do almoço/sistema.

Apesar de evidente neste caso, estados errados em sistemas mais complexos podem passar desapercebidos, podendo causar sérios problemas. Neste ponto as ferramentas de verificação formal são de grande valia.

4.5.3 – Descrevendo o sistema e a especificação para a ferramenta.

Na linguagem do SMV da Cadence pode-se descrever o problema conforme mostra a figura 4.3. Foram incluídos comentários ao longo da linguagem, apresentados aqui em negrito, buscando uma melhor compreensão da linguagem que pode ser encontrada em [Mcm98].

```
module main()

/* Todo "programa" escrito em SMV tem o módulo principal, que pode conter as instanciações,
ou chamadas para os outros módulos, as especificações e variáveis globais */

{
f1 : filosofo( );
f2 : filosofo( );

/* Pode-se ver duas instanciações, uma que roda o filosofo 1 e a outra o filosofo 2*/

colisao: assert G (~((f1.estf=Beb)&(f2.estf=Beb)));

/* Na liguagem desta ferramenta é possível rotular cada especificação para melhor
identificá-las e para utilização das declarações prove [Mcm98]
A especificação afirma que nunca vai haver um estado em que ambos filosofos bebam*/

}
```

```
module filosofo()

/* Este é o módulo que realmente descreve comportamento dos filosofos*/

{
estf:{Med, Com, Beb};

/* o estado do fisolofo (estf) pode estar dentro destas três opções, conforme figura 4.1
Med = Meditando, Com = Comendo e Beb = Bebendo*/

init(estf):=Med;

/* O estado inicial é Meditando. O próximo estado pode variar em função do estado anterior
conforme figura 4.1*/

next(estf):=
  switch(estf){
          Med:{Med, Com};
          Com:{Med, Com, Beb};
          Beb:{Beb, Com};
  };
}
```

Figura 4.3 – O comportamento dos filósofos na linguagem da ferramenta SMV

4.5.4 – Considerações

Na figura 4.3. a especificação é dada por:

```
colisao: assert G (~((f1.estf=Beb)&(f2.estf=Beb)));
```

O resultado da verificação é falso, como era esperado. A especificação afirma que nunca vai haver um estado em que o filosofo 1 esteja bebendo conjuntamente com o filosofo 2.

A ferramenta, além de afirmar que a especificação é falsa, fornece a seqüência de estados de uma das possibilidades que invalidam a especificação. Como a ferramenta funciona por varredura, geralmente esta seqüência é a possibilidade mais direta.

No caso apresentado a ferramenta forneceu o resultado mostrado na tabela 4.1:

	Instantes		
estados	1	2	3
f1.estf	Med	Com	Beb
f2.estf	Med	Com	Beb

Tabela 4.1 – Seqüência de estados que invalidam a especificação.

Além da seqüência de estados que invalidam a especificação a ferramenta SMV gera um registro que indica, entre outras coisas, os recursos utilizados na verificação do sistema. Neste exemplo o resultado mostrado após a verificação pode ser visto na figura 4.4.

```
Resources used
==============
user time.....................................................0.445773 s
system time...................................................0.459491 s
BDD nodes allocated.....................................................107
data segment size.....................................................0
```

Figura 4.4 – Resultado dos recursos utilizados pela ferramenta SMV (LTL)

O resultado mais importante na figura 4.4 é a quantidade de nós BDD alocados (BDD nodes allocated). Tal parâmetro é função da complexidade do modelo, da especificação a ser verificada e se reflete diretamente na quantidade de memória utilizada.

Além dos nós BDD alocados a ferramenta fornece duas outras informações importantes, que não foram mostradas na figura 4.4.: a quantidade de iterações, que neste caso foram duas, e a quantidade de estados alcançáveis, igual a nove, como era esperado. Mostrar-se-á, no capítulo 7, que analogamente aos nós BDDs, o número de iterações é proporcional a complexidade do modelo, e a "profundidade" da varredura para verificação de uma especificação. Por hora pode-se afirmar que, como a ferramenta termina a verificação quando a especificação testada é falsa, o número de iterações de uma verificação que aponta todas as especificações verdadeiras é maior ou igual àquele de uma verificação de uma especificação falsa.

Explorando a versatilidade da ferramenta pode-se utilizar especificações em LTL bem como em CTL [Mcm98c]. A própria ferramenta contém um dispositivo para tradução (quando

possível) de LTL para CTL [Mcm98c], o que, conforme mostrado nos capítulos 2 e 3, pode reduzir a complexidade da verificação. No exemplo dos filósofos, entretanto, trocando a especificação LTL por uma especificação CTL conforme a figura 4.5, obteve-se uma quantidade de nós BDDs alocados de 105 contra os 107 anteriormente obtidos. A quantidade de estados permanece constante bem como o número de iterações necessárias.

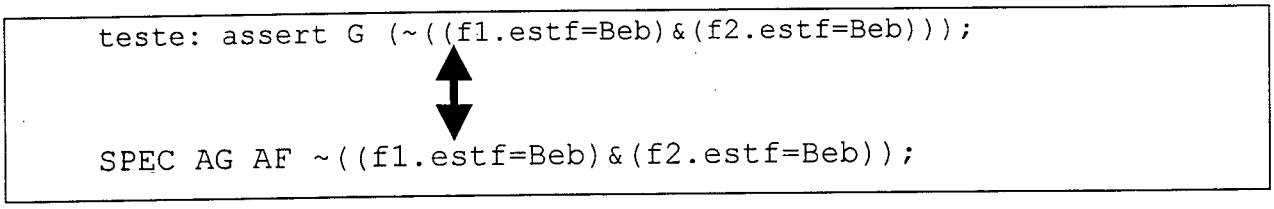


Figura 4.5 – A mesma especificação para os filósofos escrita em LTL e CTL

```
Resources used
==============
user time.....................................................0.445773 s
system time...................................................0.459491 s
BDD nodes allocated.....................................................105
data segment size.....................................................0
```

Figura 4.6 – Resultado dos recursos utilizados pela ferramenta SMV (CTL)

4.6 – Considerações sobre as ferramentas de Checagem de Modelos

Apesar de ser considerada poderosa, a modelagem através de autômatos de estados não é considerada conveniente de se trabalhar para a verificação através de algoritmos [Hol91]. Assim, as duas ferramentas, o SPIN e o SMV utilizam um linguagem de modelagem que é muito semelhante ao C definindo as características do sistema. O SPIN escreve o modelo diretamente em termos de três tipos específicos de objetos: processos, canais de mensagem e variáveis de estado [Hol91], enquanto que o SMV utiliza processos e variáveis [CGL96].

A utilização de LTL pelo SPIN leva o grau de complexidade do algoritmo a ser alto, conforme discutido no Capítulo 2, consumindo recursos elevados e obrigando a técnicas de redução da complexidade. Uma das estratégias adotadas pelo SPIN são os rótulos de caracterização dos processos, que permite uma análise rápida de condições como

congelamentos e ciclos incorretos. Já no SMV da Cadence busca-se traduzir especificações de LTL para CTL.

No capítulo 7 ver-se-ão maiores detalhes sobre os recursos utilizados, considerações entre a utilização de CTL ou LTL e demais detalhes sobre a ferramenta SMV da Cadence.

Capítulo 5 - Validando Protocolos de Comunicação

Mostrar-se-ão, neste capítulo, quais são as principais características que devem ser consideradas na modelagem e verificação de protocolos, e o papel destas nos sistemas de comunicação. Mostrar-se-á que, para realizar adequadamente o serviço para um sistema dividido em camadas, o protocolo deve considerar questões quanto ao meio de transmissão, controle de erros, vocabulários e formatação de mensagens, e o mais importante, deve possuir regras de procedimento que sejam completas e não contraditórias. Como suporte a todas estas questões estão as ferramentas de verificação formal.

5.1 – Definindo Protocolos

Um protocolo pode ser definido como:

> "Conjunto de regras e formatos (semânticos e sintáticos) que determinam o comportamento de comunicação das entidades, na execução de funções." [Fra95].

Protocolos, além de estabelecerem um conjunto de regras e formatos para comunicação entre as partes, servem de base para a execução de funções pelas entidades comunicantes. Existem diversas áreas de utilização para protocolos, desde os sinais de fumaça trocados entre índios, até os mais complexos protocolos de comunicação de dados, que permitem uma integração de diversas funções entre diversos sistemas, constituindo a base de sistemas distribuídos [Tan94].

A primeira utilização do termo protocolo para procedimentos de comunicações de dados foi feita por R.A. Scantlebury e K.A. Barteltt, no National Physical Laboratory na Inglaterra, em um memorando de abril de 1967 [Hol91].

O conjunto de regras estabelecidos pelo protocolo busca atingir determinados objetivos específicos. Entretanto, nem sempre os protocolos estão livres de erros. No livro "Design and Validation of Computer Protocols", Gerard J. Holzmann [Hol91] cita um exemplo simples de um protocolo de comunicação com problemas, onde as entidades comunicantes são dois controladores do fluxo de trens dentro de um túnel. O objetivo era evitar a colisão de trens, embora, como foi relatado, uma seqüência inesperada de eventos tenha conduzido os dois controladores ao erro, ocasionando um sério acidente.

Quando se fala em protocolos geralmente relaciona-os com os campos de sistemas operacionais, redes de computadores e comunicação de dados. Essa associação começou quase cinco décadas atrás. Com o nascimento dos computadores programáveis, na década de 40, nasceram os periféricos, tais como as impressoras, terminais, unidades de armazenamento, etc. O alto custo de um computador naquela época fez nascer as primeiras redes de interligação de periféricos. Estes, com pouca ou nenhuma "inteligência" incorporada, precisavam se comunicar com o computador central. Nasciam os primeiros protocolos de comunicação para sistemas informáticos. Em 1956 ocorreram os primeiros experimentos de comunicação entre computadores através de linhas telefônicas. No início os protocolos trabalhavam no formato mestre-escravo, ou seja, dentre os dois equipamentos trocando dados um era responsável por todo o controle do processo de transferência. Com o nascimento das redes de conexão entre diversos computadores, onde cada um era conectado diretamente com outros computadores através de pares, o modelo mestre-escravo teve que ser abandonado [Hol91].

5.2 - Características de um protocolo

O protocolo formaliza a utilização de um canal de comunicação. Assim ele precisa realizar as seguintes funções básicas [Tan94]:

- Inicialização e término de trocas de dados;
- Sincronização dos transmissores e receptores;
- Detecção e correção de erros de transmissão;
- Formatação e codificação de dados.

Para realizar tais funções de maneira eficiente e eficaz é preciso que o protocolo apresente as seguintes propriedades citadas em [Hol91], e compatíveis com os objetivos da definição do modelo OSI [Bri94] e [Tan94]:

Simplicidade – O protocolo deve ser composto de pequenos pedaços bem projetados e documentados que realizam uma função corretamente. Entendendo cada peça e como elas interagem é possível entender todo o protocolo. Protocolos feitos assim são robustos, eficientes, fáceis de manter e de verificar.

Modularidade – Os pedaços de um protocolo devem interagir entre si da maneira mais modular possível, isto é, devem são ortogonais. Suas funções não devam executar funções em outros módulos (ou camadas).

Boa formação – Protocolos bem formados não contem códigos em excesso nem códigos incompletos. Os protocolos bem feitos contem limites que os impedem de sobrecarregar os sistemas que dependem destes. O protocolo bem feito deve se auto-estabilizar na ocorrência de um erro, retornando a uma operação normal num número finito de transições. O protocolo bem feito deve ser auto-adaptável, por exemplo, à taxa de dados.

Robustez - É a capacidade de trabalhar sobre circunstâncias fora da utilização normal, mas que são permitidas pelo protocolo. Protocolos antigos, por exemplo, não conseguem trabalhar com taxas de dados muito altas. Um protocolo robusto consegue se adequar a uma nova tecnologia sem exigir mudanças fundamentais. A melhor forma de garantir robustez não é super dimensionar o projeto para tentar antecipar as novas condições, mas minimizar o projeto retirando considerações não essenciais que poderiam impedir uma adaptação a condições não antecipadas.

5.3 - O protocolo como linguagem

Os protocolos podem ser entendidos como linguagens. Eles são uma espécie de acordo entre as partes sobre como será feita a troca de informações. Assim, eles definem um formato para os quais a mensagem é válida, constituindo o que se determina ser sua sintaxe. Também definem regras de procedimento para a troca de dados, ou seja uma gramática, e definem um vocabulário de mensagens válidas que podem ser trocadas e seus significados, isto é, uma semântica [Fra95], [Hol91].

Não apenas deve-se ter um conjunto de regras para transferência de mensagem como também deve-se ter um acordo na utilização de tais regras. Muitos sistemas adotam variações de regras de acordo com sua própria necessidade. Infelizmente muitas vezes sistemas de diferentes companhias, utilizando o mesmo protocolo, não conseguem se comunicar.

Durante muitos anos diversas empresas multinacionais de equipamento de comunicações de voz e dados produziam sistemas proprietários. Apenas elas detinham o acesso aos detalhes do sistema de comunicação projetados. Assim, para instalar um sistema de gerencia, por exemplo, num sistema desenvolvido pela Siemens era necessário comprar o equipamento desenvolvido por ela. Esta estratégia garantiu durante algum tempo a formação de mercados cativos e de grandes companhias.

Nos anos 80 começou a revolução do mercado de telecomunicações. Iniciou nos EUA com a quebra do monopólio e a criação das BabyBells e se estendeu por todo o mundo através de uma grande onda de privatizações e quebras de monopólios. Novos "players" entraram no mercado de telecomunicações quebrando o paradigma de sistemas proprietários e defendendo a bandeira da intercomunicação e interoperacionalização de sistemas. De fato, ainda existe muito a ser feito para que existam sistemas realmente abertos e intercambiáveis.

Antes da revolução nas telecomunicações já se pensava na padronização de interfaces e sistemas, principalmente buscando ganho com a economia de escala na produção de peças e equipamentos. Os diversos institutos de padronização internacional, tais como o ISO e o ITU-T (Antigo CCITT), nasceram com este objetivo, mas foi realmente com a revolução das telecomunicações que se intensificou o foco na intercomunicação e interoperacionalização de equipamentos de diferentes fornecedores.

São necessários métodos para projeto e descrição de protocolos e métodos para checar se o protocolo funciona corretamente. A padronização de protocolos por si só não resolve os

problemas do projeto. Existem três linguagens de especificação de protocolos para documentos de padronização: SDL, LOTOS e Estelle. São conhecidos como FDT's, ou seja, "Formal Description Techniques" (Técnicas de Descrição Formal). Entretanto, nenhuma destas linguagem ataca o problema da verificação dos protocolos. Nada garante que um protocolo escrito em qualquer destas linguagem seja livre de erros. Além disto o LOTOS e o SDL podem descrever o comportamento de sistemas infinitos, que não podem ser checados através de ferramentas de verificação [Hol91].

5.4 - Protocolo nos sistemas de comunicacão

Dois tipos de erros são difíceis de se evitar em qualquer projeto de protocolo: um conjunto incompleto de regras e o projeto de regras que são contraditórias. A tarefa de análise e especificação de um protocolo complexo, onde existem diversos detalhes a serem considerados, exige que diversas técnicas sejam utilizadas para auxiliar o projetista/analista. Para projetar corretamente um protocolo é necessário especificar corretamente todos os pedaços importantes, e separá-los em tópicos ortogonais usando o princípio de modularidade.

Tanenbaum [Tan94] cita em seu livro Redes de Computadores que, para reduzir a complexidade do projeto, a maioria das redes é organizada em camadas, cada uma construída sobre sua predecessora. O propósito de cada uma, de acordo com o autor, é oferecer certos serviços às camadas superiores. Além disto, ela não se preocupa com os detalhes de como a camada inferior implementa o serviço. Entre duas entidades comunicantes existem duas estruturas de camadas. Cada camada se comunica com a sua equivalente da outra entidade através de regras e convenções estabelecidas pelo protocolo da camada em questão. De fato não há uma troca de informações diretamente entre o mesmo tipo de camada, e na verdade ela sempre utiliza o serviço da camada imediatamente inferior para enviar as informações.

Uma proposta de divisão de camadas foi feita em 1980 pelo International Standards Organization (ISO), conhecida como Modelo OSI (Reference Model of Open Systems Interconnection – Modelo de Referência de Interconexão de Sistemas Abertos). São 7 camadas com diferentes serviços conforme mostra a figura 5.1.

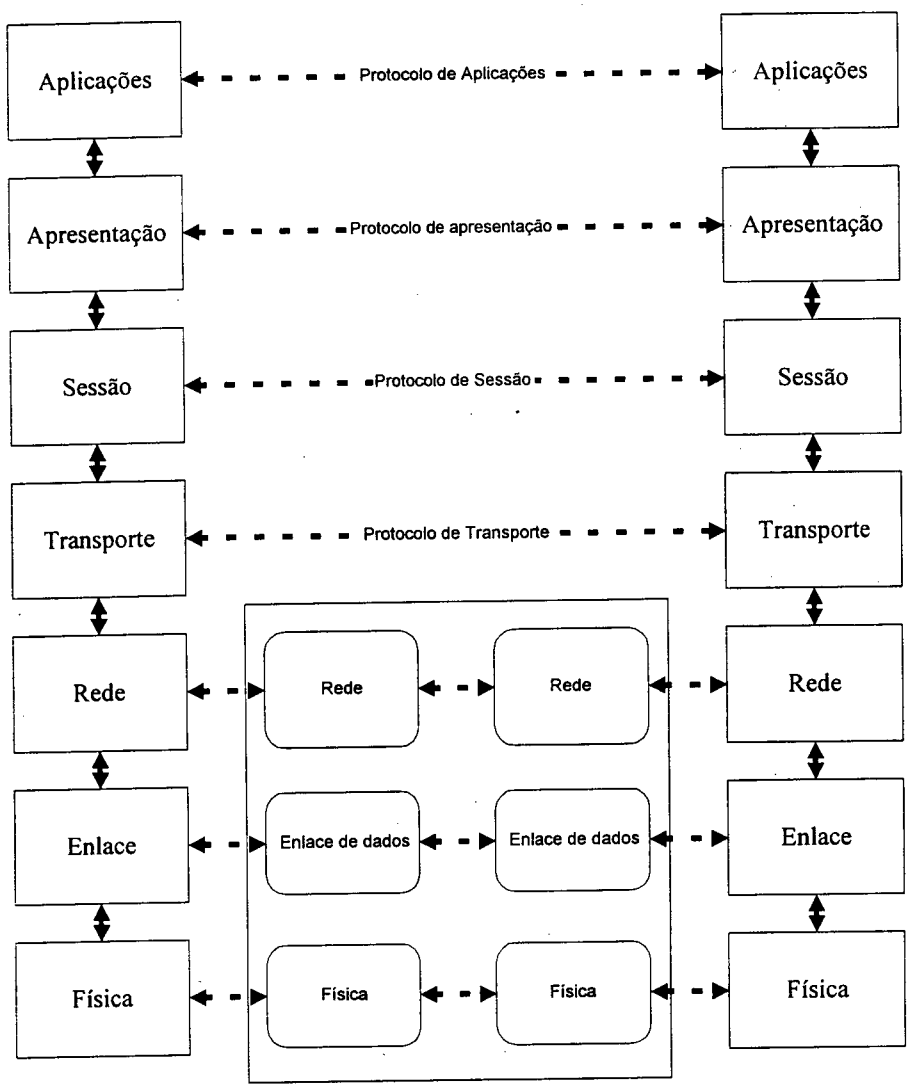


Figura 5.1 - Modelo OSI (Reference Model of Open Systems Interconnection)

Tanenbaum [Tan94] enumera os princípios utilizados para delimitação das camadas. Tais princípios são consistentes com as idéias de Holzmann [Hol91], que permitem a simplificação de modelos que descrevem o funcionamento de protocolos. Tem-se os principais:

1. Uma camada deve ser criada onde é necessário um nível de abstração diferente;
2. Cada camada deve desempenhar uma função bem definida;
3. As fronteiras entre as camadas devem ser escolhidas de forma a minimizar o fluxo de informações através das interfaces entre as mesmas.

Nem sempre os sistemas de comunicação são modelados utilizando-se todas as sete camadas do modelo OSI. O ATM, por exemplo, apresenta três camadas (física, camada ATM e adaptação), correspondendo a apenas duas camadas (física e enlace) do modelo OSI. Isto se deve ao objetivo da rede ATM de encaminhar células, conforme será visto no capítulo 6.

Cada camada na hierarquia define um serviço distinto e implementa um protocolo diferente. Ela define um nível de abstração no protocolo, agrupando funções intimamente relacionadas e as separando das funções ortogonais. Desta forma mudanças futuras feitas numa camada não afetarão o projeto das demais. Num projeto de um sistema de comunicações as camadas são separadas por interfaces que devem ser concisas e bem definidas.

Todas as camadas tem as mesmas características a serem consideradas: o serviço e o meio de transmissão. Para qualquer camada o serviço é fornecido para a camada superior, e o meio de transmissão é a camada inferior. Na camada mais baixa as preocupações com o meio de transmissão entram no detalhe de cabos ópticos, pares metálicos, etc [Tan94].

Com a divisão em camadas o problema de controle de erros, recuperação de erros, endereçamento, roteamento, controle de fluxo, codificação, e outros, podem ser resolvidos passo a passo de uma forma organizada. Com esta metodologia também é possível analisar sistemas complexos de forma modular, sem atingir os limites impostos pela explosão de estados das ferramentas de verificação formal que utilizam métodos por exaustão [Hol91].

5.5 - Delimitando os protocolos

Secionando um sistema de comunicações em camadas delimita-se claramente a atuação de um protocolo. Verifica-se que este é responsável pela comunicação com a camada equivalente no outro extremo da comunicação, e que ele rege as ações da camada para prover um serviço para a camada superior e utiliza os serviços fornecidos pela camada inferior.

A especificação de um protocolo consiste de cinco partes distintas [Hol91]:

1 - O serviço a ser provido pelo protocolo;

2 - As considerações sobre o meio no qual o protocolo é executado;

3 - O vocabulário de mensagens usadas para implementar o protocolo;

4 - A codificação, ou formatação de cada mensagem no vocabulário;

5 - As regras de procedimento que garantem a consistência das trocas de mensagens;

Destes cinco itens, de acordo com HOLZMANN [Hol91], o mais difícil de se verificar e de se projetar são as regras de procedimento.

5.5.1 - O serviço a ser provido pelo protocolo

TANENBAUM [Tan94], cita que serviços e protocolos são conceitos distintos, apesar de serem constantemente confundidos. De acordo com o autor:

> "O serviço é um conjunto de primitivas que uma camada oferece à camada acima dela. O serviço define que operações a camada está preparada para realizar em nome de seus usuários, mas não diz nada sobre como essas operações são implementadas.
>
> O protocolo é um conjunto de regras que governa o formato e significado de quadros, pacotes, ou mensagens que são trocados entre entidades parceiras dentro de uma mesma camada. As entidades usam protocolos para implementar suas definições de serviços." [Tan94]

O autor ressalta também que o protocolo das camadas pode ser alterados desde que o serviço seja mantido o mesmo. Assim, serviços e protocolos são desvinculados.

5.5.2 - As considerações sobre o meio no qual o protocolo é executado

O meio de transmissão tem enorme influência nas características de um protocolo. Existem enormes diferenças entre transmitir um bit através de um barramento de computador, com probabilidade de erro menor do que 10^{-15} e num par metálico com probabilidade de erro na ordem de 10^{-4}. A própria evolução das redes de comunicação de dados são um exemplo disto. Na década de 70, quando as redes de pacote X.25 foram desenvolvidas, o meio de transmissão apresentava alta taxa de erros. Assim, os protocolos do X.25 foram projetados

com forte ênfase na correção de erros, elevando a quantidade de controles e processos nos nós da rede e introduzindo sérios atrasos e "overheads" [SLC95]. Por outro lado, as novas redes de comunicação de dados, como Frame Relay e ATM foram projetadas considerando que o meio é muito mais confiável. Assim, diminuiu-se a quantidade de controles simplificando os protocolos.

Considerando que o meio de transmissão é composto da linha propriamente dita e de uma rede de comunicação, os dados transmitidos podem sofrer diversos tipos de erros, causados basicamente por distorções lineares, como as oriundas das limitações de banda de um canal, e distorções não lineares, causadas por ecos, "cross-talk" (diafonia), ruído branco e ruído impulsivo.

Felizmente existem diversas técnicas que permitem minimizar os efeitos de erros, como utilização de blindagem em cabos, filtros de linha, etc. Entretanto, alguns erros, oriundos do meio externo e alguns inerentes às camadas superiores à física, devem ser tratados pelos protocolos de comunicação.

Não existe um método de controle de erro que consiga capturar todos os erros que possam ocorrer [Hol91], [Tan94]. Entretanto, o método de controle de erro aumenta enormemente a confiabilidade da transmissão, chegando aos níveis alcançados pelos computadores.

Os esquemas de controle de erros devem ser construídos conforme as características do meio de transmissão que ocasionam os erros. Se um canal apenas produz erros de inserção então um protocolo que garante os erros de perda de dados é desnecessário. Em sistemas nos quais a probabilidade de erro é baixa, e os erros são isolados, por exemplo, um simples sistema de paridade pode ser um excelente método para controle de erro. Qualquer inclusão de sistemas para controle de erros, inadequados ao erro do meio de transmissão, irá diminuir a performance do protocolo e do sistema de transmissão.

Os principais erros de transmissão em sistemas de comunicação de dados são [Hol91], [Tan94]:

- Inserção
- Eliminação
- Duplicação
- Distorção
- Reordenação

Existem dois grandes conjuntos de métodos para resolver estes problemas. Para resolver os problemas de inserção e distorção utilizam-se os métodos para verificar a consistência dos dados, tais como a redundância, que verificam a consistência da mensagem recebida versus a mensagem transmitida através de códigos de correção e/ou detecção de erros. Utilizam-se basicamente dois tipos de códigos: os de blocos e os convolucionais. Já para os erros de duplicação, eliminação e reordenação utilizam-se esquemas de controle de fluxo, que tem o objetivo de evitar que os meios de transmissão fiquem congestionados. Ao mesmo tempo que evitam congestionamento os esquemas de controle de fluxo devem otimizar a utilização do canal. Nas redes de dados determinísticas a tarefa de controle de fluxo é muito mais simples do que nas redes estatísticas. Para controle de fluxo utilizam-se métodos desde o XON e XOFF, passando polo "Stop and Wait" até os protocolos de janelas deslizantes [Tan94], [Bri95].

Considerando numa rede apenas as conexões ponto-a-ponto pode-se incorrer num erro que acontece numa rede completa. Quando existem um ou mais pontos intermediários entre a entidade transmissora e a receptora então pode-se fazer dois tipos de controle de fluxo: entre nós e entre o transmissor e o receptor. No primeiro caso o tamanho da janela de um protocolo deslizante num controle entre nós é calculado separadamente para cada enlace. Quando dois links consecutivos possuem capacidades de transmissão diferentes, o controle de fluxo da conexão fim-a-fim fica comprometido. O transmissor tentará saturar o canal. No enlace mais rápido a quantidade de dados que chega ao nó receptor do enlace é maior do que a que sai. Assim, o nó nega o reconhecimento de mensagens, após o obvio congestionamento do buffer. Entretanto o transmissor continuará tentando saturar o canal fazendo retransmissões e novas transmissões.

Assim, o controle de fluxo deve considerar [Hol91]:

- Tamanho dos buffers nos nós da rede
- A banda disponível para conexão em todos os links envolvidos numa rede

O uso ótimo de uma conexão com diversos enlaces só ocorre quando o transmissor utiliza uma taxa igual ao enlace mais lento. Já nos protocolos fim-a-fim o problema não existe, pois a janela é ajustada para se adaptar ao mais lento. Entretanto nem sempre é possível identificar qual é o mais lento numa rede complexa. Além disto, o problema deixa de

ser estático para ser dinâmico. Muitos computadores podem estar acessando um mesmo enlace e isto pode torna-lo mais lento temporariamente [Hol91], [Tan94], [SLC95].

5.5.3 – O vocabulário de mensagens

Apesar de fundamental, definir o vocabulário de mensagens usado para implementar o protocolo é uma tarefa relativamente fácil comparada com a definição das regras de procedimento. Entretanto o vocabulário de mensagens é a base para a definição das regras de procedimento e para garantir certas características do serviço provido pelo protocolo. Os três itens (vocabulário, regras e características do serviço) estão casados, pois um vocabulário pobre limita características dos serviços e exige um esforço maior no projeto das regras de procedimento buscando contornar a limitação do vocabulário.

Cada sistema de comunicações possui vocabulários adequados próprios, que são visivelmente diferentes quando se analisa cada camada. Entretanto, uma base de mensagens é comum no vocabulário, principalmente aquelas que estabelecem e desconectam um canal entre duas camadas pares.

5.5.4 - A formatação de cada mensagem no vocabulário

Os sistemas de comunicação tem três métodos principais de formatação [Tan94]:

1 - Orientados a bit;
2 - Orientados a caracter;
3 - Orientados a byte.

Os orientados a bit transmitem dados como uma sucessão de bits. Para separar os quadros de dados utilizam-se "flags", que sinalizam o início e fim de cada quadro. Técnicas são utilizadas para diferenciar as "flags" dos fluxos normais de dados. A técnica mais utilizada é a de "bit stuffing", onde bits são inseridos para diferenciar os dados de uma flag.

Um representante desta categoria de método de formatação é o HDLC da ISO [Tar86]. [Equ92].

No método orientado a caracter os bits são agrupados em pacotes fixos tipicamente de 7 ou 8 bits. Nestes pacotes são transmitidos tanto os dados como os códigos de controle. Para diferenciar os quadros dentro de um protocolo orientado a caracter deve se utilizar técnicas para separar os blocos de dados. Deve-se também utilizar técnicas para evitar confusões entre os dados normais e as delimitações. A técnica mais conhecida é a de "caracter stuffing", análoga ao "bit stuffing".

Na formatação orientada a byte apenas a delimitação de início do pacote é enviada, pois o pacote tem um tamanho fixo, ao contrário dos outros dois casos citados.

Os três métodos citados podem gerar combinações e criar outros métodos. Entretanto tais métodos só funcionam na ausência de erro de transmissão. Se, por exemplo, uma flag ou caracter de controle for corrompido, então o estrago na comunicação pode existir. Para evitar isto é necessário que aos dados do usuário sejam acrescidas as informações para controle de fluxo, de erro, etc. Assim, tem-se:

Formato =

Cabeçalho	Dados	Trailer
Tipo, destino, número de seqüência, contador, etc	Dados	"Checksum", endereço de retornos

5.5.5 - As Regras de procedimento

As regras de procedimento são interpretadas concorrentemente por um número de processos que interagem entre si. Assim, dependendo da complexidade dos processos e da sua interação, pode ser muito difícil conseguir visualizar se as regras garantirão um bom funcionamento dos processos [Hol91].

Para raciocinar sobre um protocolo é necessário utilizar alguma técnica que prove sua correção. A ferramenta mais tradicional é o diagrama de seqüências, que é muito útil para detecção de erros isolados, mas não garante correção [CGL96].

Para garantir uma validação das regras de procedimento deve-se expressar o comportamento sem ambigüidades do protocolo numa notação formal [Chr98]. Tabelas de

transição ou autômatos de estados finitos podem ser utilizados para este propósito, entretanto conforme apontado no capítulo 4, as ferramentas geralmente utilizam uma linguagem própria semelhante ao C.

Não há uma metodologia geral que possa garantir o projeto de um conjunto de procedimentos sem ambigüidades. O que existe são ferramentas que automaticamente verificam a consistência lógica das regras e a observância dos requerimentos de correção [Hol91].

Muito do projeto de um novo protocolo utiliza técnicas já desenvolvidas e comprovadas. Para a camada física, dependendo do meio de transmissão adotado, existem diferentes técnicas de codificação. Não é necessário reinventar a roda. Porém existe ainda muito espaço não mapeado de técnicas para as diversas camadas do modelo OSI, principalmente para os novos protocolos das novas redes de comunicação de dados, como o ATM.

5.6 - Requerimentos para Correção de um Protocolo

O modelo do protocolo deve ser tão sucinto quanto possível, permitindo estudar a sua estrutura e consistência lógica. Para fazer isto abstrai-se de outros tópicos do projeto do protocolo, como o formato da mensagem [Hol91], [CLG96]

O modelo de validação define as interações de processos num sistema distribuído. Ele não resolve problemas da implementação de como a mensagem é transmitida, codificada e armazenada. Assim foca-se no projeto de um conjunto de regras completo e consistente para determinar as interações num sistema distribuído. Busca-se um modelo que represente todos os tipos de problemas de coordenação que podem ocorrer em sistemas distribuídos.

Para validar um projeto de um sistema de comunicações, HOLZMANN [Hol91], cita que é necessário que o modelo esteja livre de congelamentos, ciclos incorretos e terminações impróprias. ALPERN [AS85], define duas propriedades que contém todas as necessidades citadas por Holzmann, que são a propriedade de "safety" e de "liveness". A primeira é uma garantia que nenhuma 'coisa ruim' aconteça durante a execução. O autor cita como exemplos desta propriedade a exclusão mutua e a ausência de congelamentos. Já a segunda propriedade

estabelece que uma 'coisa boa' aconteça durante a execução. Como exemplos desta propriedade tem-se ausência de ciclos incorretos e terminações impróprias.

Definir que uma 'coisa boa' deva ocorrer e que uma 'coisa ruim' não pode, não é o principal problema na validação de sistemas. O problema reside em como traduzir tal especificação para uma linguagem que será verificada junto ao modelo do sistema em análise e como construir este modelo.

Os objetivos de um teste de conformidade e uma validação de protocolo são facilmente confundidos. O primeiro é usado para verificar se o comportamento externo de uma dada implementação de um protocolo é equivalente a sua especificação formal. O segundo é usado para verificar se uma especificação formal é ela própria logicamente consistente. Se um especificação formal tem um erro de projeto então uma implementação conforme a especificação passará num teste de conformidade, pois deve conter o mesmo erro, apresentando-se idêntico. Uma validação de consistência do protocolo deve sempre mostrar o erro de projeto [Hol91].

5.7 - Validação de Protocolos

A maioria dos sistemas de validação automáticos são baseados em análises exaustivas de acessibilidade [Mcm92], [CGL96]. Para garantir a validação de um modelo é suficiente verificar sua correção com um simples teste booleano para cada estado que é acessível a partir de um dado estado inicial do sistema.

Existe usualmente bem menos transições entre estados do que estados alcançáveis, assim muitos métodos se baseiam em analisar as transições ao invés dos estados. Muitas vezes o número de estados alcançáveis pode ir para o infinito enquanto as transições se mantém dentro de um pequeno valor [Hol91].

Apesar dos autores concordarem que existem limites computacionais para trabalhar com grandes conjuntos de estados geralmente eles não explicitam qual é este tamanho. Uma razão para isto pode ser a obsolescência causada pelo avanço tecnológico, onde o número de processamentos por segundo dos hardwares utilizados cresce continuamente. Outra razão pode ser o fato de que tais valores dependem da eficiência dos algoritmos, sendo diferentes

em cada ferramenta. Por exemplo, HOLZMANN, [Hol91], em 1991 cita que existem três tipos principais de análises:

- Análise completa (sistemas até 10^5 estados)
- Análise parcial controlada (sistemas até 10^8 estados)
- Simulações randômicas (sistemas maiores)

Com o avanço tecnológico e o avanço das técnicas de representação simbólica os limites apresentados por Holzmann [Hol91] estão muito abaixo do que se consegue atualmente. Já em 1990 estudos como o apresentado no artigo "Symbolic Model Checking: 10^{20} States and Beyond" [BCM90] mostraram soluções para verificação de sistemas acima de 10^{20} estados. Entretanto, limitações de hardware e de algoritmos podem diminuir este alcance.

Dado um estado inicial para cada autômato no sistema pode-se dividir o conjunto de estados em estados alcançáveis e não alcançáveis. Normalmente o conjunto de estado não alcançáveis é algumas vezes maior do que o conjunto dos estados alcançáveis. O importante é que o conjunto de todos os estados não alcançáveis inclua todos os estados de erro.

O principal problema com a estratégia de análise total é a aplicabilidade. Se a quantidade de memória de uma ferramenta é estourada pela explosão de estados então não há uma análise completa. *"Para grandes protocolos o algoritmo de análise por exaustão se deteriora numa análise parcial de baixa qualidade."* [Hol91].

Considere um protocolo com dois processos de 100 estados cada, uma fila de mensagens e cinco variáveis para cada processo. Para as variáveis locais é assumida variação entre dez valores distintos. O número de mensagens trocadas é igual a dez. Neste sistema existem $10^{5.2}$ possíveis estados para as variáveis do protocolo. Se cada processo pode estar em um dos 10^2 diferentes estados então os dois processos podem estar no máximo em 10^4 estados. Se cada fila pode reter de 0 a 5 mensagens então o número total máximo de estados do sistema é:

$$10^{10} \cdot 10^{4} \cdot \left[\sum_{i=0}^{5} 10^{i}\right]^{2} = 10^{24}.$$

Considerando que cada estado pode ser codificado em 1 byte de memória e pode ser analisado em 10^{-6} segundos então precisar-se-ia de uma máquina capaz de memorizar 10^{24} bytes e precisaria em torno de 10^{11} anos para terminar a análise.

Felizmente o número de estados alcançáveis é muito menor do que o número total. Mesmo assim os modelos dos protocolos a serem verificados devem otimizar a funcionalidade, reduzindo o número de estados. Análises sobre os modelos mais eficientes podem permitir a síntese de protocolos mais eficientes, e em muitos casos o método de análise completa só é viável se reduzir a complexidade dos modelos de validação.

5.8 – Conclusão

Neste capítulo foram mostradas algumas estratégias para modelar o sistema em análise permitindo uma verificação adequada. Infelizmente tais estratégias não são uma garantia de menor esforço, pois modelar corretamente um sistema exige um domínio sobre as características deste.

No capítulo 6 serão mostradas algumas características do sistema de comunicação de dados ATM, que serão a base para os modelos descritos e verificados no capítulo 7, onde se utilizará os conceitos e estratégias recém apresentados.

Capítulo 6 - ATM – Modo de transferência assíncrono

O ATM é um sistema de comunicação de dados que nasceu para suportar todos os tipos de aplicações em uma única rede. Para isto ele apresenta diversas características que o distinguem das redes mono-serviços. Neste capítulo mostrar-se-ão quais são estas características e as vantagens e desvantagens sobre outras redes especializadas. Além disto, mostrar-se-á que para conseguir trafegar diversos tipos de tráfegos de dados de diversas aplicações, mantendo os níveis de qualidade de serviço, é necessário uma gerência de tráfego adequada, que possua diversos instrumentos de controle e gerência, constituindo um conjunto de procedimentos interessante para verificação de propriedades no capitulo 7.

6.1 - Introdução

O ATM (Asynchronous Transfer Mode - Modo de Transferência Assíncrono) foi desenvolvido na década de 60 quando pesquisadores começaram a investigar tecnologias para uma nova plataforma de redes de alta velocidade. Nesta época ainda não se tinha a idéia de quanto a evolução tecnológica iria criar um mercado ávido por soluções de comunicação. Conforme cita SOARES [SLC95], foi com o desenvolvimento da tecnologia digital que os serviços de dados, voz e vídeo se integraram constituindo os sistemas multimídia.

No início da integração voz, dados e imagem, não haviam sistemas de comunicação adequados a suportar simultaneamente os três serviços. Existiam redes paralelas especializadas, como por exemplo redes de pacotes para dados, sistemas analógicos para comunicação de vídeo, e os sistemas de comunicação de voz.

A mudança do perfil das aplicações, substituindo gradualmente operações textuais por interações multimídia, e a proliferação das redes locais, levou, na década de 80, a que a idéia embrionária do ATM tomasse corpo. Começaram a surgir tecnologias adequadas para suportar os primeiros produtos. Na década de 90 entidades como o ATM Forum e o ITU-T iniciaram a elaboração de recomendações, definindo aspectos do ATM. Os trabalhos continuam até hoje.

6.2 - Forças que impulsionaram o nascimento do ATM

As principais forças que impulsionaram o nascimento do ATM foram [Gdc98]:

- **Mudança no perfil das aplicações** – Aplicações de consumo e entretenimento, onde se destacam o "Home Shopping" (Compras em casa), e as aplicações multimídias interativas; aplicações públicas, onde se destacam o ensino à distância e a vídeo conferência; e aplicações comerciais, com as interconexões LAN/CAN/MAN/WAN, acesso a base de dados distribuída, aplicações CAD/CAM, e outros;

- **Evolução dos meios de transmissão** – Substituição das malhas tradicionais; proliferação das fibras ópticas; aumento da capacidade com o aumento das taxas de transmissão; maior alcance e a melhoria da qualidade com um menor nível de erros;

- **Aumento da capacidade de processamento dos componentes das redes de computadores** – Os equipamentos terminais de dados podem executar tarefas antes executadas por componentes do sistema de comunicação;

- **Necessidade de integração de serviços** – A integração de dados, voz e imagem constituindo os sistemas multimídia e interligação de sistemas computacionais entre as diversas partes de uma empresa, permitindo comunicação multimídia;

- **Inadequação das redes existentes** – Os sistemas de comunicação antes do advento das redes faixa larga não eram adequados às exigências dos novos serviços e aplicações, pois eram especializados

6.3 - Classificando o ATM

A figura 6.1. mostra, dentro do universo das redes de comunicações de dados, voz e imagem, onde o ATM se encontra, ou seja, é uma rede de comutação de dados, do tipo

armazenamento e envio, baseado em pacotes de tamanho fixo, denominados células, que são encaminhados através de conexão de circuitos virtuais.

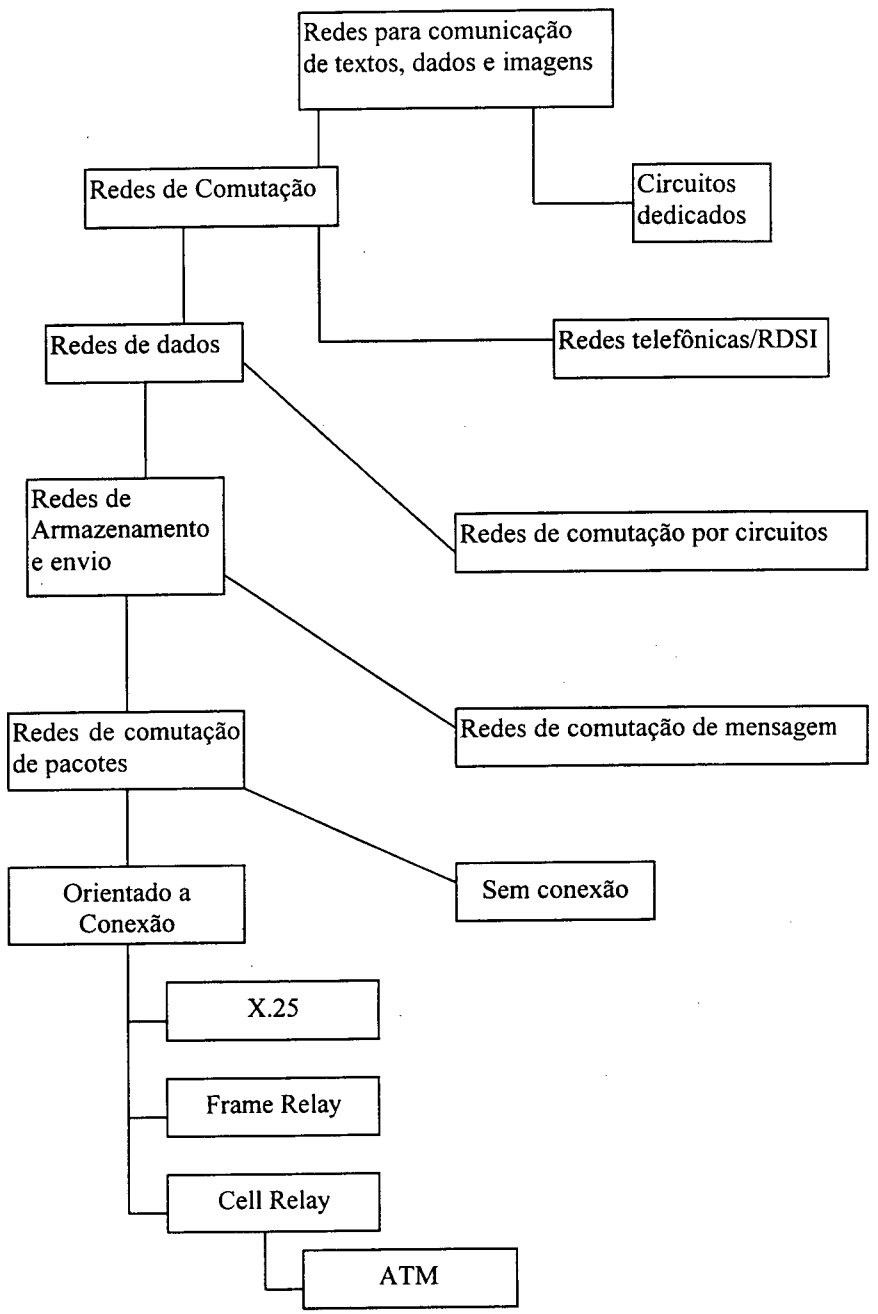


Figura 6.1 – Classificação das redes de comunicação de textos e dados

São dois os pontos mais relevantes que podem ser extraídos da figura 6.1: O modo de transferência da informação (pacotes ou circuitos) e o tipo de comutação de pacotes.

6.3.1 – Modo de transferência da informação

Segundo o ITU-T existem dois modos de transferência de informações [Tan96]: O modo circuito, onde as informações podem sofrer uma divisão, ou espacial, ou por freqüências, ou temporal; e o modo de pacotes, onde pode ser feita a comutação por blocos ou o encaminhamento de blocos.

As principais diferenças entre os dois modos de transferência são mostradas na tabela 6.1.

Característica	Modo Circuito	Modo Pacotes
Conexão	Circuitos Reais	Sem conexão, ou circuitos virtuais
Endereçamento	Implícito	Explícito
Atrasos / Alocação de banda	Determinístico	Estatístico
Protocolos	Transparente	Não transparente
Tarifação	Não admite tarifação por volume	Admite tarifação por volume

Tabela 6.1 - Comparação dos modos de transferência

As redes de armazenamento e envio, cujas redes de comutação de pacotes são integrantes, possuem nós ao longo dos pontos terminais, permitindo uma alocação de banda estatística e uma quantidade maior de usuários transferindo informações do que o caso de circuitos determinísticos. Entretanto, o preço desta vantagem é a indeterminação e aumento dos atrasos na transferência de dados.

A função de estabelecimento de conexões e endereçamento dos dados no modo circuito é inexistente pois é implícita. Já no modo pacotes pode-se, ou não, estabelecer conexão entre dois pontos através de circuitos virtuais. No estabelecimento deste, os nós escolhidos entre dois pontos de transmissão serão utilizados sempre por todos os pacotes ou mensagens transmitidos durante o tempo que o circuito estiver estabelecido, e, durante a conexão, serão alocados recursos em cada nó, tais como "buffers" e banda, para suportar o transporte de dados.

6.3.2 - Tipos de transferência por pacotes orientados à Conexão

A figura 6.1 mostrou que dentro do modo pacotes tem-se um grupo orientado à conexão, cujos principais componentes são o X.25, o Frame Relay e o Cell Relay, ao qual pertence o ATM. Mostrar-se-á que, apesar de utilizarem pacotes transmitidos por circuitos virtuais, a questão dos erros do meio de transmissão e das necessidades das aplicações, que utilizam o serviço de transmissão de dados, mostram diferenças fundamentais entre os três componentes citados.

6.3.2.1 - O X.25

O X.25 foi concebido para funcionar numa rede sujeita a muitos erros. Assim, cada pacote recebido por cada nó da rede é desempacotado e analisado. Estando livre de erros o pacote é enviado para o próximo nó ou destino. Há a perda de tempo no processamento e no "overhead" necessário em cada pacote. Os pacotes não tem tamanho fixo, porém no máximo o X.25 trabalha com 4096 octetos [Tar86], [Equ92].

6.3.2.2 - O Frame Relay

O Frame Relay é um encaminhador de quadros. Nasceu com o foco voltado para dados, apesar de existirem diversos fornecedores de soluções para voz para esta plataforma. O Frame Relay explora o aumento da confiabilidade das redes de comunicação e da inteligência das pontas. Os pacotes são encaminhados sem a desempacotação, análise e reempacotação que ocorrem no X.25. Com isso o overhead e o processamento ficam bastante reduzidos. Além disto, o Frame Relay trabalha com pacotes muito maiores do que o cabeçalho. O pacote no Frame Relay tem tamanho variável podendo chegar a 8 Kbytes. A taxa útil é também variável e pode ter valor elevado. Em circuitos de velocidade inferior a 512K o ganho em taxa útil aumenta a performance [Ger98]. Ver-se-á que tal característica torna o Frame Relay a escolha natural para ser o acesso de baixa velocidade do usuário com um nó ATM.

Embora o tamanho do pacote muito maior do que o cabeçalho seja vantajoso para o trafego de dados, esta característica, associada à não priorização do tráfego, limita a

performance das aplicações que necessitam de dados em tempo real, tais como a transmissão de voz e imagem [SLC95], pois o pacote destas aplicações deve esperar o procesamento de eventuais pacotes longos de dados que chegaram antes ao nó.

6.3.2.3 - Cell Relay/ATM

Como o Frame Relay apresenta um atraso nos pacotes de aplicações de tempo real decorrente dos grandes pacotes de dados, a idéia foi "disciplinar" os pacotes de forma a manter os tamanhos adequados a suportar serviços de tempo real. Assim nasceu o Cell Relay, que é um sistema de comunicação por encaminhamento de pacotes de tamanho fixo (células), cujo padrão mais conhecido é o ATM [Gdc98].

O processamento no ATM é mínimo e permite velocidades mais elevadas. Ao contrário do Frame Relay existe a priorização do tráfego, permitindo atender tanto a serviços de taxa de bits constante como aqueles de taxa de bits variável. Entretanto o "overhead" corresponde a 9,43% da célula e torna problemática a questão dos acessos com baixa velocidade.

6.3.2.4 - O híbrido ATM/Frame Relay

De acordo com o ATM Fórum tudo aponta para a utilização da tecnologia de quadros (Frames) para acesso a uma rede ATM utilizada como backbone. Isto fica claro a partir das diversas propostas de padrões para permitir o Frame Relay trafegar na rede ATM, sem a necessidade de desmontagem e remontagem de quadros. Para combinar o ATM com o Frame Relay o ATM Fórum apresenta o protocolo Frame UNI ou FUNI, que tem o objetivo de viabilizar o acesso a redes ATM via E1 ou Nx64k através de quadros [AF98].

6.4 - Características do ATM

As redes baseadas no Modo de Transferência Assíncrono (Asynchronous Transfer Mode – ATM) foram escolhidas e padronizadas para uso em redes públicas integradas como o RDSI-FL (Redes Digitais de Serviços Integrados de Faixa Larga). Os aspectos relacionados ao ATM são todos padronizados pelo ATM Forum, que é um consórcio de empresas de informática e telecomunicações, cujo principal objetivo é assegurar a interoperabilidade entre os equipamentos privativos e os equipamentos de redes públicas de comunicação como a RDSI-FL. Para isto o ATM Forum trabalha em cooperação com o ITU-T, que elabora as recomendações para RDSI-FL [SLC95].

6.4.1 – Categorias de Serviços

No ATM as aplicações são classificadas em cinco categorias de serviços [AF96]:

- **CBR (Constant bit rate – Taxa de bits constante)** - Destinado a aplicações que exigem banda fixa e sensíveis a retardo. Exige a garantia de banda. Alguns exemplos de aplicações CBR são: vídeo Interativo (por exemplo videoconferência), áudio interativo (por exemplo telefonia), distribuição de vídeo (televisão, vídeo-aula, etc), e outros;

- **VBR-rt (Variable Bit Rate real time – Taxa de bits variável em tempo real)** - Destinado a aplicações com elevado "burtiness" (rajadas – picos de tráfego) e sensíveis a retardo. Há um comprometimento com a entrega da informação em tempo real e há uma garantia de banda. Os principais usuários desta categoria são as aplicações que utilizam de tempo real (incluindo as listadas para CBR), no qual os sistemas terminais podem se beneficiar de uma multiplexação estatística mandando um fluxo variável e que pode tolerar ou recuperar uma perda pequena de células;

- **VBR-nrt (Variable Bit Rate non-real time – Taxa de bits variável em tempo não real)** - Destinado a aplicações com elevado "burtiness" (com muitas rajadas) e não sensíveis a retardo. Não há um comprometimento com a entrega da informação em tempo real, apesar de haver garantia de banda. Um exemplo de aplicação típico desta categoria é a interconexão de Frame Relay;

- **ABR (Available Bit Rate – Taxa de bits disponível)** - Destinado a aplicações que não exigem grandes garantias de tráfego e que possam tirar proveito do protocolo de controle de fluxo do ABR para atingir baixa taxa de perda de células, tais como transferencia de dados críticos (informações de segurança nacional) e aplicações de comunicação de dados exigindo menores atrasos;

- **UBR (Unspecified Bit Rate – Taxa de bits não especificada)** - Destinada a aplicações que não requerem garantias de tráfego. Por exemplo: transferencia de textos, dados e/ou imagem (transações bancárias, verificação de cartão de crédito, fotos/desenhos), mensagens de texto, dados e/ou imagem. (e-mail, telex, fax), interconexão de LANs ou emulação de LANs

6.4.2 - Classes de Serviço

Classes de serviços foram criadas para que fossem suportadas as diferentes características das aplicações. A camada de adaptação utiliza os serviços de transporte de células da camada ATM para oferecer serviços com requisitos específicos [SLC95]

Existem quatro classes de serviço definidas pelo ATM FORUM, divididas em função da sua natureza (VBR, ou CBR) e a necessidade ou não de manter a relação temporal da informação no destino. Na tabela 6.2 mostram-se características das quatro classes de serviço, tais como taxa de bits, sensibilidade a tempo, orientação a conexão, e exemplos de aplicações.

	Classe de Serviço			
	Classe A	Classe B	Classe C	Classe D
Sensível a Tempo	Sim		Não	
Taxa de Bits	Constante	Variável		
Orientada a conexão?	Sim			Não
Exemplos	Emulação de circuitos	Voz	Frame Relay, Ethernet	IP, SMDS

Tabela 6.2 – Classes de Serviço do ATM

6.4.3 - Aplicação x Categorias de Serviços

A Tabela 6.3, desenvolvida pelo ATM Fórum [AF96], mostra as aplicações mais comuns em ATM em relação as categorias de serviços.

Aplicação	CBR	VBR-rt	VBR-nrt	ABR	UBR
Dados	**	*	***	*	Na
Interconexão de LANs	*	*	**	***	**
Frame Relay sobre ATM	*	*	**	***	**
Emulação de Circuitos	***	**	ND	ND	ND
Multimídia sobre ATM	***	***	**	**	*

Legenda: * - desaconselhado, ** - Satisfatório, *** - Ideal, ND – Não disponível

Tabela 6.3 – Aplicações em ATM versus categorias de serviços.

Analisando a tabela 6.3 pode-se perceber que a classificação em ideal, satisfatório e desaconselhado não apenas considera a qualidade do serviço, mas também a otimização da ocupação dos recursos oferecidos pela rede.

6.4.4 - As interfaces

Uma rede ATM possui diversos nós que encaminham as células. Para prover a interligação entre estes nós de encaminhamento e entre os equipamentos que utilizam a rede e os nós utilizam-se as interfaces conforme mostrado na figura 6.2.

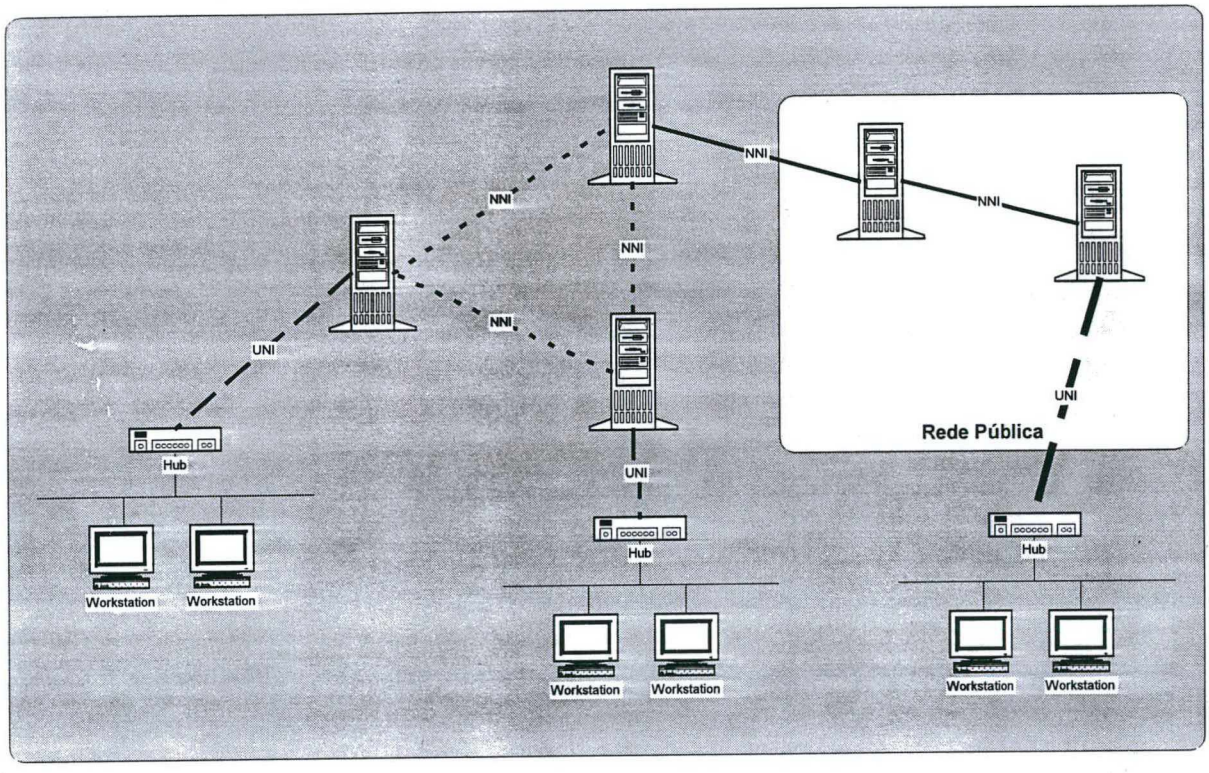


Figura 6.2 – Interfaces em redes ATM

As interfaces NNI são interfaces entre dois nós da rede ATM (NNI – Network-Network Interface – Interface Rede-Rede). As interfaces UNI são interfaces entre os equipamentos do usuário e um nó da rede ATM (UNI – User Network Interface – Interface Usuário Rede).

6.4.5 – Transferindo células no ATM

Para transmitir informações, os dados da aplicação são encapsulados em unidades de tamanho fixo, as células, com 53 bytes, sendo 5 bytes de cabeçalho e 48 de área útil para dados e informações de controle. Com o tamanho fixo garante-se rapidez e eficiência. O tamanho foi escolhido como uma média entre a proposta Européia e dos Estados Unidos da

América. Aqueles pretendiam favorecer as aplicações de voz e imagem, e sugeriam um tamanho de 32 bytes (sem incluir o cabeçalho), enquanto estes pensavam mais nas aplicações de dados, com um pacote de 64 bytes. (também sem o cabeçalho) [Gdc98]

Para encaminhar as células através da rede ATM os comutadores analisam dois campos contidos em cada célula: o VCI (Virtual Channel Identifier) e o VPI (Virtual Path Identitifier). Eles contém respectivamente as informações do VC (Virtual Channel – Canal Virtual), que estabelece as conexões fim-a-fim, e do VP (Virtual Path – Caminho Virtual), que é o conjunto de todos os VCs. Os VCs e os VPs são estabelecidos entre os nós da rede ATM.

6.4.6 – Estruturação em camadas

Na figura 6.3 é apresentado o Modelo de Referência para Protocolos (Protocol Reference Model – PRM), definido pela recomendação I.321 do ITU-T, que apresenta a estruturação do ATM em camadas e planos. As camadas seguem os princípios de ortogonalidade e modularidade apresentados no capítulo 5. Além da estruturação por camadas existe também o conceito de planos, que são pilhas de protocolos do mesmo tipo localizadas entre dois ou mais sistemas conectados, permitindo a comunicação entre eles.

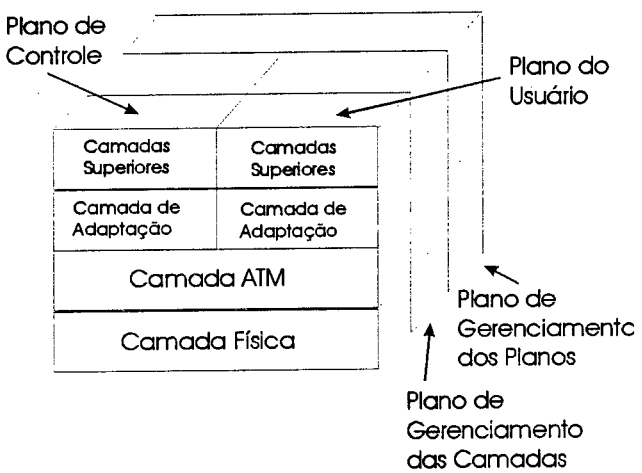


Figura 6.3 – Modelo de referência ATM

A camada de adaptação é a responsável pela adaptação dos diversos serviços das camadas superiores para se adequar a camada ATM. Realiza as funções de convergência e quebra e remontagem dos dados de entrada/saída da/para as camadas superiores. A camada

ATM faz o controle genérico de fluxo, inserção e remoção de cabeçalho, definição dos caminhos de encaminhamento e multiplexação/demultiplexação de células. A camada física é a responsável pela transmissão pelo meio físico.

O plano de usuário é utilizado para a transferência de informação dos usuários. O plano de controle é responsável pelas funções de controle, como a sinalização necessária para ativar, manter e desativar chamadas e conexões. O plano de gerenciamento é responsável pelo gerenciamento dos planos e o das camadas. O gerenciamento dos planos é utilizado para o gerenciamento dos planos do usuário, de controle e do próprio plano de gerenciamento [SLC95].

6.4.7 – O plano de controle em redes ATM

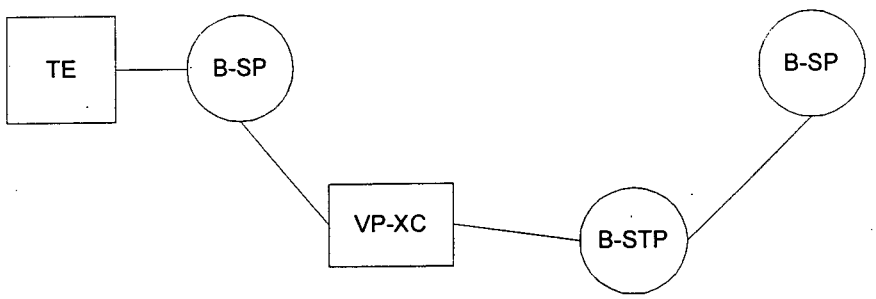
O plano de controle é responsável pela ativação, manutenção e desativação de chamadas e conexões. Chamadas são conexões ou conjuntos de conexões entre dois ou mais participantes [SLC95].

O plano de controle utiliza a sinalização em redes ATM como a base para execução de suas funções, que podem ser descritas como:

- **Controle de admissão de conexões** – Determinando se uma conexão pode ser concedida, garantindo a qualidade de serviço especificada, e buscando minimizar a probabilidade de congestionamentos;
- **Policiamento** – Ajuda na prevenção de situações onde uma determinada fonte de tráfego ultrapassa os limites negociados no momento do estabelecimento de uma conexão;
- **Controle de congestionamento** - Que permite o retorno a um estado normal após um congestionamento;

A sinalização é feita através do plano de controle, que utiliza a camada ATM, comum aos planos de controle e de usuário, para o transporte de células com informações de sinalização. As informações de sinalização em redes ATM são transportadas em conexões próprias, separadas das conexões utilizadas para transporte de informações do usuário ("out of band signalling" – sinalizações fora da banda). Esta estratégia de sinalização exige a

existência de elementos funcionais específicos conforme exemplifica a configuração apresentada na recomendação I.311 e reproduzida na figura 6.4 [SLC95]



- TE – Equipamento Terminal (Terminal Equipament)
- B-SPs Ponto de sinalização (B-ISDN signalling point) - são elementos que geram e processam as mensagens de sinalização.
- B-STPs recebem, roteam e encaminham mensagens de sinalização
- VP-XC são comutadores de mensagens de sinalização

Figura 6.4 – Configuração de uma rede de sinalização do ATM

São três as funções básicas de sinalização em redes ATM:

- Estabelecimento de conexão
- Rompimento de conexão
- Obtenção de status da conexão

Sempre três entidades estão envolvidas: O usuário chamador, a rede e o usuário chamado.

A figura 6.5 mostra um exemplo de estabelecimento de conexão ponto a ponto bem sucedido entre as três entidades da rede.

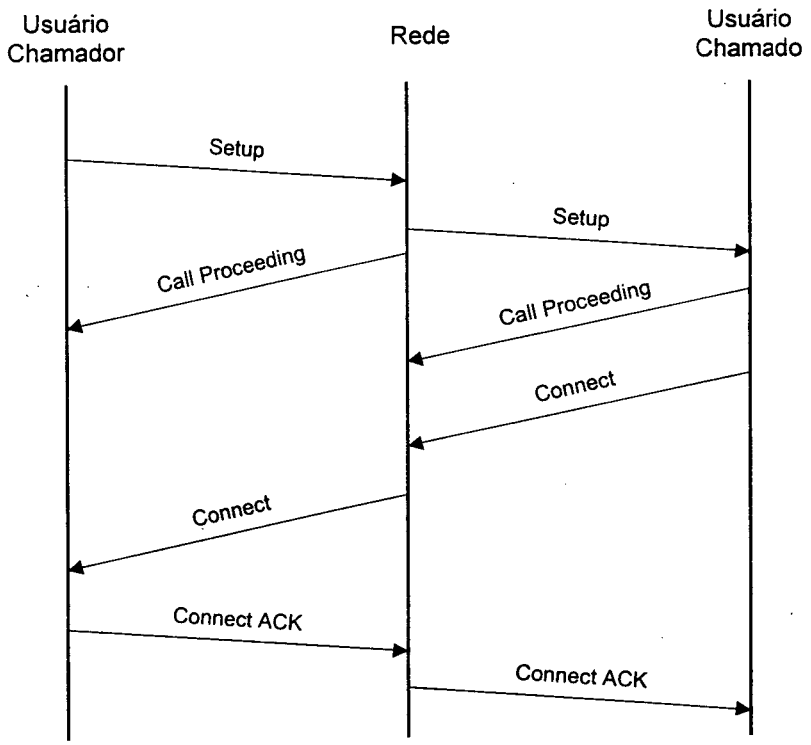


Figura 6.5 – Estabelecimento de conexão ponto a ponto pela rede ATM

O SETUP é utilizado para solicitar a conexão e é transmitido pela rede até atingir um ponto de sinalização (B-SP). O B-SP verifica se existe rota para o destino desejado e se existe disponibilidade da rede atender a qualidade de serviço desejada. Não existindo uma rota direta utiliza-se o elemento B-STP, que deverá permitir o atendimento da qualidade de serviço desejada. Se não há possibilidade a conexão é recusada. O Call Proceeding é uma mensagem opcional utilizada para estender o "time-out" do recebimento de confirmação de pedido de Setup.

As funções do plano de controle estão contidas no documento denominado *Especificação de Gerência de Tráfego* [AF96], desenvolvido pelo ATM Forum.

6.5 – A gerência de tráfego

A seção 6.5 foi baseada nas informações contidas na especificação de gerência de tráfego do ATM Forum [AF96].

A especificação de Gerência de Tráfego (Traffic Management Specification_[AF96]) do ATM Forum define procedimentos e parâmetros relacionados à gerência de tráfego e à qualidade de serviço. A função principal é evitar o congestionamento ao mesmo tempo em que deve aproveitar eficientemente os recursos da rede. Para isto a especificação apresenta mecanismos para atingir o QoS (Quality of Service) em função das cinco categorias de serviço.

6.5.1 – Os parâmetros de QoS

A alocação de recursos e a Qualidade de Serviço (QoS) dependem do tipo de categoria de serviço, influenciando desde o estabelecimento da conexão, passando pela gerência da conexão e de tráfego até a desconexão. Das cinco categorias duas são de tempo real, o CBR e o rt-VBR, e três são de fluxo variável independente do tempo. Para cada categoria de serviço existe uma estratégia própria para garantir o QoS [SLC95]

De acordo com o ATM forum são seis os parâmetros de QoS [AF96]

Os três primeiros são negociados na conexão da comunicação:

- Variação do atraso de células entre picos (Peak-to-peak Cell Delay Variation - peak-to-peak CDV)
- Atraso máximo de transferencia de células (Maximum Cell Transfer Delay - maxCTD)
- Taxa de perda de células (Cell Loss Ratio - CLR)

Uma CDV é a variação de tempo de transmissão de um célula entre dois pontos, geralmente próximo ao transmissor e ao receptor da célula. o CTD é o tempo que uma célula pode demorar. O CLR é a razão das células perdidas e as células transmitidas. Este parâmetro não é aplicável ao UBR.

Os outros três não são negociáveis:

- Taxa de erro de células - CER (Cell Error Ratio)
- Taxa de Blocos de células severamente errados – SECBR (Severely Errored Cell Block Ratio)

- Taxa de células inseridas incorretamente – CMR (Cell Misinsertion Rate)

O CER é igual a razão entre a quantidade de células erradas e as células corretas mais as células erradas. O SECBR é igual à razão do número de Blocos com células erradas sobre o total de blocos transmitidos. O CMR é a razão do número de células enviadas para o destino errado em função do tempo. Uma célula pode ser extraviada se existe um erro no cabeçalho que não foi detectado. (O termo "misinsertion" é relacionado com a inserção errada num dos caminhos, o que leva a um encaminhamento incorreto)

Os parâmetros negociáveis são acordados entre o usuário e a rede no estabelecimento da conexão, estabelecendo o que é denominado de contrato de tráfego. Baseado nos parâmetros contidos neste contrato a gerência de tráfego, através do plano de controle, pode executar suas funções.

Existem diversos fatores que afetam os parâmetros de QoS, entre eles tem-se:

- **Atraso de propagação e erros estatísticos** - inerentes à camada física;
- **Arquitetura do Comutador** – que impacta sobre a performance, sendo que os principais fatores são o projeto da matriz de comutação e a estratégia de bufferização (porta única, partilhada entre várias portas, FIFO, múltiplas filas com regras controladas por algoritmos);
- **Capacidade dos buffers**;
- **Capacidade de tráfego;**
- **Número de nós;**
- **Alocação de recursos;**
- **Falhas.**

6.5.2 – Outros parâmetros da rede ATM

Além dos parâmetros de QoS existem diversos outro que regem o comportamento das redes ATM e sistemas usuários, tais como:

- **Taxa de pico de células – PCR (Peak Cell Rate)** – é o parâmetro que define a taxa máxima que os terminais podem trafegar células. Este parâmetro pode ser

desrespeitado pelas fontes de tráfego sob a pena de descarte de células pela rede ATM;

- **Taxa máxima média de células – SCR (Sustained Cell Rate)** – define a taxa média máxima, para a categoria VBR, em que os terminais podem trafegar células. Da mesma forma que o PCR, a rede pode descartar células que não respeitem este parâmetro. Os recursos alocados para garantir este parâmetro são menores do que aqueles alocados para garantir o PCR [AF96];
- **Tamanho máximo da rajada – MBS (Maximum Burst Size)** – é o máximo período de tempo trafegando na taxa de pico PCR. Aplicável na categoria VBR;
- **Taxa de células mínima – MCR (Minimum Cell Rate)** – define a taxa de células mínima para a categoria ABR.

6.5.3 – O controle de congestionamentos

Existem diversas funções especificadas para atuar dentro do ATM buscando evitar o aparecimento e minimizar a intensidade, abrangência e duração dos congestionamentos. Evitar e tratar os congestionamentos são uma conseqüência da necessidade de garantir os objetivos de QoS. Para isto pode-se utilizar uma ou mais funções em conjunto das seguintes estratégias [AF96]:

- **Controle de Admissão de Conexões – CAC (Connection Admission Control)** – Controla a possibilidade de estabelecimento de Conexões;
- **Parâmetro de controle de uso - UPC (Usage Parameter Control)** – É o conjunto de ações executados pela rede para controlar e monitorar o tráfego;
- **Adequação do Tráfego (Traffic Shaping)** – Atua-se sobre o fluxo de células, reduzindo picos e rajadas;
- **Indicador de Congestionamento explícito para frente** – É feito pela rede ou usuários finais atuando sobre o tráfego gerado;
- **Gerência de Recursos** – Atua sobre os VP (Virtual Paths).

6.5.3.1 – CAC – Controle de Admissão de Conexões

O CAC – Deve ter acesso as informações acertadas no contrato de tráfego. Esta unidade define se uma conexão pode ser estabelecida, ou não, baseado na disponibilidade de recursos da rede em função do QoS desejado e mantendo o QoS das demais conexões anteriormente estabelecidas.

O parâmetro CLP (Cell Loss Priority – Prioridade de descarte de célula) define as células que terão prioridade de descarte em caso de comprometimento do QoS. Assim, um fluxo de células CLP = 0 é mais seguro, no caso de eventual congestionamento, do que um fluxo misto CLP = 1 e CLP =0 (CLP = 0 + 1), e do que um fluxo puro CLP = 1.

Conexões com fluxo CLP = 0, CLP =1, ou CLP = 0 + 1 podem ser estabelecidas dependendo da categoria do serviço a ser oferecido e da estratégia da operadora da rede ATM. Diferentes estratégias de alocação de recursos podem ser adotadas para cata tipo de fluxo, e são específicas de cada rede.

6.5.3.2 – O UPC – Parâmetro de controle de uso

A função de UPC é opcional na interface UNI, e deve ser posicionada o mais próximo aos pontos terminais da rede ATM. São um conjunto de ações para monitorar e controlar o tráfego. Seu objetivo é proteger a rede de ações perniciosas ou equivocadas que possam prejudicar o QoS das conexões já estabelecidas atuando sobre aquelas conexões que estão violando o contrato de tráfego.

Os algoritmos de UPC são próprios de cada rede, apesar de algumas características serem comuns a todas elas.

O CTD e o CDV (vide item 6.5.1) são monitorados pelo UPC, comparando os valores atuais com os do contrato para cada conexão, permitindo um desvio de acordo com definição de cada operadora.

Uma célula analisada pelo UPC pode:

- Passar transparentemente
- Ter seu parâmetro CLP = 0 alterado para CLP =1
- Ser descartada

O parâmetro CLP (Cell Loss Priority – Prioridade de descarte de célula) permite a marcação de células que terão prioridade de descarte em caso de comprometimento do QoS. A alteração do CLP pode ser feita tanto pela rede como pelo usuário. Assim, o próprio usuário pode estabelecer um fluxo de células garantido, e outro que pode sofrer cortes em caso de congestionamento. O UPC pode marcar células para conformá-las, se no estabelecimento da conexão o usuário solicitou marcação e se a rede possuir tal funcionalidade.

Na rede ATM em condições normais, isto é, sem congestionamento, a célula com CLP igual a 1 não será descartada, exceto se estiver fora do contrato de tráfego. Se no contrato de tráfego o fluxo é CLP = 0 + 1, células com CLP marcado com 1 que estão conforme passam, caso contrário são descartadas. Quando o fluxo CLP = 1 "conforme" ocupou todos os recursos da rede alocados, então as células não conformes CLP =0 são descartadas.

O PCR do fluxo CLP = 0 + 1 deve ser analisada pelo UPC. Análise de SCR e MBS é específica de cada rede.

A possibilidade de falhas no UPC foi considerada [AF96]: no caso de um dispositivo estar atuando sobre o tráfego em desacordo com os parâmetros acordados no estabelecimento da conexão, então o plano de gerência da rede ATM deve atuar, por exemplo isolando o enlace com defeito.

6.5.3.3 - Adequação de Tráfego

É uma função que busca atender à QoS e atingir melhor eficiência de utilização da rede. A adequação do tráfego deve manter a integridade da seqüência de células, podendo atuar sobre o pico de tráfego, limitação das rajadas de células e redução do CDV (espaçando as células uniformemente ao longo do tempo.)

Cada rede deve decidir se e onde implementar a adequação de tráfego. São duas as opções:

- Não implementar, dimensionando a rede para acomodar qualquer fluxo de tráfego sem necessidade de adequações.
- Implementar, adequando o tráfego aos recursos da rede, otimizando a eficiência.

6.5.3.4 - Indicador de Congestionamento explícito para frente

O ATM dispõe de uma estratégia para avisar ao destino que o fluxo de células está passando por nós congestionados. Assim, as camadas de aplicação mais altas podem atuar sobre o volume de tráfego enviado, buscando diminuir ou eliminar o congestionamento. Apenas o indicativo de congestionamento é função do ATM, a ação é de protocolos superiores não cobertos pelo ATM.

6.5.3.5 - Gerência de recursos utilizando os caminhos virtuais VP (Virtual Path)

Os caminhos virtuais simplificam o CAC e permitem alocação de recursos em função da categoria do serviço, associando a um ou mais grupos de VPs conexões da mesma natureza.

6.5.4 – Conformidades dos fluxos de células para manutenção dos QoS

O descritor do tráfego da fonte define as características do tráfego gerados. Os parâmetros de tráfego descritos na especificação do ATM Fórum [AF96] são a taxa de pico de Células - "Peak Cell Rate" (PCR), a Taxa sustentável de células – "Sustainable Cell Rate" (SCR), o tamanho máximo de rajada – "Maximum Burst Size" (MBS), e a taxa mínima de células – "Minimum Cell Rate" (MCR).

O descritor do tráfego da conexão contém os parâmetros do descritor de tráfego da fonte mais o CDVT, e uma definição de conformidade, fornecida pelo GRCA (Algoritmo genérico de taxa de células - "Generic Cell Rate Algorithm"), que especifica quais células estão dentro do negociado para a conexão. Na conexão, através do CAC, o descritor de tráfego da conexão será usado para alocar recursos e definir parâmetros para a ação do UPC. O GRCA é um algoritmo de agendamento virtual mais conhecido como algoritmo do balde furado, que define operacionalmente relações entre PCR e CDVT, SCR e BT (Burst Tolerance – Tolerância a rajadas) e define a conformidade para as categorias CBR, rt-VBR, nrt-VBR, e UBR. Múltiplas instâncias de GCRA podem ser abertas para os múltiplos fluxos de células. Embora o GCRA possa definir se uma célula está ou não conforme ele não promove uma

ação. Cabe ao UPC implementar o GRCA ou outros algoritmos para garantir o QoS. A especificação de gerência de tráfego [AF96] explica o funcionamento do algoritmo, que é baseado em critérios de performance.

Os parâmetros de tráfego podem ser explícitos, determinados pelos sistemas terminais ou pelo sistema de gerenciamento de rede (NMS – "Network Management System" – Sistema de gerencia de rede), ou implícito, quando a rede determina valores dos parâmetros usando regras padrão baseadas nas informações dos equipamentos terminais. A conformidade das células é função do algoritmo de compressão e parâmetros correspondentes especificados no descritor de tráfego e está especificado no contrato de tráfego. O conjunto de definições de conformidade é específico de cada rede.

A definição de conformidade para conexões CBR, rt-VBR, nrt-VBR, ou UBR é específica de cada rede, que deve garantir o QoS de todas as conexões conformes. Para conexões não conformes a rede não precisa respeitar o QoS

Para operações da rede existem dois modelos para o fluxo de células CLP = 1:

- **CLP – transparente** - Não utiliza a estratégia de marcação e o fluxo CLP = 1 está sujeito ao mesmo CLR do que o fluxo CLP = 0.
- **CLP – significante** – Os objetivos de CLR se aplicam apenas para o fluxo CLP = 0. O CLR para o fluxo CLP = 1 não é especificado, bem como para o agregado CLP = 0 + 1. A marcação é opcional.

Cada categoria de serviço é regida por um ou mais parâmetros principais para controle do QoS [SLC95]

6.5.4.1 – O CBR

O tráfego de serviços CBR são avaliados pela taxa de pico de células (PCR) e CDVT para o tráfego CLP = 0 + 1. O tráfego CLP = 1 é CLP transparente, e a marcação de bits não é aplicável, pois não há uma especificação em separado para a taxa CLP = 0. Os objetivos de CLR são aplicados apenas ao CLP = 0 + 1. A rede deve garantir a entrega de todas as células que estejam conforme o contrato de tráfego. A fonte pode transmitir células até no máximo a taxa PCR, podendo transmitir a taxas menores ou mesmo não transmitir nada.

6.5.4.2 – O rt-VBR e o nrt-VBR

São caracterizados pela taxa média sustentável de células (SCR), e o tamanho de rajada máximo (MBS) para os fluxos de tráfego, e a taxa de pico de células (PCR) e o CDVT para o CLP = 0 + 1. A diferença entre o rt-VBR e o nrt-VBR está nos parâmetros de QoS e principalmente na magnitude do MBS. MBS maiores são característicos de conexões nrt-VBR.

As fontes podem transmitir em taxas que podem variar com o tempo. O interessante deste serviço é a possibilidade dos equipamentos utilizarem multiplexação estatística, otimizando a ocupação da banda. O atraso de células acima do maxCTD deve ser muito baixo.

Existem dois modelos de conformidade para o VBR:

- **Modelo CLP significante** - que contém dois GCRA, um para controle do fluxo CLP = 0 + 1 e o outro para controle do fluxo CLP = 0. Uma célula CLP = 0 é dita conforme se atende aos dois GCRA, enquanto que para a célula CLP = 1 basta atender a primeira definição de GCRA
- **modelo CLP transparente** - É utilizado um GCRA para definir o CDVT em relação ao PCR para o fluxo CLP = 0 + 1 e um GCRA definindo a soma do BT com o CDVT em relação ao SCR do fluxo CLP = 0 + 1. Uma célula é dita conforme se atende aos dois GCRAs. O fluxo de células CLP = 0 e o CLP =1 devem ser no máximo iguais ao PCR, sendo que o CLR se aplica a estes dois fluxos.

6.5.4.3 - O UBR

Esta categoria de serviço é para aplicações não suscetíveis a atrasos. Não existe qualquer garantia de banda. Não existe nenhum comprometimento quanto a CLR ou CTD. A rede pode ou não aplicar o PCR nas funções CAC e UPC. Mesmo nos casos onde a rede não controla o PCR é importante ter este parâmetro informado pela rede para que a fonte saiba as limitações de banda ao longo da conexão. O controle de congestionamento do UBR é feito

por camadas superiores ao ATM. A categoria de serviço UBR não é sujeita a um contrato de trafego mas pode ser sujeita a políticas locais de comutadores e sistemas terminais.

6.5.4.4 - O ABR

O objetivo do ABR é prover um acesso rápido para a banda não utilizada da rede. A categoria recebe uma seção exclusiva no documento [AF96]. Ela usa os recursos disponíveis na rede, adaptando-se as mudanças de condições. As informações sobre a rede, tais como banda disponível e situações de congestionamento, são enviadas para a fonte através de um tipo de célula denominado RM ("Resource Management" – gerência de recursos). As células da fonte para o destino são denominadas células RM para frente, enquanto que as enviadas do destino para a fonte são denominadas células RM para trás. As células RM que são enviadas pela fonte, ao chegar no destino podem ter seus parâmetros alterados e são enviadas de volta. Além do destino a própria rede pode alterar os parâmetros das células RM, tanto as para frente como as para trás. A rede pode setar no cabeçalho, por exemplo o bit EFCI, que indica um estado de congestionamento, para que o destino e/ou a fonte atuem sobre o congestionamento. Além de alterar parâmetros de células RM que estão circulando pela conexão a rede pode, se necessário, gerar as células RM.

Apesar da especificação do ATM não definir esquemas de controle de fluxo de células utilizando as células RM para o CBR, rt-VBR, nrt-VBR ou UBR, estas células são permitidas, e são consideradas parte do fluxo de dados do usuário, podendo ser utilizadas por camadas superiores as camadas do ATM.

No estabelecimento da conexão, o sistema terminal deve negociar com a rede o máximo e o mínimo de largura de banda necessários, que serão o PCR e o MCR. A banda disponível da rede pode variar mas não deve ficar menor do que MCR. Para isto a rede reserva recursos, garantindo que o controle de realimentação não faça com que a taxa de células caia abaixo do MCR. O controle de Conexões CAC pode aceitar inúmeras conexões com MCR = 0, entretanto para conexões com MCR >0 os recursos da rede devem ser analisados.

O próprio algoritmo que define a conformidade das conexões ABR também define os parâmetros esperados, utilizando o feedback através de células RM com informações de taxas explícitas de células - ER (Explicit rate), indicação de congestionamento – CI (Congestion Indication), e indicação de não aumento da taxa – NI (Non Increase).

Nas conexões ABR existe comprometimento de CLR, que é específico de cada rede, sendo que o objetivo é minimizá-lo adaptando o tráfego dos sistemas terminais para as características da rede. As células de dados são CLP = 0 e as de RM são CLP = 0 ou CLP = 1. A marcação não é suportada, e as células RM com CLP = 1 podem ser descartadas pela rede.

A rede pode dividir a conexão ABR em segmentos de controle separados utilizando uma função que simula a fonte e/ou o destino.

Os critérios de alocação de recursos para garantir a banda para o ABR atendem a critérios de justiça específicos de cada rede. A taxa de cada conexão ABR tem dois parâmetros: MCR, que pode ser igual a zero e um parâmetro elástico que, negociado com a rede busca garantir que a banda será compartilhada de forma justa entre as conexões ABR.

A taxa de células permitida (ACR) enviada pelos sistemas terminais nunca pode ser menor do que o MCR. Quando conexões ABR com MCR > 0 são setadas então o CAC pode bloquear futuras solicitações de conexões com MCR > 0, entretanto o CAC geralmente não deve bloquear conexões MCR=0.

Como o ABR tem problemas de controle sobre taxa de células RM durante períodos em que nenhum dado é transmitido, a rede pode reservar determinada quantidade de banda, que é específica de cada rede. Isto pode causar a recusa de conexões inclusive de MCR =0.

Na regulação do tráfego da categoria de serviço ABR participam as três entidades, o usuário que está gerando o fluxo de células (denominado fonte), a rede ATM (principalmente o comutador), e o destino das células.

O comportamento da fonte

Embora a rede tenha se comprometido a suportar MCR a fonte pode receber indicativos para reduzir a taxa abaixo de MCR. Neste caso deve utilizar o MCR.

ACR nunca deve exceder PCR nem ficar abaixo de MCR. Depois da conexão e antes da primeira célula o ACR deve ser igual a taxa de células inicial (ICR), característico de cada rede. A própria fonte calcula o ACR dentro de determinados parâmetros e envia através de uma célula RM para frente.

A primeira célula a ser mandada deve ser uma RM para frente. Após isto as próximas células devem seguir a seguinte ordem: RM se e somente se o número de células atingiu o Mrm (parâmetro de alocação de banda das células RM) e se o tempo entre células RM

ultrapassou o valor Trm (limite máximo de tempo entre duas células RM para frente), ou se o Nrm - 1 (parametro que especifica o número máximo de células entre duas transmissões de células RM) foi atingido.

Se não existir imposição de envio de uma célula RM para frente então a próxima célula deve ser uma célula RM para trás, se existir uma célula RM para trás esperando para transmissão, ou se não foi mandada nenhuma célula RM para trás desde a ultima RM para frente, ou se nenhuma célula de dados está esperando para ser transmitida. A próxima célula é uma célula de dados se nenhuma das condições anteriores for verdadeira.

Como as conexões ABR utilizam os recursos disponíveis da rede é importante que constantemente a fonte receba uma realimentação da situação desta rede. Quando a realimentação falha é preciso que a fonte se antecipe a possíveis problemas de congestionamento que resultarão em descarte de células. Para isto existe o cuidado com a constante realimentação e existem os tempos e contadores de controle. Por exemplo, antes de mandar uma célula RM para frente, se ACR é maior do que ICR e decorreu um tempo maior do que ADTF, então o ACR volta a ser igual a ICR. Outro mecanismo de antecipação baseado na falta de feedback funciona da seguinte forma: se células para frente forem mandadas acima de uma quantidade antes do recebimento de uma célula RM para trás com BN = 0, então o ACR é decrescido de um fator ACR*CDF, sendo que CDF é o denominado fator de decremento de corte (Cutoff Decrease Factor).

Se uma célula RM para trás tiver o indicador de congestionamento (CI) igual a 1, então o ACR deve ser decrementado do fator ACR*RDF, exceto se este decremento levar a um valor abaixo de MCR, adotando-se então este valor mínimo. O fator RDF é denominado RDF (Rate Decrease Factor). Se a célula RM para trás chegar com CI igual a 0 e o indicador NI (No Increase) igual a 0 então o ACR pode ser aumentado de um fator igual a RIF (Rate Increase Factor) *PCR, ou até PCR. O indicador NI determina se a fonte pode ou não aumentar o ACR de um fator determinado por RIF. Se o NI for igual a 1 então o ACR deve ficar inalterado.

O comportamento do Destino

Células RM recebidas pelo destino devem ser retornadas para a fonte. Se o EFCI = 1 então o destino deve setar CI = 1. Se o destino estiver congestionado pode reduzir o

parâmetro de taxa de células explicito ER e/ou setar CI e NI em 1. Quando a célula RM para trás voltar para a fonte ela poderá mudar a taxa de células para se adequar a rede. Ao mesmo tempo, protocolos de camadas superiores ao ATM podem ler os indicativos de congestionamento tanto na fonte como no destino e ativar os mecanismos de controle implementados.

Quando o destino recebe uma célula RM deve reenviar a célula para a origem, sendo que o CCR (Current Cell Rate – taxa de células corrente), o MCR, o ER, o CI, e o NI devem permanecer constantes exceto se o EFCI está setado, então o CI é setado em 1 e o EFCI é resetado.

Comportamento do Comutador ATM

A rede não deve solicitar que a fonte trafegue menos do que MCR. Se a fonte não solicitar um MCR maior do que zero então a rede adota o padrão 0.

O comutador deve implementar pelo menos um dos diversos métodos de controle de congestionamento nos pontos de filas:

- Marcação de EFCI;
- comutador pode setar CI=1 ou NI=1 nas células RM para frente ou para trás;
- comutador pode reduzir o campo ER;
- comutador pode gerar células RM para trás;
- comutador pode implementar uma política de "use ou perca" para reduzir o ACR.

6.6 – Conclusão

Neste capítulo foram mostradas as características do sistema de comunicações de dados ATM. Dentro de um nível de abstração e modularização adequados, conforme conceitos apresentados no capítulo 5, serão construídos modelos deste sistema, que serão apresentados no capítulo 7. Também neste capítulo utilizar-se-á a ferramenta SMV, apresentada no capítulo 4, para verificação e projeto de alguns comportamentos do ATM.

Capítulo 7 – Verificando características de uma rede ATM

Neste capítulo serão utilizados vários conceitos apresentados nos capítulos anteriores. Serão mostradas formas de aplicação de uma ferramenta de verificação formal para confirmar a validade de características da rede ATM, e apresentar algumas situações que devem ser consideradas no projeto, tais como estratégias de atuação, dimensionamento, escolha das categorias de serviços, e outras. Para a verificação de propriedades utilizou-se a ferramenta SMV cujos principais aspectos foram mostrados no capítulo 4. Para modelar o sistema proposto foram utilizados alguns dos conceitos apresentados no capítulo 5, sendo que a abstração e a modularização se destacam como as principais formas de trabalhar o complexo problema da modelagem. O sistema a ser modelado é baseado principalmente nas especificações do ATM Forum [AF96], apresentadas no capítulo 6.

Buscar-se-á ao longo do capítulo a apresentação e verificação de propriedades tais como: partilhamento justo de recursos, ausência de congelamentos, ausência de ciclos não progressivos, comportamento dentro do especificado, e outros.

7.1 – Considerações sobre o problema

Várias das especificações do ATM são baseadas em performance, que não são alvo da estratégia de verificação utilizada pelas ferramentas e metodologias apresentadas neste documento.

A modelagem utilizou estratégias de abstração para reduzir a complexidade do modelo final sem perder o foco nos principais pontos a serem verificados. Canais de mensagem, por exemplo, foram representados por variáveis, conforme sugerem experiências realizadas por BIERE e KICK [BK95]

Apesar de não ter-se utilizado diretamente o artifício de Diagramas de Transição de Estados (DTE) (ou mais especificamente autômatos de estados) na verificação, os mesmos foram utilizados buscando uma representação mais clara do comportamento dos subsistemas, ou módulos.

Buscou-se modelar o comportamento de um sistema composto por uma rede ATM e algumas fontes de tráfego de células. Todas as considerações feitas no modelo se referem a células. Não foi considerado o isolamento como técnica de controle de tráfego, e assim a abstração desconsidera definições de VPs e VCs.

Cada fonte de tráfego foi modelada em função de uma das seguintes categorias de serviço: CBR, VBR ou ABR. Não foi criado um modelo para fonte UBR, pois no nível de abstração utilizado não existem diferenças relevantes entre esta categoria e o VBR, a não ser a desconsideração do parâmetro CLR pelo UBR. Pelo mesmo motivo utilizou-se apenas um modelo para VBR, pois o VBR de tempo real e o VBR de tempo não real diferem apenas pelas magnitudes de seus parâmetros.

Cada fonte de tráfego correspondente de cada categoria foi representada por um módulo. Como a rede apresenta reações especificas para cada categoria, foi utilizado um módulo de rede para especificar o comportamento para cada categoria.

Muitas das preocupações mostradas nas especificações de controle de tráfego [AF96] referem-se ao congestionamento, pois ocorrendo este problema os níveis de QoS acordados no estabelecimento da conexão podem ser comprometidos. Uma conexão dentro de um VP pode passar por diversos nós, cada um com recursos alocados e disponíveis, e dependendo de aleatoriedades, dinamicidades do tráfego e problemas no provisionamento dos recursos, alguns nós podem estar no estado de congestionamento. Algumas estratégias de controle de congestionamento atuam sobre a fonte do fluxo, através de camadas superiores ao ATM, alertadas através de células de feedback, e/ou através do descarte de células/quadros em cada nó. Apesar de existir diversas combinações de rotas possíveis e existir uma dinâmica de estabelecimento de conexões temporárias, para a modelagem da rede do ponto de vista da fonte utilizou-se de abstração e considerou-se a rede como um único nó que pode ou não estar congestionado em função do uso inadequado dos recursos, ou seja, para a fonte de tráfego a rede é vista através da interface usuário-rede (UNI), recebendo através dela informações desde o sucesso ou fracasso do estabelecimento de conexões, e parâmetros associados, bem como feedbacks de parâmetros de tráfego.

Existem diversas especificações do ATM Forum relativas ao gerenciamento de tráfego [AF96] que são opcionais e específicas para cada rede. Assim existem diversas possibilidades e características que devem ser consideradas nas redes ATM que dependem dos fornecedores tanto dos equipamentos de ponta como dos equipamentos da rede. Na modelagem buscou-se a utilização das especificações obrigatórias do ATM Forum [AF96], bem como as

especificações opcionais que se sobressaiam as abstrações feitas e que contribuíam para a manutenção dos níveis de QoS acordados.

7.2 – Modelando

7.2.1 – A modelagem de uma fonte geral

Antes de mostrar a modelagem de cada fonte em função das categorias de serviço, mostra-se na figura 7.1 um modelo mais abstrato que busca representar todos os diversos tipos de fontes.

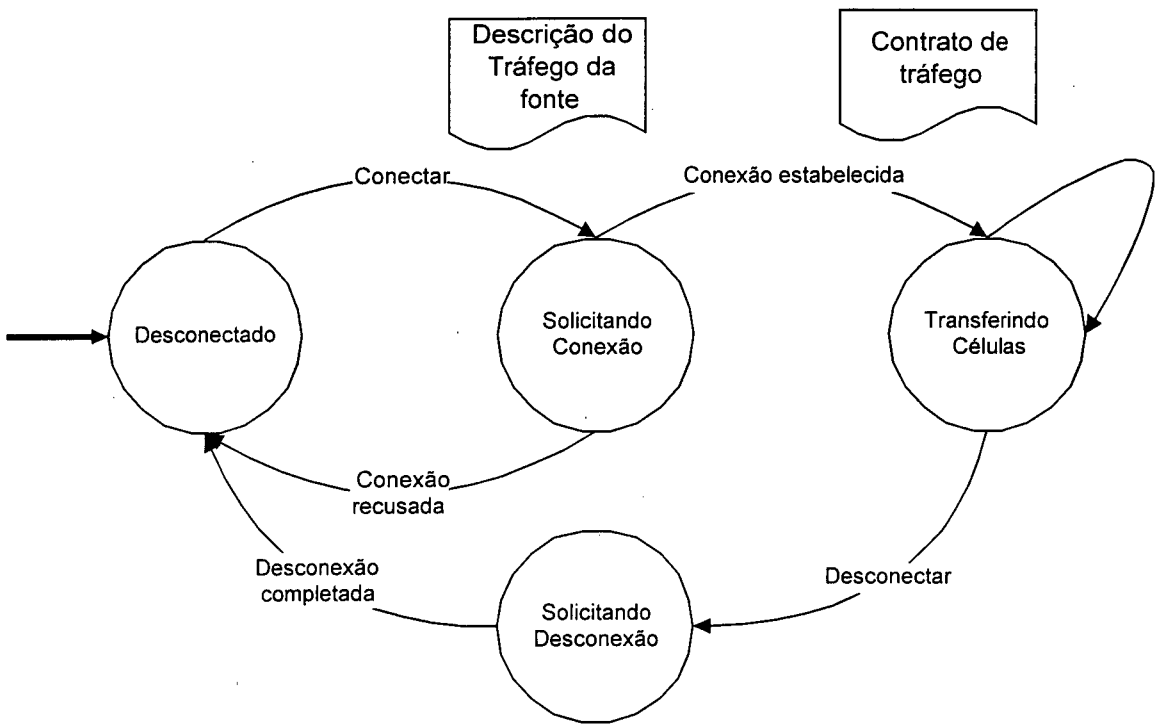


Figura 7.1 – Modelo abstrato das fontes de tráfego

Na figura 7.1 existe a representação, utilizando a simbologia de fluxogramas, de dois documentos. Eles são uma adaptação do DTE, ressaltando dois conjuntos de informações que são vitais para as ações da rede. O primeiro informa para a rede as necessidades solicitadas pela fonte quanto à conexão. Este conjunto de informações é a base para o funcionamento do

CAC (Controle de Admissão de Conexões), da elaboração dos parâmetros de QoS, e do contrato de tráfego.

O modelo geral mostrado na figura 7.1 é comum para todos as fontes, o que muda são os conteúdos dos dois conjuntos de informações e a explosão do estado de transferência de células, representando principalmente as variações das taxas dos fluxos de tráfego e possíveis ações sobre informações de "feedback", especificamente para o caso do ABR.

Antes de especificar o que contém cada conjunto de informações é necessário introduzir o modelo geral para a rede.

7.2.2 – A modelagem da rede do ponto de vista da fonte

Na figura 7.2 mostra-se o modelo abstrato que busca modelar todos os tipos de comportamentos da rede vistos pela fonte em função das características desta.

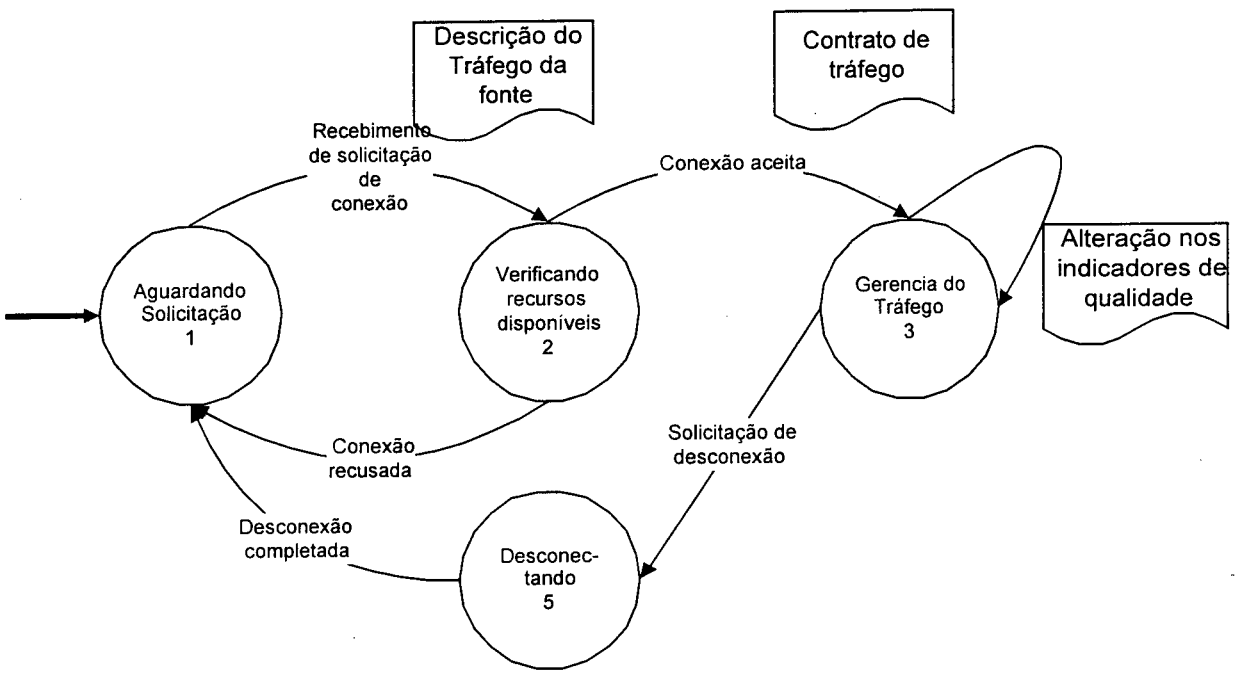


Figura 7.2 – Modelo abstrato da rede vista pelas fontes

Analogamente a modelagem geral das fontes de tráfego, a modelagem da rede sob o ponto de vista das fontes deve ser adaptado para cada tipo de categoria de serviço conectado ou em conexão. O estado 3 da figura 7.2, por exemplo deve modelar as ações da rede sobre células em desacordo com o contrato de tráfego. Devem ser representados todos os

comportamentos da rede que envolvam mudanças nas características dos fluxos gerados pelas fontes, como ações do UPC, adaptações do tráfego, descarte de células, geração de feedbacks, etc.

Na figura 7.2 pode-se ver, analogamente à figura 7.1, a existência dos dois conjuntos de informação que são a base para o estabelecimento e gerência das conexões.

7.2.3 – As informações gerais trocadas entre as fontes e a rede

Para cada categoria o descritor de tráfego da fonte possui um ou mais parâmetros:

- CBR – PCR
- VBR – PCR, SCR e MBS
- ABR – PCR e MCR

Na modelagem não será considerado o parâmetro de CDVT das conexões, que é relativo à performance. Uma modelagem pode considerar todas as opções possíveis, modelando as causas para atrasos de células, introduzindo fatores aleatórios e verificando estados de atraso, porém fatores de atraso de células não serão considerados nos modelos deste documento.

Baseado nas informações recebidas pela fonte o modelo da rede deve, baseado no comportamento do CAC estabelecer ou não a conexão.

7.2.4 – Os recursos compartilhados na rede

O modelo proposto possui dois tipos de recursos compartilhados pelos usuários da rede:

- Quantidade de células máximas transmitidas simultaneamente – maxC;
- Quantidade máxima de buffers – maxB.

Quando conexões são estabelecidas os recursos são alocados. Assim tem-se:

- Quantidade de células reservadas – QCR;

- Quantidade de buffers reservados – QBR.

Assim, para cada estabelecimento de conexão o valor dos parâmetros da fonte é verificado e, dependendo da categoria do serviço e das grandezas do descritor de tráfego da fonte, a conexão é ou não estabelecida. Se estabelecida, a conexão irá utilizar recursos da rede, reduzindo os valores dos recursos disponíveis.

A alocação de banda (tráfego disponível), para o CBR, VBR e ABR é diretamente relacionada ao PCR, SCR e MCR respectivamente. Por outro lado, a estratégia de alocação de buffers não é tão direta. Para algumas categorias os parâmetros PCR, SCR, MBS e MCR das conexões exigem maior ou menor utilização de buffers.

A figura 7.3 mostra um resumo das variáveis relativas aos recursos compartilhados do modelo de uma rede ATM vista pelo ponto de vista das fontes.

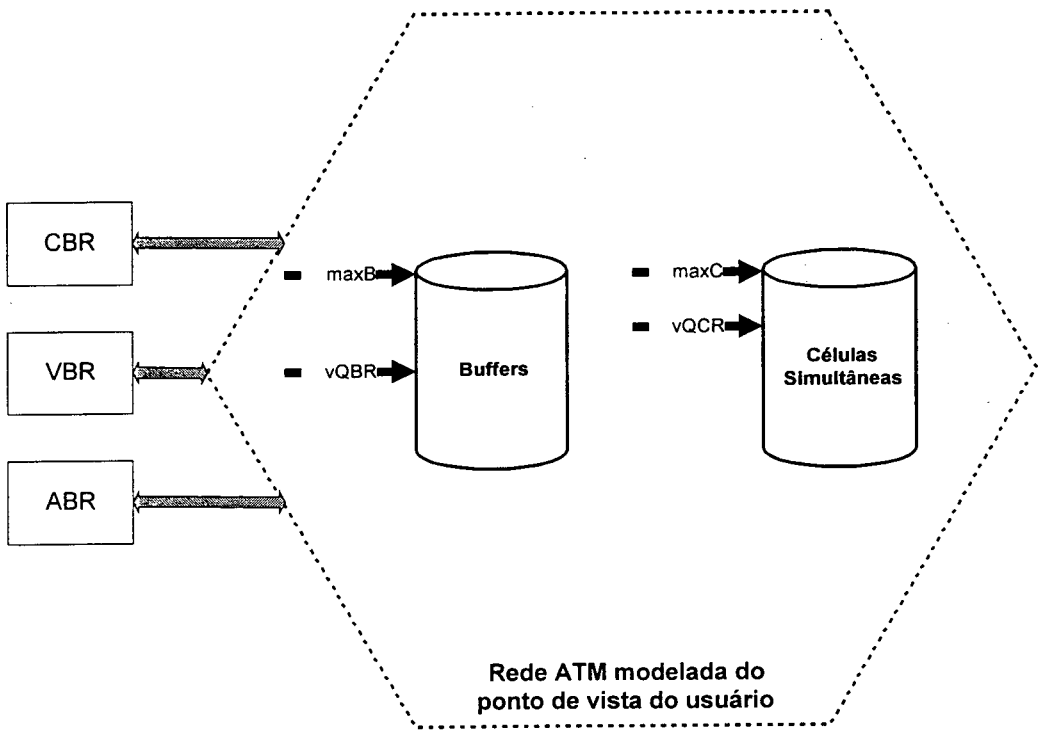


Figura 7.3 – Recursos compartilhados no modelo de rede ATM

7.2.5 – Descrição dos comportamentos das categorias de serviços

Cada fonte buscará o estabelecimento de conexões CLP = 0 + 1 e ocupará recursos do modelo de rede (buffers e células) em função da sua categoria de serviço e parâmetros do

descritor de tráfego. Para analisar o comportamento da rede quanto a correções de tráfego e problemas em manter o QoS, utilizar-se-ão fontes "não comportadas" isto é, serão alterados alguns parâmetros para que exista a possibilidade das fontes transmitirem mais do que o acordado no contrato de tráfego.

O modelo do comportamento da rede deve implantar a estratégia de UPC. Como a estratégia de marcação não existe nos modelos adotados então existem duas opções para uma célula: Ela passa ou é descartada.

7.2.6 – As linguagens do modelo

Para descrever o comportamento das fontes e redes dos modelos do sistema serão utilizadas a linguagem cotidiana, a linguagem de autômatos (representados por Diagramas de Transição de Estados (DTE)), e a linguagem de entrada da ferramenta SMV [Mcm98].

Cada linguagem apresenta vantagens e desvantagens para descrição dos comportamentos dos sistemas. A linguagem cotidiana transmite mais facilmente a idéia geral, entretanto palavras podem ter diferentes interpretações e detalhes são facilmente ignorados. A representação por DTEs apresenta a vantagem de reduzir os erros de interpretação, porém consome mais tempo para transmitir a idéia do comportamento, e pode conter falhas na descrição. Por último, a descrição através da linguagem de uma ferramenta como o SMV apresenta uma dificuldade maior de descrição e entendimento, mas, apesar de existir a possibilidade de não especificar certos comportamentos ela é menor do que a das outras linguagens. Entretanto, mesmo mínimas, tais falhas acarretam erros perigosos na utilização da ferramenta, pois quando o comportamento de uma variável ou estado não for mapeado para a ferramenta, ela adotará valores indefinidos, fornecendo resultados incorretos e imprevisíveis. Por causa disto, além de uma minuciosa revisão das descrições de comportamento, utilizou-se, e propõe-se a utilização de especificações de comportamentos esperados, com resultados absolutamente conhecidos. Tal prática funciona como uma prova dos 9 para a descrição.

A descrição através de DTEs não é um passo necessário para se obter sucesso na descrição utilizando a linguagem da ferramenta. Poder-se-ia ter traduzido a descrição do ATM Forum [AF96], feito em linguagem cotidiana, diretamente para a linguagem da ferramenta. Entretanto para ajudar na abstração, que desconsiderou determinados comportamentos

atrelados a performance e tempo, utilizou-se alguns esquemas em DTE. Além disto os DTEs dão uma boa visão geral de diversos módulos trabalhando interrelacionados.

7.2.7 – Os parâmetros adotados

Os modelos que serão mostrados nos itens 7.3, 7.4, 7.5 e 7.6 adotarão os valores para os parâmetros conforme a tabela 7.1. Os valores escolhidos não são os valores encontrados em sistemas reais, pois as ferramentas de verificação não suportariam trabalhar com as magnitudes envolvidas em tais sistemas, gerando um número de estados alcançáveis extraordinário.

Apesar dos modelos não utilizarem os valores dos parâmetros encontrados na prática isto não invalida as conclusões obtidas, pois o principal objetivo, quando da verificação de um protocolo ou sistema de comunicação de dados, é a identificação de regras de procedimento incompletas ou contraditórias, conforme apresentado no capítulo 5.

Categoria	Parâmetro 1	Parâmetro 2	Parâmetro 3	ΔvQCR	ΔvQBR
CBR	10≤ PCR ≤15	-	-	+ PCR	+PCR*2
VBR	10≤ PCR ≤15	SCR=PCR-5	3≤ MBS ≤6	SCR	MBS* (PCR-SCR)
ABR	10≤ PCR ≤15	-	0≤ MCR ≤5	MCR	MCR*2

Tabela 7.1 – Resumo dos parâmetros envolvidos na modelagem das fontes

7.2.8 – O sincronismo dos módulos

A dinâmica dos sistemas descritos nos itens 7.3, 7.4, 7.5 e 7.6 é condicionada a um relógio virtual, que trabalha em passos. Todos os estados e variáveis dos diversos módulos são atualizados simultaneamente a cada passo, o que garante o sincronismo em transições que não são foram sincronizadas através de eventos. Para seguir tal premissa na verificação dos

modelos, utilizando a ferramenta SMV, foi escolhido e indicado para esta o modo síncrono de evolução de variáveis.

7.3 – O CBR

7.3.1 – Descrição do comportamento da fonte CBR

A fonte CBR determina um nível de PCR para o seu tráfego. Este nível é informado no descritor de tráfego pela variável cbrQCR. O modelo da fonte permitirá a tentativa de estabelecimento de conexões com diversos PCRs. A rede poderá recusar ou aceitar a conexão em função dos recursos disponíveis. O valor de maxC – QCRtotal (que é o somatório de todas as reservas de células das outras conexões estabelecidas), deve ser maior do que o cbrQCR, e se estabelecida a conexão, o valor de QCRtotal deve ser acrescido de cbrQCR. Para o modelo proposto as conexões CBR possuem altas garantias de QoS, assim para cada célula simultânea indicada por cbrPCR utilizar-se-ão dois buffers. De acordo como o item 6.6.8 não existe distinção entre as células CLP = 0 e CLP = 1, e assim modela-se apenas a transmissão de células.

Algumas variações podem ser feitas no modelo, tornando-o mais ou menos comportado, mas todas serão feitas em função do cbrPCR e do maxC. A variável cbrCPS (CPS – Células por segundo) será utilizada para representar uma taxa de transferência possível no modelo.

7.3.2 – Descrição do comportamento da rede CBR

Para o modelo específico de CBR a rede deve monitorar o tráfego e verificar se este está acima do PCR firmado no descritivo de tráfego da fonte. Células não conformes serão descartadas. No estabelecimento da conexão, se não houver recursos disponíveis para a demanda exigida pela fonte, a conexão será recusada.

7.3.3 - A representação do CBR

Basicamente são quatro os DTE que representam o modelo do CBR:

- Os estados da fonte;
- Os estados da rede;
- Os estados das mensagens da fonte para rede;
- Os estados das mensagens da rede para fonte.

Os estados das mensagens variam em função do estado da rede ou da fonte, e vice versa. Poder-se-ia descrever variações do estado da fonte ou da rede diretamente em função dos estados da rede ou da fonte respectivamente, porém optou-se pela explicitação dos estados de mensagens, descrevendo cada subsistema integrante do sistema maior de transmissão de dados. Com módulos descrevendo o comportamento das mensagens, fica mais fácil introduzir alterações, como por exemplo erros no canal de transmissão que levem a perda de mensagem.

7.3.4 – Os estados da fonte CBR

O DTE dos estados da fonte é semelhante ao mostrado no item 7.2.1, porém a representação de eventos está ligada às mensagens recebidas, bem como os estados influenciam nas mensagens enviadas. Na figura 7.4 pode-se ver que as mensagens enviadas pela rede, designadas como msgrf (msg = mensagem, r=rede, f=fonte → mensagem rede para fonte), tem influencia nos estados da fonte. Analogamente, dependendo do estado da fonte, poderá haver uma mudança do estado da mensagem da fonte para a rede.

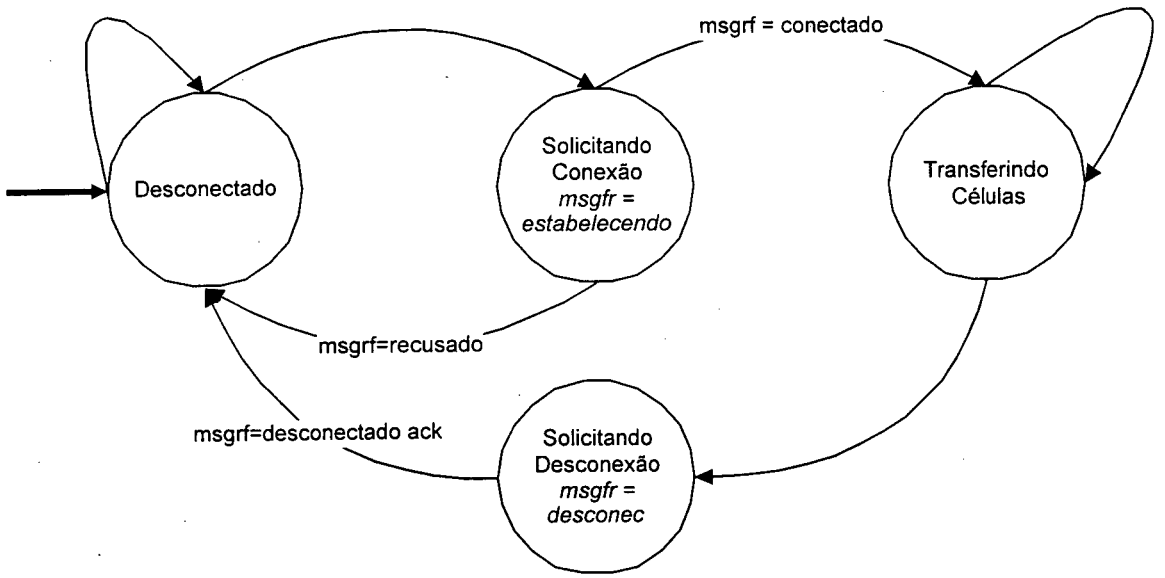


Figura 7.4 – O DTE da fonte CBR

Aparentemente desnecessário, o estado solicitando desconexão permite sincronizar os comportamentos da fonte e da rede, possibilitando que a rede esvazie os "buffers", se necessário, bem como interagir nas variáveis não mencionadas.

7.3.5 – Os estados da rede CBR

O DTE dos estados da rede é uma expansão do modelo mostrado no item 7.2.2, acrescentando estados necessários ao sincronismo com a fonte.

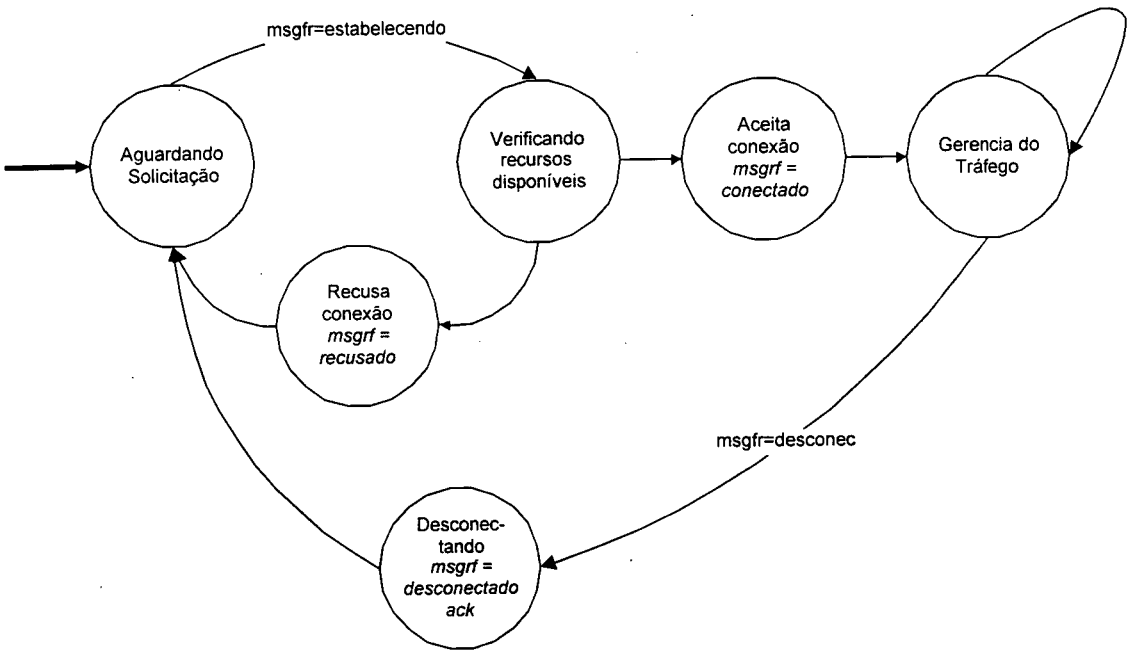


Figura 7.5 – O DTE da rede CBR

7.3.6 – Os estados das mensagens fonte→rede

A transição de estados da fonte causam a transição dos estados das mensagens enviadas da fonte para a rede conforme a figura 7.6.

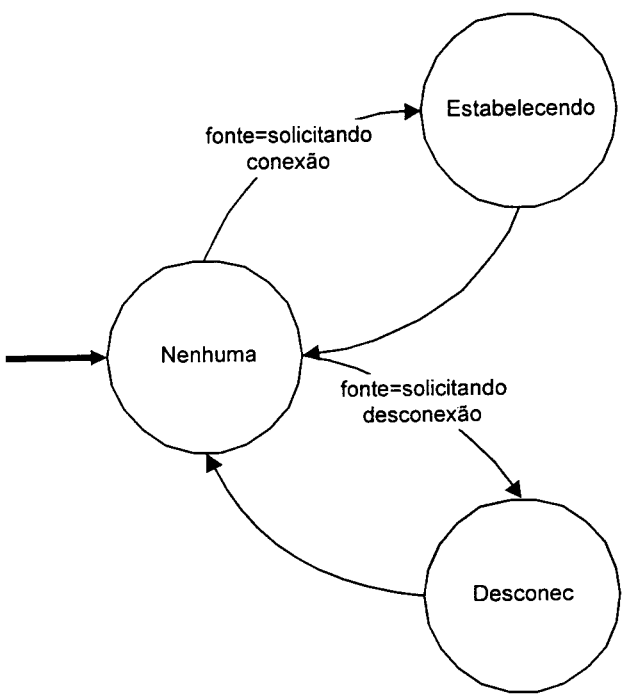


Figura 7.6 – O DTE das mensagens fonte→rede

7.3.7 – Os estados das mensagens rede→fonte

De forma análoga ao item anterior, a transição dos estados da rede causam a transição dos estados da mensagem enviada da rede para a fonte, conforme a figura 7.7.

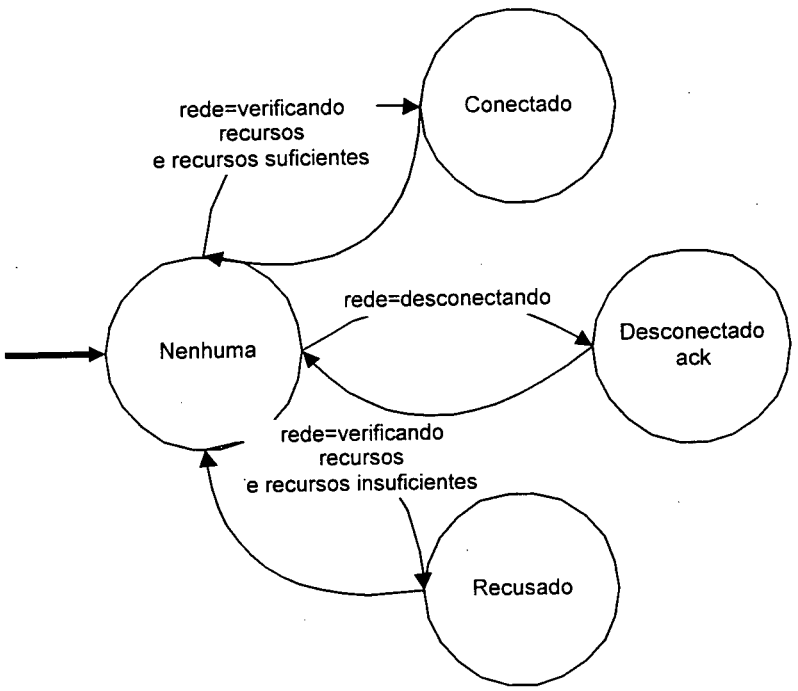


Figura 7.7 – O DTE das mensagens rede→fonte

7.3.8 – Utilizando a ferramenta para verificar o CBR

Para representar o comportamento da fonte CBR interagindo com a rede ATM foram utilizados três módulos interrelacionados: o módulo main (principal), que é obrigatório e que inicia os outros dois módulos, o da fonte e o da rede.

A figura 7.8 mostra a linguagem utilizada para alimentar a ferramenta SMV. Comentários foram adicionados ao longo do texto buscando facilitar o entendimento. O nome da maioria das variáveis utilizou a designação cbrXXX. Este expediente foi utilizado para que, quando diversos módulos de diversas redes forem reunidos, existam diferenças claras entre variáveis, por exemplo, PCR da rede CBR e PCR da rede VBR.

Existem diversos parâmetros que são trocados entre os módulos de rede e fonte, se destacando as mensagens msgcbrfr e msgcbrrf, isto é, mensagem (msg) da rede cbr enviada da fonte para rede (fr) ou da rede para fonte (rf).

As diversas variáveis trabalham dentro de intervalos arbitrados no item 7.2.8.

No modelo da figura 7.8, células transmitidas pela fonte acima do valor do QCR (Quantidade de células reservadas) negociada são, num primeiro momento armazenados no buffer, e, quando este é preenchido, elas são descartadas aumentando o CLR. A quantidade de células simultâneas reservadas pelo CBR é determinada por QCR (Quantidade de Células reservadas) e é igual ao PCR máximo da conexão. Da mesma forma que existe um número máximo de células que a fonte reservou, existe um número de buffers reservado, QBR – Quantidade de Buffers Reservados, que no estabelecimento da conexão é comparado com a quantidade máxima de buffers (maxB), e a rede permite ou não a conexão.

No modelo descrito na figura 7.8, a quantidade de células e buffers reservados não ultrapassa os valores máximos, e assim toda conexão solicitada deve ser estabelecida e sempre o CLR deve ser igual a zero. Toda solicitação de desconexão feita pela fonte também deve obter uma desconexão. Tais propriedades foram especificadas na figura 7.8 como:

```
conexao: assert G ((fcbr.estado=solconex)->F (fcbr.estado=txcel));
desconexao: assert G ((fcbr.estado=soldesc)->F (fcbr.estado=desc));
perda: assert G (rcbr.cbrCLR<1);
```

A especificação "conexao" afirma que sempre que o estado da fonte for igual a solicitando conexão (solconex) então sempre no futuro o sistema atingira um estado de transferindo células. Analogamente a especificação "desconexao" afirma que sempre que a fonte solicitar uma desconexão isto implica num estado futuro de desconexão. A especificação "perda" afirma que sempre o valor de CLR será menor do que 1, isto é, zero.

```
#define maxC 15
#define maxB 30

module main()
{

fcbr : fontecbr(msgcbrfr,msgcbrrf, cbrQCR, cbrQBR, cbrCPS);
rcbr : redecbr(msgcbrfr,msgcbrrf, cbrQCR, cbrQBR, cbrCPS, cbrBPS);


/* ------------------------Especificação---------------------------------- */
```

```
    conexao: assert G ((fcbr.estado=solconex)->F (fcbr.estado=txcel));
    desconexao: assert G ((fcbr.estado=soldesc)->F (fcbr.estado=desc));
    perda: assert G (rcbr.cbrCLR<1);


    }


    module fontecbr(msgcbrfr,msgcbrrf, cbrQCR, cbrQBR, cbrCPS)
    {

    /* ------------------------------------------------------------------------
    São quatro partes principais: Declaração, inicialização e especificações a
    serem testadas */
    /* No módulo da fonte cbr são 6 as variáveis: estado, mensagem da fonte para
rede, (msgcbrfr), taxa de pico de células(cbrPCR), quantidade de células reservadas
(cbrQCR) (relativo a banda reservada), células por segundo (cbrCPS) e quantidade de
buffers reservados (cbrQBR)
    ------------------------------------------------------------------------*/


    /* Declaração */

    estado:{desc, solconex, txcel, soldesc};
    msgcbrfr:{nenhuma, estabelecendo,desconec};
    cbrPCR:10..15;
    cbrQCR:0..15;
    cbrCPS:0..15;
    cbrQBR:0..30;


    /* ------------------------------Inicialização-----------------------------
    Nem todas as variáveis são inicializadas porque são definidas em função de
outras variáveis e não pode haver redefinição (Ver capítulo 7)
    ------------------------------------------------------------------------*/

    init(estado):=desc;
    init(cbrPCR):=10..15;

    /* ------------------------------------------------------------------------
    Evolução - Os comportamentos de cada variável são definidos aqui. Algumas
variáveis apresentam delay, isto é, são definidas para o próximo instante de tempo,
enquanto outras são definidas no mesmo instante em função de outras variáveis. É
importante sempre determinar qual(is) o(s) próximo(s) estado(s) possíveis de cada
variável, caso contrário a ferramenta associa a variável a um estado indefinido
    ------------------------------------------------------------------------*/

    next(estado):=
      switch(estado){
            desc:{desc, solconex};
            solconex:   (msgcbrrf=conectado)?txcel:   (msgcbrrf   =   recusado)?
desc:solconex;
            txcel:{txcel,soldesc};
            soldesc: (msgcbrrf=desconecack)?desc:soldesc;
            };

    msgcbrfr:=
      case{
      (estado = solconex): estabelecendo;
      (estado = soldesc) : desconec;
      default : nenhuma;
      };

    next(cbrPCR):=
      case{
            (estado=desc):10..15;
```

```
            default: cbrPCR;
      };

     if (estado=desc)
      {
      cbrQCR:=0;
      cbrQBR:=0;
      }
     else
      {
      cbrQCR:=cbrPCR;
      cbrQBR:=2*cbrPCR;
      }

     if (estado=txcel)
      cbrCPS:=cbrPCR;
     else
      cbrCPS:=0;

     }

     module redecbr(msgcbrfr, msgcbrrf, cbrQCR, cbrQBR, cbrCPS, cbrBPS)
     {

     /* ------------------------------------------------------------------------
     No módulo da rede cbr são 4 as variáveis: estado, mensagem da rede para fonte,
     (msgcbrrf), taxa de buffers por segundo (cbrBPS) e taxa de perda de células
(cbrCLR)
     ------------------------------------------------------------------------*/

     /* ---------------------------Declaração ----------------------------------*/

     estado:{agsol,verrec, aceitconex, recusconex, vertra, desc};
     msgcbrrf:{nenhuma, conectado, recusado, desconecack};
     cbrBPS:0..30;
     cbrCLR:0..15;

     /* ------------------------------Inicialização---------------------------- */

     init(estado):=agsol;
     init(cbrBPS):=0;

     /* -------------------------Evolução---------------------------------------*/

     next(estado):=
      switch(estado){
            agsol: (msgcbrfr=estabelecendo)? verrec : agsol;
            verrec : ((maxC >= cbrQCR)&(maxB >=cbrQBR)) ? aceitconex : recusconex;
            aceitconex:vertra;
            recusconex: agsol;
            vertra : ((msgcbrfr=desconec)&(cbrBPS=0))? desc : vertra ;
            desc: agsol;
      };


     msgcbrrf:=
      case{
      (estado=aceitconex) : conectado;
      (estado=recusconex) : recusado;
      (estado=desc):desconecack;
      default : nenhuma;
      };

     if (cbrCPS>=cbrQCR)
     {
      if ((cbrBPS+(cbrCPS-cbrQCR))>cbrQBR)
      {
```

```
            next(cbrBPS):=cbrQBR;
            cbrCLR:=cbrCPS-cbrQCR-(cbrQBR-cbrBPS);
        }
        else
        {
            next(cbrBPS):=cbrBPS+(cbrCPS-cbrQCR);
            cbrCLR:=0;
        }
    }
    else
    {
        if ((cbrQCR-cbrCPS)>=cbrBPS)
        {
            next(cbrBPS):=0;
            cbrCLR:=0;
        }
        else
        {
            next(cbrBPS):=(cbrBPS-(cbrQCR-cbrCPS));
            cbrCLR:=0;
        }
    }

    }
```

Figura 7.8 – A linguagem de entrada da categoria CBR para a ferramenta SMV

Utilizando a ferramenta SMV para verificar as especificações, obteve-se a confirmação de que todas são verdadeiras. A ferramenta indicou a utilização de 8 iterações e 42 estados alcançáveis e utilizou os recursos mostrados na figura 7.9.

```
Resources used
==============
user time.....................................................0.445773 s
system time...................................................0.459491 s
BDD nodes allocated.....................................................3928
data segment size.....................................................0
```

Figura 7.9 – Recursos usados para verificar o CBR utilizando especificação LTL

A ferramenta SMV da Cadence permite que se especifique em LTL e/ou CTL. Alterando as especificações "conexao", "desconexao" e "perda" de LTL para CTL, conforme mostrado na figura 7.10, obtêm-se o resultado mostrado na figura 7.11.

```
conexao: assert G ((fcbr.estado=solconex)->F (fcbr.estado=txcel));
desconexao: assert G ((fcbr.estado=soldesc)->F (fcbr.estado=desc));
perda: assert G (rcbr.cbrCLR<1);
```

```
SPEC AG (fcbr.estado=solconex)-> AG AF (fcbr.estado=txcel);
SPEC AG (fcbr.estado=soldesc)-> AG AF (fcbr.estado=desc);
SPEC AG AF (rcbr.cbrCLR<1);
```

Figura 7.10 – A mesma especificação para os filósofos escrita em LTL e CTL

```
Resources used
==============
user time.....................................................0.445773 s
system time...................................................0.459491 s
BDD nodes allocated.....................................................4098
data segment size.....................................................0
```

Figura 7.11 – Recursos usados para verificar o CBR utilizando especificação CTL

A ferramenta utilizou 5 iterações para verificação utilizando CTL, sendo que a quantidade de estados permaneceu em 42. Infelizmente, a versão do SMV da Cadence que roda para Windows 95, sempre informa o mesmo valor de tempo utilizado nas verificações, provavelmente por ser uma versão do software original e não poder, ou não possuir, verificação de tempo no Windows 95. Assim, não é possível a comparação deste indicador, e tal informação será omitida nos próximos indicativos. Entretanto foi sensível a diferença de tempo na verificação utilizando CTL contra o LTL.

A figura 7.8 descreveu uma fonte "bem comportada", isto é, ela respeita os valores negociados com a rede. Se for permitido que a fonte trafegue no mínimo uma célula acima de QCR então existe a possibilidade de uma conexão saturar o buffer e obrigar que a rede, através da característica UPC, faça o descarte de células.

Alterando a quantidade de Células por Segundo (CPS), permitido um valor acima de QCR, e rodando novamente a ferramenta obtêm-se o resultado falso para a afirmativa perda. Alterou-se a linguagem em três pontos:

- #define maxC 16 (Anteriormente era 15)

- cbrCPS:0..16; (Anteriormente era 0..15)
- cbrCPS:=cbrPCR+1; (Anteriormente era apenas cbrPCR)

A ferramenta indicou a utilização de 22 iterações e 336 estados alcançáveis e utilizou os recursos mostrados na figura 7.12.

```
Resources used
==============
BDD nodes allocated.....................................................7669
```

Figura 7.12 – Recursos usados para verificar uma fonte CBR "mal comportada" (LTL)

No caso do comportamento da categoria CBR o resultado era esperado. Se existir uma possibilidade de transmissão de uma célula acima da quantidade reservada então a manutenção deste estado saturará o buffer e ocasionará perda de células igual ao excedente do valor transmitido. A figura 7.13 mostra a evolução dos estados que levaram a especificação "perda" ser falsa.

Variáveis

Evolução dos estados

	1	2	3	4	5	6	7	8	9	10	11	12	13
cbrBPS	0	0	0	0	0	1	2	3	4	5	6	7	8
cbrCPS	0	0	0	0	11	11	11	11	11	11	11	11	11
cbrQBR	0	20	20	20	20	20	20	20	20	20	20	20	20
cbrQCR	0	10	10	10	10	10	10	10	10	10	10	10	10
fcbr.cbrPCR	10	10	10	10	10	10	10	10	10	10	10	10	10
fcbr.estado	desc	solconex	solconex	solconex	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel
msgcbrfr	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
msgcbrrf	nenhuma	nenhuma	nenhuma	conectac	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rcbr.cbrCLR	0	0	0	0	0	0	0	0	0	0	0	0	0
rcbr.estado	agsol	agsol	verrec	aceitcor	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra

	14	15	16	17	18	19	20	21	22	23	24	25
cbrBPS	9	10	11	12	13	14	15	16	17	18	19	20
cbrCPS	11	11	11	11	11	11	11	11	11	11	11	11
cbrQBR	20	20	20	20	20	20	20	20	20	20	20	20
cbrQCR	10	10	10	10	10	10	10	10	10	10	10	10
fcbr.cbrPCR	10	10	10	10	10	10	10	10	10	10	10	10
fcbr.estado	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel
msgcbrfr	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
msgcbrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rcbr.cbrCLR	0	0	0	0	0	0	0	0	0	0	0	1
rcbr.estado	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra

Figura 7.13 – Seqüência de variáveis que invalida a especificação perda

Os recursos utilizados neste caso são maiores do que as verificações anteriores, por exemplo, o número de estados que era 42 passou a 336.

Utilizando as especificações em CTL obtêm-se os resultados mostrados na figura 7.14.

```
Resources used
==============
BDD nodes allocated.....................................................9179
```

Figura 7.14 – Recursos usados para verificar uma fonte CBR "mal comportada" (CTL)

Utilizando CTL foram necessárias 23 iterações e 9.179 nós BDDs, com 336 estados alcançáveis. Aparentemente, para o sistema CBR, os recursos utilizados para verificar especificações CTL são muito próximos aos utilizados para verificar LTL. Acredita-se que a razão disto está na conversão de especificações LTL em CTL realizadas pela ferramenta [Mcm98c] e em aspectos internos e proprietários do algoritmo.

Alterando a especificação de perda para: "`perda: assert G (rcbr.cbrCLR<2);`" e verificando novamente as especificações verifica-se que todas são verdadeiras, isto é, no máximo o CLR é igual a 1. Os recursos utilizados são mostrados na figura 7.15.

Foram utilizadas 38 iterações, contra as 22 iterações para "`perda: assert G (rcbr.cbrCLR<1);`",já a quantidade de nós BDDs caiu consideravelmente, enquanto a quantidade de estados alcançáveis ficou em 336.

```
Resources used
==============
BDD nodes allocated.....................................................6205
```

Figura 7.15 – Recursos usados para verificar uma fonte CBR "mal comportada" (LTL)

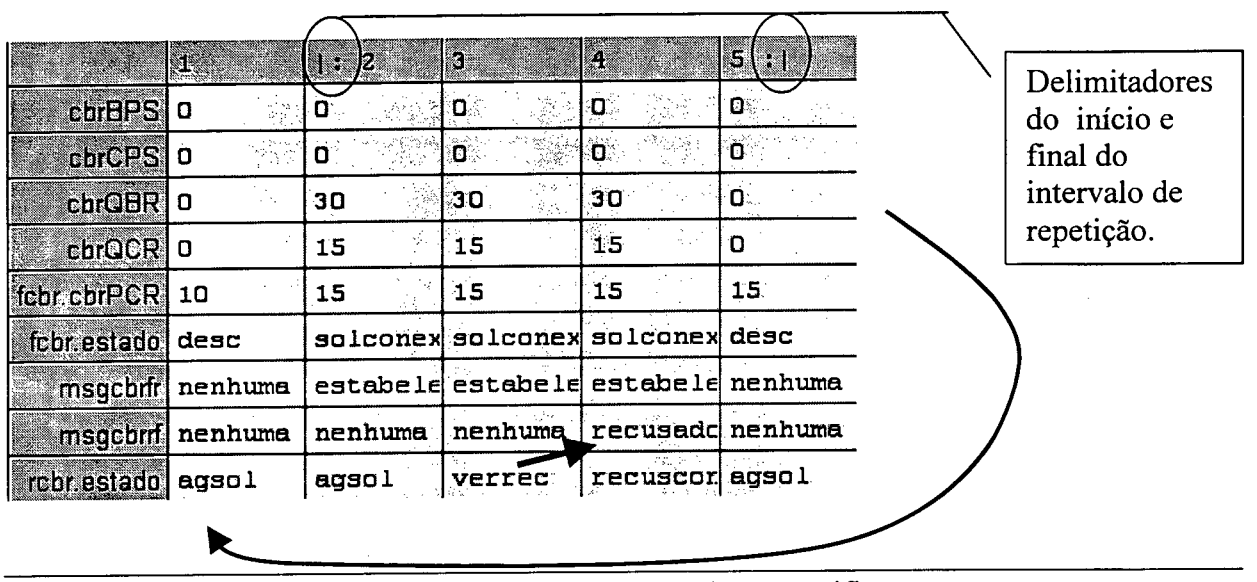
Os resultados para uma verificação semelhante utilizando o CTL foram: 4 iterações, 7.404 nós BDDs alocados e 336 estados alcançáveis

Tornando a fonte "mal comportada" verificou-se que uma especificação antes verdadeira tornou-se falsa. Da mesma forma, para os valores maxC e maxB da figura 7.8, a afirmação "conexao" é verdadeira. Entretanto, se forem reduzidos os valores de maxC e/ou maxB, e mantidos os outros valores da fonte comportada, verifica-se, através da ferramenta, que nem sempre uma solicitação de conexão leva a um estado de transferencia de células. Os

recursos utilizados neste caso, para o LTL são mostrados na figura 7.16. Na figura 7.17 é mostrado um dos conjunto de estados que torna a especificação falsa. Foram necessárias 7 iterações, mas pode-se verificar que a quantidade de nós BDDs alocados foi menor do que as verificações anteriores, provavelmente porque a ferramenta encerra a verificação quando encontra uma situação que contradiz alguma especificação. Neste caso, a especificação "perda" não foi verificada.

```
Resources used
==============
BDD nodes allocated.....................................................3860
```

Figura 7.16 – Recursos usados para verificar uma fonte CBR "mal comportada" (LTL)



	1	2	3	4	5
cbrBPS	0	0	0	0	0
cbrCPS	0	0	0	0	0
cbrQBR	0	30	30	30	0
cbrQCR	0	15	15	15	0
fcbr.cbrPCR	10	15	15	15	15
fcbr.estado	desc	solconex	solconex	solconex	desc
msgcbrfr	nenhuma	estabele	estabele	estabele	nenhuma
msgcbrrf	nenhuma	nenhuma	nenhuma	recusadc	nenhuma
rcbr.estado	agsol	agsol	verrec	recuscor	agsol

Figura 7.17 – Seqüência de variáveis que invalida a especificação conexão

A ferramenta SMV permite que se verifique todas as especificações ou apenas uma. Se uma especificação for falsa a ferramenta encerra, e apresenta o resultado para a especificação falsa, e informa que outras especificações não foram verificadas. Então, escolhendo a especificação "perda" e utilizando a ferramenta verifica-se que "perda" é verdadeiro, isto é, CLR<1 (Utilizou-se o modelo de fonte comportada para CPS). Os recursos utilizados são mostrados na figura 7.18. Foram necessárias 5 iterações. A perda sempre será igual a zero pois não há estabelecimento de conexão.

```
Resources used
==============
BDD nodes allocated.....................................................4180
```

Figura 7.18 – Recursos usados para verificar uma fonte CBR "mal comportada" (LTL)

Dependendo da complexidade da especificação a ser verificada, e da complexidade do modelo, o número de nós BDDs alocados varia. No modelo do CBR algumas variáveis podem ocupar diversos estados, como por exemplo CPS: 0..15;, o que contribui com 16 possíveis estados a serem compostos sincronamente com os demais estados. Como a ferramenta trabalha por exaustão quanto maior for o número de estados alcançáveis maior será a memória necessária para processamento, e maior será o tempo.

7.4 – O VBR

7.4.1 – Descrição do comportamento da fonte VBR

O comportamento do modelo da categoria de serviço VBR deve modelar o fluxo de células variável. O pico do tráfego de células deve ser menor ou igual ao PCR e no máximo o pico pode ser mantido conforme o valor do MBS. Por mais variável que seja, o fluxo das fontes VBR deve respeitar a média máxima SCR. Este parâmetro associado ao MBS permite que se conheça o fluxo da conexão com maior detalhe, alocando mais eficientemente os recursos.

No modelo adotar-se-á para cada conexão VBR alocação de células simultâneas igual ao SCR, sendo que os buffers serão adotados igual a MBS*(PCR-SCR).

Considerando que o ATM Forum [AF96] especifica que o MBS determina o que pode ser transmitido à taxa de pico, e que o SCR é medido em períodos maiores do que o PCR, utilizar-se-á média dos 5 últimos valores de CPS para cálculo do SCR.

A única distinção entre o rt-VBR e o nrt-VBR no modelo é a de que o MBS desta última categoria será maior do que o do rt-VBR.

7.4.2 – Descrição do comportamento da rede para o VBR

Para o modelo específico de VBR a rede deve monitorar o tráfego e verificar se está acima do PCR firmado no descritivo tráfego da fonte. Células não conformes serão eliminadas. Além disto é necessário manter um histórico das cinco últimas taxas e verificar se a média excede o SCR, se exceder as células que forem transmitidas acima do SCR serão descartadas.

No estabelecimento da conexão se não houverem recursos disponíveis para a demanda exigida pela fonte então a conexão será recusada.

7.4.3 – Representação dos DTE para o VBR

Os DTEs do estabelecimento de conexão, mensagens trocadas entre a fonte e rede e vice versa, são iguais aqueles mostrados para o CBR.

7.4.4 – Utilizando a ferramenta para verificar o VBR

Da mesma forma que a categoria CBR, a VBR utiliza três módulos interrelacionados: o módulo main (principal), que tem é obrigatório e que inicia os outros dois módulos, o da fonte e o da rede.

A figura 7.19 mostra a linguagem utilizada para alimentar a ferramenta SMV. O nome da maioria das variáveis utilizou a designação vbrXXX. Da mesma forma que no CBR existem diversos parâmetros que são trocados entre os módulos de rede e fonte, se destacando as mensagens msgvbrfr e msgvbrrd, isto é, mensagem (msg) da rede vbr enviada da fonte para rede (fr) ou da rede para fonte (rf).

```
#define maxC 10
#define maxB 30
module main()

{

fvbr : fontevbr(msgvbrfr,msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media);
rvbr : redevbr(msgvbrfr,msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media,vbrBPS);

/* ----------------------Especificação------------------------------------ */

conexao: assert G ((fvbr.estado=solconex)-> F (fvbr.estado=txcel));
desconexao: assert G ((fvbr.estado=soldesc)-> F (fvbr.estado=desc));
perda: assert G (rvbr.vbrCLR<1);

}

module fontevbr(msgvbrfr,msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media)
{

/* --------------------------Declaração-------------------------------------
São 12 variáveis: estado, mensagem da fonte para a rede, taxa de pico de
células (vbrPCR), taxa de pico médio (vbrSCR), rajada máxima (vbrMBS), quantidade
de células reservadas (vbrQCR - Banda), células por segundo (vbrCPS), quantidade de
buffers reservados (vbrQBR), armazenador dos últimos vbrCPS, média dos vbrCPS (para
comparar com o SCR), contador de vbrMBS e variação do vbrCPS.
---------------------------------------------------------------------------- */

estado:{desc, solconex, txcel, soldesc};
msgvbrfr:{nenhuma, estabelecendo,desconec};
vbrPCR:10..15;
vbrSCR:5..10;
vbrMBS: 3..6;
vbrQCR:0..15;
vbrCPS:0..15;
vbrQBR:0..30;
mediaCPS: array 0..4 of 0..15;
media:0..15;
contadorMBS:0..6;
delta:{-2,-1,0,1,2};

/* --------------------------Inicialização------------------------------- */

init(estado):=desc;
init(vbrPCR):=10..15;
init(vbrMBS):=3..6;
init(vbrCPS):=0;
for (i = 0; i<5; i = i+1){
  init(mediaCPS[i]):=0;
  }
init(contadorMBS):=0;

/* -------------------------Evolução ----------------------------------- */

next(estado):=
  switch(estado){
        desc:{desc, solconex};
        solconex:    (msgvbrrf=conectado)?txcel:    (msgvbrrf    =    recusado)?
desc:solconex;
        txcel:{txcel,soldesc};
        soldesc: (msgvbrrf=desconecack)?desc:soldesc;
        };

msgvbrfr:=
  case{
```

```
  (estado = solconex): estabelecendo;
  (estado = soldesc) : desconec;
  default : nenhuma;
  };

next(vbrPCR):=
  case{
        (estado=desc):10..15;
        default:vbrPCR;
  };

vbrSCR:=vbrPCR-5;

next(vbrMBS):=
  case {
        (estado = desc): 3..6;
        default: vbrMBS;
  };

if (estado=desc)
  {
  vbrQCR:=0;
  vbrQBR:=0;
  }
else
  {
  vbrQCR:=vbrSCR;
  vbrQBR:=vbrMBS*(vbrPCR-vbrSCR);
  }


if (estado=txcel)
  {
  next(vbrCPS):=
        case {
              (media > vbrSCR): vbrSCR-(media-vbrSCR);
              (contadorMBS-1=vbrMBS): vbrSCR;
              default : vbrCPS+delta;
        };
  }
  else
        next(vbrCPS):=0;

for (i = 0; i<4; i = i +1){
  next(mediaCPS[i]) := mediaCPS[(i+1)];
  }
next(mediaCPS[4]):=vbrCPS;

media:= ((mediaCPS[0]+mediaCPS[1]+mediaCPS[2]+mediaCPS[3]+mediaCPS[4])/5)+1;

next(contadorMBS):=
  case{
        (vbrCPS=vbrPCR) : contadorMBS+1;
        default : 0;
  };

delta:=
  case
  {
  (vbrCPS=vbrPCR): {-2,-1,0};
  (vbrCPS=0):{0,1,2};
  (vbrCPS=vbrPCR-1) : {-2,-1,0,1};
  (vbrCPS=1):{-1,0,1,2};
  default : {-2, -1,0, 1, 2};
  };

}
```

```
module redevbr(msgvbrfr, msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media, vbrBPS)
{

/* --------------------------Declaração-------------------------------------
São 4 variáveis: estado, mensagem da rede para a fonte, quantidade de buffers
utilizados por segundo (vbrBPS) e taxa de perda de células (vbrCLR).
---------------------------------------------------------------------------- */

estado:{agsol,verrec, aceitconex, recusconex, vertra, accor, desc};
msgvbrrf:{nenhuma, conectado, recusado, desconecack};
vbrBPS:0..30;
vbrCLR:0..15;

/* -------------------------Inicialização---------------------------------- */

init(estado):=agsol;
init(vbrBPS):=0;

/* ----------------------Evolução---------------------------------------- */

next(estado):=
  switch(estado){
        agsol: (msgvbrfr=estabelecendo)? verrec : agsol;
        verrec : ((maxC >= vbrQCR)& (maxB >= vbrQBR)) ? aceitconex :
recusconex;
        aceitconex:vertra;
        recusconex: agsol;
        vertra : ((msgvbrfr=desconec)&(vbrBPS=0))? desc : vertra ;
        desc: agsol;
  };

msgvbrrf:=
  case{
  (estado=aceitconex) : conectado;
  (estado=recusconex) : recusado;
  (estado=desc):desconecack;
  default : nenhuma;
  };

if (vbrCPS>=vbrQCR)
{
  if (media>vbrQCR)
  {
        next(vbrBPS):=vbrBPS;
        vbrCLR:=vbrCPS-vbrQCR;
  }
  else
  {
        if ((vbrBPS+(vbrCPS-vbrQCR))>vbrQBR)
        {
              next(vbrBPS):=vbrQBR;
              vbrCLR:=vbrCPS-vbrQCR-(vbrQBR-vbrBPS);
        }
        else
        {
              next(vbrBPS):=vbrBPS+(vbrCPS-vbrQCR);
              vbrCLR:=0;
        }
  }
}
else
{
  if ((vbrQCR-vbrCPS)>=vbrBPS)
  {
        next(vbrBPS):=0;
        vbrCLR:=0;
```

```
    }
    else
    {
            next(vbrBPS):=vbrBPS-(vbrQCR-vbrCPS);
            vbrCLR:=0;
    }
  }

  }
```

Figura 7.19 – A linguagem de entrada da categoria VBR para a ferramenta SMV

A fonte VBR descrita na figura 7.19 busca estabelecer conexões com a rede e pode transmitir células dentro do intervalo máximo determinado por PCR, entretanto, diferentemente do CBR, que pode manter o tráfego no máximo, o VBR deve manter no máximo a média acordada de SCR. Além disto os picos de PCR não podem exceder o valor determinado por MBS. Com este comportamento pode-se alocar menor quantidade de recursos para a categoria VBR do que os alocados para o CBR e tirar maior proveito da característica variável da fonte.

No modelo da categoria CBR, descrito no item 7.3, a função do buffer foi utilizada para as fontes que não respeitavam os limites acordados, e mesmo assim com a função de absorver temporariamente o excesso do tráfego acima do QCR, pois como existe a possibilidade da fonte manter a transmissão acima do acordado, o buffer finito era esgotado. Já no modelo de VBR, devido as variações naturais desta categoria, o buffer é utilizado mesmo para fontes "bem comportadas".

A vazão de células do modelo VBR é igual a SCR, que é a média máxima acordada. Temporariamente células podem ser transmitidas acima de SCR e serão armazenadas no buffer, que posteriormente será esvaziado quando o tráfego for abaixo de SCR para manter a média. Na modelagem a conexão pode estabelecer um PCR de no máximo 15 células simultâneas, e o tráfego oscilará dentro da faixa de 0 a 15 (quando conectado) variando no máximo 2 células a mais ou a menos.

Tanto a fonte "bem comportada" como a rede devem ter a estrutura de memória, a primeira para poder trafegar células dentro dos parâmetros contratados com a rede, o que é necessário para um baixo CLR, e a segunda para poder executar o controle do tráfego de células (função UPC), utilizando este parâmetro para eventuais descartes.

O sistema com a memória implementada demanda uma quantidade de recursos grande da ferramenta de verificação. Definido como "`mediaCPS: array 0..4 of 0..15;`"

a memória introduz 80 estados alcançáveis que farão a composição síncrona com os estados das demais variáveis. Por isto, e uma vez que tal estrutura já se encontra no modelo da fonte, o comportamento da rede utilizará o valor calculado naquela.

Utilizar o mesmo sistema de memória da fonte foi adotado pois num teste com um sistema com memória, tanto na fonte como na rede, não obteve-se resposta da ferramenta, uma vez que a quantidade de recursos necessários na 15 iteração extrapolou os recursos do hardware no qual o SMV foi utilizado. (Utilizou-se um Pentium II – 300MHz com 64 Mbytes de RAM e 4.7 Gbytes de Winchester, sendo que 1 Gigabyte foi disponibilizado para utilização pelo Windows como memória).

No modelo da figura 7.19, células transmitidas pela fonte acima do valor do QCR (Quantidade de células reservadas) negociado são, num primeiro momento, armazenadas no buffer. Como o QCR é igual a SCR, se a fonte respeitar a média então quando o tráfego for menor do que SCR os buffers serão esvaziados. Porém, se a fonte trafegar acima da média SCR e/ou se o buffer reservado não conseguir armazenar as variações entorno da média, então este é preenchido e as células serão descartadas aumentando o CLR. Em cada solicitação de conexão, o QCR solicitado é comparado com a quantidade máxima de células simultâneas (maxC), e a rede permite ou não a conexão.

Da mesma forma que existe um número máximo de células que a fonte reservou, existe um número de buffers reservado, QBR – Quantidade de Buffers Reservados, que no estabelecimento da conexão é comparado com a quantidade máxima de buffers (maxB), e a rede permite ou não a conexão.

A ferramenta não conseguiu verificar o modelo descrito na figura 7.19, pois extrapolou os recursos de memória. Assim, foi necessária a adoção de uma estratégia para reduzir a quantidade de estados do modelo, fixando determinadas variáveis como PCR, SCR, MBS. Com o novo modelo mostrado na figura 7.20 a verificação tornou-se possível.

```
#define maxC 10
#define maxB 30
module main()

{

fvbr : fontevbr(msgvbrfr,msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media);
rvbr : redevbr(msgvbrfr,msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media,vbrBPS);

/* ----------------------Especificação---------------------------------- */

conexao: assert G ((fvbr.estado=solconex)-> F (fvbr.estado=txcel));
desconexao: assert G ((fvbr.estado=soldesc)-> F (fvbr.estado=desc));
perda: assert G (rvbr.vbrCLR<1);
```

```
}

module fontevbr(msgvbrfr,msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media)
{

/* -------------------------Declaração-----------------------------------------
São 12 variáveis: estado, mensagem da fonte para a rede, taxa de pico de
células (vbrPCR), taxa de pico médio (vbrSCR), rajada máxima (vbrMBS), quantidade
de células reservadas (vbrQCR - Banda), células por segundo (vbrCPS), quantidade de
buffers reservados (vbrQBR), armazenador dos últimos vbrCPS, média dos vbrCPS (para
comparar com o SCR), contador de vbrMBS e variação do vbrCPS.
------------------------------------------------------------------------------ */

estado:{desc, solconex, txcel, soldesc};
msgvbrfr:{nenhuma, estabelecendo,desconec};
vbrPCR:{15};
vbrSCR:{10};
vbrMBS: {3};
vbrQCR:{0,10};
vbrCPS:0..15;
vbrQBR:{0,15};
mediaCPS: array 0..4 of 0..15;
media:0..15;
contadorMBS:0..3;
delta:{-2,-1,0,1,2};

/* ------------------------Inicialização---------------------------------- */

init(estado):=desc;
init(vbrPCR):=15;
init(vbrMBS):=3;
init(vbrCPS):=0;
for (i = 0; i<5; i = i+1){
  init(mediaCPS[i]):=0;
  }
init(contadorMBS):=0;

/* ------------------------Evolução ---------------------------------------- */

next(estado):=
  switch(estado){
        desc:{desc, solconex};
        solconex:    (msgvbrrf=conectado)?txcel:    (msgvbrrf    =    recusado)?
desc:solconex;
        txcel:{txcel,soldesc};
        soldesc: (msgvbrrf=desconecack)?desc:soldesc;
        };

msgvbrfr:=
  case{
  (estado = solconex): estabelecendo;
  (estado = soldesc) : desconec;
  default : nenhuma;
  };

next(vbrPCR):=15;

vbrSCR:=vbrPCR-5;

next(vbrMBS):=3;

if (estado=desc)
  {
  vbrQCR:=0;
  vbrQBR:=0;
  }
```

```
      else
        {
        vbrQCR:=vbrSCR;
        vbrQBR:=vbrMBS*(vbrPCR-vbrSCR);
        }

      if (estado=txcel)
        {
        next(vbrCPS):=
              case {
                    (media > vbrSCR): vbrSCR-(media-vbrSCR);
                    (contadorMBS-1=vbrMBS): vbrSCR;
                    default : vbrCPS+delta;
              };
        }
        else
              next(vbrCPS):=0;

      for (i = 0; i<4; i = i +1){
        next(mediaCPS[i]) := mediaCPS[(i+1)];
        }
      next(mediaCPS[4]):=vbrCPS;

      media:= ((mediaCPS[0]+mediaCPS[1]+mediaCPS[2]+mediaCPS[3]+mediaCPS[4])/5)+1;

      next(contadorMBS):=
        case{
              (vbrCPS=vbrPCR) : contadorMBS+1;
              default : 0;
        };

      delta:=
        case
        {
        (vbrCPS=vbrPCR): {-2,-1,0};
        (vbrCPS=0):{0,1,2};
        (vbrCPS=vbrPCR-1) : {-2,-1,0,1};
        (vbrCPS=1):{-1,0,1,2};
        default : {-2, -1,0, 1, 2};
        };

      }

      module redevbr(msgvbrfr, msgvbrrf, vbrQCR, vbrQBR, vbrCPS, media, vbrBPS)
      {

      /* -------------------------Declaração-----------------------------------------
      São 4 variáveis: estado, mensagem da rede para a fonte, quantidade de buffers
utilizados por segundo (vbrBPS) e taxa de perda de células (vbrCLR).
      ------------------------------------------------------------------------------ */

      estado:{agsol,verrec, aceitconex, recusconex, vertra, accor, desc};
      msgvbrrf:{nenhuma, conectado, recusado, desconecack};
      vbrBPS:0..30;
      vbrCLR:0..15;

      /* ------------------------Inicialização---------------------------------- */

      init(estado):=agsol;
      init(vbrBPS):=0;

      /* ------------------------Evolução----------------------------------------- */

      next(estado):=
        switch(estado){
              agsol: (msgvbrfr=estabelecendo)? verrec : agsol;
```

```
            verrec  :  ((maxC  >=  vbrQCR)&  (maxB  >=  vbrQBR))  ?  aceitconex  :
recusconex;
            aceitconex:vertra;
            recusconex: agsol;
            vertra : ((msgvbrfr=desconec)&(vbrBPS=0))? desc : vertra ;
            desc: agsol;
      };

    msgvbrrf:=
      case{
      (estado=verrec)&(maxC >= vbrQCR)&(maxB >= vbrQBR) : conectado;
      (estado=recusconex) : recusado;
      (estado=desc):desconecack;
      default : nenhuma;
      };

    if (vbrCPS>=vbrQCR)
    {
      if (media>vbrQCR)
      {
            next(vbrBPS):=vbrBPS;
            vbrCLR:=vbrCPS-vbrQCR;
      }
      else
      {
            if ((vbrBPS+(vbrCPS-vbrQCR))>vbrQBR)
            {
                  next(vbrBPS):=vbrQBR;
                  vbrCLR:=vbrCPS-vbrQCR-(vbrQBR-vbrBPS);
            }
            else
            {
                  next(vbrBPS):=vbrBPS+(vbrCPS-vbrQCR);
                  vbrCLR:=0;
            }
      }
    }
    else
    {
      if ((vbrQCR-vbrCPS)>=vbrBPS)
      {
            next(vbrBPS):=0;
            vbrCLR:=0;
      }
      else
      {
            next(vbrBPS):=vbrBPS-(vbrQCR-vbrCPS);
            vbrCLR:=0;
      }
    }

    }
```

Figura 7.20 – A linguagem de entrada da categoria VBR com fixação de variáveis

Verificando as especificações apresentadas na figura 7.20 verificou-se que a especificação "perda" é falsa, enquanto as demais são verdadeiras. Foram necessárias 112 iterações num total de 203.373 estados alcançáveis. Os nós BDDs alocados são mostrados na figura 7.21.

Mesmo com a fixação de variáveis a quantidade de recursos e estados alcançáveis necessários para verificação ficaram muito acima das grandezas trabalhadas no CBR.

```
Resources used
==============
BDD nodes allocated.....................................................182196
```

Figura 7.21 – Recursos para verificar VBR com fonte "bem comportada"

Como a ferramenta encerra quando verifica se uma especificação é falsa, os recursos consumidos para esta verificação são menores do que a especificação fosse verificada mais "profundamente".

Alterando a especificação para "`perda: assert G (vbrCLR<6);`" e utilizando a ferramenta verifica-se que a especificação é verdadeira. Foram necessárias 115 iterações e 207.074 nós BDDs, conforme mostra a figura 7.22, num conjunto de 203.373 estados alcançáveis.

```
Resources used
==============
BDD nodes allocated.....................................................207074
```

Figura 7.22 – Recursos para verificar VBR com fonte "bem comportada"

Além da redução de fixação dos intervalos poder-se-ia ter utilizado outras estratégias para reduzir o tamanho do universo de estados. Uma solução seria isolar o comportamento de descarte de células, considerando outros comportamentos do sistema como fixos dentro da pior possibilidade para o CLR. Por exemplo, construir-se-ia um modelo considerando o sistema já conectado e transferindo o máximo de células possível. A consideração das situações de pior possibilidade é extremamente delicada, uma vez que não há garantia que o projetista esteja fazendo a escolha pela pior opção realmente.

Reduzindo o número CLR da especificação "`perda: assert G (vbrCLR<6);`" de 6 para 0, buscando o maior valor que torna falsa a especificação verifica-se que existe a possibilidade do modelo ter CLR=5. Mas se a fonte é "bem comportada" porque existe a possibilidade de uma perda tão alta? Afinal, perder 5 unidades de célula num modelo que tem

no máximo 15 células simultâneas eqüivale a, em determinados instantes, a perder 33,33% das células transmitidas.

Analisando os estados que levaram a está perda, conforme mostrado na figura 7.23, conclui-se que a estratégia de comportamento da fonte e da rede não estão conseguindo controlar o tráfego de acordo com o adequado. Da forma que está modelado, todo o tráfego que ultrapassa a média é descartado. Como a ação da fonte sobre o tráfego gerado é comedida, quando a fonte fica durante alguns instantes com um baixo tráfego a média de tráfego dos últimos 5 instantes de tempo é baixa ou igual a zero, assim, nos próximos instantes a fonte pode transmitir no máximo (dentro do limite de MBS), e quando a média ultrapassa o valor máximo a taxa de transmissão está alta e a rede descarta este excesso. Existem algumas estratégias para alterar este comportamento do modelo. Uma estratégia seria eliminar o controle de tráfego da rede quanto à média SCR, assim, o buffer poderia reduzir a perda no transitório, entretanto, fontes "não comportadas" que transmitissem acima da média esgotariam o buffer e acabariam gerando descarte de células, além de comprometer outras categorias de serviço no caso de recursos compartilhados. Outra estratégia seria a de implantar uma tolerância no descarte de células da rede. A documentação do ATM Forum [AF96] várias vezes fala de tolerâncias, justamente para amenizar o rigor dos parâmetros permitindo algum desvio.

	1	2	3	4	5	6	7	8	9	10	11	12	13
fvbr.contadorMBS	0	0	0	0	0	0	0	0	0	0	0	0	0
fvbr.delta	0	0	0	0	2	2	2	2	2	2	2	1	0
fvbr.estado	desc	solconex	solconex	solconex	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	soldesc
fvbr.mediaCPS[0]	0	0	0	0	0	0	0	0	0	0	2	4	6
fvbr.mediaCPS[1]	0	0	0	0	0	0	0	0	0	2	4	6	8
fvbr.mediaCPS[2]	0	0	0	0	0	0	0	0	2	4	6	8	10
fvbr.mediaCPS[3]	0	0	0	0	0	0	0	2	4	6	8	10	12
fvbr.mediaCPS[4]	0	0	0	0	0	0	2	4	6	8	10	12	14
fvbr.vbrMBS	3	3	3	3	3	3	3	3	3	3	3	3	3
fvbr.vbrPCR	15	15	15	15	15	15	15	15	15	15	15	15	15
fvbr.vbrSCR	10	10	10	10	10	10	10	10	10	10	10	10	10
media	1	1	1	1	1	1	1	2	3	5	7	9	11
msgvbrf	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	desconec
msgvbrf	nenhuma	nenhuma	nenhuma	conectac	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
fvbr.estado	agsol	agsol	verrec	aceitcor	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra
fvbr.vbrCLR	0	0	0	0	0	0	0	0	0	0	0	0	5
vbrBPS	0	0	0	0	0	0	0	0	0	0	0	2	6
vbrCPS	0	0	0	0	0	2	4	6	8	10	12	14	15
vbrQBR	0	15	15	15	15	15	15	15	15	15	15	15	15
vbrQCR	0	10	10	10	10	10	10	10	10	10	10	10	10

Figura 7.23 – Seqüência de variáveis que invalida a especificação "perda"

Se fosse eliminado o controle de tráfego da rede sobre a média SCR, de acordo com os parâmetros do modelo, ainda existiria CLR máximo de 5. O problema é que da forma que está implementado a fonte pode transmitir bem acima da média, por períodos curtos, retornando para um valor levemente abaixo da média até ser possível novamente transmitir no máximo. Como a média tem a memória apenas dos últimos 5 instantes o controle não funciona pois na verdade a fonte está transmitindo numa média acima de SCR, saturando o buffer, independente do tamanho, conforme mostrado nas figuras 7.24. e 7.25.

media	1	1	1	1	1	1	2	3	4	6	8	9
msgvbrfr	nenhuma	estabele	estabele	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
msgvbrrf	nenhuma	nenhuma	conectad	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rvbr.estado	agsol	agsol	verrec	aceitcor	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra
rvbr.vbrCLR	0	0	0	0	0	0	0	0	0	0	0	0
vbrBPS	0	0	0	0	0	0	0	0	0	0	1	3
vbrCPS	0	0	0	0	2	4	5	7	9	11	12	10

media	10	11	10	10	9	10	11	13	13	12	11	10
msgvbrfr	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
msgvbrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rvbr.estado	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra
rvbr.vbrCLR	0	0	0	0	0	0	0	0	0	0	0	0
vbrBPS	3	1	0	0	1	4	9	14	13	10	7	5
vbrCPS	8	6	9	11	13	15	15	9	7	7	8	9

media	9	9	10	11	12	12	11	10	10	10	10	12
msgvbrfr	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	desconec
msgvbrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rvbr.estado	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra
rvbr.vbrCLR	0	0	0	0	0	0	0	0	0	0	0	5
vbrBPS	4	5	8	10	12	11	9	7	6	7	10	15
vbrCPS	11	13	12	12	9	8	8	9	11	13	15	15

Figura 7.24 – Seqüência de variáveis que invalida a especificação "perda"

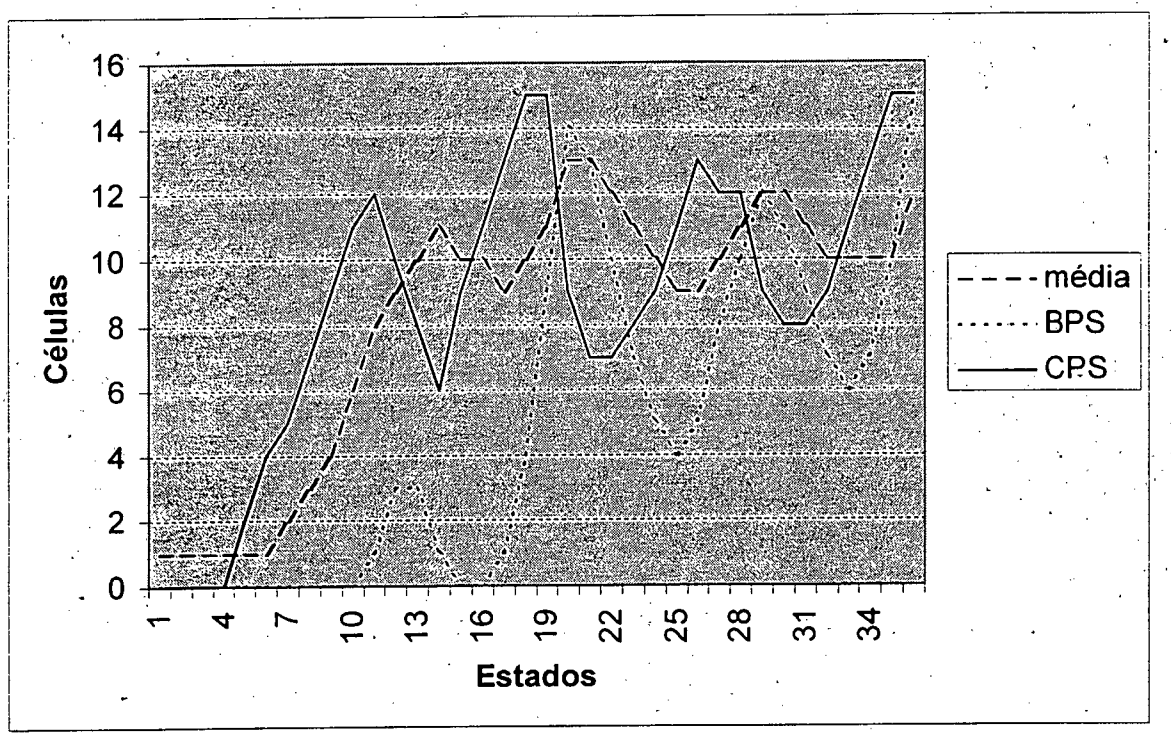


Figura 7.25 – Variação de CPS, BPS e média na categoria VBR sem controle de SCR

Uma solução mais adequada seria controlar melhor o tráfego da fonte, de forma a não permitir, por exemplo, que as variações nas taxas de transmissão fossem tão bruscas acima de SCR. Para isto, uma opção é reduzir o tamanho da memória da fonte dos CPS.

Incluí-se na declaração a variável `mediafonte: 0..15;` , que é igual a `mediafonte:=((mediaCPS[3]+mediaCPS[4])/2)+1;` e altera-se:

```
next(vbrCPS):=
        case {
                (media > vbrSCR): vbrSCR-(media-vbrSCR);
```

para:

```
next(vbrCPS):=
        case {
                (mediafonte > vbrSCR): vbrSCR-(media-vbrSCR);
```

Reduzindo o número de memórias da fonte últimos 5 para 2 CPS tem-se que a CLR é menor do que 5. Reduzindo o CLR da especificação para menor do que 4 verifica-se que a especificação é falsa, isto é, pode haver perda de 4 células. A razão é a mesma apresentada anteriormente, a única melhoria que houve foi uma maior velocidade na redução da média da fonte, o que reduziu a perda máxima de 5 para 4.

É necessário que a ação sobre a média seja mais rápida por parte da fonte para que a rede não descarte células. Assim, se quando houvesse violação da média fosse tomada uma medida severa, como por exemplo baixar o CPS para zero, o problema tenderia a desaparecer.

Implementando tal estratégia na figura 7.18 muda-se a estrutura:

```
next(vbrCPS):=
        case {
                (media > vbrSCR): vbrSCR-(media-vbrSCR);
```

para:

```
next(vbrCPS):=
        case {
                (media > vbrSCR): 0;
```

Entretanto a estratégia não funcionou, pois como utilizou-se a média da fonte semelhante a da rede, não houve tempo do corte radical no CPS prever o descarte de células.

Associando as duas estratégias busca-se resolver o problema. Verificando CLR<4 obtêm-se que a especificação continua falsa. Existe uma possibilidade de variar o CPS de forma que a média da fonte fica no valor de SCR enquanto a média da rede sobe até que no instante final tanto a média da fonte como a da rede sobem acima do valor de SCR ocasionando o descarte, conforme mostra a figura 7.26.

	4	5	6	7	8	9	10	11	12	13	14	15
fvbr.contadorMBS	0	0	0	0	0	0	0	0	0	0	0	0
fvbr.delta	0	2	1	2	2	2	1	-1	1	2	2	0
fvbr.estado	solconex	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	soldesc
fvbr.mediaCPS[0]	0	0	0	0	0	0	0	2	3	5	7	9
fvbr.mediaCPS[1]	0	0	0	0	0	0	2	3	5	7	9	10
fvbr.mediaCPS[2]	0	0	0	0	0	2	3	5	7	9	10	9
fvbr.mediaCPS[3]	0	0	0	0	2	3	5	7	9	10	9	10
fvbr.mediaCPS[4]	0	0	0	2	3	5	7	9	10	9	10	12
fvbr.mediafonte	1	1	1	2	3	5	7	9	10	10	10	12
fvbr.vbrMBS	3	3	3	3	3	3	3	3	3	3	3	3
fvbr.vbrPCR	15	15	15	15	15	15	15	15	15	15	15	15
fvbr.vbrSCR	10	10	10	10	10	10	10	10	10	10	10	10
media	1	1	1	1	2	3	4	6	7	9	10	11
msgvbrr	estabele	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	desconec
msgvbrf	conectac	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rvbr.estado	aceitcor	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra
rvbr.vbrCLR	0	0	0	0	0	0	0	0	0	0	0	4
vbrBPS	0	0	0	0	0	0	0	0	0	0	0	2
vbrCPS	0	0	2	3	5	7	9	10	9	10	12	14
vbrQBR	15	15	15	15	15	15	15	15	15	15	15	15

Figura 7.26 – Seqüência de variáveis que invalida a especificação "perda"

Para resolver este problema poder-se-ia implantar uma estratégia de buffer no dispositivo da fonte, e um mecanismo que simulasse o descarte de células da rede para então tornar o tráfego da fonte mais estável, ou aumentar o valor da média da fonte obrigando a uma ação antes de violar a média da rede. Adotando esta última estratégia altera-se

```
next(vbrCPS):=
        case {
                (media > vbrSCR): vbrSCR-(media-vbrSCR);
```

para:

```
next(vbrCPS):=
        case {
                ((mediafonte+1) > vbrSCR): 0;
```

E finalmente obtêm-se CLR< 1.

7.5 – O ABR

7.5.1 – Descrição do comportamento da fonte ABR

O modelo da fonte considera os procedimentos descritos no capitulo 6. Existe uma definição de PCR e MCR, sendo que o MCR pode ser igual a zero.

7.5.2 - Descrição do comportamento da rede para o UBR

O modelo da fonte considera os procedimentos descritos no capitulo 6.

7.5.3 – Representação dos DTE para o VBR

Os DTEs do estabelecimento de conexão, mensagens trocadas entre a fonte e rede e vice versa, são iguais aqueles mostrados para o CBR e o VBR.

7.5.4 – Utilizando a ferramenta para verificar o ABR

O ABR utiliza quatro módulos interrelacionados: o módulo main (principal), que tem é obrigatório e que inicia os outros três módulos, o da fonte, o da rede, e aquele que simula variação de recursos da rede.

A figura 7.27 mostra a linguagem utilizada para alimentar a ferramenta SMV. O nome da maioria das variáveis utilizou a designação abrXXX. Da mesma forma que no CBR e no VBR existem diversos parâmetros que são trocados entre os módulos de rede e fonte, se destacando as mensagens msgabrfr e msgabrrf, isto é, mensagem (msg) da rede abr enviada da fonte para rede (fr) ou da rede para fonte (rf).

```
     #define maxC 100
     #define maxB 100
     #define RIF 1
     #define RDF 1
     #define CRM 3
     #define CDF 3
     #define ICR 5

     module main()
     {

     fabr  :  fonteabr(msgabrfr,msgabrrf,  abrQCR,  abrQBR,  abrACR,  abrDIR,  abrCI,
abrNI);
     rabr  :  redeabr(msgabrfr,msgabrrf,  abrQCR,  abrQBR,  abrACR,  abrDIR,  abrCI,
abrNI, CD, BD);
     rtot : redetot(CD,BD);

     /* ------------------------Especificação---------------------------------- */

     conexao: assert G ((fabr.estado=solconex)->F (fabr.estado=txcel));
     desconexao: assert G ((fabr.estado=solconex)->F (fabr.estado=txcel));
     perda: assert G (rabr.abrCLR<1);
     maisquedez: assert G (fabr.abrACR<11);
     }

     module  fonteabr(msgabrfr,msgabrrf,  abrQCR,  abrQBR,  abrACR,  abrDIR,  abrCI,
abrNI)
     {

     /* ------------------------------------------------------------------------
     No módulo da fonte cbr são 6 as variáveis: estado, mensagem da fonte para
rede, (msgcbrfr), taxa de pico de células(CBRPCR), quantidade de células reservadas
(CBRQCR) (relativo a banda reservada), células por segundo (CBRCPS) e quantidade de
buffers reservados (CBRQBR)
     ------------------------------------------------------------------------*/

     /* Declaração */
```

```
      estado:{desc, solconex, txcel, soldesc};
      msgabrfr: {nenhuma, estabelecendo,desconec};
      abrPCR:10..15;
      abrMCR:0..5;
      abrQCR:0..5;
      abrACR:0..15;
      abrQBR:0..10;
      contCRM:0..3;


      /* ----------------------------Inicialização-----------------------------
      Nem todas as variáveis são inicializadas porque são definidas em função de
outras  variáveis e não pode haver redefinição (Ver capítulo 7)
      ----------------------------------------------------------------------*/

      init(estado):=desc;
      init(abrPCR):=10..15;
      init(abrMCR):=0..5;
      init(abrACR):=ICR;
      init(contCRM):=0;

      /* ----------------------------------------------------------------------
      Evolução - Os comportamentos de cada variável são definidos aqui. Algumas
variáveis apresentam delay, isto é, são definidas para o próximo instante de tempo,
enquanto outras são definidas no mesmo instante em função de outras variáveis. É
importante sempre determinar qual(is) o(s) próximo(s) estado(s) possíveis de cada
variável, caso contrário a ferramenta associa a variável a um estado indefinido
      ----------------------------------------------------------------------*/

      next(estado):=
        switch(estado){
              desc:{desc, solconex};
              solconex:    (msgabrrf=conectado)?txcel:    (msgabrrf    =    recusado)?
desc:solconex;
              txcel:{txcel,soldesc};
              soldesc: (msgabrrf=desconecack)?desc:soldesc;
              };

      msgabrfr:=
        case{
        (estado = solconex): estabelecendo;
        (estado = soldesc) : desconec;
        default : nenhuma;
        };

      next(abrPCR):=
        case{
              (estado=desc):10..15;
              default: abrPCR;
        };

      next(abrMCR):=
        case{
              (estado=desc):0..5;
              default: abrMCR;
        };

      if (estado=desc)
        {
        abrQCR:=0;
        abrQBR:=0;
        }
      else
        {
        abrQCR:=abrMCR;
        abrQBR:=2*abrMCR;
        }
```

```
     next(contCRM):=
       case{
              (abrDIR=0)&(contCRM<CRM):contCRM+1;
              (abrDIR=1)&(abrCI=1):(contCRM<CRM)? contCRM+1:contCRM;
              (abrDIR=1)&(abrCI=0):0;
              (abrDIR=0)&(contCRM=CRM):contCRM;
       };

     if (estado=txcel)
     {
       next(abrACR):=
              case{
                     (contCRM=CRM):((abrACR-CDF)<=abrMCR) ? abrMCR : (abrACR-CDF);
                     (abrCI=1):((abrACR-RDF)<=abrMCR) ? abrMCR : (abrACR-RDF);
                     ((abrCI=0)&(abrNI=1)):abrACR;
                     ((abrCI=0)&(abrNI=0)):((abrACR+RIF)>=abrPCR)    ?    abrPCR    :
(abrACR+RIF);
              };
     }
     else
       next(abrACR):=0;
     }

     module  redeabr(msgabrfr,msgabrrf,  abrQCR,  abrQBR,  abrACR,  abrDIR,  abrCI,
abrNI, CD, BD)
     {

     /* ------------------------------------------------------------------------
     No módulo da rede cbr são 4 as variáveis: estado, mensagem da rede para fonte,
     (msgcbrrf), taxa de buffers por segundo (CBRBPS) e taxa de perda de células
(CBRCLR)
     ------------------------------------------------------------------------*/

     /* ----------------------Declaração ----------------------------------*/

     estado:{agsol,verrec, aceitconex, recusconex, vertra, desc};
     msgabrrf:{nenhuma, conectado, recusado, desconecack};
     abrBPS:0..30;
     abrCLR:0..15;

     abrDIR:boolean;
     abrCI:boolean;
     abrNI:boolean;

     /* ---------------------Inicialização---------------------------------- */

     init(estado):=agsol;
     init(abrBPS):=0;
     init(abrDIR):=0;
     init(abrCI):=0;
     init(abrNI):=0;

     /* -------------------------Evolução---------------------------------------*/

     next(estado):=
       switch(estado){
              agsol: (msgabrfr=estabelecendo)? verrec : agsol;
              verrec : ((maxC >= abrQCR)&(maxB >=abrQBR)) ? aceitconex : recusconex;
              aceitconex:vertra;
              recusconex: agsol;
              vertra : ((msgabrfr = desconec)&(abrBPS=0))? desc : vertra ;
              desc: agsol;
       };

     msgabrrf:=
       case{
```

```
  (estado=aceitconex) : conectado;
  (estado=recusconex) : recusado;
  (estado=desc):desconecack;
  default : nenhuma;
  };

if (abrACR>=CD)
{
  if ((abrBPS+(abrACR-CD))>BD)
  {
        next(abrBPS):=abrBPS;
        abrCLR:=abrACR-CD-(BD-abrBPS);
  }
  else
  {
  next(abrBPS):=abrBPS+(abrACR-CD);
  abrCLR:=0;
  }
}
else
{
  if ((CD-abrACR)>=abrBPS)
  {
        next(abrBPS):=0;
        abrCLR:=0;
  }
  else
  {
        next(abrBPS):=abrBPS-(CD-abrACR);
        abrCLR:=0;
  }

}


next(abrDIR):=
  case {
        (abrCLR>0) : 0;
        (abrCLR=0)&(abrDIR=0):1;
        (abrCLR=0)&(abrDIR=1):0;
  };

next(abrCI):=
  case {
        (abrACR>CD)|(abrBPS>BD) : 1;
        default:0;
  };

next(abrNI):= case {
        (abrACR=CD) : 1;
        default:0;
  };

}

module redetot (CD, BD)

{

CD:5..20;
BD:10..40;

init(CD):=20;
init(BD):=20;

next(CD):=20;
next(BD):=20;
```

```
}
```

Figura 7.27 – A linguagem de entrada da categoria ABR

O módulo redetot permite a simulação de variação de recursos da rede e seu impacto nas especificações. De acordo com o descrito na figura 7.27 os recursos estão acima do máximo que a fonte pode trafegar, desta forma as três especificações mostradas na figura 7.28 são verdadeiras:

```
conexao: assert G ((fabr.estado=solconex)->F (fabr.estado=txcel));
desconexao: assert G ((fabr.estado=solconex)->F (fabr.estado=txcel));
perda: assert G (rabr.abrCLR<1);
```

Figura 7.28 – Especificações verdadeiras para categoria ABR

Foram necessárias 22 iterações, um total de 37.349 nós BDDs alocados em 2.196 estados alcançáveis

```
Resources used
==============
BDD nodes allocated.....................................................37349
```

Figura 7.29 – Recursos para verificar ABR com recursos de rede fixos

No modelo da figura 7.27, a quantidade de células e buffers que a rede possui disponível é maior do que o máximo que a categoria ABR pode ocupar, assim todas as especificações são verdadeiras. Alterando o valor de células disponíveis (CD) no modelo, de 20 para 10 observa-se as especificações continuam verdadeiras. Apesar da fonte possuir um PCR de até quinze células a fonte nunca trafega este valor, pois, partindo do ICR (taxa de células inicial), a fonte incrementa o ACR até o valor que a rede permite, neste caso 10.

A especificação "`maisquedez: assert G (fabr.abrACR<11);`" é verdadeira, indicando que a fonte fica limitada ao máximo de células disponíveis.

No modelo da figura 7.27, e na variação utilizando CD = 10, não há uma variação brusca nos recursos da rede, assim, a fonte pode manter o tráfego adaptado aos recursos

oferecidos. Entretanto, a entrada de novas fontes de tráfego numa rede podem variar os recursos compartilhados bruscamente afetando a categoria ABR.

Para simular a entrada de uma nova fonte que utilizará os recursos aproveitados pela fonte pode-se alterar:

```
next(CD):=20;
next(BD):=20;
```

para

```
next(CD):={5,20};
BD:=2*CD;
```

A estratégia de variação do ACR em função da rede é muito rudimentar. Variações bruscas na rede não serão rapidamente acompanhadas pelo controle de ACR da fonte. Verifica-se que existe perda de células após uma variação brusca dos recursos disponíveis, conforme mostra a figura 7.30.

	4	5	6	7	8	9	10	11	12	13	14	15	16
BD	10	10	10	10	10	10	40	40	40	40	10	10	10
CD	5	5	5	5	5	5	20	20	20	20	5	5	5
abrACR	0	0	1	2	3	4	5	6	7	8	9	10	9
abrCI	0	0	0	0	0	0	0	0	0	0	0	1	1
abrDIR	1	0	1	0	1	0	1	0	1	0	1	0	1
abrNI	0	0	0	0	0	0	0	0	0	0	0	0	0
abrQBR	0	0	0	0	0	0	0	0	0	0	0	0	0
abrQCR	0	0	0	0	0	0	0	0	0	0	0	0	0
fabr.abrMCR	0	0	0	0	0	0	0	0	0	0	0	0	0
fabr.abrPCR	10	10	10	10	10	10	10	10	10	10	10	10	10
fabr.contCRM	1	0	1	0	1	0	1	0	1	0	1	0	1
fabr.estado	solconex	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel
msgabrfr	estabele	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
msgabrrf	conectac	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rabr.abrBPS	0	0	0	0	0	0	0	0	0	0	0	4	9
rabr.abrCLR	0	0	0	0	0	0	0	0	0	0	0	0	3
rabr.estado	aceitcon	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra

Figura 7.30 – Seqüência de variáveis que invalida a especificação "perda"

Variando de 11 a 2 o valor de X da especificação "`perda: assert G (rabr.abrCLR<X);`" verifica-se que o máximo CLR é igual a 10, conforme mostra a figura 7.31.

	9	10	11	12	13	14	15	16	17	18	19	20	21
BD	10	40	40	40	40	40	40	40	40	40	40	10	10
CD	5	20	20	20	20	20	20	20	20	20	20	5	5
abrACR	4	5	6	7	8	9	10	11	12	13	14	15	15
abrCI	0	0	0	0	0	0	0	0	0	0	0	0	1
abrDIR	0	1	0	1	0	1	0	1	0	1	0	1	0
abrNI	0	0	0	0	0	0	0	0	0	0	0	0	0
abrQBR	0	0	0	0	0	0	0	0	0	0	0	0	0
abrQCR	0	0	0	0	0	0	0	0	0	0	0	0	0
fabr.abrMCR	0	0	0	0	0	0	0	0	0	0	0	0	0
fabr.abrPCR	15	15	15	15	15	15	15	15	15	15	15	15	15
fabr.contCRM	0	1	0	1	0	1	0	1	0	1	0	1	0
fabr.estado	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel	txcel
msgabrfr	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
msgabrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
rabr.abrBPS	0	0	0	0	0	0	0	0	0	0	0	0	10
rabr.abrCLR	0	0	0	0	0	0	0	0	0	0	0	0	10
rabr.estado	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra	vertra

Figura 7.31 – Seqüência de variáveis que invalida a especificação "perda"

A perda de células mostrada na figura 7.31, decorrente da variação brusca dos recursos disponíveis, ocorre porque o buffer pequeno não consegue armazenar o excedente de células transmitidas após uma redução brusca da banda disponível. Poder-se-ia aumentar a capacidade do buffer para suportar variações bruscas dos recursos disponíveis, porém a característica do ABR é aproveitar capacidade ociosa da rede. Se para evitar a perda é necessário que tenha recurso alocado para a categoria então o não há sentido em se usar o ABR.

Sem alterar o buffer disponível (BD) pode-se aumentar os valores de RIF e RDF para mais rapidamente forçarem o retorno da fonte a uma situação adequada, porém as variações simuladas no estabelecimento de conexão de uma nova fonte de tráfego são muito bruscas o que exigiria altos valores de RIF e RDF, o que é ineficiente para a dinâmica de variações suaves no tráfego.

Uma estratégia adequada é a determinação da taxa explícita de células, ER, pelo controle do CAC, que pode informar com antecipação para o ABR a solicitação de entrada de uma nova fonte de tráfego que irá reduzir a capacidade disponível. Assim a fonte ABR poderá se ajustar antes da mudança se realizar. A figura 7.32 mostra as mudanças da linguagem da figura 7.27 para adotar a estratégia de taxa explícita, bem como simular a entrada e saída de uma nova fonte que requisita os recursos disponíveis para o ABR.

```
ER:0..15;
```

```
if ((estado=txcel)&(ER=0))
{
        next(abrACR):=
                case{
                        (contCRM=CRM):((abrACR-CDF)<=abrMCR) ? abrMCR : (abrACR-CDF);
                        (abrCI=1):((abrACR-RDF)<=abrMCR) ? abrMCR : (abrACR-RDF);
                        ((abrCI=0)&(abrNI=1)):abrACR;
                        ((abrCI=0)&(abrNI=0)):((abrACR+RIF)>=abrPCR)    ?    abrPCR    :
(abrACR+RIF);
                };
}
else

        if (estado=txcel)
                next(abrACR):=ER;
        else
                next(abrACR):=0;
}

module redetot (CD, BD, ER)

{

CD:5..20;
BD:10..40;
estado:{desc, solconex, txcel, soldesc};

init(estado):=desc;

next(estado):=
        switch(estado){
                desc:{desc, solconex};
                solconex: txcel;
                txcel:{txcel,soldesc};
                soldesc: desc;
                };

CD:=
        case {
                (estado=txcel):5;
                default:20;
        };

ER:=
        case {
                (estado=solconex):5;
                default:0;
        };
}
```

Figura 7.32 – Mudança na linguagem SMV para modelar a utilização de taxas explícitas

A verificação das especificações do sistema modelado pela linguagem da figura 7.27, alterado pelas inclusões mostradas na figura 7.32, mostrou que todas são verdadeiras, utilizando 19 iterações, 10.116 estados alcançáveis e a quantidade de nós BDDs mostrada na figura 7.33.

```
Resources used
==============
BDD nodes allocated.....................................................44745
```

Figura 7.33 – Recursos utilizados para verificar ABR com utilização de taxas explícitas

A utilização da taxa explícita pode resolver o problema das grandes e bruscas variações nos recursos oferecidos. Para pequenas variações, a escolha de valores de RIF e RDF adequados, associado a um pequeno buffer, pode evitar o descarte de células.

7.6 – O sistema completo

Um sistema de uma conexão usuário-rede ATM pode conter diversas categorias de serviço, e a ferramenta SMV pode ser utilizada para verificar como diferentes categorias se comportam de acordo com o dimensionamento da rede e das características de tráfego de cada fonte. Entretanto para modelar este sistema é necessário considerar algumas questões.

7.6.1 – A explosão de estados

No item 7.5 foi trabalhada a categoria VBR isoladamente. Verificou-se que o modelo inicial proposto extrapolava a capacidade de hardware que suportava a ferramenta. Assim, utilizou-se uma redução do número de estados possíveis através da fixação de algumas variáveis como o SCR e o PCR. Mesmo com esta estratégia a quantidade de estados do modelo desta categoria exigiu um esforço computacional considerável, atingindo 203.373 estados alcançáveis.

Tentativas de verificar a categoria VBR em paralelo com as outras categorias, utilizando a combinação dos módulos apresentados nos itens 7.3 e 7.5, fracassaram, pois a ferramenta extrapolou a capacidade de hardware. Assim, foi necessário adotar uma estratégia de separação dos comportamentos para conseguir realizar a verificação.

Existem dois comportamentos comuns nas categorias de serviço que são bem distintos. O primeiro é o comportamento de conexão e desconexão da fonte com a rede, o segundo é a

dinâmica da taxa de células da fonte e de ocupação de buffer de uma conexão estabelecida. Assim, é possível separar os dois comportamentos e analisá-los individualmente para um grupo de categorias simultâneas.

A dinâmica da taxa de células, com as variações na taxa indicadas por CPS e ACR, e ocupação dos buffers, indicadas por BPS, tem uma interligação. Para uma rede conceder a desconexão solicitada por uma fonte é necessário aguardar até que o buffer esteja vazio. Entretanto tal interligação não impede que os comportamentos sejam analisados em separado.

A dinâmica da taxa de células além de ser bem desconectado do comportamento de conexão e desconexão, é isolado em cada categoria de serviço simulada nos itens 7.3, 7.4 e 7.5, pois as fontes só conseguem trafegar dentro dos recursos reservados no estabelecimento da conexão, ou, no caso do ABR, naqueles determinados pela rede.

7.6.2 – Analisando o comportamento de conexão e desconexão

A figura 7.34 mostra a linguagem utilizada para alimentar a ferramenta SMV com o modelo.

```
#define maxC 15
#define maxB 30

module main()
{

QCR : array 0..2 of 0..15;
QBR : array 0..2 of 0..30;
CD:-15..15;
BD:-30..30;

fr1 : fonteerede(QCR[0], QBR[0], CD, BD);
fr2 : fonteerede(QCR[1], QBR[1], CD, BD);
fr3 : fonteerede(QCR[2], QBR[2], CD, BD);

CD:= maxC-QCR[0]-QCR[1]-QCR[2];
BD:=maxB-QBR[0]-QBR[1]-QBR[2];

conexao: assert G ((fr1.estadofonte=solconex)->F (fr1.estadofonte=txcel));
desconexao: assert G ((fr1.estadofonte=soldesc)->F (fr1.estadofonte=desc));
ambos: assert G F((fr1.estadofonte=txcel)->~(fr2.estadofonte=txcel));

}

module fonteerede(QCR, QBR, CD, BD)
{

estadofonte:{desc, solconex, txcel, soldesc};
msgcbrfr:{nenhuma, estabelecendo,desconec};
estadorede:{agsol,verrec, aceitconex, recusconex, vertra, desc};
```

```
     msgcbrrf:{nenhuma, conectado, recusado, desconecack};
     PCR:10..15;

     init(estadofonte):=desc;
     init(PCR):=10..15;
     init(estadorede):=agsol;

     next(estadofonte):=
       switch(estadofonte){
              desc:{desc, solconex};
              solconex:    (msgcbrrf=conectado)?txcel:    (msgcbrrf    =    recusado)?
desc:solconex;
              txcel:{txcel,soldesc};
              soldesc: (msgcbrrf=desconecack)?desc:soldesc;
              };

     msgcbrfr:=
       case{
       (estadofonte = solconex): estabelecendo;
       (estadofonte = soldesc) : desconec;
       default : nenhuma;
       };

     next(PCR):=
       case{
              (estadofonte=desc):10..15;
              default: PCR;
       };

     if ((estadofonte=desc)|(estadofonte=solconex))
       {
       QCR:=0;
       QBR:=0;
       }
     else
       {
       QCR:=PCR;
       QBR:=2*PCR;
       }

     next(estadorede):=
       switch(estadorede){
              agsol: (msgcbrfr=estabelecendo)? verrec : agsol;
              verrec : ((CD >= QCR)&(BD >=QBR)) ? aceitconex : recusconex;
              aceitconex:vertra;
              recusconex: agsol;
              vertra : (msgcbrfr=desconec)? desc : vertra ;
              desc: agsol;
       };


     msgcbrrf:=
       case{
       (estadorede=aceitconex) : conectado;
       (estadorede=recusconex) : recusado;
       (estadorede=desc):desconecack;
       default : nenhuma;
       };

     }
```

Figura 7.34 – A linguagem de entrada do comportamento de conexão e desconexão

O problema ocorre quando mais de uma fonte solicita conexão simultaneamente. Como os recursos de banda e buffers são compartilhados a verificação de recursos disponíveis será simultânea e serão aceitas mais conexões do que o adequado, tornando falsa a especificação:

```
ambos: assert G F((fr1.estadofonte=txcel)->~(fr2.estadofonte=txcel));
```

A figura 7.35 mostra os estados que levaram a invalidar a especificação anterior.

	1	2	3	4	5	6	7	8
BD	30	30	30	30	-30	-30	-30	-30
CD	15	15	15	15	-15	-15	-15	-15
QBR[0]	0	0	0	0	30	30	30	30
QBR[1]	0	0	0	0	30	30	30	30
QBR[2]	0	0	0	0	0	0	0	0
QCR[0]	0	0	0	0	15	15	15	15
QCR[1]	0	0	0	0	15	15	15	15
QCR[2]	0	0	0	0	0	0	0	0
fr1.PCR	15	15	15	15	15	15	15	15
fr1.estadofonte	desc	solconex	solconex	solconex	txcel	txcel	txcel	txcel
fr1.estadorede	agsol	agsol	verrec	aceitcor	vertra	vertra	vertra	vertra
fr1.msgcbrfr	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma	nenhuma	nenhuma
fr1.msgcbrrf	nenhuma	nenhuma	nenhuma	conectac	nenhuma	nenhuma	nenhuma	nenhuma
fr2.PCR	15	15	15	15	15	15	15	15
fr2.estadofonte	desc	solconex	solconex	solconex	txcel	txcel	txcel	txcel
fr2.estadorede	agsol	agsol	verrec	aceitcor	vertra	vertra	vertra	vertra
fr2.msgcbrfr	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma	nenhuma	nenhuma
fr2.msgcbrrf	nenhuma	nenhuma	nenhuma	conectac	nenhuma	nenhuma	nenhuma	nenhuma
fr3.PCR	15	15	15	15	15	15	15	15
fr3.estadofonte	desc	desc	desc	solconex	solconex	solconex	desc	solconex
fr3.estadorede	agsol	agsol	agsol	agsol	verrec	recuscor	agsol	agsol
fr3.msgcbrfr	nenhuma	nenhuma	nenhuma	estabele	estabele	estabele	nenhuma	estabele
fr3.msgcbrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	recusadc	nenhuma	nenhuma

Figura 7.35 – Evolução das variáveis que levou mais conexões do que o adequado

Assim é necessário que apenas uma fonte tenha suas características de tráfego avaliadas de cada vez. As demais fontes que estão no estado de "solicitando conexão" continuarão esperando. Para modelar este comportamento utilizou-se uma estratégia de prioridade. Cada fonte que solicita conexão incrementa um parâmetro denominado IP (índice de prioridade). Quando a fonte consegue a conexão, ou desiste de buscar a conexão, o parâmetro é zerado. Mais de uma categoria pode ter o mesmo índice de prioridade, porém apenas a primeira da

lista do modelo que apresenta o índice mais alto poderá se conectar, enquanto a outra tem seu índice incrementado.

O modelo de três fontes com a estratégia de priorização é mostrado na figura 7.36.

```
#define maxC 15
#define maxB 30

module main()
{

QCR : array 0..2 of 0..15;
QBR : array 0..2 of 0..30;
CD:-15..15;
BD:-30..30;
ident: array 0..2 of 0..2;
ficha:0..3;
IP: array 0..2 of 0..3;

for (i=0;i<3;i=i+1)
  init(ident[i]):=i;

fr1 : fonteerede(QCR[0], QBR[0], CD, BD, ident[0], IP[0], ficha);
fr2 : fonteerede(QCR[1], QBR[1], CD, BD, ident[1], IP[1], ficha);
fr3 : fonteerede(QCR[2], QBR[2], CD, BD, ident[2], IP[2], ficha);

CD:= maxC-QCR[0]-QCR[1]-QCR[2];
BD:=maxB-QBR[0]-QBR[1]-QBR[2];
for (i=0;i<3;i=i+1)
  next(ident[i]):=i;

chain (j=0;j<11;j=j+1)
  {
  chain (i=0; i<2;i=i+1)
        if (IP[i]=j) ficha:=i;
  }

conexao: assert G ((fr1.estadofonte=solconex)->F (fr1.estadofonte=txcel));
desconexao: assert G ((fr1.estadofonte=soldesc)->F (fr1.estadofonte=desc));
ambos: assert G F((fr1.estadofonte=txcel)->~(fr2.estadofonte=txcel));

}

module fonteerede(QCR,QBR,CD,BD,ident,IP,ficha)
{

estadofonte:{desc, solconex, txcel, soldesc};
msgcbrfr:{nenhuma, estabelecendo,desconec};
estadorede:{agsol,verrec, aceitconex, recusconex, vertra, desc};
msgcbrrf:{nenhuma, conectado, recusado, desconecack};
PCR:10..15;

init(estadofonte):=desc;
init(PCR):=10..15;
init(estadorede):=agsol;

next(estadofonte):=
  switch(estadofonte){
        desc:{desc, solconex};
        solconex:   (msgcbrrf=conectado)?txcel:   (msgcbrrf   =   recusado)?
desc:solconex;
        txcel:{txcel,soldesc};
        soldesc: (msgcbrrf=desconecack)?desc:soldesc;
        };
```

```
    msgcbrfr:=
      case{
      (estadofonte = solconex): estabelecendo;
      (estadofonte = soldesc) : desconec;
      default : nenhuma;
      };

    next(PCR):=
      case{
            (estadofonte=desc):10..15;
            default: PCR;
      };

    if ((estadofonte=desc)|(estadofonte=solconex))
      {
      QCR:=0;
      QBR:=0;
      }
    else
      {
      QCR:=PCR;
      QBR:=2*PCR;
      }

    next(estadorede):=
      switch(estadorede){
            agsol: (msgcbrfr=estabelecendo)? verrec : agsol;
            verrec : ((CD >= PCR)&(BD >=2*PCR)&(ficha=ident)) ? aceitconex :
recusconex;
            aceitconex:vertra;
            recusconex: agsol;
            vertra : (msgcbrfr=desconec)? desc : vertra ;
            desc: agsol;
      };


    msgcbrrf:=
      case{
      (estadorede=aceitconex) : conectado;
      (estadorede=recusconex) : recusado;
      (estadorede=desc):desconecack;
      default : nenhuma;
      };

    next(IP):=
      case
      {
      (estadofonte=solconex):IP+1;
      (estadofonte~=solconex):0;
      };


    }
```

Figura 7.36 – A linguagem de entrada do comportamento de conexão e desconexão

A estratégia de priorização funcionou em não permitir que duas fontes estabelecessem conexões simultaneamente, ou seja, a especificação "ambos" é verdadeira. Já a especificação "conexao: assert G ((fr1.estadofonte=solconex)->F

(fr1.estadofonte=txcel));" é falsa, o que era esperado, pois existe a possibilidade da fonte 2 ocupar os recursos durante todo o tempo, não permitindo a conexão da fonte 1.

Alterando a especificação de conexão para "conexao: assert G (((fr1.estadofonte=solconex)&(fr2.estadofonte=desc))-> F (fr1.estadofonte=txcel));" e verificando novamente descobriu-se uma falha do projeto. A figura 7.37 mostra outra possibilidade de sucessão de estados que leva a especificação a ser falsa, isto é, é possível que a fonte número 1 nunca consiga estabelecer uma conexão.

	2	3	4	5	6	7	8	9	10	11	12	13	14
BD	30	30	30	0	0	0	30	30	30	30	0	0	0
CD	15	15	15	0	0	0	15	15	15	15	0	0	0
IP[0]	0	0	0	0	0	1	2	3	0	0	0	0	1
IP[1]	0	1	2	3	0	0	0	0	1	2	3	0	0
QBR[0]	0	0	0	0	0	0	0	0	0	0	0	0	0
QBR[1]	0	0	0	30	30	30	0	0	0	0	30	30	30
QBR[2]	0	0	0	0	0	0	0	0	0	0	0	0	0
QCR[0]	0	0	0	0	0	0	0	0	0	0	0	0	0
QCR[1]	0	0	0	15	15	15	0	0	0	0	15	15	15
QCR[2]	0	0	0	0	0	0	0	0	0	0	0	0	0
ficha	1	1	1	1	1	0	0	0	1	1	1	1	0
fr1.PCR	15	15	15	15	15	15	15	15	15	15	15	15	15
fr1.estadofonte	desc	desc	desc	desc	solconex	solconex	solconex	desc	desc	desc	desc	solconex	solconex
fr1.estadorede	agsol	agsol	agsol	agsol	agsol	verrec	recuscor	agsol	agsol	agsol	agsol	agsol	verrec
fr1.msgcbrfr	nenhuma	nenhuma	nenhuma	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma	nenhuma	nenhuma	estabele	estabele
fr1.msgcbrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	recusadc	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma
fr2.PCR	15	15	15	15	15	15	15	15	15	15	15	15	15
fr2.estadofonte	solconex	solconex	solconex	txcel	soldesc	soldesc	desc	solconex	solconex	solconex	txcel	soldesc	soldesc
fr2.estadorede	agsol	verrec	aceitcor	vertra	vertra	desc	agsol	agsol	verrec	aceitcor	vertra	vertra	desc
fr2.msgcbrfr	estabele	estabele	estabele	nenhuma	desconec	desconec	nenhuma	estabele	estabele	estabele	nenhuma	desconec	desconec
fr2.msgcbrrf	nenhuma	nenhuma	conectac	nenhuma	nenhuma	desconec	nenhuma	nenhuma	nenhuma	conectac	nenhuma	nenhuma	desconec

Figura 7.37 – Evolução das variáveis que impede a conexão da fonte 1

Existe a possibilidade da fonte número 2 estabelecer conexão e ocupar os recursos antes da fonte 1 solicitar a conexão. Esta então irá encontrar os recursos ocupados, sendo negada sua conexão pela rede. Então a prioridade de transmissão da fonte 1 volta a zero e a fonte 2 paralelamente consegue se desconectar e reconectar. Esta situação pode se repetir infinitamente.

O erro do projeto foi permitir que a prioridade da fonte 1 voltasse a zero. Alterando o comportamento de IP para:

```
next(IP):=
  case
  {
  (estadofonte=solconex):(IP<10)?IP+1:IP;
  (estadofonte=txcel):0;
  default:IP;
  };
```

Verifica-se que a especificação "conexao" continua falsa, pois existem três possibilidades consideradas normais:

- A fonte 2 estabelece a conexão antes da fonte 1, e fica transmitindo continuamente
- A fonte 2 estabelece a conexão antes da fonte 1, e esta tem a conexão recusada e não tenta novamente estabelecer a conexão
- A fonte 2 estabelece a conexão antes da fonte 1, e esta tem a conexão recusada e fica sem estabelecer a conexão até que a fonte 2 tenha prioridade maior do que esta.

A especificação "conexao" é falsa, como esperado, entretanto a especificação "ambos" também foi apontada como não verdadeira, mostrando outro erro do projeto. Como existe a possibilidade do IP ser incrementado sempre que o estado da fonte estiver em "solconex", ocorre a verificação de recursos quase simultaneamente. Pode-se ver, pela figura 7.38, que a mudança de prioridade ocorre logo após a verificação de recursos para estabelecimento de conexão da fonte 1, permitindo que a fonte 2 ainda encontre os recursos disponíveis.

	17	18	19	20	21	22	23	24	25	26	27	28	29
BD	30	30	30	30	30	30	30	30	30	30	0	-30	-30
CD	15	15	15	15	15	15	15	15	15	15	0	-15	-15
IP[0]	9	9	9	9	9	9	9	9	10	10	10	0	0
IP[1]	3	4	5	6	6	7	8	9	9	10	10	10	0
QBR[0]	0	0	0	0	0	0	0	0	0	0	30	30	30
QBR[1]	0	0	0	0	0	0	0	0	0	0	0	30	30
QBR[2]	0	0	0	0	0	0	0	0	0	0	0	0	0
QCR[0]	0	0	0	0	0	0	0	0	0	0	15	15	15
QCR[1]	0	0	0	0	0	0	0	0	0	0	0	15	15
QCR[2]	0	0	0	0	0	0	0	0	0	0	0	0	0
ficha	0	0	0	0	0	0	0	1	0	1	1	1	1
fr1.PCR	15	15	15	15	15	15	15	15	15	15	15	15	15
fr1.estadofonte	desc	desc	desc	desc	desc	desc	desc	solconex	solconex	solconex	txcel	txcel	txcel
fr1.estadorede	agsol	agsol	agsol	agsol	agsol	agsol	agsol	agsol	verrec	aceitcon	vertra	vertra	vertra
fr1.msgcbrfr	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma	nenhuma
fr1.msgcbrrf	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	nenhuma	conectad	nenhuma	nenhuma	nenhuma
fr2.PCR	15	15	15	15	15	15	15	15	15	15	15	15	15
fr2.estadofonte	solconex	solconex	solconex	desc	solconex	solconex	solconex	desc	solconex	solconex	solconex	txcel	txcel
fr2.estadorede	agsol	verrec	recuscon	agsol	agsol	verrec	recuscon	agsol	agsol	verrec	aceitcon	vertra	vertra
fr2.msgcbrfr	estabele	estabele	estabele	nenhuma	estabele	estabele	estabele	nenhuma	estabele	estabele	estabele	nenhuma	nenhuma
fr2.msgcbrrf	nenhuma	nenhuma	recusadc	nenhuma	nenhuma	nenhuma	recusadc	nenhuma	nenhuma	nenhuma	conectad	nenhuma	nenhuma

Figura 7.38 – Evolução das variáveis que permite a conexão das fontes 1 e 2

A solução para o problema é permitir que somente a fonte com a "ficha" possa solicitar a conexão. Assim altera-se o comportamento de estabelecimento de conexão da fonte, escrito na linguagemem SMV, para:

```
next(estadofonte):=
       switch(estadofonte){
              desc: (ficha=ident)?solconex:desc;
              solconex:    (msgcbrrf=conectado)?txcel:    (msgcbrrf    =    recusado)?
desc:solconex;
              txcel:{txcel,soldesc};
              soldesc: (msgcbrrf=desconecack)?desc:soldesc;
              };
```

Utilizando a ferramenta para verificar o modelo alterado comprova-se que todas as especificações são verdadeiras. Foram necessárias 7 iterações e 12.208 nós BDDs alocados, de um total de 1.728 estados alcançáveis.

Para evitar que os recursos da rede ATM fossem constantemente solicitados por um grupo de fontes, deixando outras sem conexão, poderia ter-se utilizado a aleatoriedade na escolha de qual a seria a próxima fonte. Entretanto a estratégia de aleatoriedade não é justa. Outra solução seria a de mestre-escravo, porém ela foi evitada, pois apresenta uma baixa eficiência. Utilizou-se a solução por prioridade.

7.7 – Considerações sobre a verificação dos modelos

Nos itens 7.3, 7.4, 7.5 e 7.6 foi possível trabalhar algumas das características do comportamento do sistema de comunicações ATM, mostrando como pode-se utilizar as estratégias de modelagem e a ferramenta de verificação. Existem infinitas outras possibilidades de projeto e verificação, que podem ser simuladas por empresas de desenvolvimento de equipamentos, e mesmo operadoras de telecomunicações, descobrindo falhas de projeto e permitindo projetar estruturas mais adequadas.

Na verificação do ATM um dos maiores problemas foi conseguir escrever corretamente os modelos na linguagem da ferramenta SMV. A sua interface e forma de uso lembram os primeiros compiladores de C, com praticamente as mesmas limitações e algumas outras, que demandam tempo desnecessário do usuário. A ferramenta diz, no início da execução que "It

does what it does, no more, no less", ou seja, ela faz o que faz, nada mais nada menos. Infelizmente com um pouco mais de funções ela economizaria muito tempo do usuário. Pode-se citar as desvantagens, ou limitações: A principal e a que obriga que toda linguagem, ou programa, de entrada seja escrito em um editor de texto externo à ferramenta, e apesar dela possuir um campo que mostra a linguagem de entrada, é impossível qualquer alteração, sendo para isto necessário sair da ferramenta, alterar o texto no editor, salvar, voltar para a ferramenta, reabrir o programa e rodar novamente a verificação. Outro problema se refere a erros no texto do programa. Qualquer erro, por exemplo, uma virgula no lugar de um ponto e virgula, é apontado como erro de sintaxe e obriga o usuário a sair da ferramenta e utilizar o editor. O problema é que a ferramenta só aponta um erro por vez, e se o programa apresentar n erros se é obrigado a repetir este processo n vezes. Outro problema ocorre quanto aos recursos utilizados na verificação. Quando a primeira vez é verificado um modelo a ferramenta gera um registro, um log, que indica os recursos utilizados. Entretanto, se for solicitado que a ferramenta verifique novamente o mesmo programa, com uma pequena alteração, algumas vezes o indicador de recursos indica bem menos do que a antiga verificação. Para evitar este problema foi necessário eliminar os arquivos gerados pela ferramentas associados à verificação em questão, pois eles armazenavam de alguma forma um histórico da última análise permitindo uma nova análise mobilizando menos recursos.

Outro problema ocorreu quando um programa continha um intervalo de uma variável digitado errado. Ao invés do intervalo ser 10..15 foi digitado 10..5, e toda vez que se carregava o programa a ferramenta parava de funcionar e trancava a execução, obrigando a uma finalização forçada da tarefa. Além deste, algumas vezes, tentativas de abertura de arquivos causavam travamento do software.

Mesmo com os problemas apresentados, a ferramenta SMV realmente faz o que ela foi feita para fazer, isto é, ela verifica qualquer possibilidade de tornar uma das especificações do modelo falsa, permitindo a identificação de uma série de problemas e permitindo, no caso de sistemas como o ATM, a construção de sistemas mais eficazes e eficientes.

Capítulo 8 – Conclusões e Recomendações

Técnicas formais ajudam os projetistas a superar sérios desafios que eles encaram com o crescimento e aumento da complexidade que os projetos vem sofrendo. Movendo da simulação com suas inferências para os métodos de dedução da verificação formal, usuários ganham confiança que seus projetos estão sendo melhor verificados. Além disto ferramentas formais completam a análise de um sistema em menos tempo, conseguindo reduzir o tempo de projeto e tempo para o mercado.

A utilização de ferramentas de verificação formal não é uma tarefa trivial. É necessário a adoção de metodologias e estratégias de modelagem e verificação adequadas antes de utilizar uma ferramenta. O importante é ter em mente que os vários pontos estão interligados: os objetivos da verificação, o tipo de sistema, a estratégia de modelagem e a ferramenta a ser utilizada, sendo que existem diversas opções para cada ponto, interrelacionada com os demais. Por exemplo, se o objetivo da verificação envolve performance e amarrações explícitas do tempo, então é necessário a adoção de uma ferramenta que manipule modelos construídos com o tempo explicito. Outro exemplo: um sistema de comunicações de dados pode ser dividido em módulos, facilitando a verificação e contornando o problema da explosão de estados, já um sistema de tráfego ferroviário poderia apresentar a simetria como uma solução mais eficaz para reduzir a complexidade.

Na literatura a explosão de estados, apesar de ser muito debatida parece uma realidade apenas para sistemas grandes e complexos. Entretanto, como foi mostrado no capítulo 7, se utilizam-se variáveis com vários estados possíveis então a composição destas num modelo pode levar a uma quantidade de estados alcançáveis além do que as capacidades do hardware podem trabalhar. Para evitar este problema, quando não for possível utilizar a abstração, recomenda-se a redução dos modelos descrevendo apenas o ponto de interesse, fixando os demais. Por exemplo, pode-se verificar como se comporta um sistema com várias fontes de tráfego, que se conectam e desconectam da rede, sem considerar as variações de seus tráfegos. Porém tal estratégia de fixação de variáveis pode mascarar alguns erros se o projetista não considerou alguma inter-relação das variáveis fixas com o sistema em foco.

Se não houver uma explosão de estados a ferramenta fornece uma solução conclusiva a respeito do modelo versus as especificações, indicando, se for verificado um erro, qual a seqüência que levou a tal situação. Porém, quando a ferramenta indica ausência de erro, não é

possível ver a seqüência normal de operação, o que poderia ser uma fonte importante para revelar erros da descrição do comportamento do sistema que passaram desapercebidos pelas especificações, que não foram construídas para detectá-los. Por exemplo, no início das verificações dos modelos do capítulo 7 foi construído, sem querer, um modelo com um erro sutil: era permitido que a fonte se desconectasse da rede enquanto o buffer ainda tinha células armazenadas, e, de acordo com a construção do modelo, no próximo instante de tempo como a rede ia para um estado de desconexão as células do buffer eram eliminadas sem acusar como descartadas. Este erro foi identificado através de inspeção visual de uma seqüência de outro erro, desta vez natural do modelo. Para tentar minimizar estas situações recomenda-se a verificação da correção do modelo através de várias especificações obvias, algumas gerando trilhas de erros que devem ser meticulosamente analisadas.

Apesar de existirem metodologias e estratégias para modelagem de protocolos de comunicação de dados não existe uma receita de bolo que garanta sucesso. O importante é o projetista conhecer todas as possibilidades de estratégias e metodologias e, conhecendo profundamente o sistema, escolher um conjunto que seja adequado. O sucesso na escolha é diretamente proporcional ao conhecimento do sistema a ser modelado. Mesmo assim o projetista pode ter que queimar uma etapa de tentativas e erros antes de obter o sucesso.

Descrever o modelo de um sistema de comunicação de dados é uma tarefa que depende muito da habilidade e conhecimento do projetista, entretanto, descrever as especificações desejadas e alimentar a ferramenta com as informações são etapas mais pragmáticas.

A ferramenta utilizada no capítulo 7 apresentou uma série de pequenos problemas e falta de recursos que consumiram mais tempo do que o previsto para realização das verificações, conforme foi mostrado naquele capítulo. Como a ferramenta que roda em Windows 95 é uma adaptação da ferramenta que roda em ambiente UNIX recomenda-se testar se os problemas são dependentes do sistema operacional e, se são, utilizar a ferramenta no seu sistema operacional nativo.

Na verificação do ATM um dos maiores problemas foi conseguir escrever corretamente os modelos na linguagem da ferramenta SMV, conforme foi descrito no item 7.8. Porém, uma vez alimentada, a verificação utilizando a ferramenta SMV apresenta todas as possibilidades de tornar uma especificação inválida, permitindo a verificação de projetos e a construção de comportamentos mais adequados para cada subsistema do ATM.

Capítulo 9 – Referências Bibliográficas

[AF96] ATM FORUM, Technical Committee. **Traffic Management Specification.** Version 4.0. af-tm-0056.000. 1996.

[AF98] ATM Forum. **Technical Specifications**. Approved Items as of October 1998.

[AS85] ALPERN, Bowen e SCHNEIDER, Fred B. **Defining Liveness**. Elsevier Publishers B.V. 1985.

[BCM90] BURCH, J.R e CLARK, E.M. e MCMILLAN, K.L. e DILL, D.L. e HWANG. J. **Symbolic Model Checking: 10^{20} states and Beyond.** Proceedings of the Fifth Annual Symposium on Logic in Computer Science. Junho, 1990.

[Bel98] BELL-LABS. **Basic Spin Manual**. Bell-Labs. 1998.

[BH94] BOWEN, Jonathan P. e HINCHEY, Michael G. **Seven more myths of Formal Methods.** Oxford University Computing Laboratory. Relatório Técnico PRG-TR-94-7. 1994.

[Bit96] BITTENCOURT, G. **Inteligência Artificial Ferramentas e Teorias.** Décima Escola de Computação, Departamento de Ciência da Computação, UNICAMP, Campinas, SP, 240 p., 1996.

[BK95] BIERE, Armim e KICK, Alexander. **A case study on different modeling approaches based on model checking – verifying numerous versions of the alternating bit protocol with SMV**. Germany. 1995.

[Bri94] BRISA. Sociedade Brasileira para Interconexão de sistemas abertos. **Arquiteturas de redes de computadores OSI e TCP/IP**. Makron Books. Rio de Janeiro. 1994.

[CFC94] CARDOSO, Janétte, FARINES, Jean-Marie e CURY, José Eduardo R. **Controle de Sistemas de Manufatura**. Material do Curso de Pós-Graduação em Automação Industrial a nível de Especialização ministrado na Universidade de Caxias do Sul. Setembro, 1994.

[CGL96] CLARKE, Edmund M, GRUMBERG, Orna, e LONG, David E. **Model Checking**. 1996.

[Chr98] Chrysalis Symbolic Design, Inc. **White Paper: What is Formal Verification** June 1998 Trademark/Copyright © 1992-1998 Chrysalis Symbolic Design, Inc.

[Cis98] CISCO. **StrataCom System Overview. Traffic Management Overview ATM Connections.** 1998.

[Cur94] CURY, José Eduardo. **Modelagem e Avaliação de Desempenho. De sistemas a Eventos Discretos**. Notas de Aula. 1994.

[CW96] CLARKE, Edmund M. e WING, Jeannette M. **Formal Methods: State of the Art and Future Directions.** ACM Strategic Directions in Computing Research Workshop. MIT, Cambridge, Estados Unidos. Junho de 1996.

[Dil98] DILL, David L. **Model Checking**. Computer Systems Laboratory. Stanford University

[Eme90] EMERSON, E. Allen. **Temporal and Modal Logic**. Chapter 16. Handbook of theoretical computer science. Elsevier Science Publishers B.V. 1990.

[Equ92] EQUITEL Telecomunicações. **Introdução à comutação de pacotes** - UN2179. Santa Catarina, 1992.

[Fer88] FERREIRA, Aurélio Buarque de Holanda. **Dicionário Aurélio Básico da Língua Portuguesa.** Editora Nova Fronteira S/A. Rio de Janeiro, RJ. 1988.

[Fra95] FRANCO, Carlos Alberto di. **Manual de Telecomunicações**. Paulo Andreoli & Associados. São Paulo. 1995.

[Gdc98] GDC (General Datacom) – Curso VIS. **Visão Estratégica do ATM**. 1998.

[Ger98] GERHARDT, Paulo. **Comercialização e marketing de serviços Frame Relay.** Treinar Treinamento Empresarial LTDA. 1998.

[Hal90] HALL, J.A. **Seven Myths of Formal Methods**. IEEE Software. Setembro 1990.

[Hel98] HELJANKO, Keijo. **Model Checking the Branching Time Temporal Logic CTL**, 1998

[Her96] HERZOG, Jonathan. **Reviews of Tools.** JPL/HMC Mathematics Clinic Report. 1996

[Hol91] HOLZMANN, Gerard J. **Design and Validation of Computer Protocols**. Bell Laboratories. Murray Hill. New Jersey. Prentice-Hall. Copyright 1991 by Lucent Technologies, Bell Laboratories, Incorporated.

[HS92] HOMER, Steven e SELMAN, Alan L. **Complexity Theory**. Department of Computer Science, Boston University e Department of Computer Science, State University of New York at Bufalo. June 8, 1992

[JH97] JPL/HMC. **Formal Verification Tools.** JPL/HMC Mathematics Clinic Report. 1997

[LM92] LAKATOS, Eva Maria, MARCONI, Marina de Andrade. **Metodologia do trabalho científico**. 4ª edição. Editora Atlas. São Paulo. 1992.

[Luc97] LUCENT Technologies Inc. **Model Checkers**. USA. 1997.

[Mar96] MARTINS, Ricardo Ferreira. **Verificação de Sistemas Dependentes do Tempo a partir de Especificações escritas em RT-LOTOS**, PGEEL/UFSC, Junho 1996.

[Mcm92] McMILLAN, K.L. **Symbolic Model Checking. An approach to the state explosion problem**. 1992

[Mcm92b] MCMILLAN, K.L. **The SMV system DRAFT**. Carnegie-Mellon University, 1992.

[Mcm98] MCMILLAN, K. L. **The SMV Language**. 1998.

[Mcm98b] McMILLAN, K.L. **Getting started with SMV**. 1998

[Mcm98c] McMILLAN,, K.L. **SMV FAQ**. 1998

[Nor96] NORDSTROM, Anders. **Formal Verification – A viable Alternative to Simulation?** IEEE, 1996. Reprinted with permission, from the International Verilog HDL Conference, Page 90, Santa Clara, CA, 1996.

[Ort96] ORTIZ, Alex Lopez. **Comp. Theory FAQ**. Faculty of Computer Science. University of New Brunswick. Canada. 1996

[SLC95] SOARES, Luiz Fernando Gomes, LEMOS, Guido, COLCHER, Sérgio. **Redes de Computadores: Das LANs, MANs e WANs às Redes ATM**. Segunda Edição. Editora Campus. Rio de Janeiro. 1995

[Tan94] TANENBAUM, Andrew S. **Redes de Computadores.** Editora Campus Ltda, 1994. Rio de Janeiro, RJ.

[Tar86] TAROUCO, Liane. M. R. **Rede de Computadores. Locais e de Longa Distância.** McGraw-Hill. Rio de Janeiro, RJ. 1986.

[Vis98] VISSER, Willem. **Pratical Model Checking: LTL vs CTL**. U. of Manchester. 1998

[Wol83] WOLPER, P. **Temporal Logic can be more expressive**. Inform. and Control. 1983

[Won94] WONG, Kai Cheung. **Discrete-Event control architecture: an algebraic approach.** Systems Control Group Report No. 9407. July 1994.

[Yan93] YANG, Bwolen, e outros. **A performance Study of BDD-Based Model Checking.** USA. 1993

[Zil92] ZILLER, Roberto M. **Exercício para a disciplina Tópicos Especiais em Sistemas a Eventos Discretos**. 1992.

Glossário

A

AAL - ATM Adapation Layer - Camada padrão que permite que múltiplas aplicações tenham seus dados convertidos para e de células ATM. É um protocolo que traduz serviços de camadas mais altas para o tamanho e formato de uma célula ATM.

ABR - Available Bit Rate - ABR é uma categoria de serviço na qual as características de transferencia da rede podem mudar as características de transmissão da conexão. Um mecanismo de controle de fluxo é especificado e suporta diversos tipos de realimentação para controlar a taxa da fonte para atender as mudanças nas características de transferencia da rede ATM. É esperado que o sistema terminal adapte seu tráfego em função da realimentação recebida, assim conseguindo uma baixa taxa de perda de células e uma divisão justa da banda disponível da rede. A variação do atraso de células não é controlada neste serviço.

ACR - Allowed Cell Rate - É um parâmetro do serviço ABR que define a taxa atual de células por segundo que a fonte pode enviar.

ATM - Asynchronous Transfer Mode - Modo de transferência assíncrono no qual a informação é organizada em células. Não há periodicidade na transmissão de células contendo informações da mesma fonte.

Autômatos a estados finitos – É uma forma de representar processos reais através da utilização de um conjunto de estados finito e um conjunto de transições que determinam como ocorrem as transições entre estados. A modelagem através de uma máquina de estados finitos é utilizada por várias ferramentas de verificação.

C

CAC - Connection Admission Control: Controle de Admissão de conexões. - É definido como um conjunto de ações adotadas pela rede durante a fase de estabelecimento de conexão, indicando se a conexão pode ser estabelecida ou não.

CBR - Constant Bit Rate - É uma categoria de serviço que suporta uma taxa constante para transporte de serviços como vídeo, voz e emulação de circuitos, .

CDF - Cutoff Decrease Factor - Fator de decremento de corte. Permite a redução do ACR associado ao CRM.

CDV - Cell Delay Variation - É um componente do atraso de transferencia de células, oriundo de buffers e agendamento de células. O Peak-to-peak CDV é um parametro de QoS associado com o CBR e o VBR

CDVT - Cell Delay Variation Tolerance - As funções da camada ATM podem alterar as características de tráfego das conexões ATM introduzindo variação de atraso de células.

Célula – A unidade de transmissão do ATM, consistindo de um cabeçalho de 5 bytes e um payload de 48 octetos.

CI - Congestion Indicator - Indicador de Congestionamento - É um campo na célula RM usado para indicar para fonte que esta deve baixar seu ACR. A fonte seta o CI =0 quando envia uma célula RM. Indicando CI=1 é como o destino indica que EFCI foi recebido numa célula prévia de dados.

CLP - Cell Loss Priority: Prioridade de perda de célula. - É um bit do cabeçalho da célula ATM que indica dois níveis de prioridade: CLP=0 tem prioridade maior do que as células com CLP=1, que podem ser descartadas durante períodos de congestionamento para preservar as células CLP=0.

CLR - Cell Loss Ratio: Taxa de perda de células - É um parâmetro de QoS negociável específicos de cada rede. É definido como a razão entre o total de células perdidas pelo total de células transmitidas.

code checker – Uma ferramenta que determinará se um pedaço do código atende a uma determinada especificação. Além disto eles podem ser usados para checar especificações e/ou para traduzir uma especificação para um código.

Código - Um programa escrito numa linguagem de programação. O programa pode ser compilado, traduzido ou interpretado.

Condições iniciais – São as condições de um processo ou sistema que devem ser consideradas ou respeitadas antes de qualquer processamento.

CRM – Contador de células RM faltantes - Limita o número de células RM para frente que podem ser enviadas na ausência do recebimento de uma célula RM para trás.

D

Deadlock – Condição que se verifica quando dois ou mais processos, que utilizam um ou mais recursos em comum, não conseguem definir qual processo deve utilizar-se do recurso primeiro, ficam parados esperando uma mudança na situação.

Diagrama de Transição de Estados – Fornecem um meio de representação gráfica de autômatos, utilizando círculos, que denotam os estados, e arcos, que denotam eventos.

DIR – É um campo da célula RM que indica a direção da célula RM. Célula enviada pela fonte é DIR = 0 e célula enviada pelo destino é DIR = 1

DTE – Ver Diagrama de Transição de Estados.

E

ER - Explicit Rate - Taxa explicita é um campo da célula RM usado para limitar o ACR da fonte num determinado valor. É inicialmente setado pela fonte para uma taxa requisitada, e pode ser reduzida pela rede para um valor que esta pode suportar

Espaço – Corresponde ao conjunto de estados que o sistema pode apresentar numa representação por um espaço de estados.

Especificação – É a descrição do comportamento desejado de um sistema. Para alimentar uma ferramenta de verificação formal geralmente é necessário que as especificações sejam escritas numa linguagem adequada

Estado – É a representação de um determinado momento de um sistema. Cada estado contém informações estáticas do sistema, porém um conjunto de estados e suas transições descrevem o comportamento total do sistema

F

Ferramentas de projeto – São programas utilizados no desenvolvimento de um projeto a partir de especificações simples. Algumas ferramentas podem se utilizar de métodos formais para garantir que o sistema final terá o comportamento desejado.

G

GCRA - Generic Cell Rate Algorithm: Algoritmo genérico de taxa de células. - É usado para definir a conformidade em função do contrato de tráfego de uma conexão. Para cada chegada de célula o GCRA determina se a célula está ou não conforme o contrato de

tráfego. A função UPC pode implementar o GCRA ou um ou mais algoritmos para garantir a conformidade.

I

ICR - Initial Cell Rate - É um parâmetro do serviço ABR em células por segundo o qual a fonte deve adotar no inicio da transmissão após o estabelecimento da conexão e após um período de inatividade.

Invariantes – É um tipo de afirmativa que define uma relação entre variáveis, e que é sempre válida mesmo com mudanças nos valores destas variáveis.

ISO - International Organization for Standardization - Uma organização internacional para padronização, baseada em Genebra na Suíca.

L

Laço invariante – É um tipo de afirmativa que permanece verdadeira de uma repetição de um laço para outra.

M

MCR - Minimum Cell Rate - É um parâmetro do serviço ABR que especifica a taxa mínima de células que a fonte sempre pode mandar pois a rede deve surportar

Métodos formais – São utilizados para desenvolvimento e verificação de processos. São baseados em rigorosos métodos matemáticos geralmente encontrados na lógica matemática.

Model Checker – É a denominação dada um grupo de ferramentas de verificação formal que analisam um sistema ou processo verificando se atendem a especificações fornecidas pelo usuário. As características do processo ou sistema em análise, bem como as especificações a serem testadas, são inseridas na ferramenta através de uma linguagem adequada. A ferramenta converte as informações das características do processo numa máquina de estados finitos para então realizar a verificação formal.

N

Nrm – É um parâmetro do serviço ABR que indica o máximo de células que uma fonte pode mandar antes de enviar uma nova célula RM para frente.

O

OSI - Open Systems Interconnection - Um modelo de arquitetura em sete camadas para sistemas de comunicações desenvolvido pela ISO.

P

PCR - Peak Cell Rate - Indica a taxa de células que a fonte não deve exceder.

proof checker – É a denominação dada a um grupo de ferramentas de verificação formal que trabalha uma especificação de um sistema aplicando sucessivas regras de lógica matemática até que se verifique que os objetivos são atendidos

Protocolo – Um conjunto de regras e formatos (semantica e sintaxe) que determina o comportamento de entidades em cada camada no desempenho de suas funções.

R

RDF - Rate Decrease Factor: Parâmetro do serviço ABR que controla o descrescimo da taxa de transmissão.

RIF - Rate Increase Factor: Parametro do serviço ABR que controla o acrescimo da taxa de transmissão.

S

SCR - Sustainable Cell Rate - É o limite máximo médio da taxa de células conforme no ATM

SED - Ver Sistemas a Eventos Discretos

Semântica – Concernente à significação

Sintaxe – Trata da função e disposição de símbolos em sentenças e sentenças em expressões respeitando uma boa construção gramatical

Sistemas a Eventos Discretos (SED), correspondem a uma classe de sistemas que apresentam duas características principais: apresentam grandezas discretas e evolução regida por eventos instantâneos. É um sistema a estado discreto dirigido por eventos.

T

Theorem Prover – É uma denominação de um conjunto de ferramentas de verificação que, semelhante aos proof checkers, trabalham matematicamente uma especificação inicial buscando provar que ela atinge os objetivos. Estas ferramentas são mais automáticas do que os proof checker, tendo então a vantagem de precisar de menos esforço de interação humana, e a desvantagem de permitir pouco controle do encaminhamento da prova.

U

UBR - Unspecified Bit Rate - UBR é uma categoria de serviço do ATM que não especifica garantias de tráfego. Nenhum valor é definido para o CLR.

UPC - Usage Parameter Control - O parâmetro de controle de uso é definido como o conjunto de ações usados pela rede para monitorar e controlar o tráfego de conexões ATM, mantendo o QoS em função do que foi negociado.

V

VBR - Variable Bit Rate - Categoria de serviço do ATM que suporta taxa de bits variável.

Verificação – É a ação de formalmente mostrar que um sistema opera corretamente dentro de determinadas especificações. Para realizar uma verificação geralmente utiliza-se uma ferramenta de verificação.