Uma Ferramenta para Especificação e Geração de Modelos Markovianos

Morganna Carmem Diniz† Edmundo de Souza e Silva‡

† UFRJ, COPPE-Sistemas
‡ UFRJ, NCE e Dept. Ciência da Computação do IM

Modelos markovianos têm sido muito usados na análise de sistemas de computação / comunicação. Neste trabalho descrevemos uma ferramenta implementada em Prolog para a especificação de modelos de computação / comunicação a nível de sistemas e a geração da cadeia de Markov associada. Exemplos na área de protocolos de comunicação são apresentados para ilustrar a flexibilidade da ferramenta.

1 Introdução

Modelos markovianos têm sido muito usados na análise de sistemas de computação / comunicação ([3]). Em geral, o sistema é visto como uma coleção de estados e o seu comportamento é representado pela matriz de transição de estados do processo markoviano correspondente. É comum um sistema de complexidade mediana possuir milhares de estados o que onera muito a solução do modelo correspondente, a não ser que este modelo possua características particulares que permitam a obtenção da solução de maneira eficiente sem que seja necessária resolver a cadeia de Markov. Um exemplo de modelos com características especiais são os modelos de redes de filas que possuem solução em forma de produto ([4]).

Infelizmente são relativamente raros os resultados que levam a uma solução mais eficiente. Na maioria dos casos, o analista tem que escolher entre adotar hipóteses simplificadoras para tentar obter uma solução mais eficiente do modelo ou resolver numericamente um processo markoviano que captura maiores detalhes do comportamento do sistema. Na área de modelagem de confiabilidade por exemplo, é comum a representação do comportamento de falha / reparo de um sistema por uma cadeia de Markov ([3], [2]).

Com o decréscimo do custo de memória, o aumento da capacidade de processamento e avanços nas técnicas de solução de processos markovianos, modelos com dezenas de milhares

de estados podem atualmente ser resolvidos. Tais avanços abrem novas possibilidades, permitindo a representação mais sofisticada de esquemas complexos comumente encontrados em modelos de confiabilidade e modelos de arquiteturas paralelas e protocolos de comunicação. Como consequência, é crescente o interesse por ferramentas que permitam o analista descrever o sistema a ser modelado e resolver numericamente a cadeia de Markov correspondente.

Existem duas representações de um modelo: a representação do analista e a representação analítica. A primeira é tipicamente simbólica e próxima a representação *natural* do sistema. A segunda é uma representação de baixo nível, fornecida como entrada para o software de solução. Por exemplo, a representação de uma rede de computadores pode ser feita através do conjunto de filas de pacotes diante dos canais de comunicação da rede. O analista descreve o conjunto de filas e a conecção entre elas, taxas de chegada de mensagens nos canais, a rota a ser seguida pelas mensagens, etc. A segunda representação seria a matriz de transição de estados da cadeia de Markov do sistema de filas.

É claro que um problema importante na construção de uma ferramenta diz respeito ao projeto da interface com o analista. Fornecer uma interface que exija a descrição do sistema em termos da matriz de transição de estados tornaria a ferramenta quase que inútil, uma vez que é muito difícil se evitar erros na descrição de uma matriz com poucas dezenas de estados e praticamente impossível a descrição de uma matriz com dezenas de milhares de estados. É necessário, então que seja desenvolvida uma interface que permita a descrição do sistema em termos de primitivas de mais alto nível e que a tradução para a representação matricial seja feita automaticamente.

Várias ferramentas de software adotam uma interface talhada para uma determinada aplicação. Por exemplo, a ferramenta SAVE ([2]) fornece uma interface específica para a análise de confiabilidade. Entretanto, o software de solução desta ferramenta poderia ser usado para outras aplicações. Uma solução para a reutilização do software de solução seria o desenvolvimento de interfaces distintas para aplicações distintas. Tal solução não é nada eficiente, uma vez que a tradução de uma especificação de alto nível para a representação analítica não é, em geral, trivial. Outra solução é a adoção de uma linguagem de especificação que permita facilmente a descrição de modelos de diferentes aplicações. Por exemplo, redes de Petri estocásticas ([5]) fornecem esta flexibilidade.

Redes de Petri estocásticas permitem a descrição de modelos voltados a diferentes aplicações. O analista dispõe de primitivas simples para a descrição do modelo. Infelizmente, os modelos descritos em redes de Petri estocásticas são por demais complexos e longe de uma descrição *natural* do sistema sendo estudado. Com o objetivo de se obter flexibilidade para especificação de modelos obtidos de aplicações distintas e ainda manter a descrição próxima a representação *natural* do modelo, Berson *et al* ([1]) desenvolveram uma metodologia que permite que interfaces talhadas para aplicações específicas sejam desenvolvidas com pequeno esforço a partir de uma representação básica.

Nesta metodologia, modelos de sistemas são descritos em termos de *objetos* que interagem entre si através de troca de mensagens. Aplicações específicas são desenvolvidas a partir dos tipos de objetos usados nos modelos de aplicação. Novos domínios de aplicação são facilmente incorporados pela utilização de novos tipos de objetos que podem ser armazenados em uma

biblioteca de tipos de objetos.

O detalhamento da metodologia básica de descrição de modelos é apresentada em [1]. Neste trabalho estamos interessados no desenvolvimento de uma biblioteca de tipos de objetos para aplicações específicas, e em particular apresentamos exemplos de modelos de desempenho de protocolos de comunicação. Na seção 2 fazemos uma revisão da metodologia de modelagem de [1]. Na seção 3 apresentamos exemplos na área de análise de protocolos. Os exemplos servem para ilustrar a flexibilidade e aplicabilidade da metodologia nesta área de aplicação. Na seção 4 descrevemos resumidamente uma ferramenta sendo desenvolvida na UFRJ em um micro computador PC. Na seção 5 apresentamos nossas conclusões.

2 O Modelo

2.1 Descrição

Na metodologia proposta em [1] um modelo de um sistema é composto por um conjunto de componentes chamados de objetos. As interações entre os objetos são feitas por meio de troca de mensagens. Cada objeto é uma entidade com um estado interno que pode ser alterado com o tempo. O estado de um objeto pode mudar em consequência a um evento gerado pelo próprio objeto ou a uma mensagem recebida de um outro objeto.

O estado de um objeto determina os tipos de eventos que podem ocorrer e as taxas de ocorrência destes eventos. Um evento em um objeto pode simplesmente mudar o estado interno deste objeto sem afetar nenhum outro, mas, em geral, também provocará alguma reação em outros objetos. Isto é modelado pela geração e envio de mensagens que poderão provocar alguma mudança de estado do objeto receptor. Portanto, a especificação de um objeto inclui a definição dos eventos que ele pode gerar, as ações tomadas quando da ocorrência de um evento (e.g., envio de mensagens) e a descrição de como o objeto reage ao recebimento de mensagens.

O estado do sistema é dado pelo conjunto de estados dos objetos e pelo conjunto de mensagens ainda não entregues no sistema. As mensagens são um abstração introduzida no modelo. Elas são entregues em tempo zero, portanto é nulo o tempo de reação do objeto a uma mensagem recebida. Desta forma alguns estados são transitórios e denominados de *evanescentes*. Estados que possuem tempo de permanência positiva são denominados de *tangíveis*. Os estados *evanescentes* são aqueles com uma ou mais mensagens não entregues e estados *tangíveis* são aqueles com nenhuma mensagem em trânsito.

Para propósito de análise, apenas os estados *tangíveis* são considerados. Uma sequência *evanescente* é uma sequência de transições de estados que começa e termina em um estado *tangível* onde todos os estados intermediários são *evanescentes*. As ações que podem ocorrer em resposta a um evento ou a uma mensagem determinam conjuntos de sequências *evanescentes*. Uma vez que todos os conjuntos de sequências *evanescentes* são determinados, as taxas de transição para cada par de estados *tangíveis* podem ser facilmente achadas. Estas

taxas formam a matriz de transição de estados.

Formalmente, um objeto θ é definido por uma n-upla:

$$\theta \equiv (I, S, S_0, \varepsilon, M^r, M^s, \delta', \delta)$$

onde:

- I = nome do objeto;
- S = conjunto de possíveis estados do objeto;
- $S_0 \in S$ = estado inicial do objeto;
- ε = conjunto de eventos que podem ser gerados pelo objeto;
- M^r = conjunto de mensagens que podem ser recebidas pelo objeto;
- M^s = conjunto de mensagens que podem ser enviadas pelo objeto;
- R = a função taxa do objeto

  $$R : S \times \varepsilon \rightarrow \Re^+$$

  dado um estado $s \in S$ e um evento $e \in \varepsilon$, esta função fornece a taxa na qual o evento e ocorre no estado S;
- δ' = a função evento do objeto

  $$\delta' : S \times \epsilon \rightarrow \{(0,1] \times S \times [M^s]\}$$

  dado um estado s e um evento e, esta função retorna um conjunto de possíveis respostas, onde cada resposta é composta de um novo estado, uma lista ordenada de mensagens a serem entregues e a probabilidade desta resposta ocorrer;
- δ = função mensagem do objeto

  $$\delta : S \times M^r \rightarrow \{(0,1] \times S \times [M^s]\}$$

  dado um estado s e uma mensagem m, esta função retorna o conjunto de possíveis respostas, onde cada resposta consiste de um novo estado, uma lista ordenada de mensagens e a probabilidade desta resposta ocorrer.

2.2 Implementação

A ferramenta de software em desenvolvimento permite a especificação de modelos e geração da cadeia de Markov correspondente. Basicamente, a matriz de transição de estados é obtida pela geração dos estados alcançáveis a partir de um estado inicial e das regras que descrevem o comportamento do sistema. As regras fornecem as condições para que determinadas ações sejam tomadas a partir de certos estados de um objeto.

A ferramenta é organizada em quatro níveis como mostra a figura 1.

1. o núcleo aceita a descrição do sistema em termos de objetos, eventos e mensagens. A partir desta descrição e de um estado inicial, o núcleo gera a matriz de transição de estado do sistema.
2. Objetos podem ter o mesmo comportamento *básico*, diferindo apenas no valor dos parâmetros especificados para cada objeto. No nível de definição do tipo de objeto, diferentes tipos de objetos são especificados. Um determinado modelo pode conter mais de uma ocorrência do mesmo tipo de objeto. Objetos diferentes mas do mesmo tipo terão parâmetros diferentes como por exemplo, taxas de eventos, valores de variáveis de condições, etc. Neste nível pode se criar uma biblioteca de tipos de objetos, a ser utilizada pelos níveis superiores para definir um modelo.
3. No nível de aplicação o usuário define um modelo criando objetos utilizando-se da biblioteca de tipos de objetos no nível abaixo e especificando os parâmetros necessários a cada objeto.
4. O nível de interface com o usuário permite que o usuário se abstraia dos níveis anteriores. Uma linguagem de alto nível pode ser definida para se fazer uso dos objetos existentes talhados para um determinado domínio de aplicação. Na seção 4 descrevemos sucintamente o nível de aplicação da ferramenta sendo desenvolvida na UFRJ.

A linguagem escolhida para a implementação da ferramenta foi Prolog. São três os motivos principais da escolha de Prolog ([1]): primeiro, Prolog não exige a especificação dos tipos de dados utilizados, permitindo inteira liberdade na descrição dos estados dos objetos; segundo, Prolog fornece *unificação*, uma forma poderosa de inicialização de variáveis e casamento de padrões usada na verificação das pré-condições; e terceiro, Prolog possui *backtrack*: quando mais de uma regra tem suas pré-condições satisfeitas ao mesmo tempo, todas elas são testadas para achar todos os estados alcançáveis.

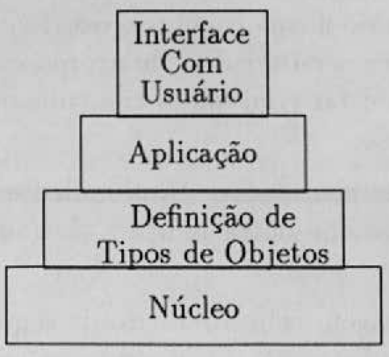

Figura 1: Níveis do Sistema

3 Exemplos

Nesta seção mostraremos alguns exemplos do uso desta ferramenta. No primeiro exemplo mostraremos o protocolo do bit alternante que, por ser bastante simples, servirá para ilustrar a metodologia; no segundo exemplo o protocolo do go back n e por último o protocolo ABRACADABRA.

3.1 Bit Alternante

Considere a definição do protocolo do bit alternante utilizada em [5]:

> Mensagens chegam exponencialmente com média $1/\lambda$ no emissor que possui capacidade de guardar apenas uma mensagem. Mensagens que chegam quando o buffer está ocupado são descartadas. Junto à cada mensagem transmitida é incorporado um bit 0 ou 1. A uma mensagem transmitida *pela primeira vez* é incorporado um bit de valor diferente da última mensagem transmitida. Ao receber uma mensagem o receptor envia a confirmação (ack) e quando da chegada do ack o emissor libera o buffer. Os canais podem, com uma certa probabilidade, transmitir com erro. Mensagens ou acks com erro de transmissão são descartados.

O sistema pode ser modelado por quatro objetos: EMISSOR, RECEPTOR e dois canais, CANAL1 que leva mensagens do emissor ao receptor e CANAL2 que leva acks do receptor ao emissor. Os objetos CANAL1 e CANAL2 representam um canal *full duplex*, onde mensagens são transmitidas em ambas as direções de forma independente. Os tempos de transmissão de mensagens ou confirmações são variáveis aleatórias com distribuição exponencial e parâmetro μ_2 e μ_3 respectivamente. Quando chega mensagem no EMISSOR, ela é colocada no CANAL1 para ser entregue ao RECEPTOR. Após um período de espera pelo ack (time-out), o EMISSOR retransmite a mensagem.

O EMISSOR é responsável pela geração de pacotes e recuperação de erros. Identificamos então dois eventos: geração de novos pacotes e ocorrência de time-outs. Para cada evento, diferentes ações podem ser tomadas, de acordo com o estado do EMISSOR. Por exemplo, um novo pacote faz com que o transmissor gere uma mensagem para o objeto canal, para transmissão.

O EMISSOR entende de 2 tipos de mensagens. Uma indica que o canal transmissor está livre para transmitir novo pacote, e outra indica a chegada de um ack vindo do canal receptor.

O RECEPTOR ao receber a mensagem com o número de sequência desejado, coloca o ack da mensagem no CANAL2. Este ack corresponde ao número de sequência da próxima mensagem a ser enviada pelo EMISSOR. Mensagens fora de sequência são descartadas e o ack da última mensagem correta é retransmitido.

É importante ressaltar que o mapeamento da descrição original do protocolo em quatro objetos é arbitrário. O mesmo comportamento pode ser descrito definindo-se outros tipos de objetos.

A descrição detalhada do EMISSOR é apresentada abaixo:

```
TIPO : transm_1_buffer;
NOME : nome_objeto;
ESTADO : (buffer,Num_Seg,Num_Pend,Sit_Canal,timeout);
EVENTO chegada de mensagens: (buffer,Num_Seg,Num_Pend,Sit_Canal,timeout) →
                             (buffer2,Num_Seg,Num_Pend2,Sit2_Canal,timeout);
        CONDIÇÃO : buffer = vazio,
                   Sit_Canal = livre;
        AÇÃO :     buffer2 = ocupado,
                   Num_Pend2 = [ ],
                   Sit2_Canal = transmitindo,
                   taxa_chegada(nome_objeto,T),
                   caminho(nome_objeto, canal, destino),
                   envia_mensagem(canal,Num_Seg);
        TAXA :     T;
        CONDIÇÃO : buffer = vazio,
                   Sit_Canal = tranmitindo;
        AÇÃO :     buffer2 = ocupado,
                   Num_Pend2 = Num_Seg,
                   Sit2_Canal = transmitindo,
                   taxa_chegada(nome_objeto,T),
        TAXA :     T;
EVENTO timeout : (buffer,Num_Seg,Num_Pend,Sit_Canal,timeout) →
                 (buffer,Num_Seg,Num_Pend,Sit2_Canal,timeout2);
        CONDIÇÃO : timeout = 1;
        AÇÃO :     Sit2_Canal = transmitindo,
                   timeout2 = 0,
                   taxa_timeout(T),
                   caminho(nome_objeto, canal, destino),
                   envia_mensagem(canal,Num_Seg);
        TAXA :     T;
MENSAGEM livre : (buffer,Num_Seg,Num_Pend,Sit_Canal,timeout) →
                 (buffer,Num_Seg,Num_Pend2,Sit2_Canal,timeout2);
            CONDIÇÃO : buffer = ocupado,
                       Num_Pend ≠ [ ];
            AÇÃO :     Num_Pend2 = [ ],
                       Sit2_Canal = transmitindo,
                       timeout2 = 0,
                       caminho(nome_objeto, canal, destino),
                       envia_mensagem(canal,Num_Seg);
            CONDIÇÃO : buffer = ocupado,
                       Num_Pend = [ ];
            AÇÃO :     Num_Pend2 = [ ],
                       Sit2_Canal = livre;
                       timeout2 = 1;
            CONDIÇÃO : buffer = vazio;
            AÇÃO :     Num_Pend2 = [ ],
                       Sit2_Canal = livre;
                       timeout2 = 0;
MENSAGEM ack : (buffer,Num_Seg,Num_Pend,Sit_Canal,timeout) →
               (buffer2,ack,Num_Pend,Sit_Canal,timeout2);
            CONDIÇÃO : ack = Num_Seg;
            AÇÃO :     buffer2 = ocupado,
                       timeout2 = timeout;
            CONDIÇÃO : ack = (Num_Seg + 1) mod 2;
            AÇÃO :     buffer2 = vazio,
                       timeout2 = 0;
```

O exemplo começa pela definição do tipo de objeto como sendo transm_1_buffer. A cláusula NOME fornece o nome da variável a ser substituida pelo nome do objeto quando da sua utilização. A cláusula ESTADO define os componentes que formam o estado do objeto. Neste exemplo o estado é definido por (buffer, Num_Seg, Num_Pend, Sit_Canal, timeout), onde buffer para transmissão é *ocupado* quando o EMISSOR possui uma mensagem cujo recebimento ainda não foi confirmado pelo RECEPTOR e é *vazio* quando está pronto para receber uma nova mensagem; Num_Seg é o número de sequência da próxima mensagem a ser enviada; Num_Pend indica o número de

sequência de uma mensagem para ser transmitida logo que o canal desocupe ou é [ ] quando não exite mensagem pendente; Sit_Canal é *livre* quando CANAL1 está desocupado e é *transmitindo* quando existe mensagem no canal sendo transmitida; e timeout é 1 quando retransmissão da última mensagem pode ocorrer e é 0 no caso contrário.

Este tipo de objeto pode gerar dois eventos: chegada de novas mensagens e timeout. A cláusula CONDIÇÃO estabelece as condições sobre as quais o evento pode ocorrer. O primeiro evento pode ocorrer com duas condições: o buffer está vazio (buffer = vazio) e o canal está desocupado (Sit_Canal = livre) ou o buffer está vazio (buffer = vazio) e o canal está transmitindo alguma mensagem (Sit_Canal = transmitindo). A cláusula AÇÃO descreve as ações do objeto quando o evento ocorre. Na primeira condição chegadas de novas mensagens são inibidas (buffer = ocupado) e a mensagem é colocada no canal para ser transmitida. Na segunda condição a mensagem tem que esperar para ser transmitida logo que CANAL1 desocupe (Num_Pend2 = Num_Seg) e o buffer fica ocupado (buffer = ocupado). Caminho(nome_objeto, canal, destino) é a leitura de um parâmetro do nível de aplicação que fornece o nome do canal e do objeto receptor da mensagem. A taxa de chegada é dada pelo parâmetro taxa_chegada(nome_objeto,T). A taxa de ocorrência deste evento é igual a taxa de chegada das novas mensagens.

No segundo evento é feita a retransmissão das mensagens após um período de espera pelo ack. Este evento só ocorre quando timeout foi habilitado. Note que o timeout só é habilitado quando não existe mensagem pendente a ser transmitida e o buffer de retransmissão está ocupado. A taxa de ocorrência deste evento é dada por taxa_timeout(T).

Dois tipos de mensagens podem ser recebidos pelo objeto: *livre* e *ack*. A primeira proveniente do objeto CANAL1 indica que este está livre para novas transmissões. Ao receber a mensagem *livre* uma nova mensagem é enviada se Num_Pend for diferente de [ ]. A segunda proveniente do CANAL2, indica a chegada de um ack sem erro. Quando uma mensagem é corretamente recebida o ack é igual ao número da próxima mensagem a ser enviada (ack = (Num_Seg + 1) mod 2). Qualquer outro tipo de mensagem recebida é ignorado pelo emissor.

O nível de aplicação para um objeto do tipo acima descrito poderia ser:

```
TIPO(emissor,transm_1_buffer).

INICIAL(emissor,(vazio,0,[ ],livre,0)).

TAXA_CHEGADA(emissor,20).

TAXA_TIMEOUT(1).

CAMINHO(emissor, canal1, receptor).
```

A cláusula TIPO especifica o nome do objeto (emissor) e o seu tipo correspondente (transm_1_buffer). A cláusula INICIAL define o estado inicial do objeto (buffer = vazio, Num_Seg = 0, Num_Pend = [ ], Sit_Canal = livre e timeout = 0). As cláusulas TAXA_CHEGADA e TIMEOUT determinam as taxas de chegada e de retransmissão das mensagens. E a cláusula CAMINHO define o caminho seguido pela mensagem quando da sua transmissão.

O estado do RECEPTOR é representado pelo número de sequência da próxima mensagem a ser recebida, pelo número de sequência do último ack transmitido e por uma variável indicando o estado do canal: *livre* quando CANAL2 está desocupado e *transmitindo* quando CANAL2 está com alguma mensagem.

```
* definição do tipo de objeto recep
TIPO : recep;
NOME : nome_objeto;
ESTADO : (Num_Seg,ultimo_ack,Sit_Canal);
MENSAGEM mens : (Num_Seg,ultimo_ack,Sit_Canal) → (Num2_Seg,ultimo2_ack,Sit2_Canal);
            CONDIÇÃO: Num_Seg = mens,
                      Sit_Canal = livre;
            AÇÃO:     Num2_Seg = (Num_Seg + 1) mod 2,
                      ultimo2_ack = Num2_Seg,
                      Sit2_Canal = transmitindo,
                      caminho(nome_objeto, canal, destino),
                      envia_mensagem(canal,Num2_Seg);
            CONDIÇÃO: Num_Seg = mens,
                      Sit_Canal = transmitindo;
            AÇÃO:     Num2_Seg = (Num_Seg + 1) mod 2;
                      ultimo2_ack = ultimo_ack,
                      Sit2_Canal = transmitindo;
            CONDIÇÃO: Num_Seg ≠ mens,
                      Sit_Canal = livre;
            AÇÃO:     Num2_Seg = Num_Seg,
                      ultimo2_ack = ultimo_ack,
                      Sit2_Canal = transmitindo,
                      caminho(nome_objeto, canal, destino),
                      envia_mensagem(canal,Num_Seg);
MENSAGEM livre : (Num_Seg,ultimo_ack,Sit_Canal) → (Num2_Seg,ultimo2_ack,Sit2_Canal);
            CONDIÇÃO: Num_Seg ≠ ultimo_ack;
            AÇÃO:     Sit2_Canal = transmitindo,
                      ultimo2_ack = Num_Seg,
                      caminho(nome_objeto, canal, destino),
                      envia_mensagem(canal,Num_Seg);
            CONDIÇÃO: Num_Seg = ultimo_ack;
            AÇÃO:     Sit2_Canal = livre;
                      ultimo2_ack = ultimo_ack,
```

O estado de cada canal é [ ] quando não estiver transmitindo mensagem ou indica o número de sequência da mensagem sendo transmitida.

CANAL1 e CANAL2 quando ocupados (N ≠ [ ]) transmitem a mensagem com uma determinada probabilidade de erro. A probabilidade de erro, a taxa de transmissão e o nome do objeto que receberá a mensagem são dados por prob_de_falhar(nome_objeto, prob), taxa(nome_objeto, T) e caminho(origem, nome_objeto, destino) respectivamente.

O objeto RECEPTOR ao receber uma mensagem com o número de sequência esperado (Num_Seg = mens) e estando o CANAL2 desocupado (Sit_Canal = livre), atualiza o código da próxima mensagem a ser recebida (Num2_Seg = (Num_Seg + 1) mod 2) e transmite o ack. Se o CANAL2 já tiver transmitindo um ack a variável ultimo_ack não é atualizada, e logo que o canal fique livre o ack é transmitido. Caso a mensagem recebida tenha uma sequência diferente, ela é descartada e o número de sequência esperado é retransmitido se o CANAL2 estiver livre. Qualquer outro tipo de mensagem é ignorado.

```
* definição do tipo de objeto canal
TIPO : canal;
```

```
NOME : nome_objeto;
ESTADO : (N);
EVENTO transmissão : N → N1;
        CONDIÇÃO : N ≠ [ ];
        AÇÃO :     taxa(nome_objeto,T),
                   prob_de_falhar(canal,prob),
                   caminho(origem, nome_objeto, destino),
                   envia_mensagem(destino,ERRO),
                   envia_mensagem(origem,LIVRE),
                   N1 = [ ];
        TAXA :     prob * T;
        CONDIÇÃO : N ≠ [ ];
        AÇÃO :     taxa(nome_objeto,T),
                   prob_de_falhar(canal,prob),
                   caminho(origem, nome_objeto, destino),
                   envia_mensagem(destino,N),
                   envia_mensagem(origem,LIVRE),
                   N1 = [ ];
        TAXA :     (1 - prob) * T;
MENSAGEM M: N → N1;
        CONDIÇÃO : N = [ ];
        AÇÃO : N1 = M;

definição do nível de aplicação
TIPO(receptor, recep).
TIPO(canal1, canal).
TIPO(canal2, canal).

INICIAL(receptor,(0,0,livre)).
INICIAL(canal1,[ ]).
INICIAL(canal2,[ ]).

CAMINHO(emissor, canal1, receptor).
CAMINHO(receptor, canal2, emissor).

TAXA(canal1,μ2).
TAXA(canal2,μ3).

PROB_DE_FALHAR(canal1,0.05).
PROB_DE_FALHAR(canal2,0.05).
```

3.2 Go Back N

Considere o protocolo Go Back N como descrito em [6]:

> Mensagens são numeradas de 0 a 2N - 1. Mensagens chegam exponencialmente com média $1/\lambda$ no emissor que possui capacidade de guardar apenas N mensagens. Mensagens que chegam quando o buffer está ocupado são descartadas. O emissor pode transmitir até N mensagens antes que fique bloqueado por falta de acks. Todas as mensagens transmitidas após uma ocorrência de erro no canal são descartadas pelo receptor até que a mensagem que falta seja retransmitida. Ao receber um ack o emissor considera que todas as mensagens anteriores a este ack foram corretamente recebidas pelo receptor. Mensagens ou acks com erro na transmissão são descartados.

Similarmente ao exemplo anterior, o sistema pode ser modelado por quatro objetos: EMISSOR, RECEPTOR, CANAL1 e CANAL2. O estado do objeto EMISSOR é representado pelo número de mensagens no buffer de retransmissão (buffer); pelo número de sequência do último ack recebido (Num_Ack); pelo número de sequência da próxima mensagem a ser enviada logo que o canal desocupe (é igual a [ ] quando não existe

mensagem pendente); pela situação de CANAL1 (*livre / transmitindo*); e por um indicador (timeout) que é 1 quando o timeout está habilitado e é 0 no caso contrário. O estado do RECEPTOR é dado pelo número de sequência da próxima mensagem a ser recebida, pelo número de sequência do último ack enviado e pela situação de CANAL2 (*livre / transmitindo*).

O objeto EMISSOR ao receber a mensagem *livre* do CANAL1 transmite a mensagem marcada em Num_Pend, se houver, e habilita o evento *timeout* (timeout = 1). Ao receber um ack, o EMISSOR diminui do buffer o número de mensagens esperando confirmação. O EMISSOR pode gerar 2 tipos de eventos: chegada de novas mensagens e *timeout*. O evento *timeout* nada mais é do que uma indicação para a retransmissão das mensagens que ainda não receberam ack: o número de sequência da próxima mensagem passa a ser igual ao número de sequência do último ack recebido.

O objeto RECEPTOR ao receber uma mensagem com o número de sequência desejado (mens = Num_Pend) transmite o ack se o CANAL2 estiver livre. Mensagens recebidas com número de sequência diferente do desejado são descartadas e o último ack é retransmitido se CANAL2 estiver desocupado. Mensagens recebidas pelo RECEPTOR quando CANAL2 está ocupado são aceitas se o número de sequência está correto, mas somente o ack da última mensagem é transmitida quando o canal fica livre. Observe que este objeto é semelhante ao objeto RECEPTOR do exemplo anterior. A única diferença consiste que o RECEPTOR do GO_BACK(N) se utiliza de uma sequência de mensagem de módulo 2N no lugar de módulo 2. Portanto não iremos aqui repetir o tipo definido anteriormente.

CANAL1 e CANAL2 são idênticos ao do exemplo anterior.

```
* definição do tipo de objeto transm_N_buffers
TIPO : transm_N_buffers;
NOME : nome_objeto;
ESTADO : (buffer,Num_Ack,Num_Pend,Sit_Canal,timeout);
EVENTO chegada : (buffer,Num_Ack,Num_Pend,Sit_Canal,timeout) →
                 (buffer2,Num_Ack,Num_Pend2,Sit2_Canal,timeout);
        CONDIÇÃO : go_back(N),                    * lê parâmetro
                   buffer < N,
                   Num_Pend ≠ [ ],
                   Sit_Canal = livre;
        AÇÃO :     buffer2 = buffer + 1,
                   Num_Pend2 = (Num_Pend + 1) mod (2 * N),
                   Sit2_Canal = transmitindo,
                   taxa_chegada(nome_objeto,T),
                   caminho(nome_objeto, canal, destino),
                   envia_mensagem(canal, Num_Pend);
        TAXA :     T;
        CONDIÇÃO : go_back(N),
                   buffer < N,
                   Num_Pend = [ ],
                   Sit_Canal = livre;
        AÇÃO :     buffer2 = buffer + 1,
                   Num_Pend2 = (Num_Ack + 1) mod (2 * N),
                   Sit2_Canal = transmitindo,
                   taxa_chegada(nome_objeto,T),
                   caminho(nome_objeto, canal, destino),
                   envia_mensagem(canal, Num_Pend);
        TAXA :     T;
        CONDIÇÃO : go_back(N),
                   buffer < N,
                   Sit_Canal = transmitindo;
        AÇÃO :     buffer2 = buffer + 1,
                   Num_Pend2 = Num_Pend,
```

```
                    Sit2_Canal = transmitindo;
                    taxa_chegada(nome_objeto,T);
          TAXA :    T;
EVENTO timeout : (buffer,Num_Ack,Num_Pend,Sit_Canal,timeout) →
                 (buffer,Num_Ack,Num_Pend2,Sit2_Canal,timeout2);
          CONDIÇÃO : buffer > 0,
                    timeout = 1,
                    Sit_Canal = livre;
          AÇÃO :    Sit2_Canal = transmitindo,
                    timeout2 = 0,
                    taxa_timeout(T),
                    go_back(N),
                    Num_Pend2 = (Num_Ack + 1) mod (2 * N),
                    caminho(nome_objeto, canal, destino),
                    envia_mensagem(canal, Num_Ack));
          TAXA :    T;
          CONDIÇÃO : buffer > 0,
                    timeout = 1,
                    Sit_Canal = transmitindo;
          AÇÃO :    Sit2_Canal = transmitindo,
                    timeout2 = 0,
                    Num_Pend2 = Num_Ack,
                    taxa_timeout(T);
          TAXA :    T;
MENSAGEM livre : (buffer,Num_Ack,Num_Pend,Sit_Canal,timeout) →
                 (buffer,Num_Ack,Num_Pend2,Sit2_Canal,timeout2);
               CONDIÇÃO : buffer > 0,
                          Num_Pend ≠ [ ],
                          go_back(N),
                          (Num_Pend - Num_Ack) mod 2 * N < buffer;
               AÇÃO :     Sit2_Canal = transmitindo,
                          Num_Pend2 = (Num_Pend + 1) mod (2 * N),
                          timeout2 = 1,
                          caminho(nome_objeto, canal, destino),
                          envia_mensagem(canal, Num_Pend);
               CONDIÇÃO : buffer > 0,
                          go_back(N),
                          Num_Pend = [ ];
               AÇÃO :     Sit2_Canal = transmitindo,
                          Num_Pend2 = (Num_Ack + 1) mod (2 * N),
                          timeout2 = 0,
                          caminho(nome_objeto, canal, destino),
                          envia_mensagem(canal, Num_Ack);
               CONDIÇÃO : buffer > 0;
                          Num_Pend ≠ [ ],
                          go_back(N),
                          (Num_Pend - Num_Ack) mod 2 * N ≥ buffer;
               AÇÃO :     Sit2_Canal = livre,
                          Num_Pend2 = Num_Pend,
                          timeout2 = 1;
               CONDIÇÃO : buffer = 0;
               AÇÃO :     Sit2_Canal = livre,
                          Num_Pend2 = [ ],
                          timeout2 = 0;
MENSAGEM ack : (buffer,ultimo_ack,Num_Pend,Sit_Canal,timeout) →
               (buffer2,novo_ack,Num_Pend2,Sit_Canal,timeout2);
               CONDIÇÃO : go_back(N),
                          buffer - (ack - ultimo_ack) mod 2 * N ≠ 0;
               AÇÃO :     go_back(N),
                          buffer2 = buffer - (ack - ultimo_ack) mod 2 * N,
                          novo_ack = ack,
                          Num_Pend2 = Num_Pend,
                          timeout2 = timeout;
               CONDIÇÃO : go_back(N),
                          buffer - (ack - ultimo_ack) mod 2 * N = 0;
               AÇÃO :     buffer2 = 0,
                          novo_ack = ack,
                          Num_Pend2 = [ ],
                          timeout2 = 0;
```

```
* definição do nível de aplicação
GO_BACK(2).

TIPO(emissor,transm_N_buffers).
TIPO(receptor,recep_N).
TIPO(canal1, canal).
TIPO(canal2, canal).

INICIAL(emissor,(0,0,[ ],livre)).
INICIAL(receptor,(0,0,livre)).
INICIAL(canal1,[ ]).
INICIAL(canal2,[ ]).

CAMINHO(emissor, canal1, receptor).
CAMINHO(receptor, canal2, emissor).

TAXA_CHEGADA(emissor,λ).
TAXA_TIMEOUT(μ1).
TAXA(canal1,μ2).
TAXA(canal2,μ3).

PROB_DE_FALHAR(canal1,p).
PROB_DE_FALHAR(canal2,k).
```

É fácil verificar a semelhança entre as especificações do EMISSOR do bit alternante e do go_back_N. As diferenças são devidas a que, no go_back_N, podemos ter uma fila para transmissão de mensagens, e existe a possibilidade de ocorrer timeout antes de N mensagens serem transmitidas. A cláusula GO_BACK(N) especifica o número máximo de mensagens que o emissor transmite antes de se bloquear. Para N = 1, temos o bit alternate descrito na seção anterior. Observe que podemos conseguir o protocolo go back N para vários valores de N pela simples mudança do valor de N na cláusula GO_BACK(N). Isto pode não ser possível em especificações feitas utilizando redes de Petri estocásticas, onde a representação gráfica do modelo pode variar com o valor de um parâmetro.

3.3 ABRACADABRA

Em [7] é proposto o protocolo ABRACADABRA que captura o comportamento básico do modelo OSI. Este protocolo tem importantes características dos protocolos normalizados como as fases de estabelecimento e de liberação de conexão.

A especificação do protocolo propõe uma conexão entre dois usuários UA e UB como mostrado na figura 2. O pedido para estabelecimento da conexão é feito por um usuário à entidade com a qual se conecta pelo canal UCEP. A entidade transmite o pedido para a outra entidade pelo uso de um canal bidirecional que possue uma determinada probabilidade de transmitir com erro. Confirmado o estabelecimento da conexão os dados são transmitidos. No término da transmissão dos dados é pedido o encerramento da conexão.

As unidades de dados do protocolo (ou PDUs) são de 3 tipos:

1. CR (pedido de conexão) e CC (confirmação de conexão) na fase de estabelecimento de conexão;
2. DT (dado) e AK (ack) na fase de transferência de dados;

3. DR (pedido de desconexão) e DC (confirmação de desconexão) na fase de liberação da conexão;

As PDU's DT e AK são enviadas junto com um número de sequência de módulo 2. A cada nova conexão esta sequência é inicializada.

As PDU's CR, DT e DR são retransmitidas pela entidade após um período de espera pela resposta. Um número máximo de tentativas por PDU é estabelecido pelo protocolo.

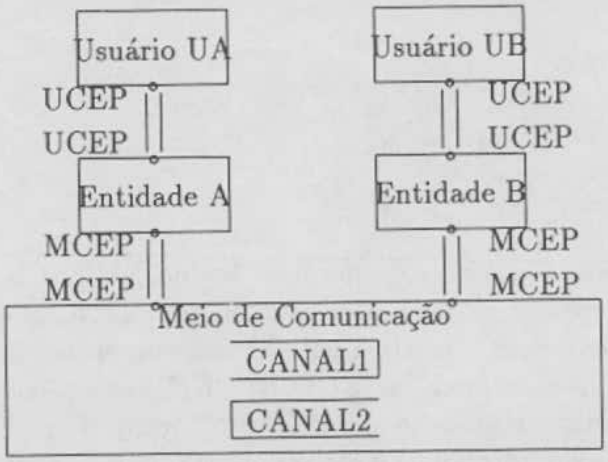

Figura 2: Arquitetura do protocolo ABRACADABRA

O sistema pode ser modelado por seis objetos relacionados aos módulos da figura 2 (entretanto, apenas três tipos de objetos são necessários): USUÁRIO_A, USUÁRIO_B, ENTIDADE_A, ENTIDADE_B, CANAL1 E CANAL2. O objeto USUÁRIO_A (USUÁRIO_B) inicializa pedindo a ENTIDADE_A (ENTIDADE_B) que estabeleça uma conexão com o USUÁRIO_B (USUÁRIO_A). Mensagens são enviadas da ENTIDADE_A (ENTIDADE_B) para a ENTIDADE_B (ENTIDADE_A) através do CANAL1 (CANAL2). Respostas da ENTIDADE_B (ENTIDADE_A) são transmitidas para a ENTIDADE_A (ENTIDADE_B) pelo CANAL2 (CANAL1). CANAL1 e CANAL2 são do mesmo tipo usado nos dois exemplos anteriores. Ao receber a confirmação da conexão a ENTIDADE_A (ENTIDADE_B) informa ao USUÁRIO_A (USUÁRIO_B) e este transmite os primeiros dados. A cada recebimento correto de um ack o USUÁRIO_A (USUÁRIO_B) é informado e novos dados são transmitidos ou o fechamento da conexão é pedido. Qualquer entidade pode pedir o fechamento da conexão se um erro for detectado. A ENTIDADE_A (ENTIDADE_B) retransmite a última PDU após um determinado período de espera. O número de retransmissões permitidas para cada PDU é dada pelo parâmetro NUM_RETRANSMISSÕES.

São estados possíveis para o usuário que está transmitindo uma mensagem:

1. (CLOSED, 0) - o componente está livre de qualquer conexão;
2. (OPEN, CR) - o componente está aguardando a resposta de um pedido de abertura de conexão;

3. (OPEN, AK) - o componente está esperando pelo ack de uma mensagem;
4. (OPEN, DT) - o componente pode transmitir novos dados ou fechar conexão;

O usuário que está recebendo uma mensagem possue apenas dois estados: (CLOSED, 0) e (OPEN,0). No primeiro estado o componente está livre de qualquer conexão e no segundo estado está aguardando o envio das mensagens. As três últimas definições de mensagens do objeto de tipo *usuário* (especificado abaixo) são utilizadas na recepção.

```
TIPO : usuário;
NOME : nome_objeto;
ESTADO : (M);
EVENTO abertura de conexão : (Sit) → (Sit2);
        CONDIÇÃO : Sit = (CLOSED,0);
        AÇÃO :     Sit2 = (OPEN, CR),
                   taxa_abertura(nome_objeto,T),
                   ucep(nome_objeto, destino),
                   envia_mensagem(destino,ucep(CR));
        TAXA :     T;
EVENTO transmite :  (OPEN,K) → (OPEN,K1);
        CONDIÇÃO : K = DT;
        AÇÃO :     K1 = AK,
                   taxa_processamento(nome_objeto,T),
                   prob_novos_dados(nome_objeto,prob),
                   ucep(nome_objeto,destino),
                   envia_mensagem(destino,ucep(DT));
        TAXA :     prob * T;
EVENTO fechamento de conexão : (OPEN,K) → (K1,0);
        CONDIÇÃO : K = DT;
        AÇÃO :     K1 = CLOSED,
                   taxa_processamento(nome_objeto,T),
                   prob_novos_dados(nome_objeto,prob),
                   ucep(nome_objeto,destino),
                   envia_mensagem(destino,ucep(DR));
        TAXA :     (1 - prob) * T;
MENSAGEM ucep(CC): (OPEN,K) → (OPEN,K1);
            CONDIÇÃO : K = DR;
            AÇÃO :     K1 = AK,
                       ucep(nome_objeto,destino),
                       envia_mensagem(destino,ucep(DT));
MENSAGEM ucep(AK): (OPEN,K) → (OPEN,K1);
            CONDIÇÃO : K = AK;
            AÇÃO :     K1 = DT;
MENSAGEM ucep(DR) : (OPEN) → (CLOSED);
MENSAGEM ucep(CR): (CLOSED,0) → (OPEN,0);
            AÇÃO : ucep(nome_objeto,destino),
                   envia_mensagem(destino,ucep(CC));
MENSAGEM ucep(DR): (OPEN,0) → (CLOSED,0);
            AÇÃO : ucep(nome_objeto,destino),
                   envia_mensagem(destino,ucep(DC));
MENSAGEM ucep(DT) : M → M;
```

São estados possíveis da entidade ABRACADABRA:

1. (CLOSED,0) - o componente está livre de qualquer conexão;
2. (WFCC,N) - o componente está esperando a resposta de um pedido por conexão (Wait For CC), onde N é o número de retransmissões feitas do CR;
3. (WFAK,M(N, E, R, K)) - o componente está esperando pelo ack do dado enviado (Wait For Ack); N é o número de retransmissões feitas do DT, E é o número de sequência do último dado transmitido, R é o número de sequência do último dado recebido e K é *true* quando nenhum DT ou AK foi recebido pelo componente;

4. (OPEN,M(0, E, R, K)) - o componente está esperando por dados;
5. (CLOSING,N) - o componente está esperando a resposta de um pedido por liberação de conexão, onde N é o número de retransmissões feitas do DR;
6. (WFUR,0) - o componente está esperando a resposta do usuário (Wait For User Response). Este estado é transitório.

```
* definição do tipo de objeto ent (entidade do ABRACADABRA)
TIPO : ent;
NOME : nome_objeto;
ESTADO : (M,N);
EVENTO timeout : (K1,N1) → (K2,N2);
        CONDIÇÃO : K1 = CLOSING,
                   num_retransmissões(nome_objeto,NUM),
                   N1 ≤ NUM;
        AÇÃO :     K2 = CLOSING,
                   N2 = N1 + 1,
                   taxa(nome_objeto,T),
                   mcep(nome_objeto, canal, destino),
                   envia_mensagem(canal,mcep(DR));
        TAXA :     T;
        CONDIÇÃO : K1 = WFCC,
                   num_retransmissões(nome_objeto,NUM),
                   N1 ≤ NUM;
        AÇÃO :     K2 = WFCC,
                   N2 = N1 + 1,
                   taxa(nome_objeto,T),
                   mcep(nome_objeto, canal, destino),
                   envia_mensagem(canal,mcep(CR));
        TAXA :     T;
        CONDIÇÃO : K1 = WFAK,
                   N1 = M(N,E,R,K),
                   num_retransmissões(nome_objeto,NUM),
                   N ≤ NUM;
        AÇÃO :     K2 = WFAK,
                   N2 = (N + 1,E,R,K),
                   taxa(nome_objeto,T),
                   mcep(nome_objeto, canal, destino),
                   envia_mensagem(canal,mcep(DT,E));
        TAXA :     T;
        CONDIÇÃO : K1 = WFCC,
                   num_retransmissões(nome_objeto,NUM),
                   N1 > NUM;
        AÇÃO :     K2 = CLOSING,
                   N2 = 0,
                   taxa(nome_objeto,T),
                   ucep(nome_objeto, destino1),
                   envia_mensagem(destino1,ucep(DR)),
                   mcep(nome_objeto, canal, destino2),
                   envia_mensagem(canal,mcep(DR));
        TAXA :     T;
        CONDIÇÃO : K1 = WFAK,
                   N1 = M(N,E,R,K),
                   num_retransmissões(nome_objeto,NUM),
                   N > NUM;
        AÇÃO :     K2 = CLOSING,
                   N2 = 0,
                   taxa(nome_objeto,T),
                   ucep(nome_objeto, destino1),
                   envia_mensagem(destino1,ucep(DR)),
                   mcep(nome_objeto, canal, destino2),
                   envia_mensagem(canal,mcep(DR));
        TAXA :     T;
        CONDIÇÃO : K1 = CLOSING,
                   num_retransmissões(nome_objeto,NUM),
                   N1 > NUM;
        AÇÃO :     K2 = CLOSED,
                   N2 = 0,
```

```
                    taxa(nome_objeto,T);
        TAXA :      T;
MENSAGEM ucep(CR): (CLOSED,0) → (WFCC,0);
        AÇÃO : mcep(nome_objeto, canal, destino),
               envia_mensagem(canal,mcep(CR));
MENSAGEM mcep(K): (WFCC,N) → (OPEN,M(0,0,0,true));
        CONDIÇÃO : K = CC ou K = CR;
        AÇÃO : ucep(nome_objeto,destino),
               envia_mensagem(destino,ucep(CC));
MENSAGEM mcep(DR): (WFCC,N) → (CLOSED,0);
        AÇÃO : ucep(nome_objeto,destino1),
               envia_mensagem(destino1,ucep(DR)),
               mcep(nome_objeto, canal, destino2),
               envia_mensagem(canal,mcep(DC));
MENSAGEM ucep(DR): (WFCC,N) → (CLOSING,0);
        AÇÃO : mcep(nome_objeto, canal, destino),
               envia_mensagem(canal,mcep(DR));
MENSAGEM mcep(CR): (CLOSED,0) → (WFUR,0);
        AÇÃO : ucep(nome_objeto,destino),
               envia_mensagem(destino,ucep(CR));
MENSAGEM mcep(DR): (CLOSED,0) → (CLOSED,0);
        AÇÃO : mcep(nome_objeto, canal, destino),
               envia_mensagem(canal,mcep(DC));
MENSAGEM ucep(CC): (WFUR,0) → (OPEN,M(0,0,0,true));
        AÇÃO : mcep(nome_objeto, canal, destino),
               envia_mensagem(canal,mcep(CC));
MENSAGEM ucep(DR): (WFUR,0) → (CLOSING,0);
        AÇÃO : mcep(nome_objeto, canal, destino),
               envia_mensagem(canal,mcep(DR));
MENSAGEM ucep(DT): (OPEN,M(N,E,R,K)) → (WFAK,M(0,E,R,K));
        AÇÃO : mcep(nome_objeto, canal, destino),
               envia_mensagem(canal,mcep(DT,E);
MENSAGEM mcep(AK,E2): (WFAK,M(N,E,R,K)) → (K2,N2);
        CONDIÇÃO : E2 = (E + 1) mod 2;
        AÇÃO :     K2 = OPEN,
                   N2 = M(0,E2,R,K),
                   ucep(nome_objeto,destino1),
                   envia_mensagem(destino1,ucep(AK));
        CONDIÇÃO : E2 ≠ (E + 1) mod 2;
        AÇÃO :     K2 = CLOSING,
                   N2 = 0,
                   ucep(nome_objeto,destino1),
                   envia_mensagem(destino1,ucep(DR)),
                   mcep(nome_objeto, canal, destino2),
                   envia_mensagem(canal,mcep(DR));
MENSAGEM mcep(DT,Q): (sit,M(N,E,R,K)) → (sit,M(N,E,R2,false));
        CONDIÇÃO : sit = WFAK ou sit = OPEN,
                   Q = R;
        AÇÃO :     R2 = (R + 1) mod 2,
                   ucep(nome_objeto,destino1),
                   envia_mensagem(destino1,ucep(DT)),
                   mcep(nome_objeto, canal, destino2),
                   envia_mensagem(canal,mcep(AK,R2)):
        CONDIÇÃO : sit = WFAK ou sit = OPEN,
                   Q ≠ R;
        AÇÃO :     R2 = R,
                   mcep(nome_objeto, canal, destino),
                   envia_mensagem(canal,mcep(AK,R));
MENSAGEM mcep(CR): (sit,M(N,E,R,true)) → (sit,M(N,E,R,true));
        CONDIÇÃO : sit = WFAK ou sit = OPEN;
        AÇÃO :     mcep(nome_objeto, canal, destino),
                   envia_mensagem(canal,mcep(CC));
MENSAGEM mcep(DR): (sit,M(N,E,R,K)) → (CLOSED,0);
        CONDIÇÃO : sit = WFAK ou sit = OPEN;
        AÇÃO :     ucep(nome_objeto,destino1),
                   envia_mensagem(destino1,ucep(DR)),
                   mcep(nome_objeto, canal, destino2),
                   envia_mensagem(canal,mcep(DC));
MENSAGEM ucep(DR): (sit,K) → (CLOSING,0);
```

```
            CONDIÇÃO : sit = WFAK ou sit = OPEN;
            AÇÃO :     mcep(nome_objeto, canal, destino),
                       envia_mensagem(canal,mcep(DR));
MENSAGEM mcep(M): (CLOSING,N) → (CLOSED,0);
            CONDIÇÃO : M = DC ou M = DR;
```

O nível de aplicação utiliza os objetos acima definidos. As entidades A e B utilizam o mesmo tipo de objeto (ent). É importante notar também a definição das ligações como mostra a Figura 2.

```
* definição do nável de aplicação
TIPO(usuário_A,usuário).
TIPO(usuário_B,usuário).
TIPO(entidade_A,ent).
TIPO(entidade_B,ent).
TIPO(canal1, canal).
TIPO(canal2, canal).

INICIAL(usuário_A,(closed,0)).
INICIAL(usuário_B,(closed,0)).
INICIAL(entidade_A,(closed,0)).
INICIAL(entidade_B,(closed,0)).
INICIAL(canal1,[ ]).
INICIAL(canal2,[ ]).

NUM_RETRANSMISSÕES(entidade_A,1).
NUM_RETRANSMISSÕES(entidade_B,1).

UCEP(usuário_A,entidade_A).
UCEP(usuário_B,entidade_B).

MCEP(entidade_A, canal1, entidade_B).
MCEP(entidade_B, canal2, entidade_A).

TAXA_ABERTURA(usuário_A,μ1).
TAXA_PROCESSAMENTO(usuário_A,μa).
TAXA(canal1,c1).
TAXA(canal2,c2).

PROB_NOVOS_DADOS(usuário_A,p1).
PROB_DE_FALHAR(canal1,p2).
PROB_DE_FALHAR(canal2,p3).
```

Observe que para o usuário_A foram especificadas a taxa de abertura de conexão, a taxa de processamento de mensagens e a probabilidade de transmitir novos dados após o recebimento do ack da última mensagem. Estes parâmetros não foram definidos para o usuário_B. Como consequência, o usuário_B não poderá iniciar uma conexão. Entretanto, esta é uma opção feita para diminuir o número de estados da cadeia de Markov, e pode portanto, ser facilmente alterada.

Como comentário final observamos que não houve preocupação na escolha das variáveis de estado de cada objeto (nos exemplos acima), de maneira a tornar a descrição de seu comportamento mais compacta. Outras descrições poderiam ser feitas desde que não alterassem o comportamento final do objeto.

4 Ferramenta de Análise

Para auxiliar o analista tanto na definição de modelos quanto na criação de novos tipos de objetos estamos desenvolvendo dois módulos de interface para o usuário: definição

de modelos e definição de tipos de objetos. No primeiro módulo temos a utilização da biblioteca de tipos de objetos existentes. Inicialmente o usuário fornece o nome do modelo que deseja especificar. Modelos anteriormente definidos podem ser alterados. Após especificar o nome do modelo são pedidos os nomes e tipos de objetos que compõem o sistema. Para cada objeto são solicitados os parâmetros correspondentes ao tipo especificado. No encerramento deste módulo é criado um arquivo com as cláusulas do nível de aplicação.

Um exemplo do uso desse módulo é mostrado na figura 3, onde são especificados os parâmetros para o EMISSOR do bit alternante discutido na seção anterior. Na primeira tela é pedido o nome do modelo. Na tela 2 o usuário entra com o nome de um componente e seu tipo correspondente. Se este componente já tiver sido definido, a tela 3 é mostrada com os parâmetros já preenchidos podendo serem modificados, de outra forma, a tela 3 é mostrada com os campos em branco.

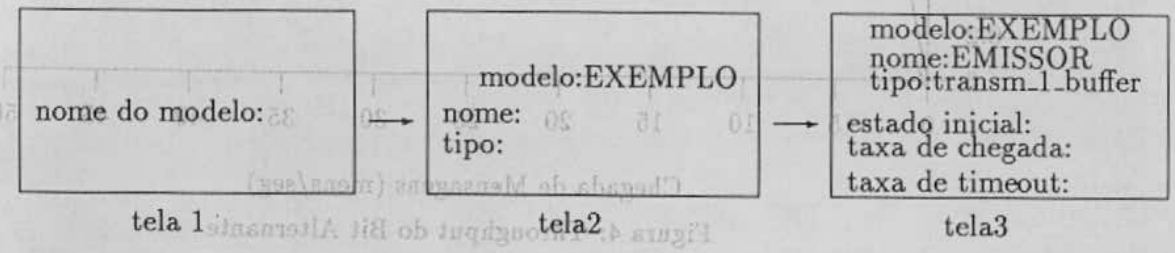

Figura 3: interface com o usuário

No segundo módulo é fornecida uma biblioteca de funções para serem usadas na definição de novos tipos de objetos. Este módulo tem por objetivo ajudar ao usuário que possui pouco conhecimento de Prolog. Cada função corresponde a uma ação a ser executada pelo objeto como por exemplo a leitura de um parâmetro ou a mudança de estado. Inicialmente o usuário fornece o nome do tipo de objeto a ser definido. Então, são solicitados os eventos que o objeto pode gerar e as mensagens que pode receber. Para cada evento e mensagem são pedidos: o estado inicial, o estado final, as condições para que ocorram e as funções utilizadas. No término deste módulo é criado um arquivo com a definição correspondente em Prolog.

Uma vez especificado o sistema, a cadeia de Markov pode ser gerada e os resultados analisados. Como exemplo, a cadeia de Markov gerada a partir do protocolo do bit alternante possui 28 estados. A cadeia de Markov pode ser gerada de forma simbólica, como indicado abaixo:

```
(EMISSOR,(vazio,0,[ ],livre,0)), (CANAL1,[ ]), (RECEPTOR,(0,0,livre)), (CANAL2,[ ])

                                    ⇓

(EMISSOR, (ocupado,0,[ ],transmitindo,0)), (CANAL1,0), (RECEPTOR,(0,0,livre)), (CANAL2,[ ]));
          taxa = taxa_chegada
```

onde o primeiro estado corresponde ao estado inicial do protocolo do bit alternante: o EMISSOR está com o buffer vazio, a próxima mensagem a ser transmitida terá bit

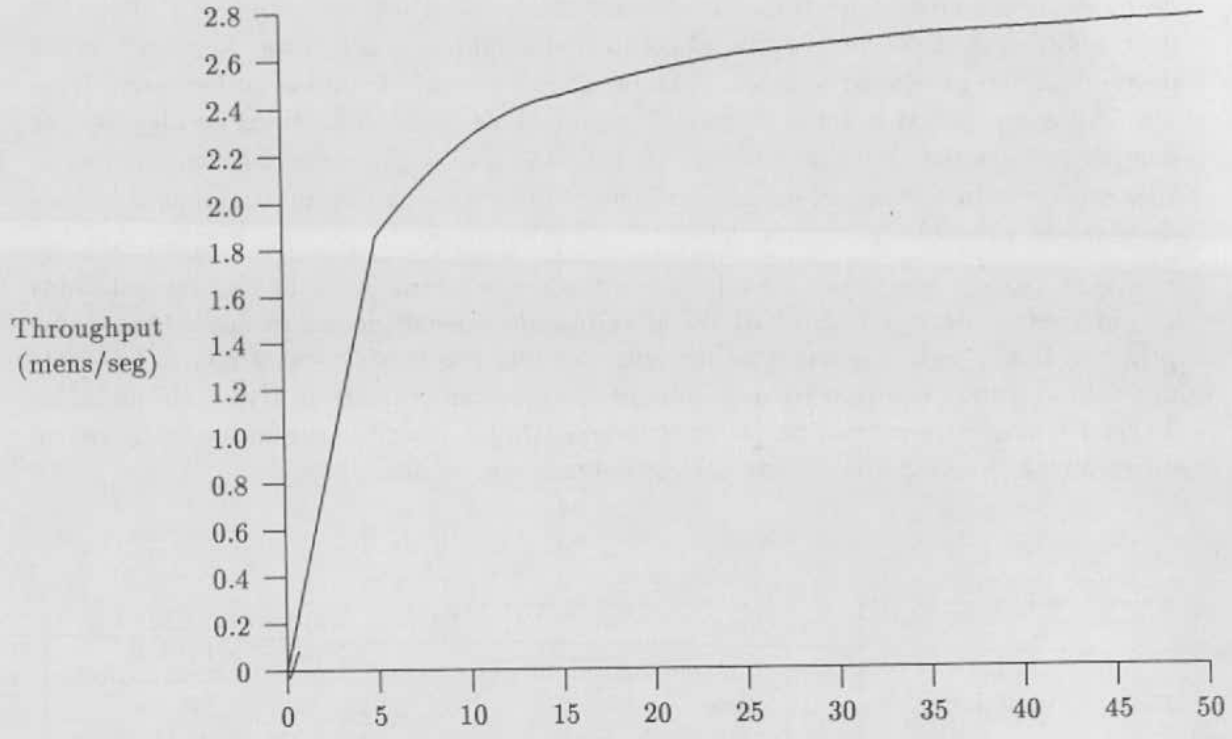

Figura 4: Throughput do Bit Alternante

0, não existe nenhuma mensagem esperando a liberação do canal para ser transmitida ([ ]), o canal que o EMISSOR utiliza está livre e o *timeout* não foi ativado; o CANAL1 está desocupado; o RECEPTOR está esperando por uma mensagem com bit 0, o último ack enviado pelo RECEPTOR tinha bit 0 e o canal que o RECEPTOR utiliza está livre; o CANAL2 está desocupado. Após a chegada de uma mensagem o sistema passa para o segundo estado com taxa *taxa_chegada*, onde o EMISSOR tem o buffer ocupado e o CANAL1 possui uma mensagem para transmitir.

Seja ρ a probabilidade do estado estacionário que o sistema esteja com o buffer cheio. No modelo as mensagens chegam com taxa Poisson λ e são descartadas se o buffer já está ocupado. Portanto o throughput do sistema é $\lambda(1 - \rho)$.

Considere um sistema que usa este protocolo e que tem canais com capacidade de 9600 e probabilidade de erro de 5 por cento e pacotes de 1024 bits. Assuma que a taxa de timeout é de 1 msg/seg. Variando a taxa de λ temos o gráfico do throughput pela taxa de chegada como mostrada na figura 4.

Na figura 5 é traçado o gráfico da probabilidade de ocorrência de timeout por erro de transmissão do canal em relação ao tempo.

O protocolo ABRACADABRA descrito acima gerou uma cadeia de Markov com 431 estados. Por exemplo considere o estado onde usuário_A e entidade_A têm estados internos (CLOSED,0), o usuário_B tem estado OPEN e a entidade_B tem estado OPEN,M(0,0,0,FALSE). Este estado pode ocorrer quando houve uma abertura de conexão e troca de mensagens, mas a entidade_A não conseguiu fechar a conexão.

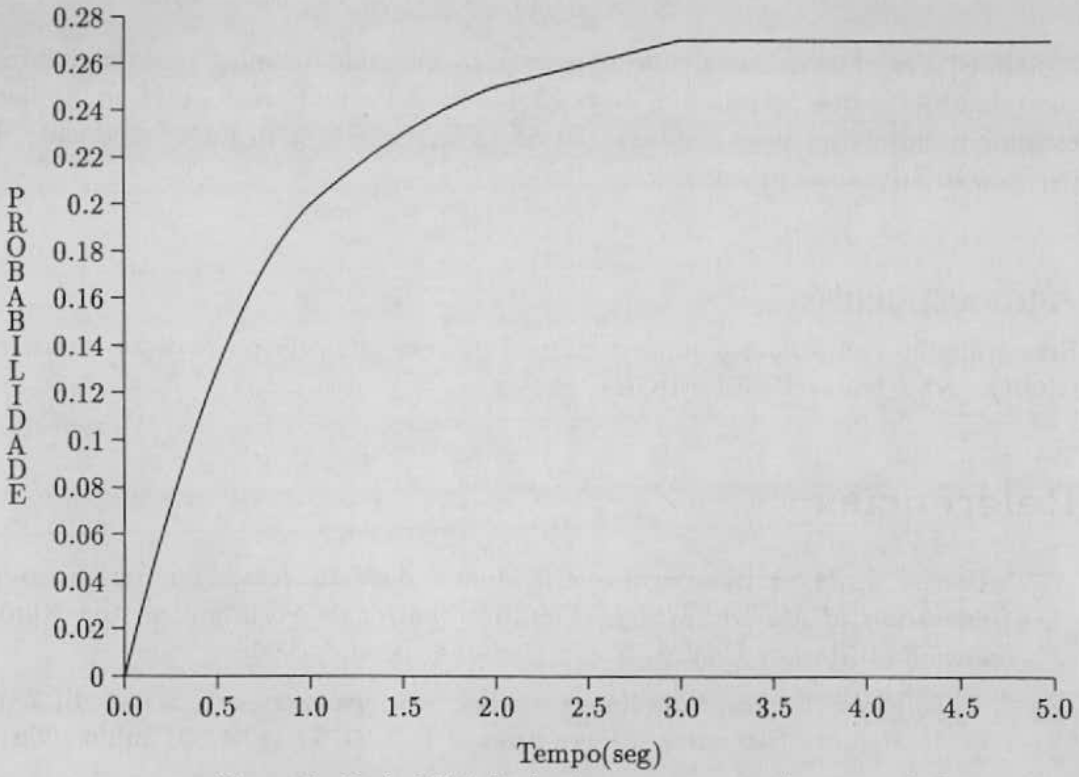

Figura 5: Probabilidade de timeout por erro de transmissão

O fechamento da conexão pode ter sido pedido pelo usuário_A quando do fim da transmissão dos dados ou pela entidade_A devido a detectação de erro no recebimento dos acks. A probabilidade estacionária deste estado quando $\mu_1 = 20$, $\mu_2 = 20$, $p_1 = 0.7$, $p_2 = 0.5$, $p_3 = 0.5$, $c_1 = 9.375$ e $c_2 = 9.375$ (estes parâmetros estão definidos no nível de aplicação) é de 0.39×10^{-4}.

5 Conclusões

A nossa experiência mostrou que a metodologia para definição de modelos é simples e poderosa. Utilizando-se de objetos previamente especificados, o usuário tem a liberdade de descrever modelos para diversas aplicações. Para facilitar a descrição do modelo, uma interface interativa para o usuário foi desenvolvida. A interface é útil não só na fase de construção do modelo, mas também na fase de construção de tipos de objetos. Para a construção de tipos de objetos, entretanto, o usuário necessita de um mínimo de conhecimento de Prolog. Porém, é importante ressaltar que o usuário final não precisa destes conhecimentos, uma vez que ele acessaria apenas a biblioteca de objetos previamente construidos.

A geração da cadeia de Markov do sistema sendo modelado é feita de forma simbólica. Sendo assim os estados são facilmente identificáveis. Isto é importante na fase de coleta de informações para a análise.

Atualmente existem desenvolvidos objetos para a área de modelagem de confiabilidade, além de objetos que permitem a descrição de modelos de filas. Já está em andamento estudos preliminares para a integração desta ferramenta com uma ferramenta de especificação formal de protocolos.

Agradecimentos

Este trabalho conta com o suporte parcial do programa de cooperação internacional CNPQ - NSF (entre UFRJ e UCLA).

Referências

[1] S.Berson, E.de Souza e Silva e R.R.Muntz, *A Methodology for Specification and Generation of Markov Models*, First International Workshop on the Numerical Solution of Markov Chains, North Carolina, janeiro 1990.

[2] A.Goyal, W.C.Carter, E.de Souza e Silva, S.S.Lavenberg e K.S.Trivedi, *The System Availability Estimator*, Proceedings of FTCS-16,pag.84-89, Julho 1986.

[3] K.S.Trivedi, *Probability e Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice - Hall, INC., Englewood Cliffs, 1982.

[4] S.S.Lavenberg, *Computer Performance Modeling Handbook*, Academic Press, INC. (LONDON) LTD, 1983.

[5] M.K.Molloy, *Performance Analysis Using Stochastic Petri Nets*, IEEE Transactions on Computers, vol. c- 31, NO. 9, setembro 1982.

[6] A.S.Tanenbaum, *Computer Networks*, PRENTICE-HALL, INC., Engledwood Cliffs, New Jersey 07632.

[7] ISO/SC2/Wg1/FDT-A, *Guidelines for the application of Estelle, LOTOS and SDL*, Project/97.21.9/Q48.2 CCITT/SG/X/Q7, agosto 1987.