UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Projeto e Implementação da Distribuição de um Simulador Multinível

por

LUÍS FERNANDO DA SILVA

Dissertação submetida à avaliação, como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Flávio Rech Wagner
Orientador

Porto Alegre, junho de 1997

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

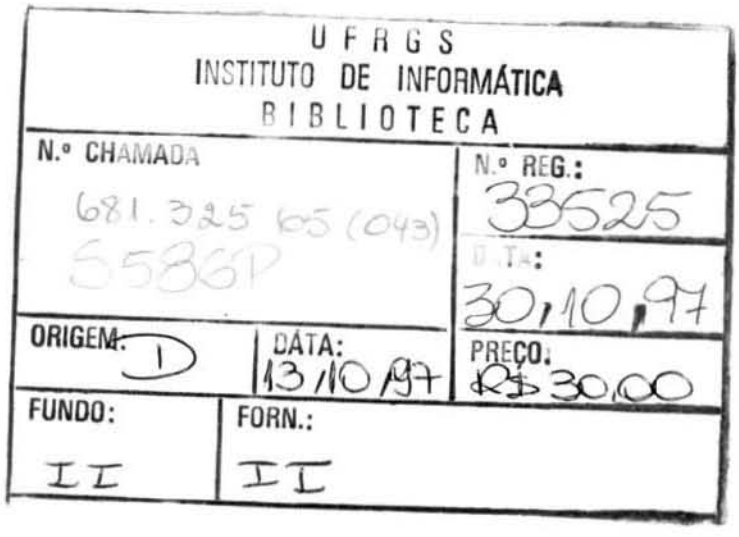
Silva, Luís Fernando

Projeto e implementação da distribuição de um simulador multinível / por Luís Fernando da Silva. – Porto Alegre: CPGCC da UFRGS, 1997.

104 f.: il.

Dissertação (mestrado)- Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1997. Orientador: Wagner, Flávio Rech.

1. CAD para sistemas digitais. 2. Simulação de hardware. 3. Simulação distribuída. 4. Simulação multinível. I. Wagner, Flávio Rech. II. Título.



Sistemas digitais. SBCL
Simulação distribuída
CAD. Sistemas digitais
CNPq 3.04.05.00-9

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Pós-Graduação: Prof. José Carlos Ferraz Hennemann
Diretor do Instituto de Informática: Prof. Roberto Tom Price
Coordenador do CPGCC: Prof. Flávio Rech Wagner
Bibliotecária-Chefe do Instituto de Informática: Zita Prates de Oliveira

Agradecimentos

Um dia um grande homem disse: "Problemas são oportunidades disfarçadas" (Henry Ford). Foi superando os diversos problemas durante este trabalho que muitas oportunidades surgiram.

Tive a oportunidade de conhecer pessoas especiais de quem jamais me esquecerei. Uma no entanto levarei na memória como um exemplo a seguir, o meu orientador e amigo Prof. Flávio Rech Wagner.

Conheci o que significa amar, pois somente amando uma pessoa que é possivel enfretar problemas como o meu amor Margrit Reni Krug o fez, jamais desanimando quando não havia mais ânimo.

Finalmente, agradeço a minha mãe, Irma Maria da Silva, que jamais deixou-me só mesmo quando não presente.

Obrigado a Todos.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

IP	Internet Protocol
PTVG	Processo responsável pelo cálculo do Tempo Virtual Global
TCP	Transmission Control Protocol
TVG	Tempo Virtual Global
UDP	User Datagram Protocol

Resumo

O uso de ferramentas de simulação para validar projetos de sistemas digitais é uma prática comum, devido às vantagens que estas trazem ao desenvolvimento destes sistemas, tais como: custo, segurança, velocidade e acuracidade. Porém, a simulação seqüencial de alguns sistemas pode levar várias horas ou até mesmo dias, fazendo desta maneira surgir a necessidade de técnicas para acelerar tal procedimento.

Uma solução encontrada para aumentar a velocidade de simulação pode estar no uso de técnicas de sistemas distribuídos, já que muitas vezes o próprio sistema real tem embutido em si um certo paralelismo, o que facilita os procedimentos de distribuição. Ao se tratar da simulação de sistemas distribuídos logo surge um dos grandes problemas inerentes a estes, o controle global do tempo, fazendo com que a sincronização entre os processos seja bastante complicada.

Neste trabalho são estudados dois paradigmas de sincronização, o otimista e o conservativo. Tendo como base estes paradigmas, formularam-se duas técnicas para solucionar o problema de sincronização, no contexto da simulação multinível de sistemas digitais.

Nos estudos realizados, utilizou-se como plataforma a API WinSock para Windows a fim de proporcionar a comunicação entre processos.

Ao final é feita uma análise comparativa das versões desenvolvidas, as quais fizeram uso das técnicas de sincronização acima mencionadas.

Palavras-Chave: Simulação de sistemas digitais, simulação multinível, simulação distribuída, técnicas de sincronização.

TITLE: "DESIGN AND IMPLEMENTATION OF THE DISTRIBUTION OF A MULTI-LEVEL SIMULATOR".

Abstract

The use of simulation tools to validate the design of digital systems is a common practice, due to the benefits these tools bring to the development of those systems: cost, security, velocity, and accuracy. However, the sequential simulation of some systems may take hours or even days, thus creating the need of techniques for speeding up this procedure.

A solution for increasing the simulation speed may be the use of techniques based on distributed systems, since very often the real system has an implicit parallelism, which makes easier the aplication of distribution procedures. When dealing with the simulation of distributed systems, one of the big problems that arise is the global control of simulation time, which makes the synchronization among processes very complex.

In this work two synchronization paradigms are studied: the optimist and the conservative ones. Based on these paradigms, two techniques for solving the problem of synchronization in the context of multi-level simulation of digital systems have been developed.

In these studies, the API WinSock for Windows has been used for supporting the communication between processes.

A comparative analysis of the versions we developed, that use the above mentioned synchronization techniques, is also presented.

Keywords: digital systems simulation, multi-level simulation, distributed simulation, synchronization techniques.

1 Introdução

A utilização de simulação em sistemas digitais tem como finalidade validar o projeto eletrônico propriamente dito, fazendo com que custos oriundos de falhas de concepção sejam praticamente nulos, pois vários testes podem ser feitos a fim de fornecer dados suficientes ao projetista para que este avalie corretamente a funcionalidade do circuito.

A simulação consiste em se ter o circuito a ser simulado descrito em um modelo teórico, o qual contém dados suficientes para que um algoritmo especializado proceda a sua validação, detectando possíveis falhas. Um circuito eletrônico pode ser modelado em vários níveis de abstração como mostrado na figura 1.1, na qual temos parte do circuito descrito a nível de sistemas e parte descrita a nível de portas lógicas, porém ambos fazem parte de um mesmo circuito. Os vários níveis de abstração que podemos utilizar para confeccionar um circuito digital, dependem do nível de detalhes que desejamos obter sobre o circuito.

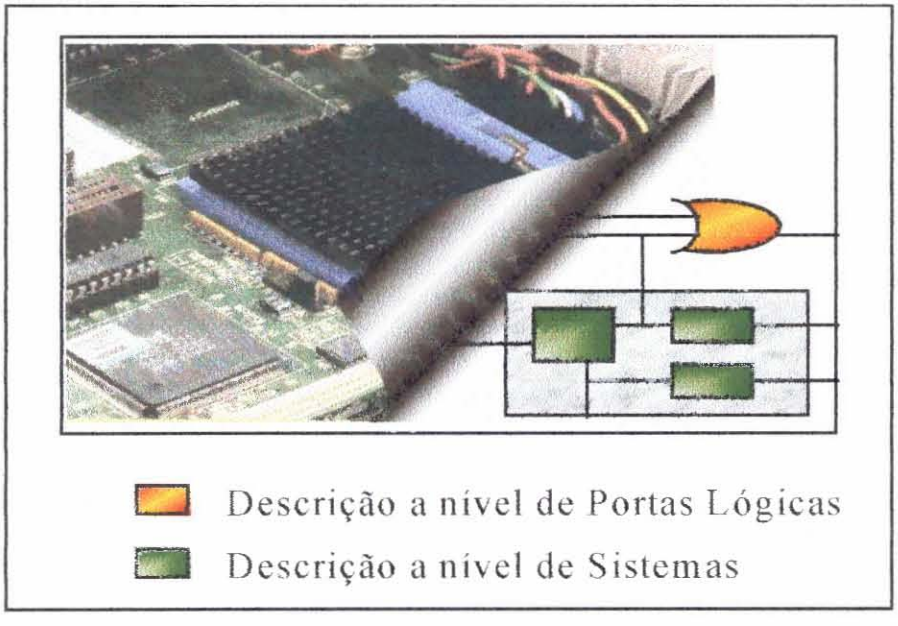


FIGURA 1.1 - Níveis de abstração de hardware

A simulação de um circuito digital é composta basicamente de três fases, conforme mostrado na figura 1.2, as quais são:

1. injeção de estímulos nos sinais de entrada da interface do circuito;

2. execução do algoritmo especializado para validar o modelo;

3. recuperação e avaliação dos sinais de saída do circuito, gerados pela execução do algoritmo de simulação sobre o modelo.

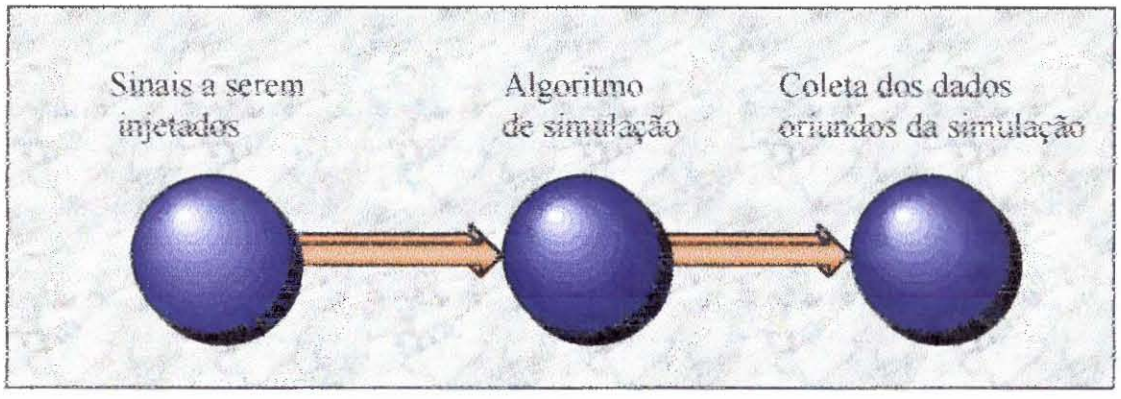


FIGURA 1.2 - Fases de uma simulação

No CPGCC da UFRGS está sendo desenvolvido um ambiente de simulação de sistemas digitais denominado AMPLO (Ambiente Integrado para Projeto Lógico de Sistemas Digitais). Este ambiente possui ferramentas que permitem a descrição e validação de sistemas digitais em três níveis de projeto: sistema, transferência entre registradores (RT) e portas lógicas. Para cada um destes níveis existe uma linguagem de descrição de hardware, respectivamente, LAÇO [SIL 88], KAPA [WAG 87b] e NILO [WAG 87a]. Os sistemas projetados neste ambiente são descritos de forma modular e hierarquizada como redes de agências [WAG 91] através da linguagem REDES [WAG 87].

A simulação dos sistemas descritos no AMPLO é feita com o auxílio dos simuladores deste, os quais trabalham sobre a filosofia mestre-escravo [WAG 89]. Existe um elemento central denominado "MESTRE" que coordena as interações entre os módulos a serem simulados e o avanço do tempo de simulação, onde cada módulo é simulado com a ajuda de um "ESCRAVO" específico, o qual é dedicado ao nível de abstração em que este módulo se encontra descrito.

O AMPLO em sua primeira fase utilizou-se de simulação seqüencial, onde o mestre era responsável pela propagação das variações dos sinais de interface entre os módulos, bem como pelo avanço do tempo de simulação. A simulação de sistemas digitais de grande porte consome muito tempo. A fim de melhorar o

desempenho, em termos de aceleração do tempo de simulação, optou-se por fazer a distribuição da simulação, para que fosse possível tirar proveito desta técnica.

Porém, ao se fazer um estudo das possibilidades de implementação da distribuição, surge um dos maiores problemas deste tipo de técnica, que é a ausência de controle centralizado do avanço do tempo de simulação, já que a simulação requer coerência no avanço do tempo global. Para que seja possível o uso da distribuição torna-se necessário o estudo dos dois paradigmas para sincronização de eventos em sistemas distribuídos existentes na literatura, o otimista e o conservativo [FUJ 90].

A distribuição da simulação, neste trabalho, será feita através do particionamento da agência a ser simulada, usando-se a ligação natural existente entre as agências primitivas que a compõem, tendo-se em mente a utilização de uma rede de equipamentos monoprocessados.

Este trabalho irá utilizar-se da teoria básica de cada um dos paradigmas, otimista e conservativo, a fim de propor soluções para controlar o sincronismo da simulação não somente para o projeto AMPLO, mas também para qualquer outro tipo de sistema de simulação discreta.

No capítulo 2, é feita uma apresentação de como é realizada a simulação seqüencial no sistema AMPLO, para que seja possível o acompanhamento da descrição das técnicas a serem abordadas neste trabalho.

No capítulo 3, são descritos os dois paradigmas de sincronização que irão reger o estudo das soluções propostas, o paradigma conservativo e o otimista.

No capítulo 4, é analisado o software utilizado como plataforma de distribuição a ser utilizada neste trabalho.

No capítulo 5, são feitos estudos sobre as propostas de distribuição da simulação no sistema AMPLO apresentadas por Figueiró [FIG 94].

No capítulo 6, é explicado como foi feita a criação do ambiente de testes utilizado para validar os algoritmos propostos.

No capítulo 7, são descritas as duas técnicas desenvolvidas para promover a distribuição do sistema AMPLO, uma conservativa e outra otimista.

No capítulo 8, são apresentados os dados coletados pelos testes feitos sobre o sistema seqüencial e sobre os sistemas distribuídos.

E finalizando o trabalho, no capítulo 9 são apresentadas as conclusões obtidas durante o seu desenvolvimento .

2 Simulação Seqüencial

2.1 Introdução

O sistema AMPLO integra várias ferramentas de projeto de sistemas digitais, tais como os simuladores e editores de linguagens de descrição de hardware. Para que seja possível a simulação de um sistema digital em AMPLO, devemos descrevê-lo através das linguagens existentes, LAÇO, KAPA, NILO e REDES.

A integração das ferramentas oferecidas pelo AMPLO é obtida através do uso de um banco de dados homogêneo especialmente desenvolvido para o mesmo, sendo armazenados por este todos os dados de projeto de forma consistente.

Os sistemas são descritos de forma modular e hierarquizada como redes de agências, através da linguagem REDES [WAG 87]. Cada agência na rede pode ser descrita como uma de duas formas, conforme a figura 2.1:

- Agência primitiva - especificada através de uma das três linguagens de descrição de hardware utilizadas em AMPLO: LAÇO, KAPA e NILO, que descrevem o comportamento da agência propriamente dita.

- Agência composta - constituída por uma rede de ocorrências de outras agências (primitivas ou compostas), nada sendo dito nesta descrição a respeito do interior das mesmas. A descrição deste tipo de agência é feita utilizando-se a linguagem REDES.

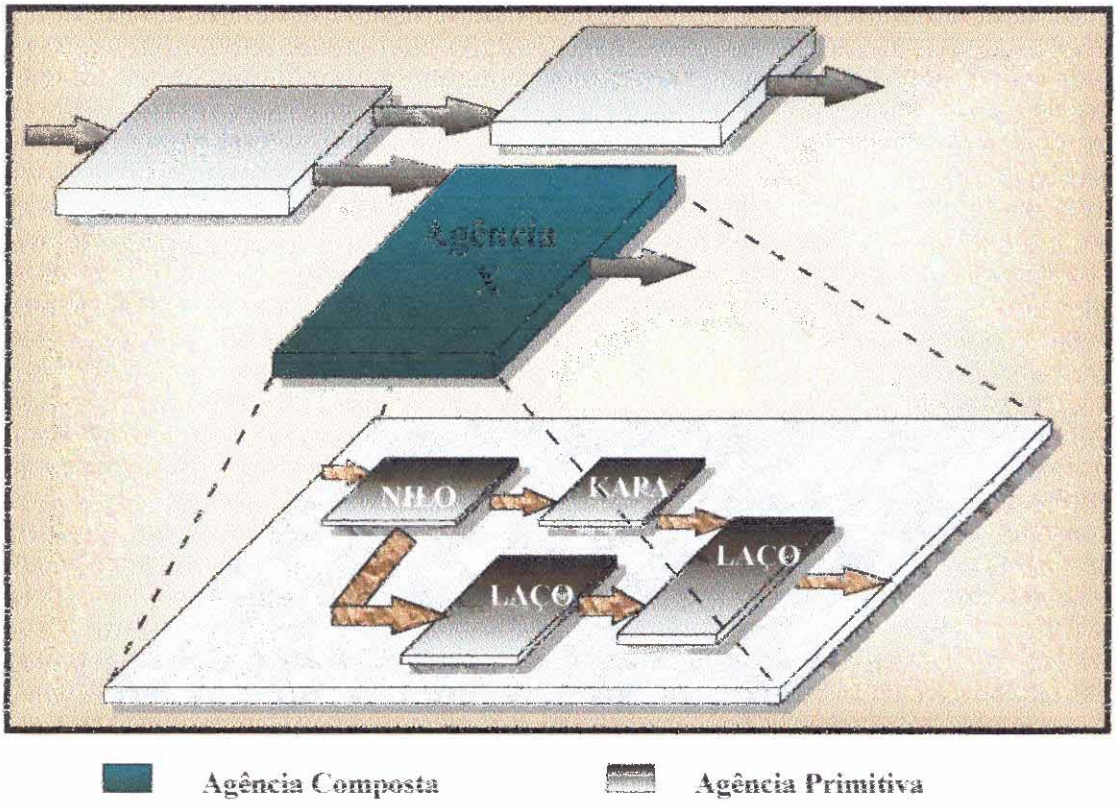


FIGURA 2.1 - Rede de agências

O AMPLO proporciona a simulação multinível através do uso conjunto de simuladores específicos para cada um dos níveis de descrição oferecidos pelo ambiente. A simulação está baseada no princípio mestre-escravo, conforme a figura 2.2, onde existe um módulo principal que controla o avanço do tempo global e a interação entre as agências (simulador mestre) e três outros módulos (um para cada linguagem de descrição de hardware) que controlam a execução das agências primitivas (simuladores escravos). Para que seja possível propor uma solução distribuída para o ambiente AMPLO, torna-se necessário o estudo dos simuladores e do ambiente de simulação.

O simulador mestre pode ativar diferentes escravos, de acordo com o nível de descrição da agência a ser simulada, o que permite a simulação multinível.

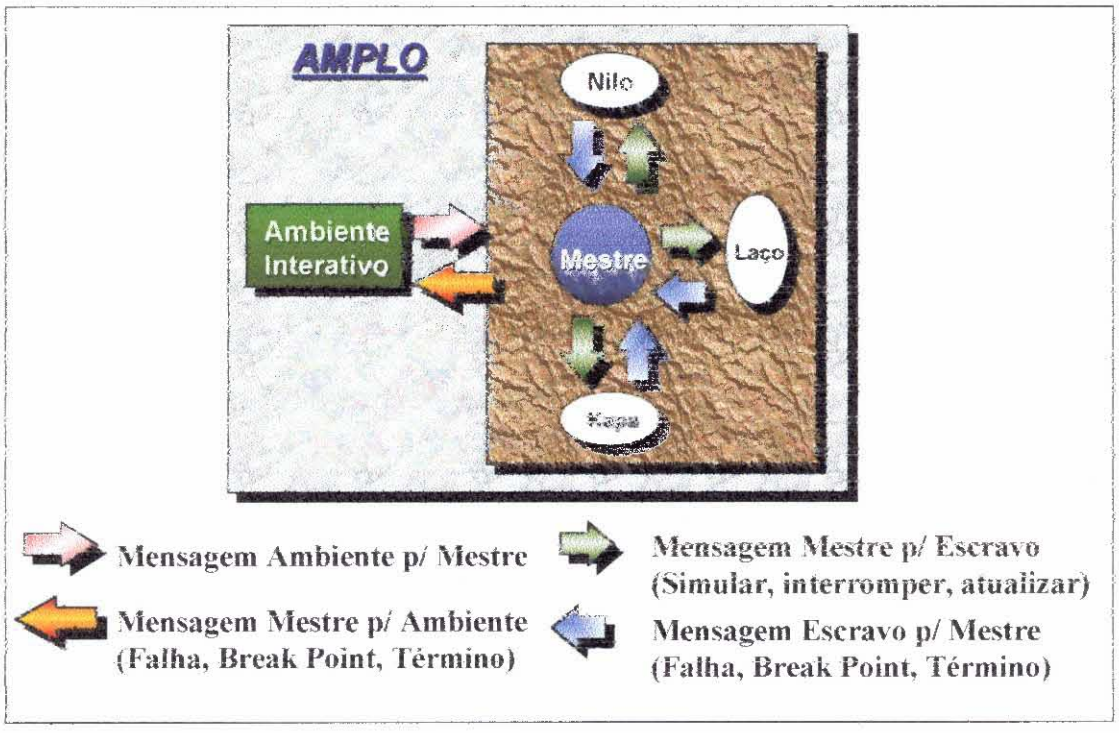


FIGURA 2.2 - Simulação seqüencial no AMPLO

De posse do projeto do sistema elaborado a partir das linguagens de descrição de hardware oferecidas pelo AMPLO, pode-se efetuar a simulação deste. Para tal será necessário executar uma seqüência de procedimentos básicos, os quais são: construção do modelo de simulação, especificação do estado inicial dos sinais, descrição e vinculação de estímulos. Estes procedimentos são suportados através de um ambiente interativo [WAG 91], que se comunica com o simulador através de um sistema de mensagens (ver figura 2.2).

Torna-se necessária a resolução das agências compostas, ou seja, é necessário que o modelo a ser simulado possua somente agências primitivas. O AMPLO possui um módulo para resolver tal problema, denominado construtor de modelos, o qual proporciona o achatamento das hierarquias encontradas dentro de agência composta, ou seja, é este módulo que monta a rede de agências primitivas que compõem o sistema a ser simulado.

A figura 2.3 mostra uma agência composta por três agências X, Y e Z. Estas três agências, por sua vez, são compostas por agências primitivas, conforme mostram as figuras 2.4 a 2.6. Após a resolução pelo construtor de modelos, a agência original é descrita por uma rede de agências primitivas, mostrada na figura 2.7.

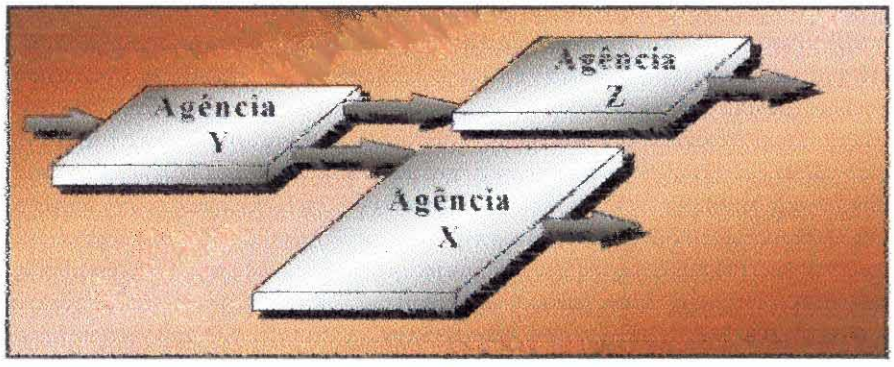


FIGURA 2.3 - Agência Composta

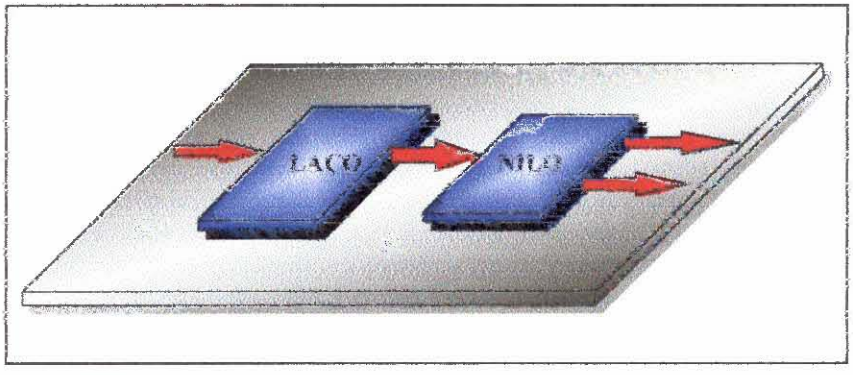


FIGURA 2.4 - Agência Primária Y

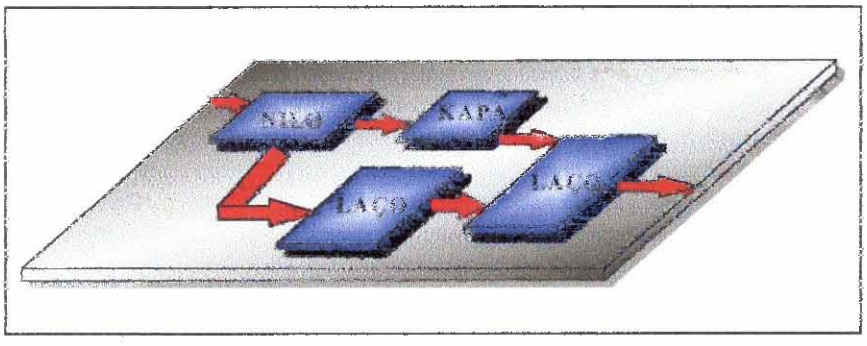


FIGURA 2.5 - Agência Primária X

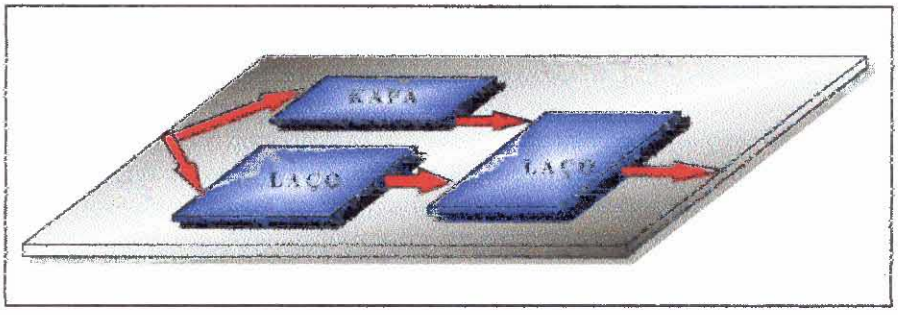


FIGURA 2.6 - Agência Primária Z

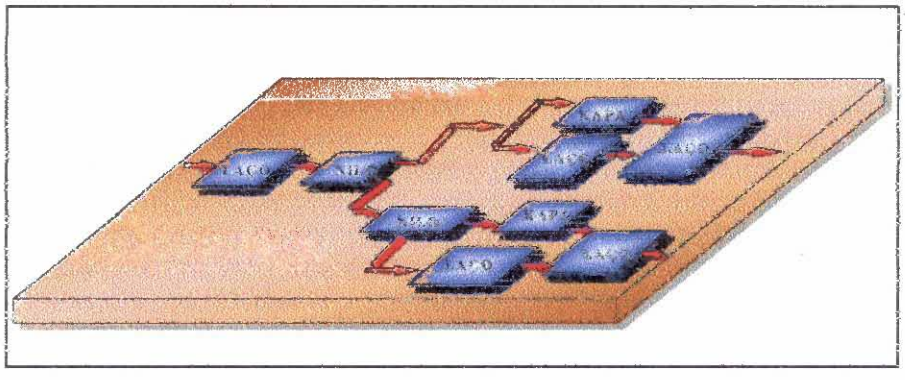


FIGURA 2.7 - Rede de agências primitivas relativa à agência composta da figura 2.3

2.1.1 Especificação do estado inicial dos sinais

É necessário que seja fornecido o valor inicial que cada sinal deverá possuir ao ser iniciada a simulação, tanto para os sinais de interface bem como para os sinais internos a cada agência.

2.1.2 Descrição de estímulos

Um estímulo é uma forma de onda que será aplicada em um ou mais sinais. A cada estímulo é associado um identificador e uma lista de trechos de formas de onda, onde cada trecho contém dados sobre a periodicidade, o tempo de duração e uma lista com a descrição comportamental do estímulo.

Para ser possível a simulação de um sistema no ambiente AMPLO, é necessária a descrição dos valores a serem aplicados nos sinais de entrada da interface da agência a ser simulada. Tal procedimento é possível através do uso de um construtor

gráfico-interativo de estímulos e de uma ferramenta de vinculação do estímulo definido com o sinal de interface desejado.

2.1.3 Processo de Simulação

Ao iniciar uma simulação o ambiente passa para o simulador mestre um ponteiro que indica uma posição de memória onde está armazenada a estrutura que representa as mensagens trocadas entre o mestre e o ambiente, bem como um ponteiro para a rede de agências primitivas e a lista de entradas primárias, a qual contém todas as descrições destas entradas, com suas respectivas listas de *fanout* e estímulos associados.

O mestre e os escravos são apenas subrotinas que estão inclusas no código do AMPLO, desta maneira a troca de mensagens não ocorre, mas sim passagem de parâmetros.

O usuário informa ao mestre, através do ambiente, os comandos a serem aplicados à simulação, bem como a rede de agências primitivas a ser simulada, que foi montada através do construtor de modelos.

Durante o processo de simulação, a cada tempo de simulação o simulador mestre verifica se existe alguma condição de parada e se esta já foi atingida. Caso isto ocorra, o mestre envia ao ambiente uma mensagem contendo o tempo atual e os valores dos sinais monitorados, e informando também qual a condição de parada que foi atingida, sendo o controle da simulação passado ao usuário. Não havendo uma condição de parada o tempo de simulação é incrementado, e conforme a necessidade são preparadas mensagens a serem enviadas às agências.

2.2 Simulador Mestre

O simulador mestre é o módulo principal da simulação proporcionada pelo sistema AMPLO, pois é ele o responsável pela interação entre os escravos e o ambiente. O mestre tem como principais funões:

- controlar o avanço do tempo de simulação;

- ativar as agências nos tempos por elas programados, de acordo com eventos futuros previstos, chamando o escravo apropriado conforme o nível em que a agência foi descrita; e
- propagar os novos valores de sinais de interface de agências para outras agências que estiverem conectadas a estes sinais.

O simulador mestre é composto de três partes:

1) o núcleo central, que é responsável pelas funções de propagação de valores de sinais entre as agências primitivas e pela administração da lista de eventos, a qual controla o avanço do tempo de simulação;

2) os adaptadores, os quais surgem devido aos diferentes níveis de abstração que podemos utilizar para descrevermos as agências que farão parte do sistema a ser simulado, onde cada uma possui tipos de dados característicos, muitas vezes sendo necessária a adaptação do sinal de saída de uma agência para o sinal de entrada de outra, e

3) o módulo de comunicação, responsável pela comunicação entre mestre e escravos, bem como entre o mestre e o ambiente.

O simulador mestre trabalha sobre quatro estruturas de dados básicas (figura 2.8) :

1) Lista de eventos: esta estrutura contém os eventos a serem disparados em um determinado tempo de simulação, ordenados por tempo de simulação. O mestre armazena apenas um evento (o mais próximo) para cada agência do modelo, enquanto que cada agência pode armazenar no seu interior um número qualquer de eventos.

2) Rede de agências primitivas - Ao ser chamado para controlar a simulação, o mestre recebe uma estrutura que possui todas as agências primitivas que compõem a agência a ser simulada, gerada pelo construtor de modelos. Esta estrutura contém os dados relativos às interconexões entre as agências.

3) Tabelas de sinais controlados - Tabela de sinais que devem ser monitorados durante a simulação.

4) Lista de entradas primárias - ao iniciar a simulação o simulador mestre deve saber quais os valores iniciais dos sinais e quais estímulos são aplicados a estes.

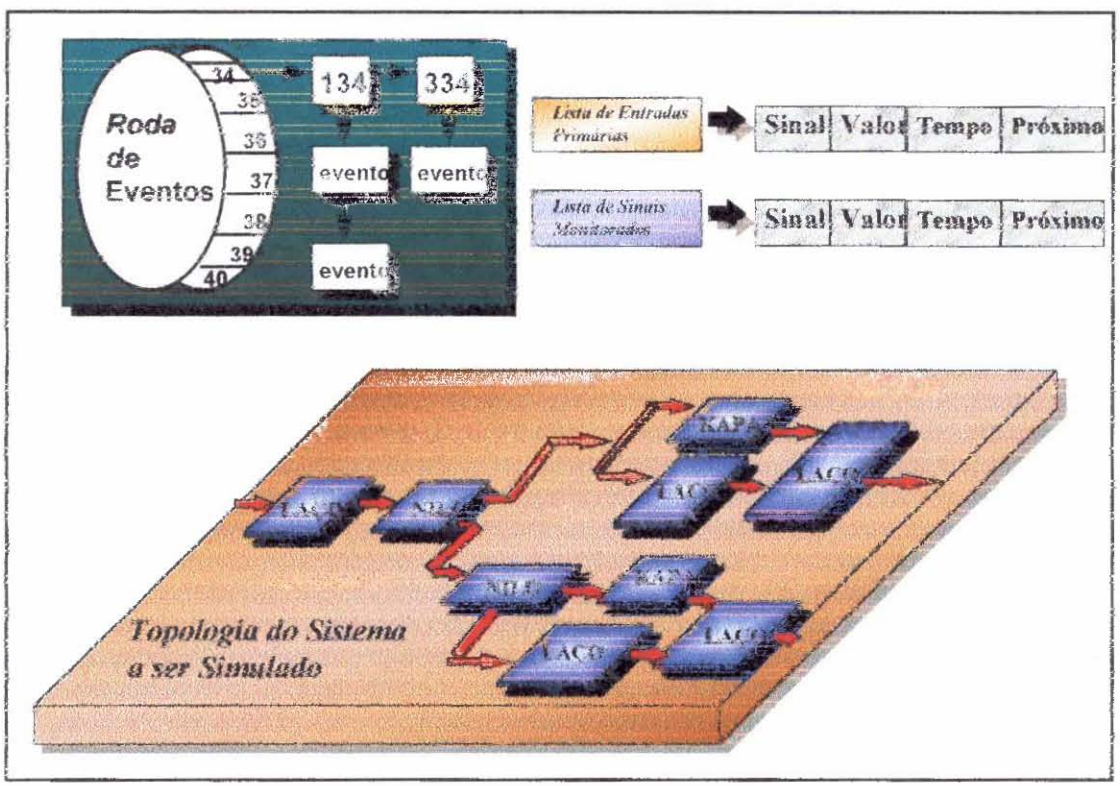


FIGURA 2.8 - Estruturas de dados do Mestre

O diálogo entre o ambiente e o mestre permanece até que o usuário ordene o início da simulação. Ao iniciar o processo de simulação, o tempo de simulação global é sempre avançado em uma unidade de tempo. O mestre então verifica se o tempo já excedeu o tempo de simulação estipulado pelo usuário. Caso o tempo atual de simulação seja maior ou igual ao tempo estipulado pelo usuário, o mestre devolve o controle ao ambiente.

A primeira tarefa que o mestre faz logo após avançar o tempo de simulação é verificar se há um evento programado para este tempo, através dos eventos inseridos na roda de eventos, figura 2.8. Caso exista algum evento programado para o tempo atual o mestre providencia o tratamento deste evento.

Existem quatro tipos de eventos:

1) do tipo **Delay**: uma agência pode provocar um atraso interno, logo esta irá prosseguir com sua função após Δt unidades de tempo, ou seja, esta deverá ser "acordada" pelo mestre no tempo **T(atual)** + Δt . Deve-se tomar conhecimento que cada agência possui uma lista de eventos futuros internos. O mestre somente necessita programar um evento do tipo Delay para cada agência, sempre o tempo de menor valor na lista de eventos futuros internos à agência;

2) do tipo **Variação-Sinal-Interface**: a programação de um evento deste tipo acontece quando ocorre uma variação de um sinal de interface de entrada de uma agência, causada por variação em sinal de saída de outra agência. Logo o escravo correspondente a esta agência deve ser acionado a fim de que esta atualize o sinal afetado, bem como seja feita a simulação da agência para que seja propagado o novo valor do sinal de interface para o interior da agência;

3) do tipo **Variação-Sinal-Externo**: quando os sinais de entrada da agência principal sofrem alterações, ou seja, há uma variação devida a um estímulo de entrada. Antes de propagar a variação do sinal, o mestre verifica se há outra variação programada e para qual tempo. Caso exista uma variação futura esta é incluida na lista de eventos a serem simulados, como um evento futuro; e

4) do tipo ***Clock*:** em uma agência podem estar conectados à interface sinais do tipo *Clock*. Tais sinais têm características especiais, e para tanto foi criado um evento especial para tratar este tipo de evento. O nodo de um evento do tipo *clock* guarda o nome do sinal de *clock* que deve ser ativado naquele tempo. Tal evento sofre o mesmo tratamento de um evento do tipo Variação-Sinal-Externo.

A comunicação entre o mestre e o ambiente de simulação é feita através da passagem de uma estrutura que contém a lista de comandos enviados pelo usuário, e ,vinculado a cada elemento desta, uma outra lista com os sinais que devem sofrer a ação destes comandos, conforme a figura 2.9.

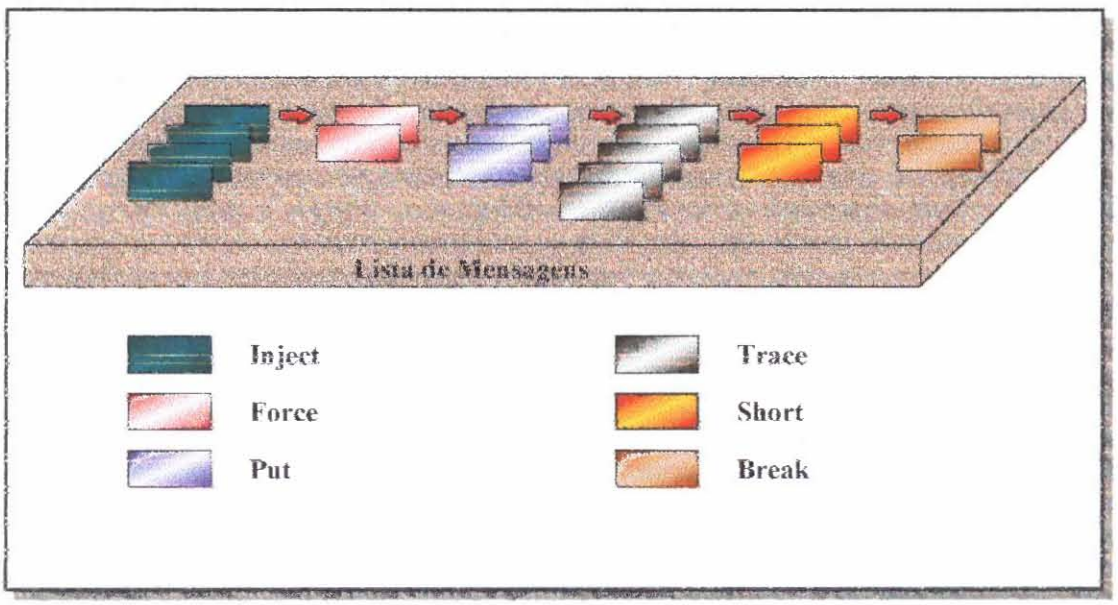


FIGURA 2.9 - Estrutura da Lista de Mensagens

Conforme o tipo do comando enviado ao mestre pelo ambiente, tem-se uma estrutura diferente, a qual conterá os dados necessários a fim de notificar ao mestre a quem se refere tal solicitação. Um modelo das estruturas menores que fazem parte da lista de mensagens é monstrado através da figura 2.10.

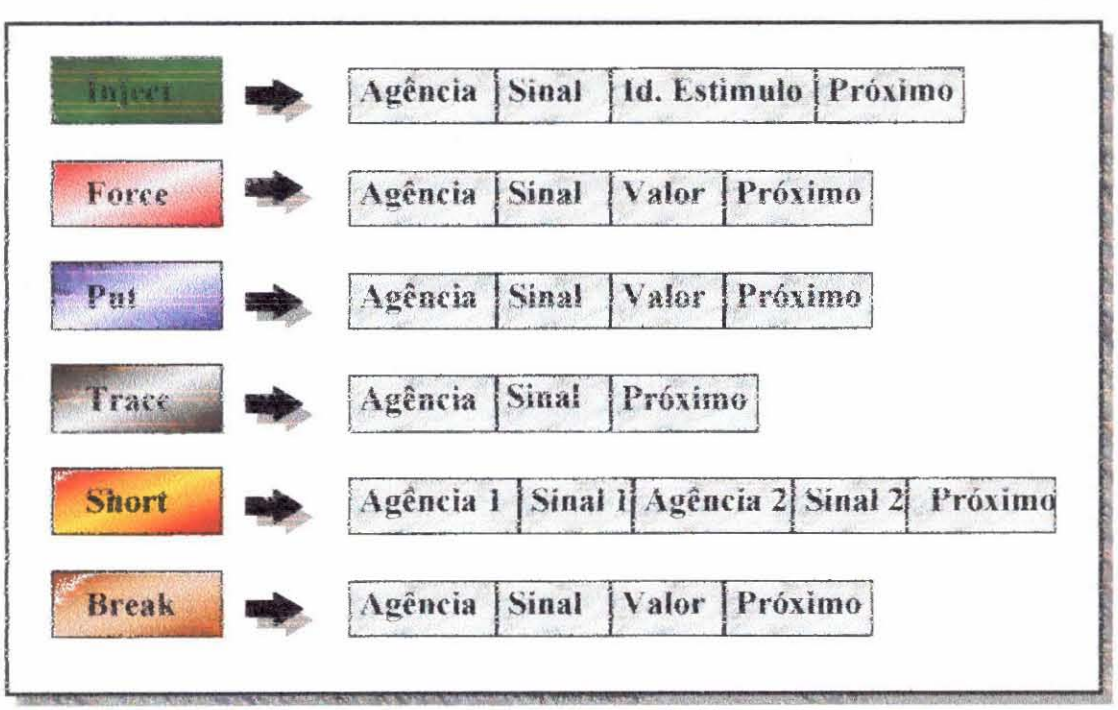


FIGURA 2.10 - Estruturas internas dos comandos

Da mesma forma que se utiliza uma estrutura para realizar a comunicação entre o mestre e o ambiente, é utilizada uma estrutura para efetuar a comunicação mestre/escravo e escravo/mestre.

O algoritmo de simulação do mestre é mostrado na figura 2.11.

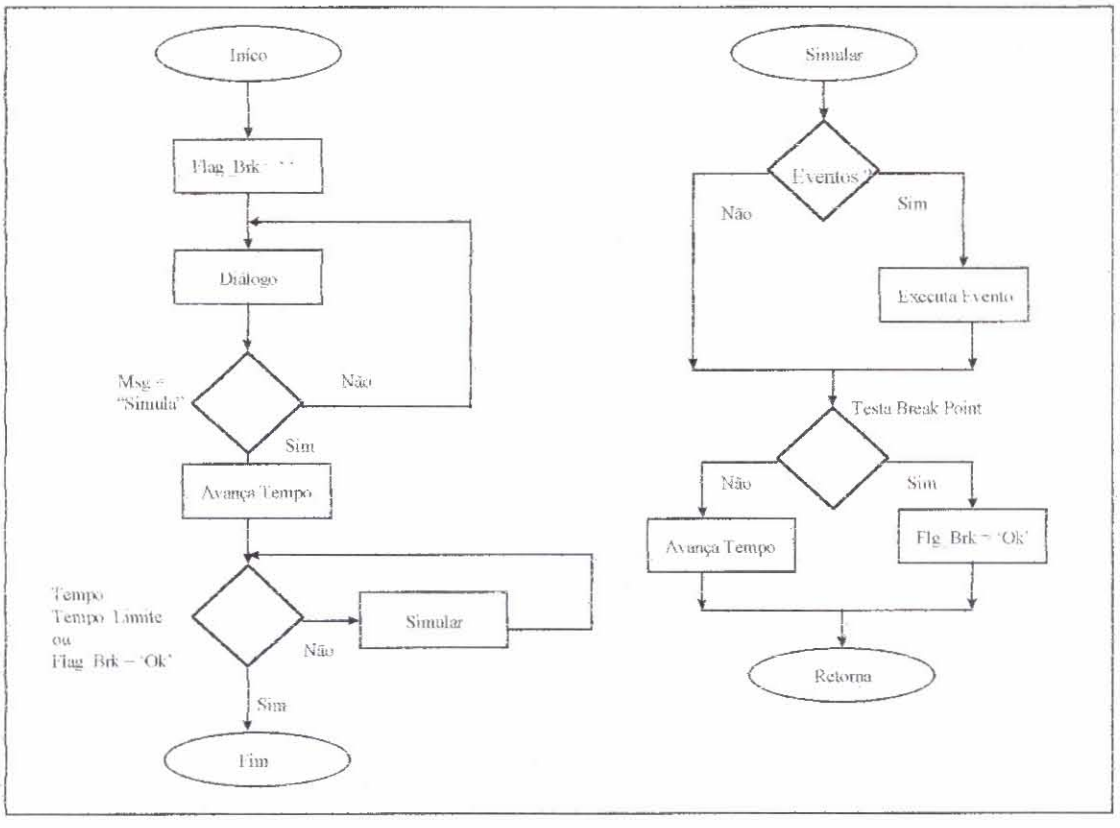


FIGURA 2.11 - Algoritmo de simulação do mestre

O mestre possui em seu corpo duas outras funções, barramento e adaptação de sinais.

1. A função de barramento tem como objetivo resolver os problemas referentes a sinais que são do tipo barramento. Tal função deve avaliar os sinais que alimentam o barramento e, a partir dos dados referentes a estes, deverá fornecer o valor do sinal de consenso do barramento a ser propagado. Como as linguagens que compõem o AMPLO possuem características diferentes referentes a alguns tipos de sinais, a função de barramento terá que prover um consenso entre sinais de diferentes tipos de dados.Um exemplo de uso da função de barramento é esquematizado através da figura 2.12.

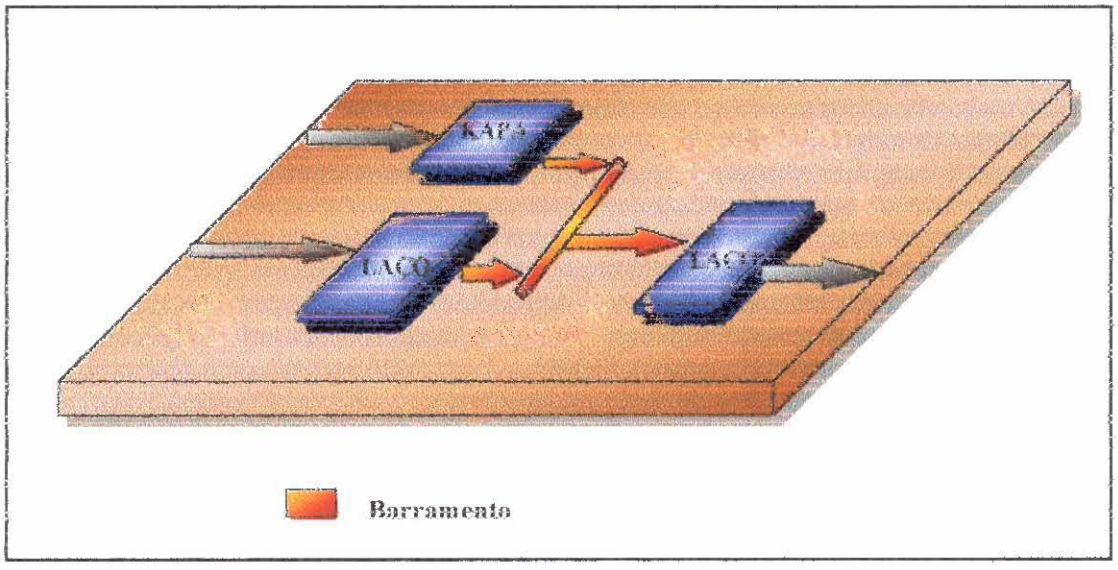


FIGURA 2.12 - Uso da função de barramento

2. A função de adaptação de sinais é utilizada sempre que a ligação entre sinais de tipos de dados diferentes, sendo necessária a adaptação entre estes.

2.3 Simulador LAÇO

O simulador LAÇO tem como objetivo simular agências descritas através da linguagem de descrição de hardware LAÇO, dedicada ao nível de sistemas. Seu funcionamento baseia-se no princípio de redes de Petri estendidas.

Em LAÇO, os nodos de um grafo de controle podem ser de dois tipos: lugares e transições, onde lugares armazenam valores 0 e 1 que representam a ocorrência de eventos e transições possuem lugares de entrada e saída. O papel do simulador escravo é seguir o grafo de controle, disparando as devidas transições quando a ocorrência de eventos nos seus lugares de entrada torna a lógica associada a estas verdadeira.

Podemos associar atrasos a cada transição descrita no grafo, fazendo desta maneira com que os eventos de saída só sejam propagados após decorrido o tempo especificado.

Ao iniciar a simulação de uma agência descrita em LAÇO, o simulador verificará se há eventos programados para o tempo atual. Caso exista algum evento para este tempo, todas as transições para as quais este evento é uma entrada são testadas e, caso estejam ativas, são inseridas na lista de transições que irão ser disparadas, denominada ***LTR***.

Ao ser disparada uma transição com atraso, tanto os lugares de entrada que causaram o disparo da transição como os lugares de saída que serão marcados pela transição são reservados, de forma que durante o período de retardo nenhuma operação pode ser executada sobre estes lugares.

A figura 2.13 ilustra um agência primitiva descrita em LAÇO.

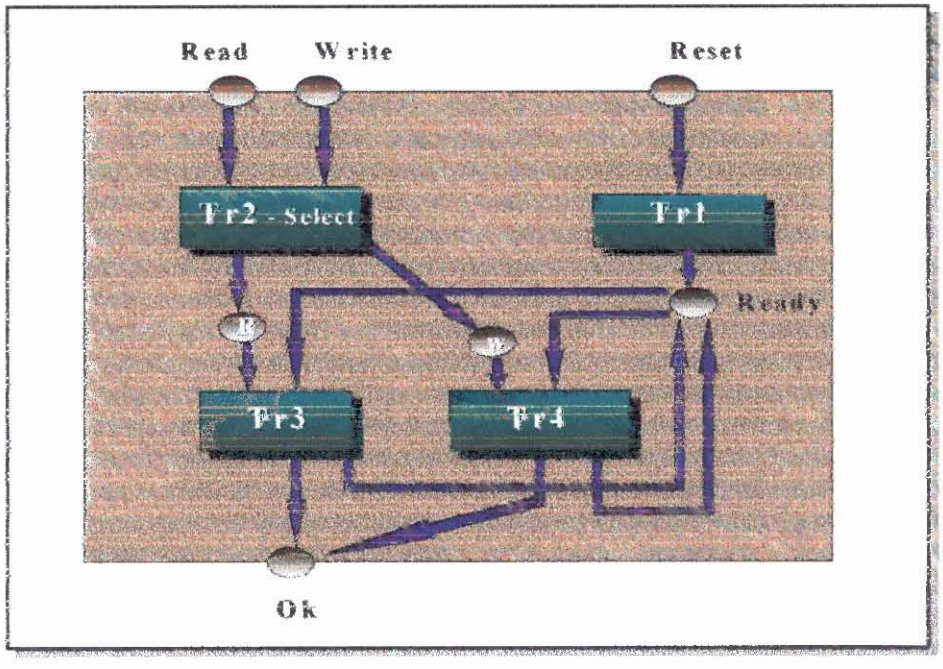


FIGURA 2.13 - Exemplo de agência primitiva LAÇO

2.4 Simulador NILO

O simulador NILO tem como objetivo simular agências descritas através da linguagem de descrição de hardware NILO [WAG 87], a qual é dedicada ao nível de portas lógicas.

Além das portas lógicas convencionais, a linguagem NILO possui construções para declarar portas de transmissão bidirecionais, *buffers* "tri-state" e resistores "pull-up"e "pull-down". A linguagem permite também a declaração de vários atrasos de propagação associados às portas, assim como de intensidades associadas às saídas das portas.

A figura 2.14 ilustra uma agência primitiva descrita em NILO.

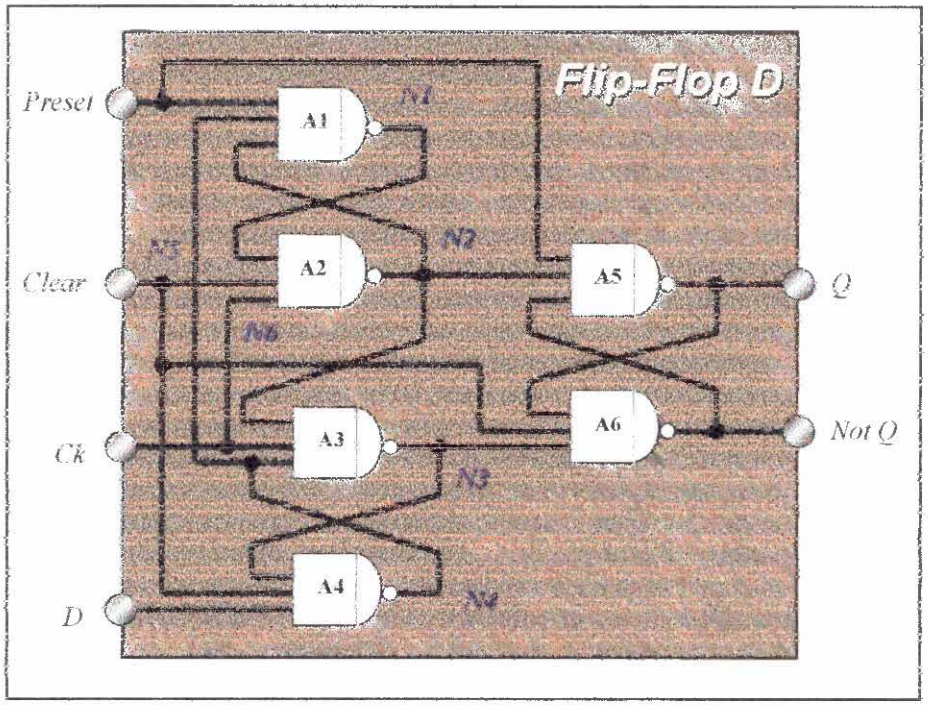


FIGURA 2.14 - Exemplo de agência descrita em NILO

3 Paradigmas de sincronização

3.1 A necessidade de sincronização

O tráfego de mensagens em um ambiente distribuído pode ser ilustrado através da figura 3.1, a qual evidencia o problema inerente a distribuição, isto é, o controle do tempo global. Torna-se clara a necessidade do estudo de técnicas de sincronização em simulação de sistemas discretos, que objetivam tirar proveito dos benefícios da distribuição.

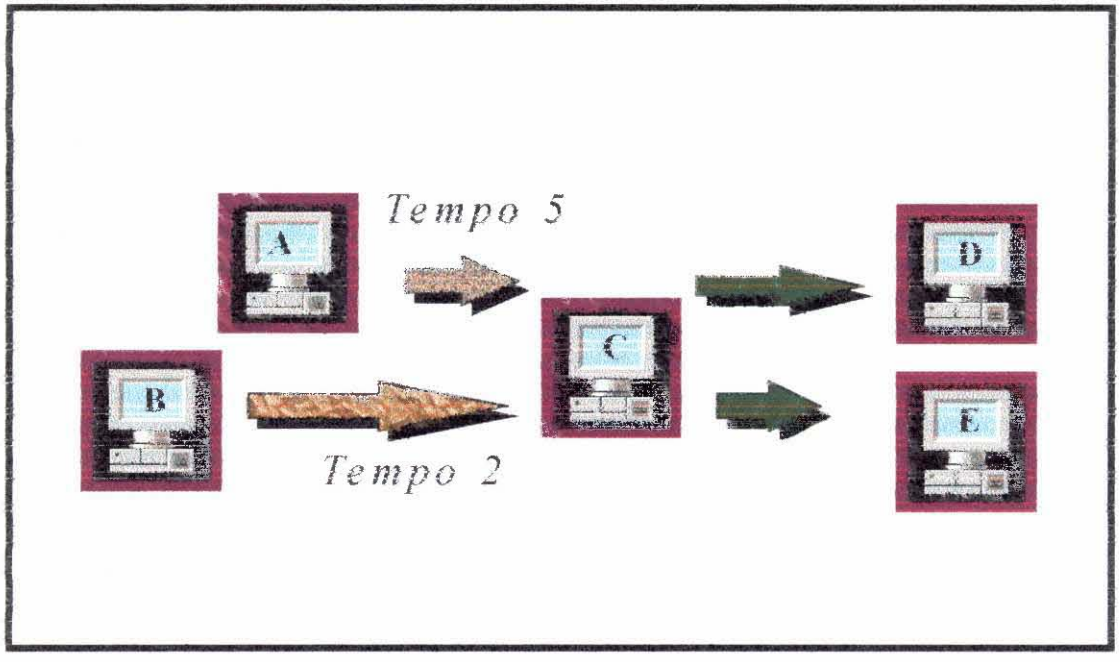


FIGURA 3.1 - Problema na distribuição da simulação

A figura 3.1 demonstra que os processos A e B estão enviando mensagens para o processo C, sendo que cada uma das mensagens foi gerada em um determinado tempo de simulação dentro do processo que as originou. Deve-se notar que a mensagem enviada pelo processo B foi gerada em um tempo de simulação anterior ao da mensagem gerada pelo processo A, porém pode ocorrer que a mensagem de B para C não chegue antes. Tal fato gera a necessidade de criar mecanismos de sincronização, para que seja garantida a ordem de execução das mensagens que irão trafegar pelo ambiente distribuído, o qual será utilizado para a simulação de um sistema discreto. O tempo de chegada das mensagens a seus destinos não depende apenas do tempo em que esta foi gerada e lançada na rede, pois vários fatores poderão contribuir para que ocorra

um atraso na chegada de uma mensagem ao seu destino. Dentre os problemas mais comuns está a utilização de uma rede de computadores com topologia híbrida, onde haja a necessidade de roteamento da mensagem, conforme ilustrado através da figura 3.2.

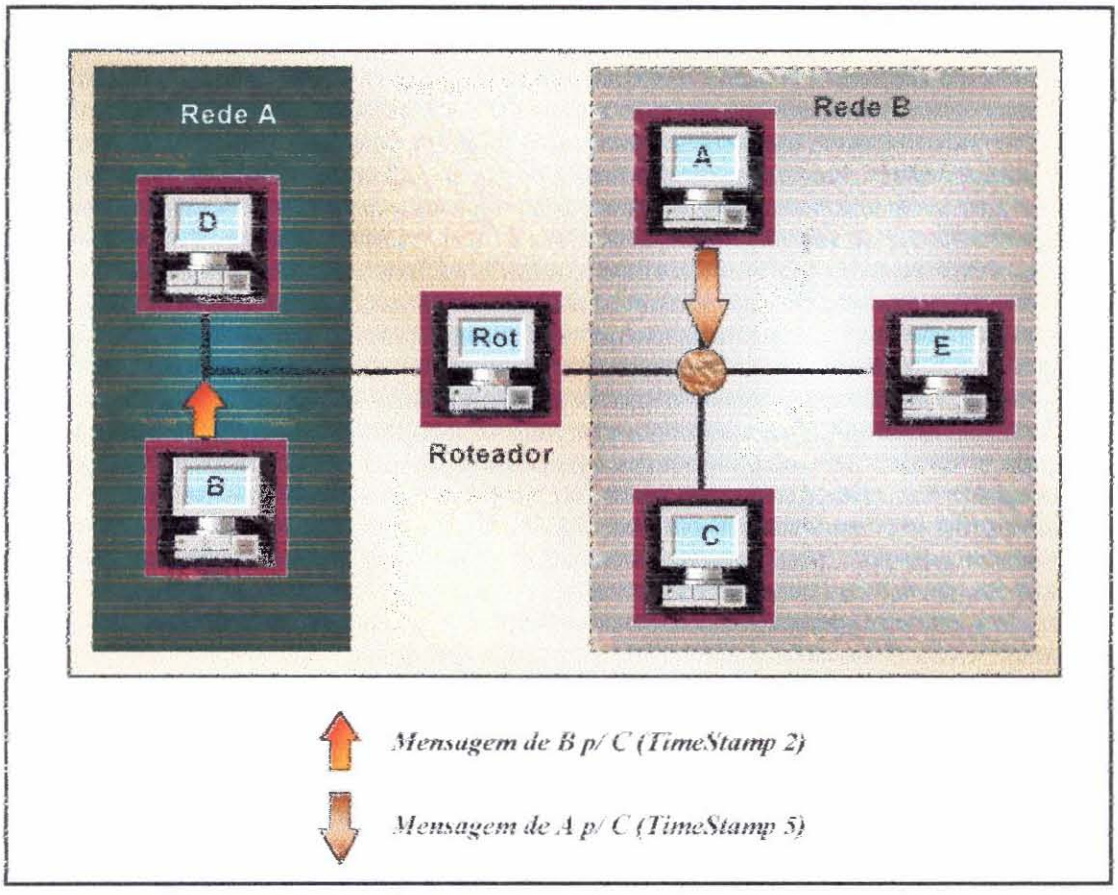


FIGURA 3.2 - Exemplo de rede

A figura 3.2 demonstra uma situação peculiar em uma rede de computadores, onde se tem vários equipamentos interligados por uma única rede, e por sua vez, diversas redes poderão estar interligadas através de um roteador. Como garantir que uma mensagem gerada por um processo no equipamento B no tempo de simulação 2 chegue ao processo que está rodando no equipamento C antes da mensagem gerada pelo processo da máquina A no tempo de simulação 5 ?

Para que se possa controlar a evolução do tempo de simulação global, são utilizados dois paradigmas de sincronização: o conservativo e o otimista, os quais serão descritos a seguir.

3.2 Paradigma Conservativo

As técnicas conservativas permitem a discrepância entre os tempos dos eventos em execução nos processos constituintes do sistema, os quais são simulados independentemente um dos outros, enquanto houver segurança de que a troca de mensagens entre estes continua íntegra [CHA 79]. Tal integridade diz respeito à chegada das mensagens ao processo receptor, onde deve ser garantido que, em todos os canais de comunicação de entrada, as mensagens devam chegar respeitando a ordem cronológica, ou seja, no mesmo canal as mensagens devem chegar em ordem de ***timestamp*** [GRO 89] (tempo de simulação em que a mensagem foi gerada). A troca de mensagens entre os processos é feita de forma assíncrona. Cada mensagem carrega consigo, além do seu conteúdo, o seu *timestamp*, o nome do processo emissor e o nome do processo destino, conforme visto na figura 3.3.

FIGURA 3.3 - Formato de uma mensagem

Cada processo deve possuir uma fila de mensagens, ordenada por *timestamps*, para cada ponto de entrada. A comunicação entre o processo emissor e o processo receptor é considerada como um canal de entrada, como demonstrado na figura 3.4. Um canal nada mais é do que uma fila das mensagens recebidas para cada sinal de interface.

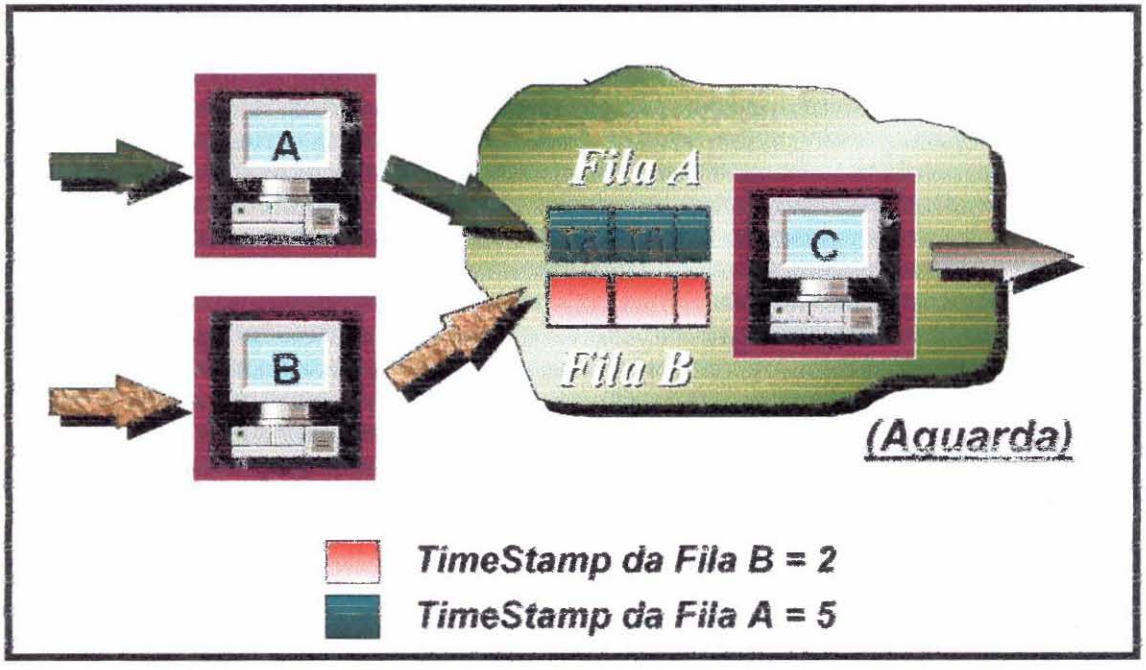
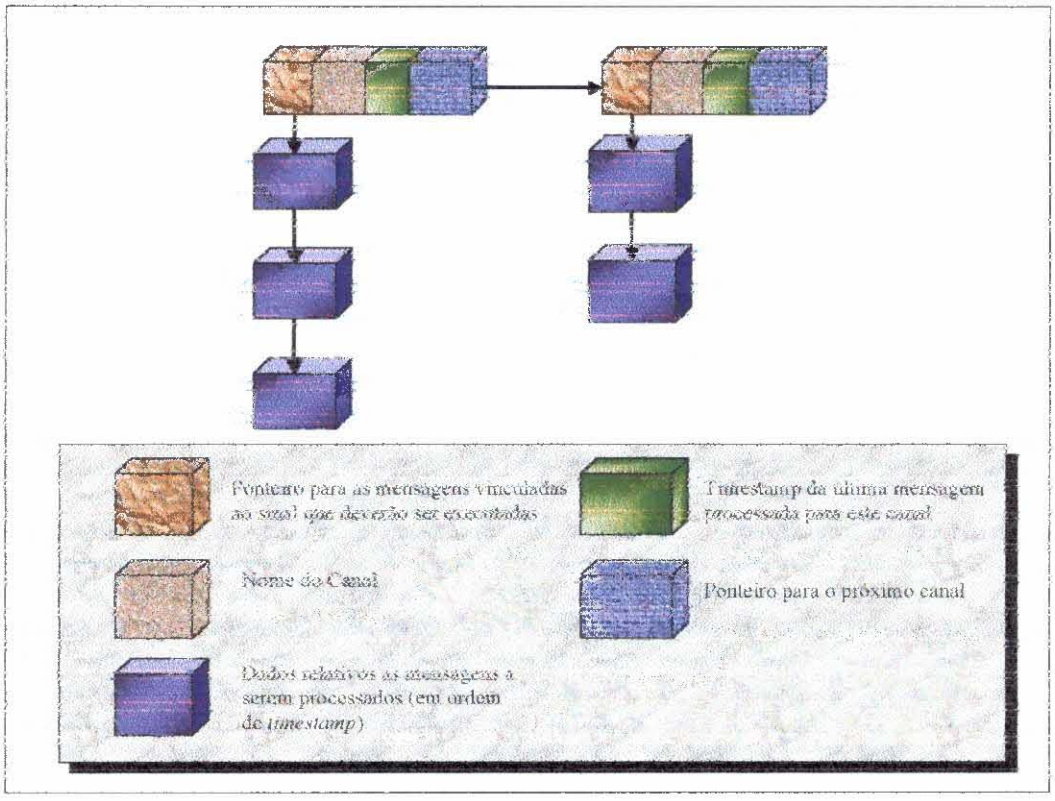


FIGURA 3.4 - Técnica conservativa

As estruturas básicas necessárias para fazer-se uso da técnica conservativa são demonstradas na figura 3.5. A figura 3.6 mostra um algoritmo fazendo uso da técnica conservativa.




```
struct  CANAIS
{
        struct MENSAGENS *Mensagens;  //Estrutura de mensagens
        struct CANAIS *next;              // Aponta para o próximo canal
        double TimeStamp;                    // Último Timestamp processado
                                              deste canal
}

// A estrutura de mensagens irá variar conforme o sistema a ser simulado

struct MENSAGENS
{
    int valor;
    ...
   struct MENSAGENS *next;
}
```

FIGURA 3.5 - Estruturas usadas pelo algoritmo conservativo

```
// Algoritmo fazendo uso do paradigma Conservativo

Início
   Mensagens = 0;
   Faça
      Se Há Nova Mensagem Enviada
            Se Mensagem != Interromper_Simulação
               Insere_Mensagem ( )
               Mensagens++;          // Contador de mensagens a serem
                                        processadas
            Senão
               Interrompe_Simulação
            FimSe
      FimSe
      Se Mensagens > 0
            Se Busca_Situaçào_Canais ( ) != Vazio
                  Executa_Mensagens ( )
            FimSe
      FimSe
   FimFaça
Fim

Insere_Mensagem ( )
      1. Busca na estrutura de canais o canal que irá receber a mensagem enviada;
      2. Insere os dados contidos na mensagem na estrutura de MENSAGENS;

Busca_Situação_Canais ( )
      1. A estrutura de Canais é percorrida a fim de detectar a situação da fila de
         mensagens do canal com o menor TimeStamp até então executado (Vazio ou
         Não);
      2. Detecta qual o próximo Timestamp menor dentre as mensagens dos canais,
         guardando-o na variável Exe_TimeStamp.

Executa-Mensagens ( )
      1. Seleciona as mensagens que estão programadas com o timestamp =
         Exe_TimeStamp;
      2. Ajustar valores dos Canais de entrada (Somente aqueles que tenham eventos
         programados para o timestamp identificado como o menor de todos dentre as
         mensagens já enviadas);
      3. Mensagens--;
```

FIGURA 3.6 - Algoritmo sobre o paradigma conservativo

A maior dificuldade encontrada na implementação deste tipo de técnica de sincronização está no fato de que é muito comum o sistema de simulação gerar situações de *deadlock*, visto que os processos estão sendo executados de forma assíncrona e que estes devem esperar, para cada fila de mensagens, uma mensagem com tempo superior ao *timestamp* da última mensagem enviada. Torna-se necessário desta maneira a implementação de mecanismos de detecção e recuperação de *deadlocks*.

A figura 3.7 demonstra uma situação de *deadlock* criada devido à espera provocada por este tipo de técnica. A figura 3.7 está retratando um *deadlock* entre os processos B, C e D, onde B está esperando uma mensagem de D, que por sua vez está esperando uma mensagem de C e este aguarda a chegada de uma mensagem de B, fechando desta maneira o ciclo de um *deadlock*.

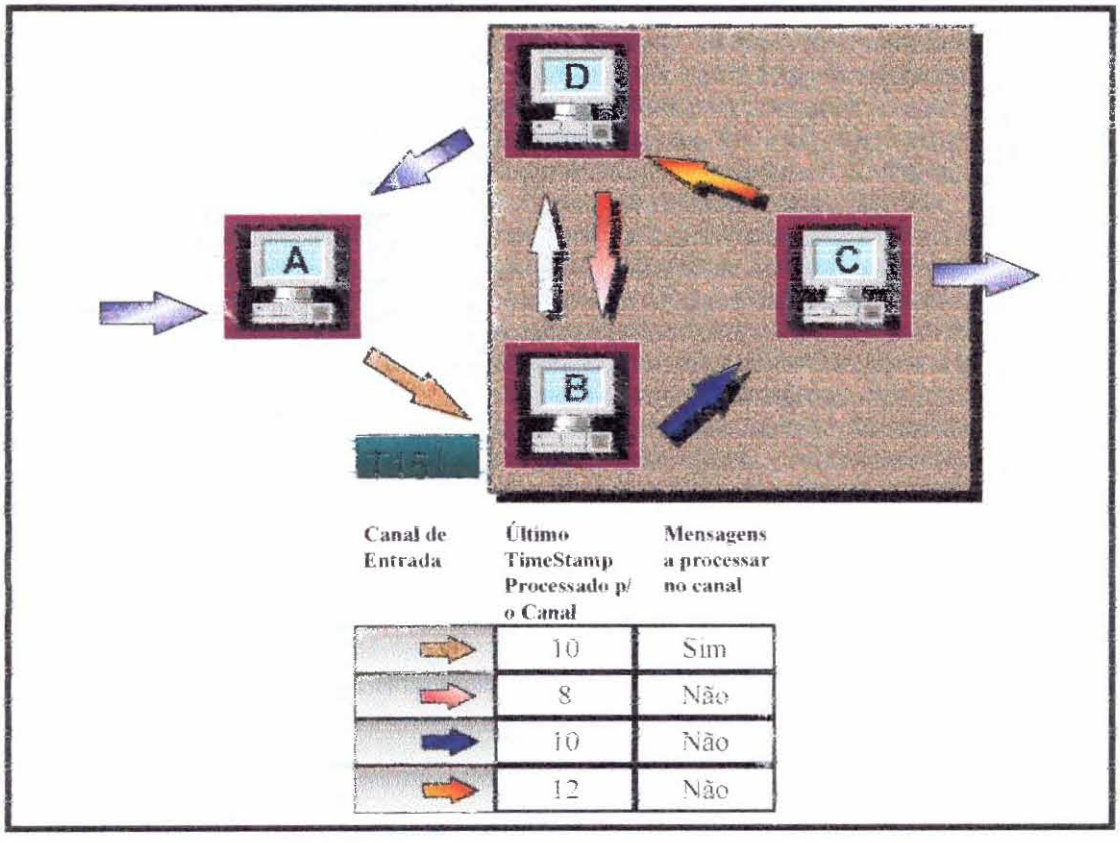


FIGURA 3.7 - Situação de *Deadlock*

Na literatura [SU 89] [LIU 90] encontra-se alguns métodos que visam evitar a ocorrência de *deadlock* em sistemas que utilizem o paradigma conservativo. Dentre estes métodos, pode-se citar os dois principais, os quais são: envio de mensagens nulas e detecção e recuperação de *deadlock*.

3.2.1 Mensagens nulas

Esta técnica [FUJ 90] faz uso de mensagens que não possuem efeito na simulação propriamente dita, as quais servem apenas para informar ao processo receptor que este poderá avançar até um determinado tempo de simulação X sem se preocupar com a chegada de mensagens atrasadas, pois o processo que enviou a mensagem nula garante que não irá gerar nenhuma mensagem até o tempo informado pela mensagem nula recebida.

O uso desta técnica poderá provocar um tráfego de mensagens muito grande, o que acarretaria na diminuição do desempenho da rede. A redução de desempenho na rede seria basicamente devida à propagação muito constante de mensagens sem valor efetivo.

O formato das mensagens que irão trafegar pelo ambiente distribuído é demonstrado na figura 3.8.

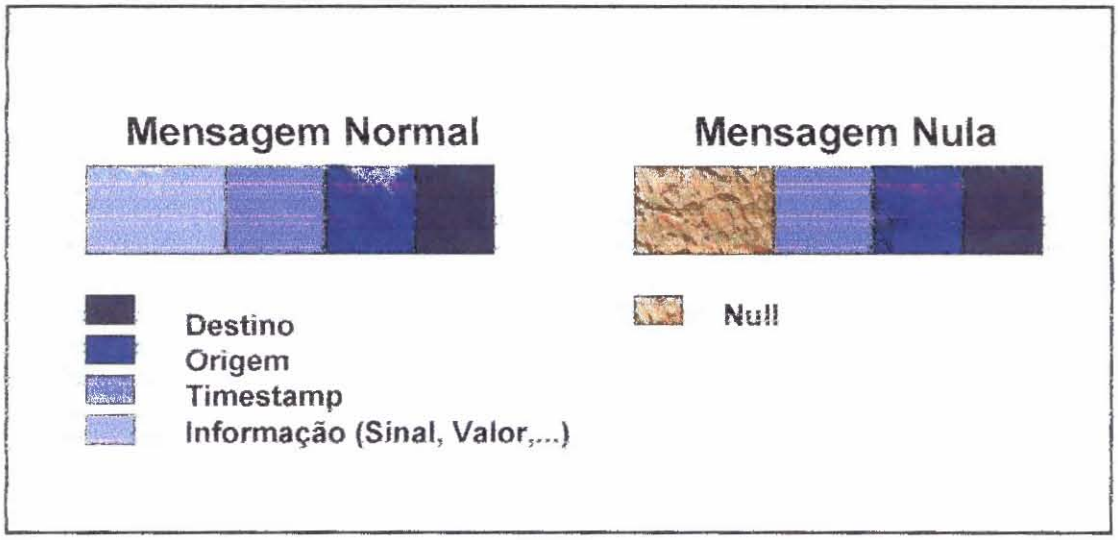


FIGURA 3.8 - Formato das Mensagens com uso da Técnica de Mensagens Nulas

3.2.2 Detecção e Recuperação de *Deadlock*

Fujimoto [FUJ 90] propõe a construção de dois mecanismos, um para detecção de *deadlock* e outro para desfazer o *deadlock*. O mecanismo de detecção seria

basicamente vistoriar os canais de entrada, verificando se o canal que está vazio é também o canal com o menor *timestamp* até então processado. Caso o seja, pode-se estar diante de um *deadlock*, logo deve-se disparar o processo para desfazer o bloqueio. O processo de detecção e recuperação de *deadlock* será disparado independente de ser ou não um deadlock real.

Uma das desvantagens do uso de detecção e recuperação de, conforme proposto por Fujimoto, está no fato de que durante a fase de desbloqueio os processos irão ficar parados, até que seja achado um consenso (tempo global entre os processos envolvidos no *deadlock* a partir do qual a simulação possa continuar), o qual possibilitará a continuação da simulação. O mecanismo de recuperação de *deadlock*, conforme Fujimoto, irá enviar mensagens a todos os processos, solicitando a estes que enviem o seu menor *timestamp* até então processado, ou seja, para cada possibilidade de *deadlock* serão necessárias $2*N$ mensagens, onde N é o número de processos que compõem o sistema que está sendo simulado. Outra situação que se pode ter ao utilizar tal técnica é a ocorrência de mais de um acionamento do mecanismo de recuperação de *deadlock*, ocorrendo desta forma um grande aumento de mensagens enviadas através da rede. Conforme a figura 3.7, tanto B, C ou D podem ativar o mecanismo de recuperação de *deadlock*.

A simples situação de um processo detectar que sua fila de menor *timestamp* processado está vazia, não condiciona que este esteja realmente em *deadlock*. Poderá simplesmente estar ocorrendo um atraso na propagação da mensagem, devido a vários fatores: rede muito lenta, ou mesmo devido ao fato de que o processo emissor esteja processando a próxima mensagem.

Lubachevky [LUB 29] propõe o uso de um tempo mínimo para que um processo se considere em *deadlock*, ou seja, que seja estipulado um tempo β o qual representa o intervalo máximo de espera para a chegada de uma mensagem, após o qual será detectada a possibilidade de um *deadlock*. Com certeza a utilização de tal mecanismo irá provocar uma redução dos falsos *deadlocks*, porém a dificuldade de se implantar tal técnica encontra-se na determinação do tamanho de β. Este valor não pode ser muito pequeno, pois não atenderia a sua real finalidade de eliminar os falsos *deadlocks*, e não pode ser muito grande, visto que o processo, ao detectar a possibilidade de *deadlock*, ficaria muito tempo parado. Só após ter transcorrido o intervalo β é que o processo poderia acionar o mecanismo de recuperação de *deadlock*,

o que geraria uma queda de desempenho quando os supostos falsos *deadlocks* fossem *deadlocks* reais.

Fujimoto sugere que se construa algoritmos a fim de explorar a topologia do sistema a ser simulado, identificando quais os processos que possuem relação direta emissor-receptor. O conhecimento da topologia poderá auxiliar na redução das transmissões de mensagens para detectar o maior *timestamp* entre os processos que têm ligação direta a fim de quebrar o *deadlock*.

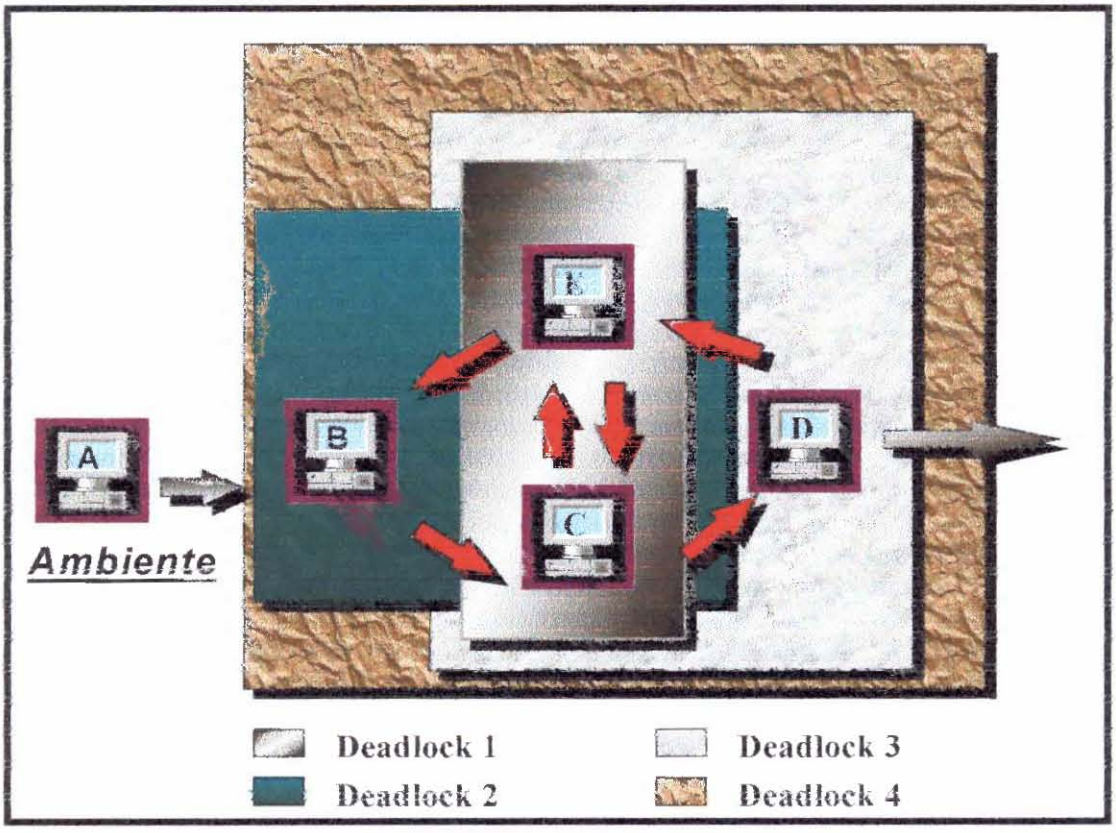


FIGURA 3.9 - Grupos de processos com possibilidade de *deadlock*

Conforme a figura 3.9, quatro grupos de processos poderão causar situações de *deadlock*, o primeiro grupo sendo composto pelos processos E e C, o segundo, por E, C e B , o terceiro, por E, C e D, e o quarto grupo composto por B, C, D e E.

3.3 Paradigma Otimista

As técnicas otimistas não controlam a discrepância de sincronismo entre os processos, ou seja, cada processo vai executando seus eventos sem se preocupar com eventos futuros, que poderão advir dos processos que abastecem suas entradas. Esse tipo de comportamento poderá fazer com que ocorra a chegada de uma mensagem fora de ordem de *timestamp*, tempo de simulação em que a mensagem ocorreu, o que deverá provocar um ***rollback***, a fim de desfazer os eventos já simulados, voltando os processos afetados até um ponto de segurança. O mecanismo de recuperação baseia-se em desfazer todos os eventos com tempo superior ao do *timestamp* da mensagem que gerou o início do *rollback*. Para que seja possível executar este mecanismo, é necessário que o processo salve constantemente os seus estados intermediários. A figura 3.10 demonstra como as mensagens são tratadas em sistemas que utilizam o paradigma otimista. O processo B está com sua fila de menor *timestamp* vazia, tal fila refere-se a comunicação com D. Porém, ao receber uma mensagem de A, o processo B irá tratá-la, mesmo que isto venha mais tarde a provocar um erro devido à chegada de uma mensagem do processo D com *timestamp* menor.

O algoritmo Time Warp é o mais conhecido da classe dos otimistas. Este se baseia no paradigma do tempo virtual. Este mecanismo detecta uma discrepância no tempo de simulação quando ocorre a chegada de uma mensagem com *timestamp* fora da ordem cronológica. Tal fato irá fazer com que o algoritmo inicie o procedimento de recuperação.

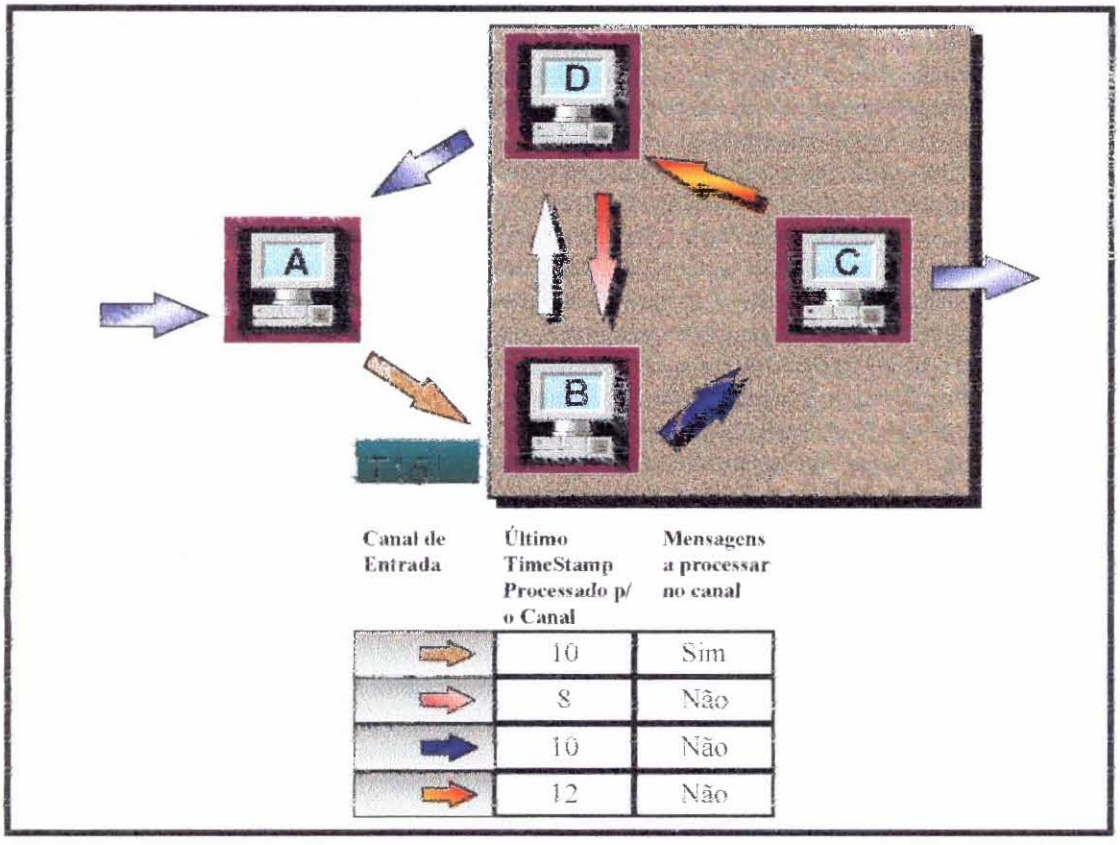


FIGURA 3.10 - Exemplo referente ao paradigma otimista

O processo de salvamento das informações que antecede o envio de uma mensagem de um processo produtor para um receptor tem um custo altíssimo, pois várias mensagens são enviadas constantemente, conforme a carga do produtor, logo a quantidade de memória a ser utilizada para armazenar tais informações pode ser muito grande. Para resolver este tipo de problema ,a literatura enfoca um método principal, o qual vem a ser o cálculo do tempo virtual global, ***TVG***, que é o menor tempo de simulação entre todos os processos envolvidos na simulação. Informações anteriores ao TVG podem ser descartadas, já que garantidamente o *rollback* não retrocederá além do TVG. Para que seja possível calcular o TVG, é necessário criar-se um processo que será responsável por tal cálculo, o qual se denominará como ***PTVG***. O PTVG deverá de tempos em tempos solicitar a todos os processos que estes enviem seu menor tempo local de simulação. Esta tarefa irá acarretar um grande fluxo de mensagens, possivelmente diminuindo o desempenho da simulação. O intervalo de tempo em que o PTVG irá calcular o TVG deverá ser muito bem definido, levando-se em conta dois aspectos:

1) a quantidade de estados armazenados pelo processo, visto que, quanto menor for o intervalo do cálculo do TVG, menor será a necessidade de memória para armazenar tais estados;

2) a interrupção do PTVG sobre os processos em simulação, causando um grande tráfego de mensagens.

A figura 3.11 demonstra uma situação onde deverá ocorrer um *rollback* devido à chegada de uma mensagem atrasada. Tal situação foi provocada devido ao fato de que o grupo de processos que estão demarcados na figura (região com fundo verde) já havia interagido várias vezes, elevando a cada passagem o tempo atual de cada um, enquanto que o processo ***E*** está processando a primeira mensagem recebida de ***A***. Quando ***E*** conseguiu propagar a sua mensagem para ***B***, o tempo de simulação atual do mesmo já estava além do *timestamp* da mensagem recebida de ***E***.

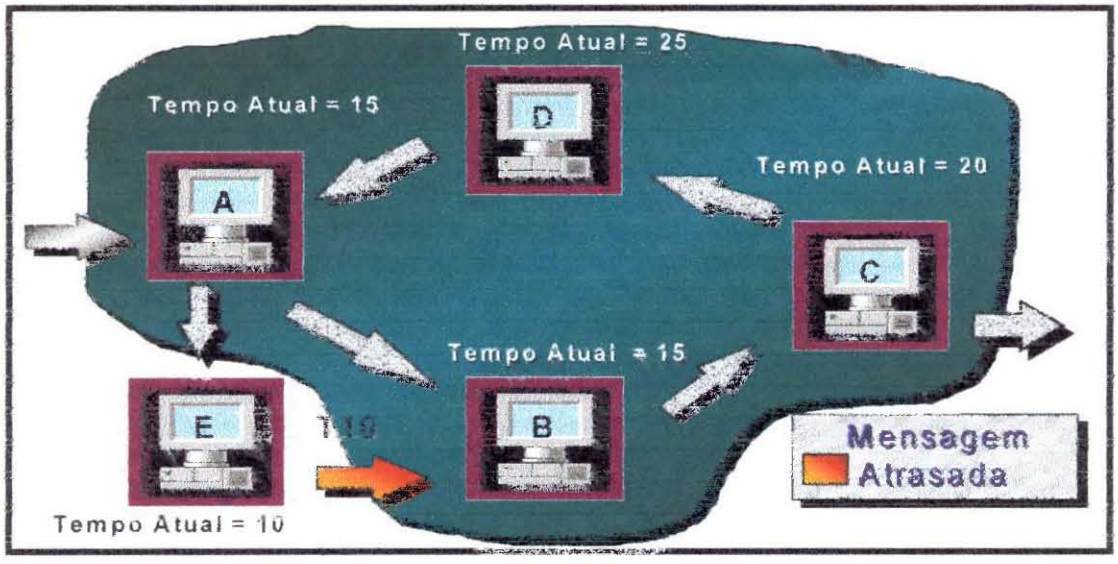


FIGURA 3.11 - Chegada de mensagem atrasada

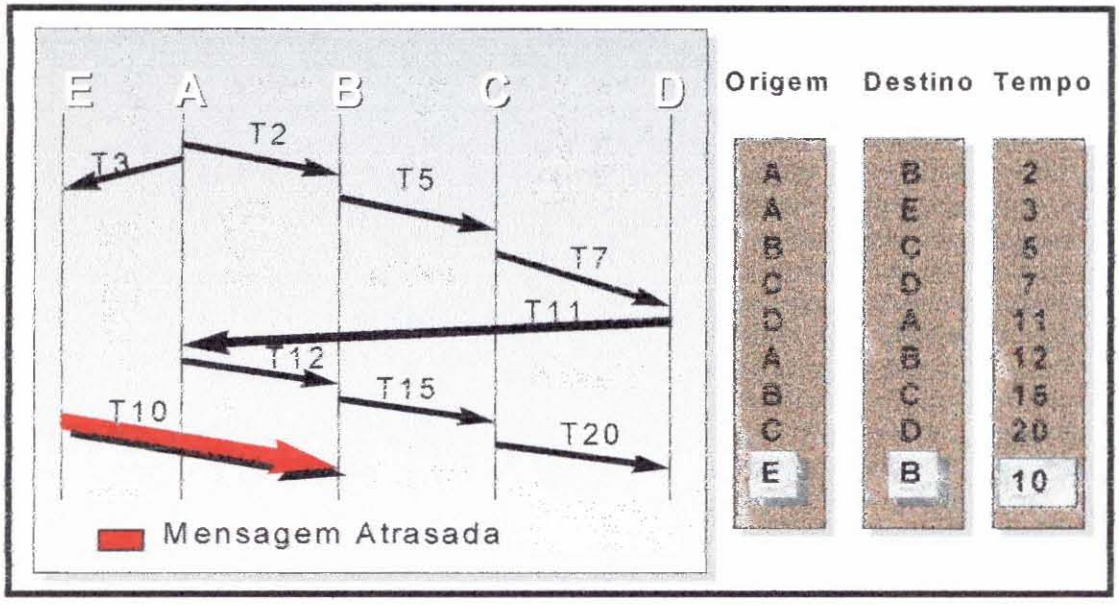


FIGURA 3.12 - Situação de retrocesso

A figura 3.12 mostra como as mensagens foram propagadas entre os processos.

Conforme as técnicas descritas na literatura, deve-se proceder o retrocesso das mensagens a fim de que os processos sejam colocados em uma situação de segurança. Esta situação poderá ser obtida através da consulta ao PTVG para que este informe o TVG atual, e que o processo onde a discrepância de tempo ocorreu (B) possa enviar antimensagens aos processos para os quais ele enviou alguma mensagem, no intervalo de tempo entre o tempo da mensagem que gerou a discrepância (10, no caso) e o tempo atual deste processo (15, no caso). Cada processo, ao receber uma antimensagem, deverá iniciar o retrocesso das já enviadas neste intervalo de tempo, caso necessário, até o TVG. Tal situação poderá provocar um efeito cascata no que diz respeito ao retrocesso, efeito este que terá como limite máximo o TVG. A figura 3.13 mostra o algoritmo baseado no paradigma otimista.

```
// Algoritmo fazendo uso do paradigma Otimista

Início
   Mensagens = 0;
   Faça
      Se Há Nova Mensagem Enviada
            Se Mensagem != Interromper_Simulação
                  Se Mensagem.Tipo != Anti mensagem
                        Se Insere_Mensagem ( ) == Ok
                           Mensagens++;          // Contador de mensagens a
                                                    serem processadas
                        Senão
                           RollBack()
                        FimSe
                  Senão
                        RollBack()
                  FimSe
            Senão
                Interrompe_Simulação
            FimSe
      FimSe
      Se Mensagens > 0
            Se Busca_Situaçào_Canais ( ) = Ok
                  Executa_Mensagens ( )
            FimSe
      FimSe
   FimFaça
Fim
```

Insere_Mensagem ( )

I. Busca na estrutura de canais o canal que irá receber a mensagem enviada;
   A. Confronta o TimeStamp da Mensagem enviada com o último TimeStamp processado, caso o TimeStamp da Mensagem seja menor que o último processado para este canal, retorna com "ERRO"
II. Insere os dados contidos na mensagem na estrutura de MENSAGENS;
III. Retorna "OK"

Busca_Situação_Canais ( )

1. Detecta qual o próximo *Timestamp* menor dentre as mensagens dos canais, guardando-o na variável Exe_TimeStamp.

Executa-Mensagens ( )

1. Salva Estados
2. Seleciona as mensagens que estão programadas com o *timestamp* = Exe_TimeStamp;
3. Ajustar valores dos Canais de entrada (Somente aqueles que tenham eventos programados para o *timestamp* identificado como o menor de todos dentre as mensagens já enviadas);
4. Mensagens--;

FIGURA 3.13 - Algoritmo fazendo uso do paradigma otimista

4 Plataforma de distribuição a ser utilizada

4.1 Introdução

A fim de realizar os testes dos algoritmos propostos neste trabalho, optou-se pela utilização da API WinSock para Windows.

Em primeira instância pensou-se na utilização de um sistema operacional de rede heterogêneo orientado à programação de aplicações paralelas e distribuídas, conhecido como HETNOS [BAR 92], tal como originalmente proposto por Figueiró [FIG 94]. O HETNOS está sendo desenvolvido na Universidade Federal do Rio Grande do Sul (UFRGS) e está disponível para rodar em ambiente Unix sob uma rede de estações de trabalho SUN. Este sistema mostrou-se interessante para os testes dos algoritmos deste trabalho, porém ao ser feita uma análise mais criteriosa, apresentada a seguir, optou-se pela utilização da API WinSock.

A API WinSock é uma biblioteca de funções que implementa a interface de *sockets* da forma popularizada pela Berkeley *Software* Distribution do UNIX .

Um *socket* pode ser considerado como um ponto de referência para o qual as mensagens são enviadas e a partir do qual as mensagens podem ser recebidas. Dessa forma, eles estabelecem um canal de comunicação e podem começar a enviar e receber mensagens um para o outro. Uma estação de trabalho A que deseje se comunicar com uma estação de trabalho B deve saber o número do *socket* aberto pela aplicação que roda em B e vice-versa.

Há vários tipos de *sockets* (*stream*, datagrama, puro, pacotes seqüenciais, etc.), sendo a disponibilidade de um determinado tipo dependente do domínio considerado.

A API WinSock suporta dois tipos de *sockets*, *stream* e datagrama. O *socket* do tipo datagrama transmite os dados de forma não confiável (não há seqüenciamento, nem controle de fluxo nem de perda de pacotes) através de mensagens. Já o *socket* do tipo *stream* transmite os dados de forma confiável, pois este

possui controle de seqüenciamento, controle de fluxo e erros), sendo este orientado a conexão.

As vantagens do *socket* do tipo datagrama são a ausência de uma fase de conexão e a não limitação quanto ao número de conexões abertas. O *socket* do tipo *stream* possui como vantagens o seu desempenho quando o fluxo de dados é grande e a sua confiabilidade.

4.2 Comparação entre HETNOS e WinSock

Para que seja possível avaliar os motivos pelos quais a plataforma WinSock foi escolhida, ao invés da plataforma HETNOS previamente proposta por Figueiró, será feita uma análise comparativa entre ambas. Foram levados em conta diversos fatores; os resultados são listados brevemente a seguir:

1. transparência de localização dos processos na rede - tanto o HETNOS como a WinSock proporcionam ao usuário a não vinculação entre processo e processador, ou seja, o usuário não precisa saber em que processador está sendo executado determinado processo, bastando enviar uma mensagem que indique para qual processo esta se destina;

2. primitivas de alto nível - o HETNOS e a WinSock disponibilizam ao usuário primitivas de alto nível para a comunicação entre processos;

3. documentação - o HETNOS possui documentação *on-line*, porém esta está constantemente sofrendo alterações devido às novas versões, visto que o HETNOS é um sistema que ainda está sendo desenvolvido. Em contrapartida, a WinSock possui uma farta documentação;

4. sistemas operacionais - o HETNOS atualmente só pode ser executado em ambiente UNIX, enquanto que a WinSock está disponível para diversos sistemas operacionais;

5. arquitetura orientada a mensagens - ambos são orientados a mensagens.

4.2.1 Vantagens do WinSock

A WinSock possui algumas vantagens em relação ao HetNOS, as quais são:

1. plataforma Windows - visto que uma provável versão do AMPLO será implantada nesta plataforma, facilitando a distribuição junto às entidades acadêmicas;

2. facilidade de implantação dos algoritmos - devido ao fato da WinSock estar vinculada a qualquer ambiente Windows, torna-se fácil a sua implantação, enquanto que para os testes utilizando o HETNOS seria necessário obtê-lo junto ao grupo de pesquisa da UFRGS.

3. facilidade em se montar um ambiente de teste.

4.3 Características da WinSock

Um *socket* pode possuir um nome e estar associado a um processo. O *socket* serve simplesmente como interface entre o aplicativo e a camada do sistema operacional, conforme a figura 4.1.

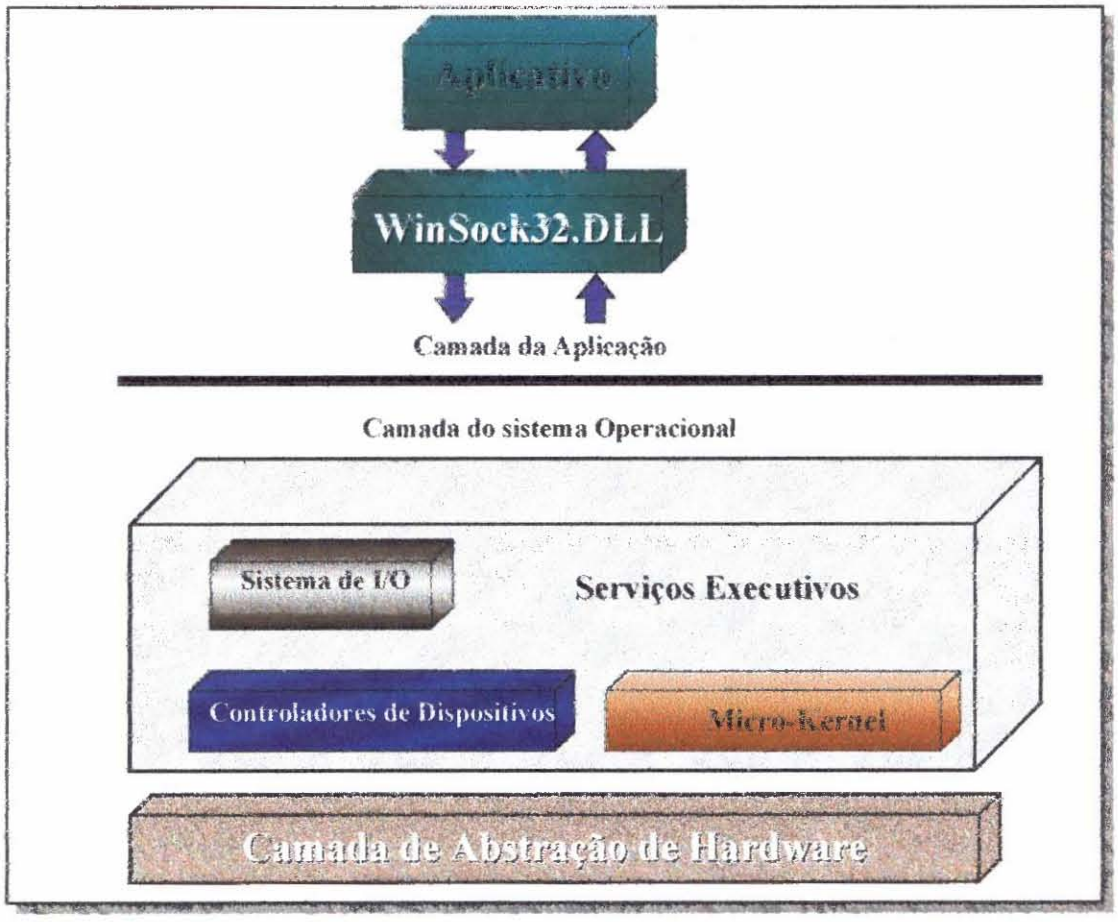


FIGURA 4.1 - Uso de *sockets*

A comunicação entre dois *sockets* se faz de forma bi-direcional, conforme demonstrado na figura 4.2.

A WinSock proporciona dois tipos de transmissão, uma confiável, obtida através do serviço de fluxo orientado, TCP (*Transmission Control Protocol*), e outra não confiável, obtida pelo uso de datagramas, UDP (*User Datagram Protocol*).

O uso da WinSock baseado no TCP faz com que um *socket* só se comunique com outros *sockets* aos quais este foi previamente conectado.

A figura 4.2 demonstra como é feita a conexão entre dois *sockets*, um sendo o servidor e outro sendo o cliente.

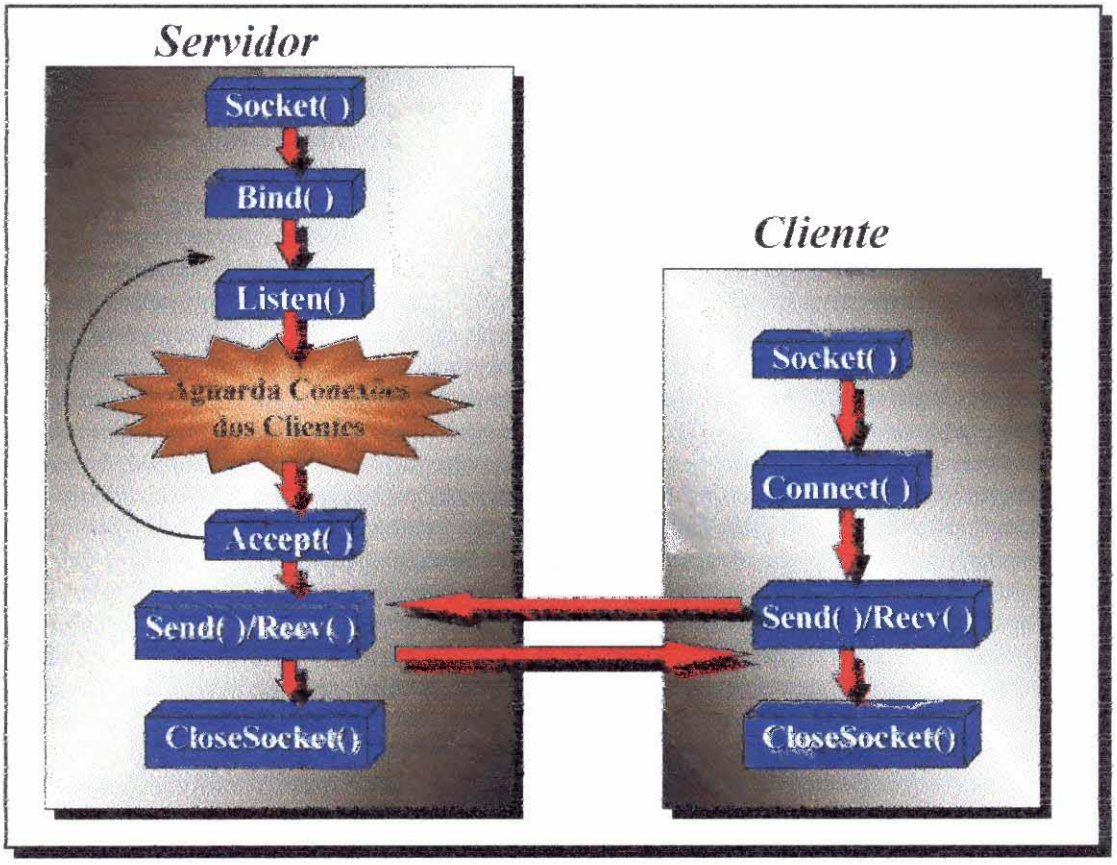


FIGURA 4.2 - Conexão entre *sockets*

A WinSock usando TCP se adapta bem à arquitetura Cliente/Servidor. Em uma típica interação Cliente/Servidor, o servidor cria um *socket*, dá a ele um nome e espera por clientes que se conectem ao *socket* por ele produzido. O cliente por sua vez cria também um *socket* e se conecta ao *socket* nomeado no servidor. Quando o servidor detecta a conexão para o *socket* nomeado, ele cria um novo *socket* e o utiliza para efetuar a comunicação com o cliente. O *socket* nomeado no servidor continua a esperar por novas conexões de outros clientes. A comunicação entre o cliente e o servidor é demonstrada através da figura 4.3.

A WinSock disponibiliza um grupo de classes, as quais contém rotinas previamente elaboradas para facilitar o trabalho de comunicação. Estas, por sua vez, fazem uso das funções básicas fornecidas para a comunicação entre processos.

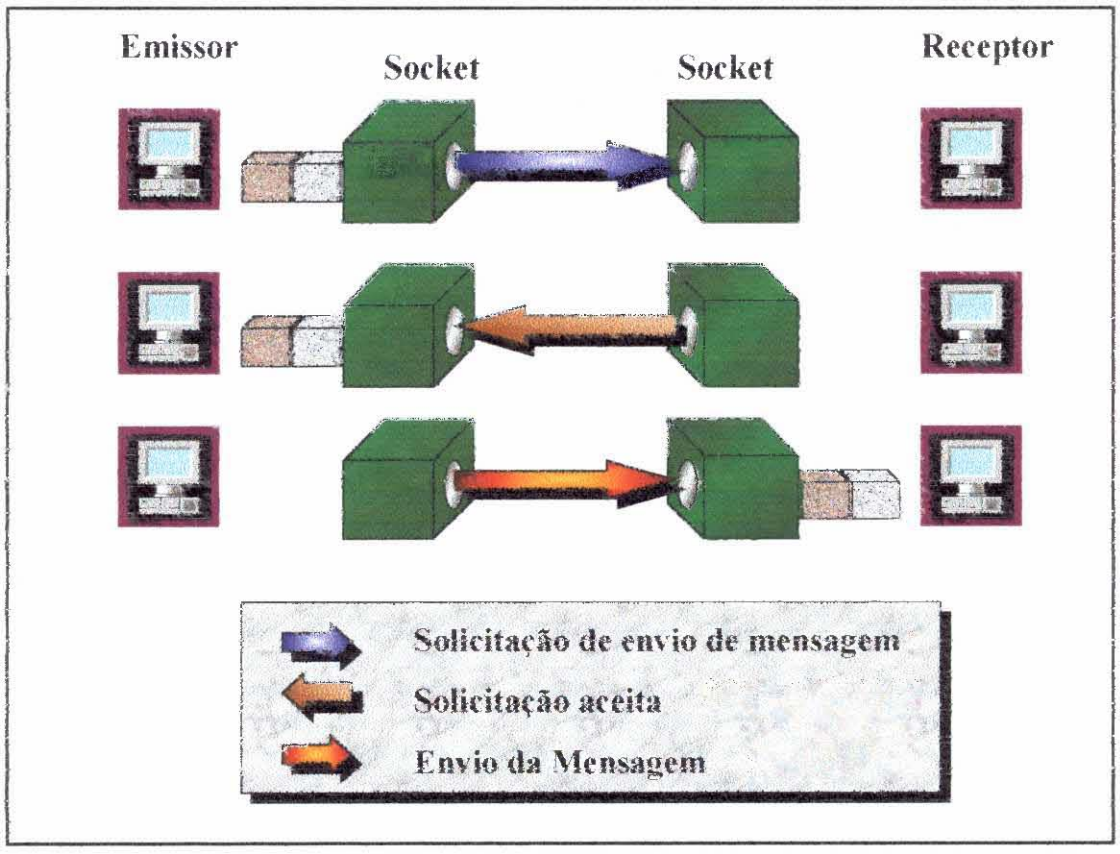


FIGURA 4.3 - Envio de mensagens via *socket*

5 Estudo das propostas de Figueiró

Torna-se necessário o estudo das técnicas propostas por Figueiró [FIG 94], visto que ambas as dissertações trabalham sobre o mesmo enfoque que é a distribuição da simulação de sistemas discretos, em especial o sistema AMPLO.

Figueiró propôs duas versões para a distribuição do sistema de simulação AMPLO, a versão parcialmente distribuída e a versão totalmente distribuída.

5.1 Versão parcialmente distribuída

5.1.1 Descrição

Nesta versão Figueiró diz que o mestre continuará a existir e que este irá receber o modelo a ser simulado, o qual contém a lista de agências primitivas e a lista de conexões entre estas.

A distribuição será aplicada somente no nível dos escravos, os quais serão processos à parte. Para cada agência primitiva será criada uma instância do escravo em que esta foi descrita. A instância por sua vez irá receber do mestre a estrutura da agência a ser simulada por esta. Cada instância disparada ficará aguardando ordens do mestre para evoluir o tempo de simulação local à agência.

O mestre irá manter, como na simulação seqüencial, suas funções básicas, as quais são:

- avançar o tempo de simulação;
- coordenar a comunicação entre as agências;
- executar o algoritmo de consenso de barramento quando necessário, e
- promover a adaptação de sinais.

Tendo-se como exemplo a topologia da agência descrita na figura 5.1, este método forçará a criação dos processos conforme a figura 5.2.

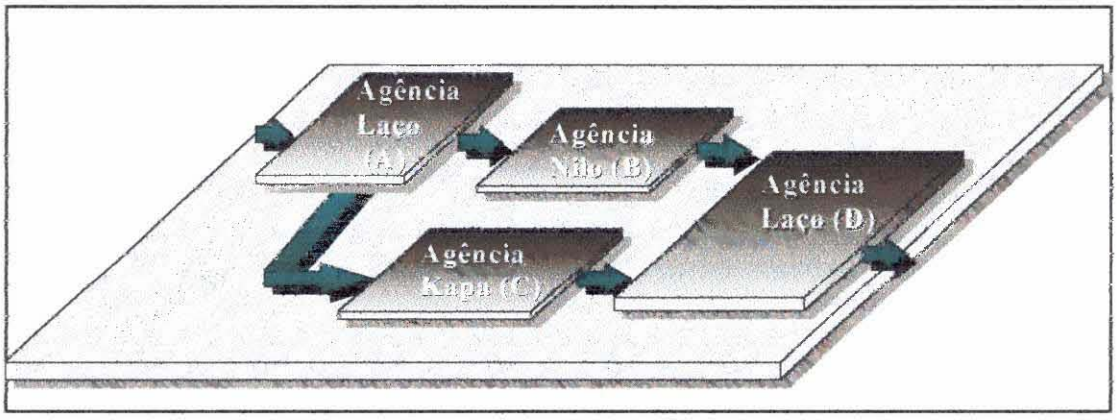


FIGURA 5.1 - Exemplo de topologia de agência

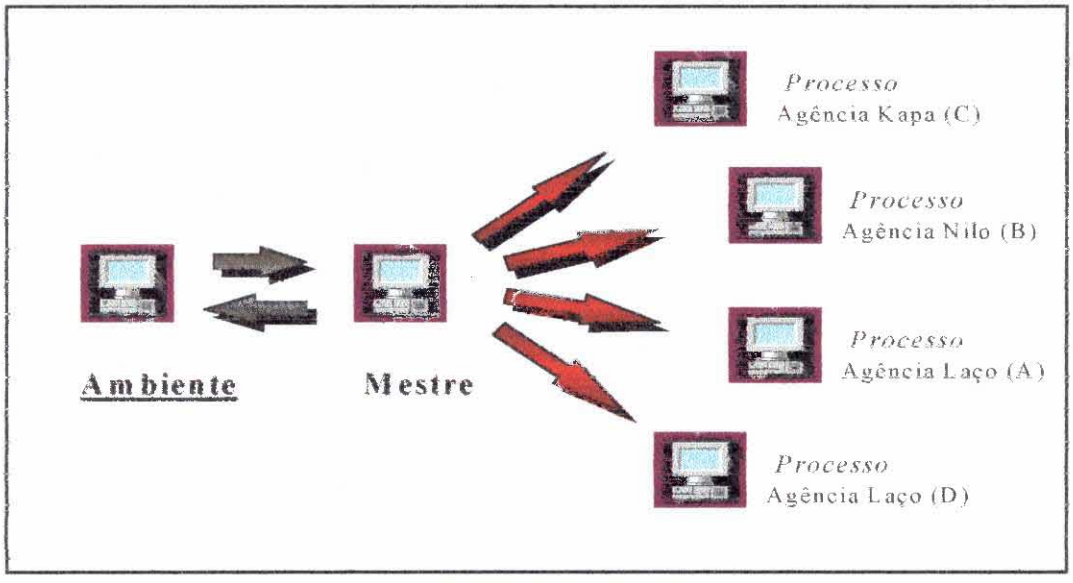


FIGURA 5.2 - Criação das instâncias dos simuladores escravos

A figura 5.3 mostra um trecho de algoritmo que, conforme Figueiró, implementa a ativação de um escravo para cada agência existente na rede de agências primitivas pertencente ao modelo a ser simulado.

```
Enquanto agencia.prox != NULL
inicio
        agencia = busca_agencia(rede_de_agencias)
        se      agencia.tipo == KAPA
        então   dispara(escravo_KAPA,agencia.nome)
        se      agencia.tipo == LACO
        então   dispara(escravo_LACO,agencia.nome)
        se      agencia.tipo == NILO
        então   dispara(escravo_NILO,agencia.nome)
fim
```

FIGURA 5.3 - Algoritmo para ativação de processos escravos

5.1.2 Críticas à versão parcialmente distribuída

O fato de ser disparado um processo para cada instância de escravo utilizado no sistema a ser simulado, ou seja, para cada agência do modelo, poderá provocar uma sobrecarga na rede usada como plataforma, visto que um sistema complexo, subdividido em muitos módulos de complexidade inferior, fará com que sejam disparados inúmeros processos causando possivelmente sérios problemas no desempenho da simulação.

Outro fato muito importante é a não utilização real das possibilidades de distribuição de um sistema, pois mantendo-se o mestre como pivô para a propagação das mensagens entre os processos pode-se causar basicamente uma situação de processamento seqüencial.

5.2 Versão totalmente distribuída

A versão totalmente distribuída prevê a independência dos simuladores escravos do controle exercido pelo simulador mestre.

Os escravos irão absorver as funções básicas do mestre, as quais são:

- o avanço do tempo de simulação;
- coordenar a comunicação com as agências vizinhas;
- executar o algoritmo de consenso de barramento quando necessário, e;
- promover a adaptação de sinais.

Os escravos deverão ter conhecimento da topologia da rede de agências primitivas que compõem o modelo a ser simulado, com o objetivo de promover a comunicação entre estas.

Os escravos deverão receber do ambiente a lista de estímulos iniciais que deverão ser aplicados antes da inicialização da simulação.

Para a versão totalmente distribuída, Figueiró propôs duas alternativas, uma utilizando-se de técnica conservativa e outra com técnica otimista.

5.2.1 Com sincronização conservativa

O proposto para esta versão é a utilização de mensagens nulas a fim de manter a sincronização, conforme introduzido na seção 3.2.1. A cada evento futuro a ser simulado dentro de cada agência será transmitida uma mensagem aos processos receptores conectados àquele que está gerando os eventos internos futuros, conforme demonstra a figura 5.4. Na figura está sendo executada no processo A a agência X (nível LAÇO), a qual possui quatro transições seqüenciais. Com a utilização de mensagens nulas para manter o sincronismo entre as agências, serão necessárias oito mensagens a serem propagadas pela rede onde a simulação está ocorrendo. Este número de mensagens é necessário visto que as agências Y e Z poderão estar aguardando

algum valor de X; logo esta tem a obrigação de informar os tempos seguros de suas próximas transmissões.

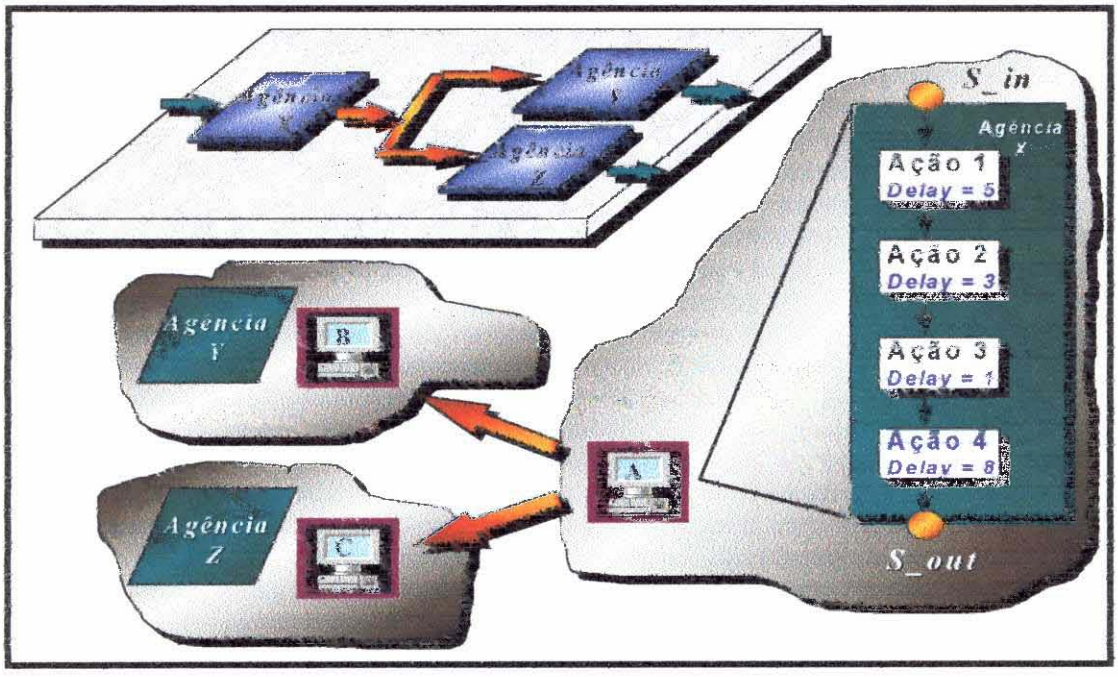


FIGURA 5.4 - Sincronização Conservativa

A agência X estando com o seu tempo local em 25 e tendo esta recebido um estímulo em seu sinal de interface S_in, promoverá a avanço do tempo através da simulação. A ação 1 é colocada na roda de eventos futuros local à agência para ser disparada após cinco unidades de tempo, logo a próxima mensagem só será gerada no tempo de simulação 30. Desta maneira o escravo promove a transmissão de uma mensagem nula da agência X para as agências Y e Z, as quais possuem conexão direta com esta. As agência Y e Z poderão avançar sua simulação até o tempo 30. À medida que a simulação vai avançando na agência X, novas mensagens nulas serão transmitidas devido à entrada de novos eventos na roda de eventos interna à agência.

5.2.1.1 Críticas à versão conservativa

A utilização de mensagens nulas para se obter a sincronização provocará um fluxo muito grande de mensagens na rede, o que acarretará na queda de desempenho.

5.2.2 Com sincronização otimista

A fim de promover o controle da discrepância entre os tempos das mensagens relativas a um sinal, é proposto que se utilize o mestre como pivô, a fim de que este calcule o tempo virtual global do sistema, conforme introduzido na seção 3.3. A partir do momento que este tempo seja calculado, o mestre deverá informar a todos os processos o valor obtido para que os mesmos possam atualizar suas filas de estados armazenados. O processo de cálculo do tempo virtual global (TVG) se dá em três fases, conforme mostra a figura 5.5.

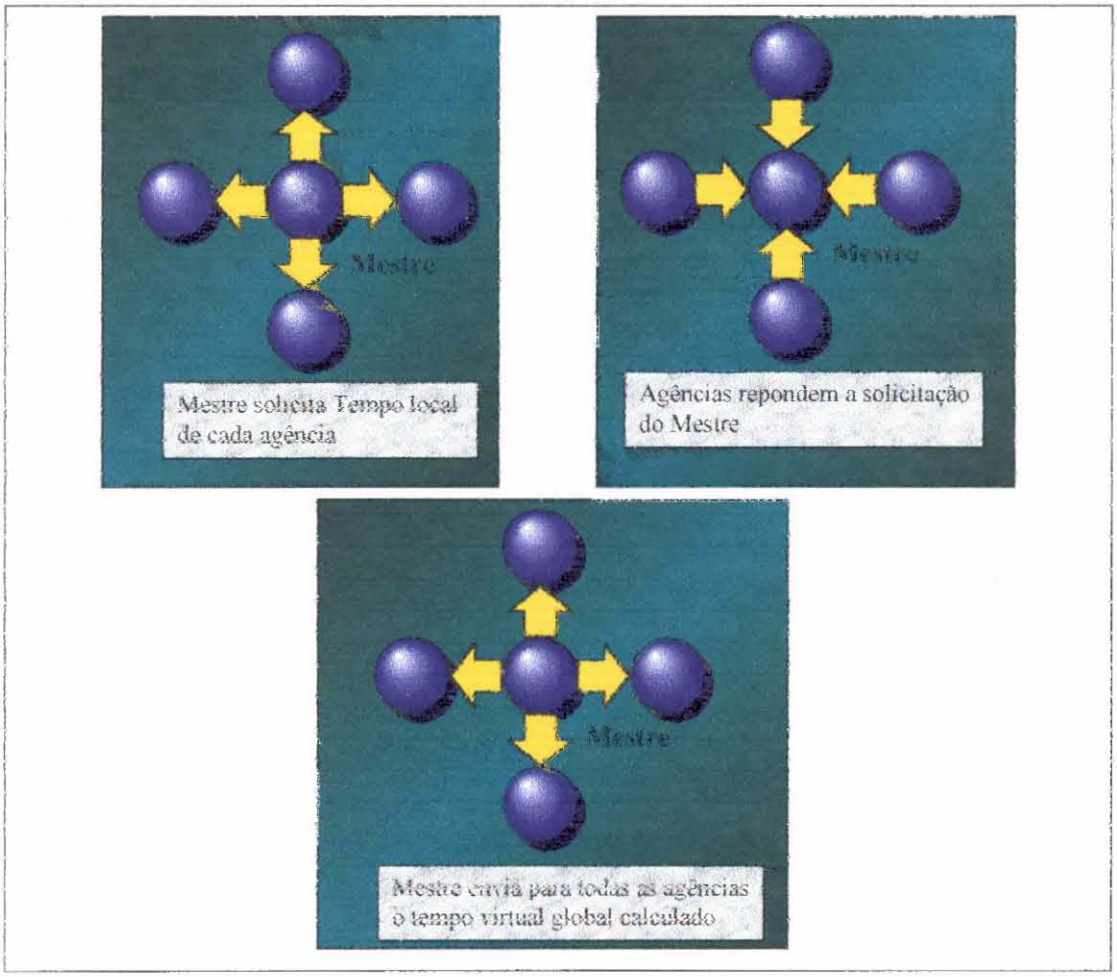


FIGURA 5.5 - Fases para cálculo do Tempo Virtual Global

É proposto também que, para cada sinal de saída das agências primitivas, deva-se ter uma fila com os dados enviados desde o último tempo virtual global informado pelo mestre. Tais dados irão constituir as antimensagens a serem enviadas aos escravos vizinhos caso seja detectada a chegada de uma mensagem atrasada de um processo escravo, causando desta maneira um processo de *rollback*.

5.2.2.1 Críticas à versão otimista

Nenhuma referência é feita por Figueiró em relação a como calcular a periodicidade na qual o mestre se baseará para interromper os escravos a fim de obter de cada um o seu tempo local de simulação, para que seja possível efetuar-se o cálculo do tempo virtual global.

Figueiró diz que as antimensagens devem possuir o mesmo formato de uma mensagem comum. Porém, esta estrutura não precisaria guardar os dados relativos aos sinais transmitidos por uma agência naquele momento, pois para acionar o processo de rollback basta o processo receptor.

6 Ambiente de Simulação

Para que se possa tirar o maior proveito possível das potencialidades da distribuição, são propostas neste trabalho duas versões, ambas totalmente distribuídas, porém uma fazendo uso do paradigma conservativo e outra do paradigma otimista.

Ambas versões irão necessitar conhecer a topologia do sistema que será simulado. A obtenção dos dados relativos à topologia da rede de agências primitivas a serem simuladas torna-se de certa forma fácil, visto que o AMPLO possui o módulo denominado construtor de modelos (ver seção 2.1.1), o qual promove o achatamento da hierarquia de agências, ou seja, monta uma estrutura a ser utilizada para a simulação que possui unicamente agências primitivas e as ligações entre estas.

Para efetuar os testes dos algoritmos propostos foi necessário criar um "falso" ambiente, pois o ambiente de simulação do projeto AMPLO, introduzido na seção 2.1, não está operacional em sua plenitude.

Os simuladores escravos utilizados para os testes foram o simulador NILO e o simulador LAÇO. Ambos simuladores não estão totalmente operacionais.

Os escravos na versão seqüencial estão separados, sem a integração através do mestre, não sendo possível desta maneira avaliar o desempenho da simulação seqüencial do AMPLO com a utilização do ambiente de simulação. A fim de se obter os dados para a comparação entre a simulação seqüencial e a simulação distribuída, criou-se um conjunto de agências primitivas em NILO e em LAÇO e através de um mestre adaptado para esta situação foi feita a coleta dos dados.

6.1 Processo de instalação do ambiente de simulação

O ambiente deverá ser executado em um equipamento. Logo após a sua instalação este fornecerá o número do IP relativo ao equipamento e em que porta o socket foi criado, para que seja possível fornecer aos outros computadores o endereço de onde se encontra o ambiente.

Deve-se disparar o escravo em cada máquina que irá auxiliar na simulação. O termo escravo foi usado no singular, pois este irá incorporar os simuladores NILO , LAÇO e KAPA em um mesmo código.

O objetivo de incorporar os escravos em um mesmo módulo está relacionado à disponibilidade de memória, pois se fosse disparado um escravo para cada agência como um processo à parte logo se teria um excesso de processos, causando provavelmente um *overflow* na memória.

Ao ser criado um escravo, este solicitará o endereço IP e a porta onde se encontra o ambiente. Logo após tomar conhecimento da localização física do ambiente na rede, o escravo enviará uma solicitação de conexão a este.

Ao receber a solicitação do escravo para se conectar, o ambiente obterá automaticamente a localização deste, ou seja, o endereço IP e a porta de comunicação do escravo. A comunicação inicial entre ambiente e escravos pode ser abstraída da figura 6.1.

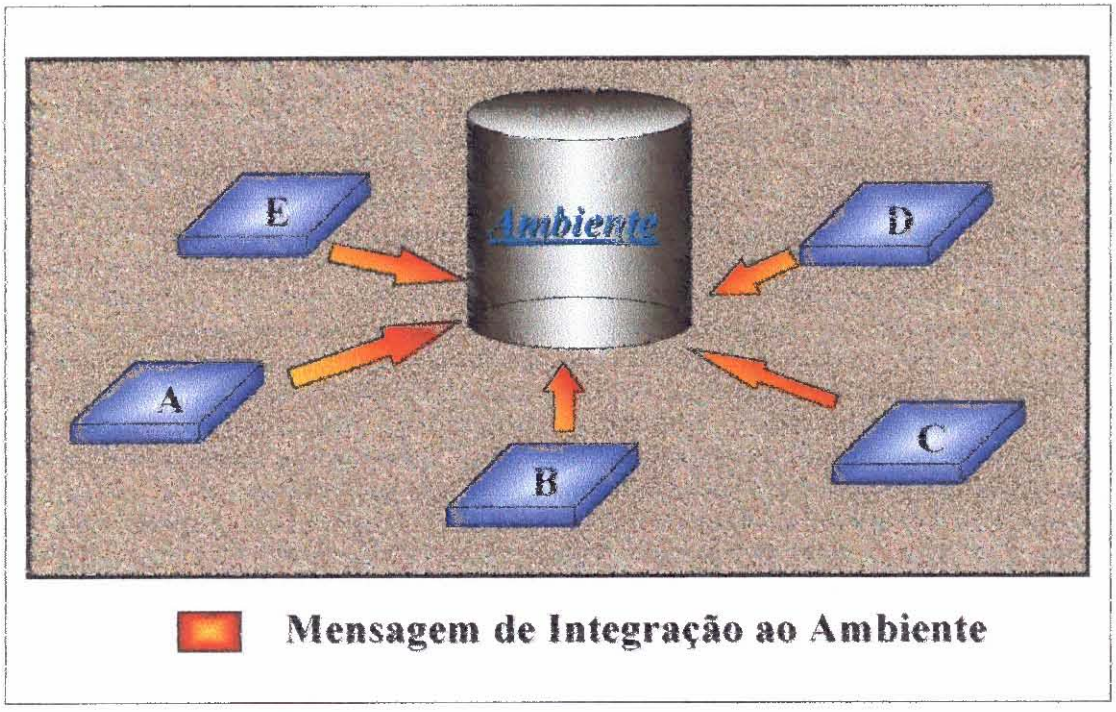


FIGURA 6.1 - Comunicação Escravos/Mestre p/ inicialização

6.2 Processos Escravos

Um escravo poderá ser responsável pela simulação de mais de uma agência. Para tal este contém uma estrutura a fim de armazenar os dados relativos a cada agência, seja ela NILO, LAÇO ou KAPA.

A figura 6.2 mostra a estrutura que será utilizada para guardar informações relativas às agências. Como o escravo poderá controlar diversos tipos de agências, torna-se necessário que esta estrutura seja heterogênea, podendo suportar dados relativos a qualquer agência, seja ela LAÇO, KAPA ou NILO.

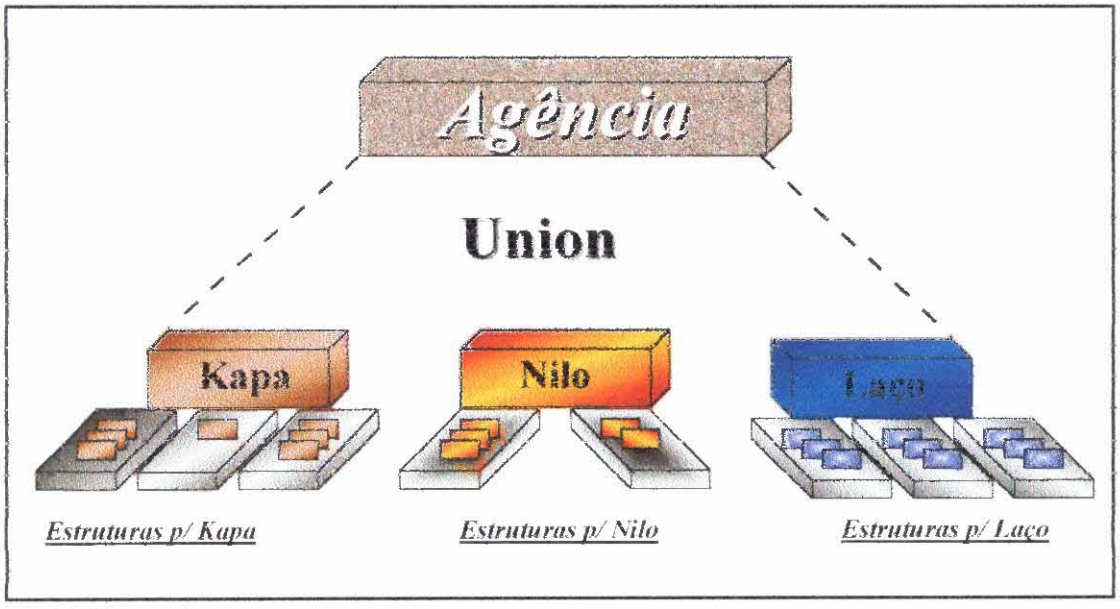


FIGURA 6.2 - Estrutura de armazenamento

O escravo absorveu dois deadlock módulos, o algoritmo de adaptação de sinais e o módulo de barramento, introduzidos na seção 2.2, os quais possuem a responsabilidade de proporcionar a comunicação entre sinais de diversos tipos de dados aceitos para cada tipo de agência, seja ela LAÇO, KAPA ou NILO.

Para que o escravo pudesse absorver o módulo de barramento tornou-se necessário alterar a estrutura que diz respeito à ligação entre as agências. Um canal de entrada de uma agência poderá ser composto por vários outros canais os quais contribuem para a resolução do barramento, ou seja, são os canais relativos aos sinais que participam do consenso.

A figura 6.3 mostra o formato básico de um escravo.

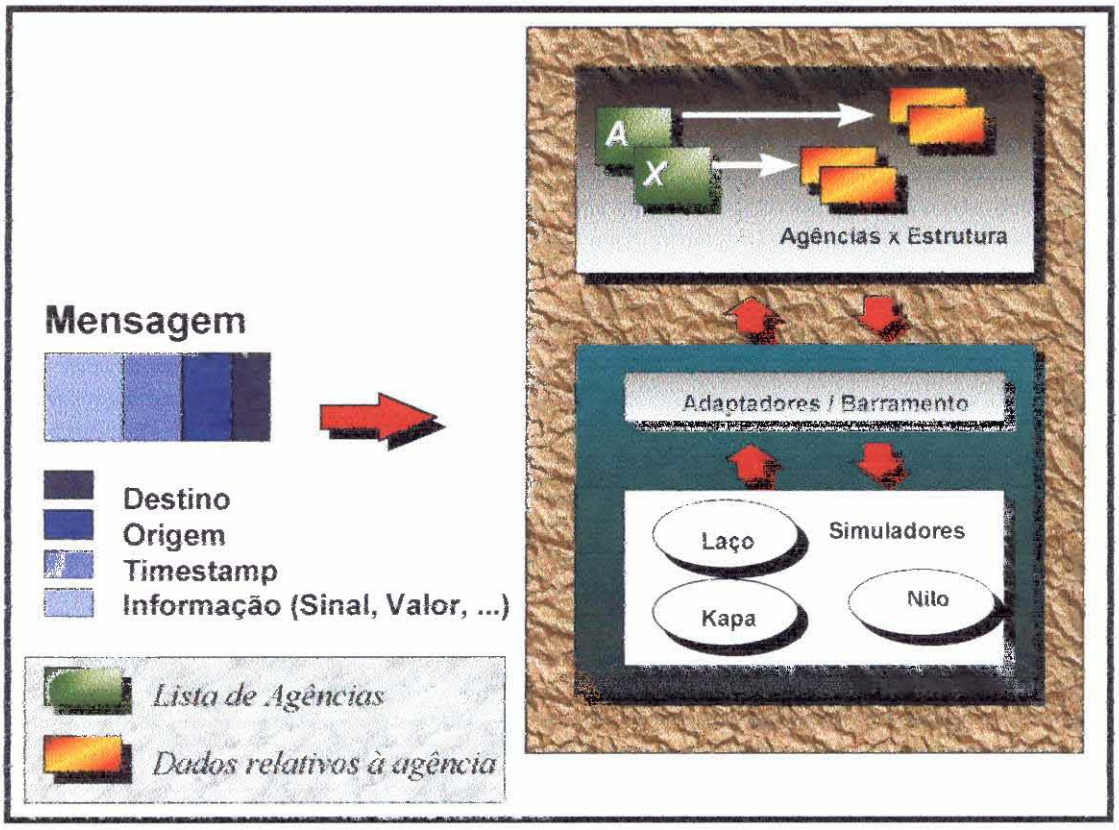


FIGURA 6.3 - Estrutura de um Escravo

Os escravos ficam em estado de prontidão aguardando o início da simulação, após terem recebido as informações relativas as agências que estes irão controlar, bem como o endereço de onde eles irão receber e enviar dados.

O escravo poderá receber diversos tipos de mensagens com tamanhos diversos.

Na simulação seqüencial era passado para o escravo um ponteiro informando a localização da estrutura que continha um conjunto de apontadores para as possíveis mensagens, sendo elas utilizadas ou não. Tal estrutura não pode mais ser utilizada, pois a comunicação será feita através de troca de mensagens pela rede, e estas devem transmitir pacotes de dados bem definidos onde deve-se indicar o tipo de mensagem e o número de elementos que ela contém.

Um exemplo de pacote de mensagens pode ser visto na figura 6.4.

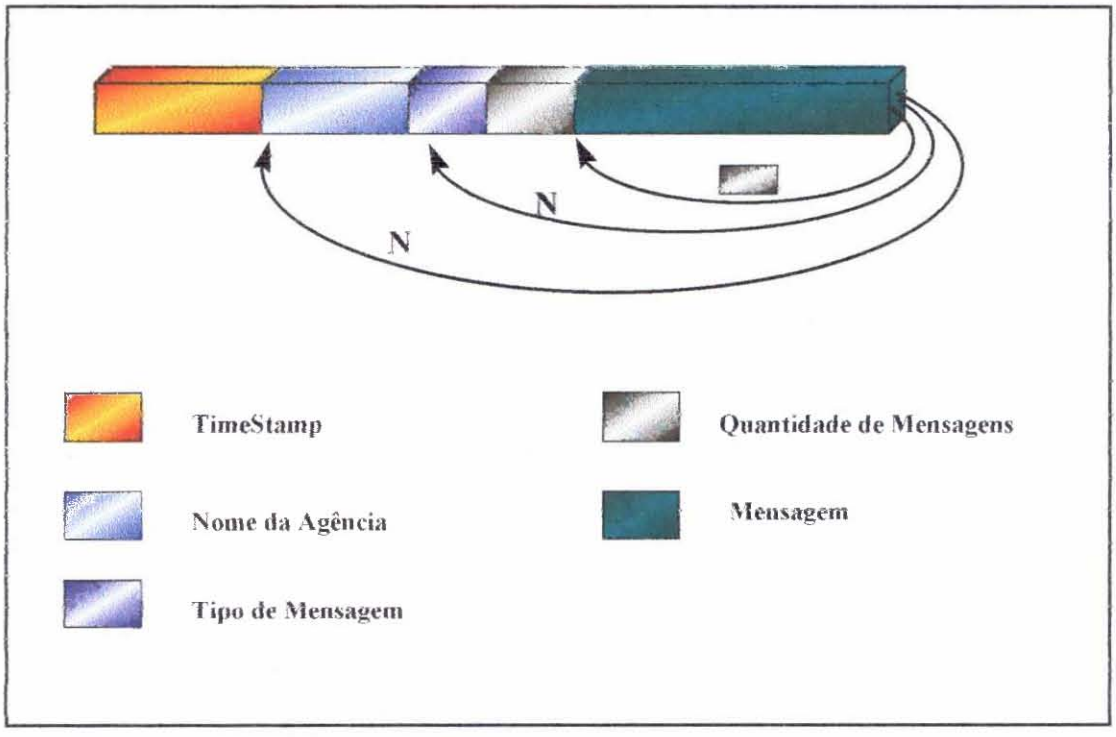


FIGURA 6.4 - Exemplo de pacote de mensagens

6.3 Distribuição de Carga

O ambiente, ao tomar conhecimento dos processos que irão auxiliar na simulação, procede à distribuição das agências entre estes.

Um dos problemas típicos de um ambiente de programação distribuída é a distribuição de carga, ou seja, a alocação de serviço segundo alguma política de distribuição [BAR 92].

Para que a distribuição das agências a serem simuladas dentro dos processos fosse feita com pelo menos um certo critério, tornou-se necessária a criação de um algoritmo, que tem como finalidade básica balancear de forma adequada a distribuição das agências que irão fazer parte da se simulação.

O critério utilizado foi a interconectividade das agências. Duas agências que possuem relação direta não devem rodar no mesmo processo, pois caso contrário

poderíamos causar um processamento seqüencial, visto que o escravo poderá controlar várias agências, porém este só tem capacidade de simular uma a cada vez. Quando é feita a referência a processos não deve-se confundir com processador, pois poderemos ter diversos processos sendo executados em um mesmo processador.

A fim de poder entender o funcionamento do algoritmo de escalonamento, será tomada como exemplo a agência da figura 6.5.

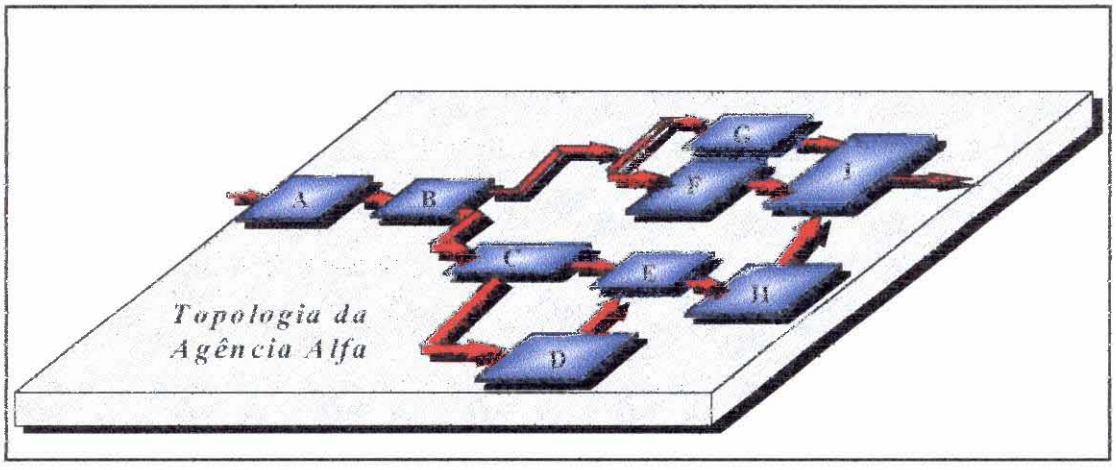


FIGURA 6.5 - Topologia da agência Alfa

A figura 6.6 representa a topologia da agência Alfa da figura 6.5 na forma de um grafo. O algoritmo de escalonamento irá percorrer este grafo, e conforme o número de processos que estejam disponíveis fará a distribuição das agências.

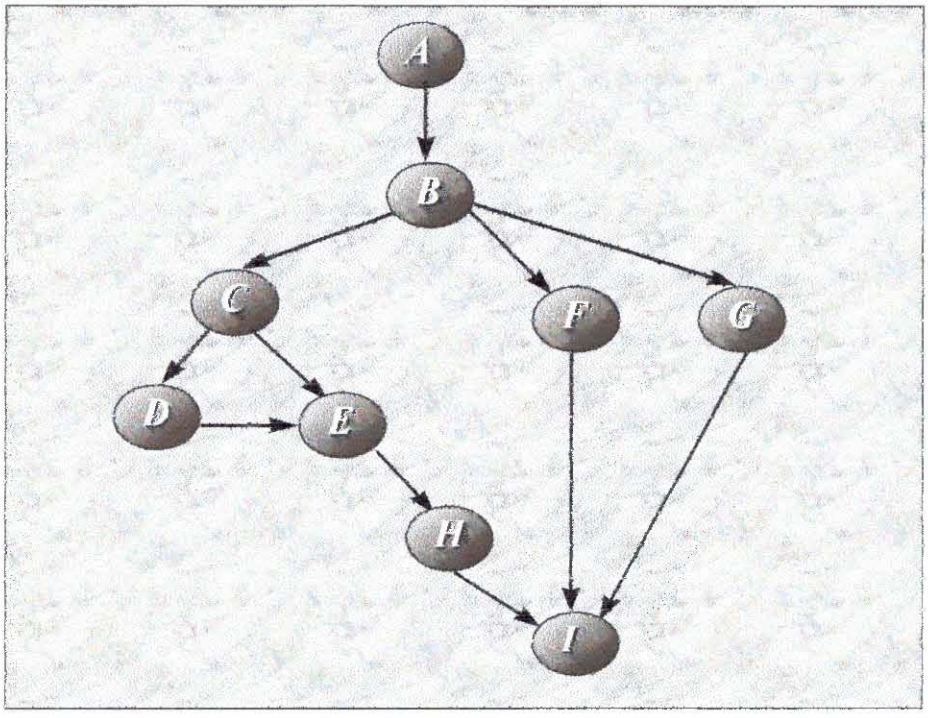


FIGURA 6.6 - Topologia da agência Alfa representada como um grafo

A figura 6.7 demonstra como ficaria a distribuição das agências primitivas que compõem a topologia da agência Alfa haja apenas três processos disponíveis para a execução da simulação.

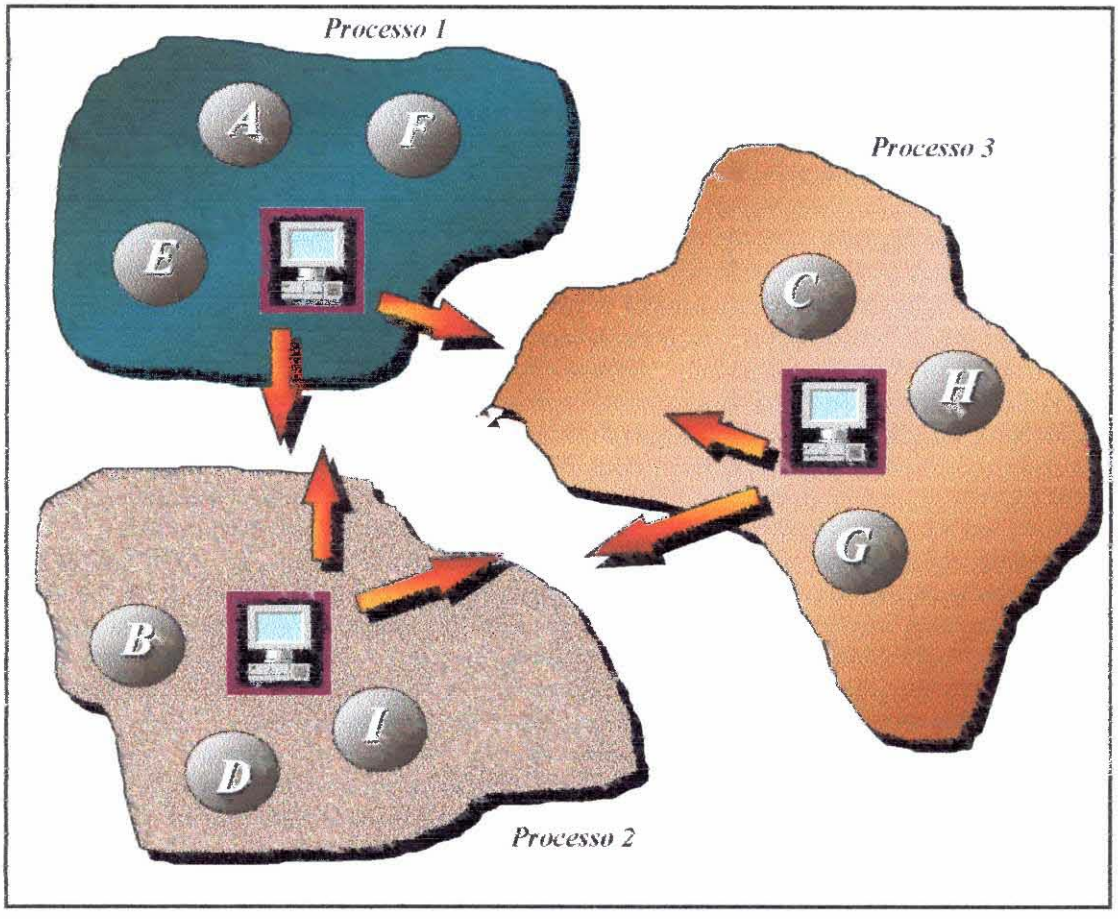


FIGURA 6.7 - Distribuição das agências entre os processos

7 Propostas de Distribuição

Para ambas versões propostas neste trabalho, os processos escravos têm a estrutura de comunicação correspondente à figura 7.1.

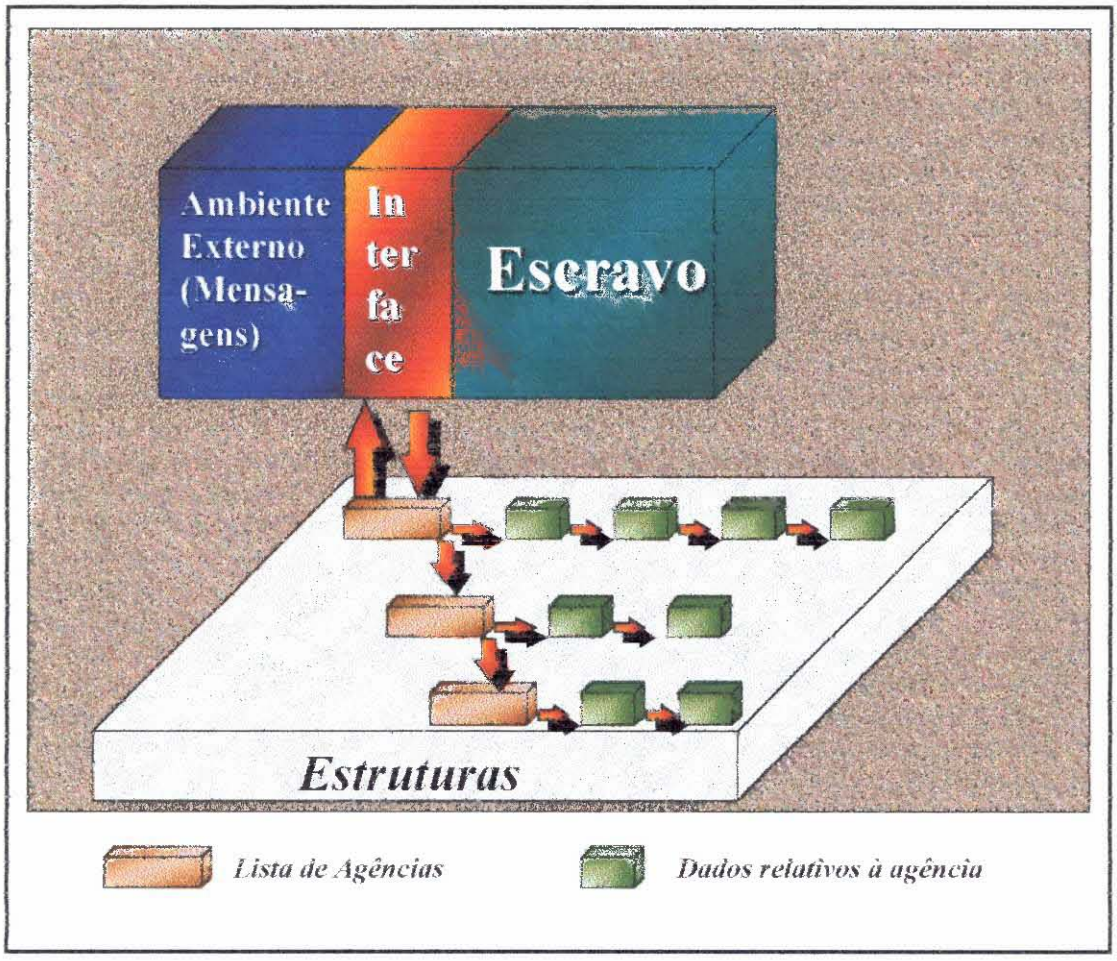


FIGURA 7.1 - Estrutura de comunicação do Escravo

A figura 7.1 retrata a ligação entre o ambiente externo e os simuladores escravos, que estão inclusos no módulo denominado "escravo". O módulo interface tem como objetivos principais:

1. promover a conexão com o ambiente;

2. promover a conexão com as outras agências que necessitarem de comunicação;

3. receber e enviar mensagens, promovendo o processo de empacotamento e desempacotamento das mensagens, e

4. ao receber uma mensagem, verificar qual a agência dentre as que este processo controla está sendo ativada, buscando na lista de agências o ponteiro para a estrutura de dados que descreve a funcionalidade desta, a fim de passar o endereço obtido e as mensagens para o escravo apropriado, conforme a agência.

7.1 Proposta de distribuição usando paradigma conservativo

Esta versão é baseada no paradigma conservativo. O ambiente delega a um dos processos que irão auxiliar na simulação a tarefa de resolver os *deadlocks*. A partir deste momento, este processo será chamado simplesmente de árbitro.

O árbitro recebe a topologia da rede a ser simulada, e a partir desta deve montar uma estrutura que terá informações relativas às agências que poderão participar de um possível *deadlock*. Tal situação pode ser exemplificada através da figura 7.2.

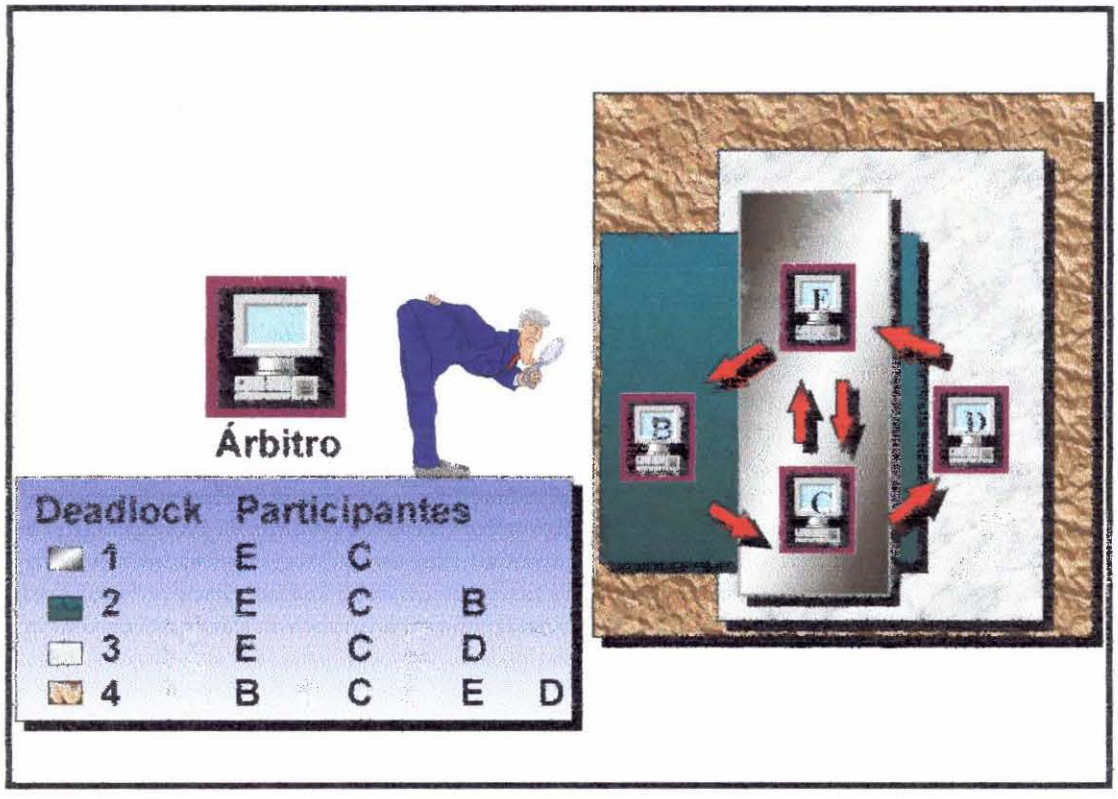


FIGURA 7.2 - Situações de possíveis deadlocks

Na figura 7.2 há quatro possibilidades de deadlock. O árbitro separou os possíveis grupos de *deadlock* a fim de facilitar a localização, quando necessária, e identificar qual o *deadlock* que está sendo desfeito.

Somente após o árbitro ter resolvido a lista de possíveis deadlocks e ter enviado uma mensagem ao ambiente informando que está pronto é que este irá habilitar ao usuário o controle do processo de simulação.

O ambiente deverá informar a cada processo o endereço do árbitro, pois é a este que os processos deverão solicitar ajuda quando tiverem uma parada.

Conforme o paradigma conservativo, um processo não poderá prosseguir com a simulação caso o canal de comunicação de entrada que possui o menor *timestamp* até então processado esteja vazio. Esta situação poderá fazer com que ocorra um *deadlock*.

Caso ocorra tal situação, o processo deverá enviar uma mensagem para o árbitro informando que está parado no tempo ***T*** e qual a agência que está provocando a parada.

Para que seja possível a compreensão da técnica utilizada será criada uma situação baseada na figura 7.2.

Supõe-se que o processo **C** esteja parado devido a sua fila de menor timestamp estar vazia, e que esta fila seja a relativa ao processo **E**. Assim que seja detectada a parada, **C** envia uma mensagem ao árbitro para que este resolva o problema. O árbitro ao receber a mensagem de **C** verifica quais os grupos de *deadlock* em que **C** pode estar envolvido.

O processo **C**, após ter enviado a mensagem ao árbitro, coloca-se em um estado de tranqüilidade, pois sabe que irá receber uma mensagem seja por parte do processo **E** seja por parte do árbitro.

O árbitro irá enviar uma mensagem ao processo **E** perguntando a situação deste.

O processo E poderá estar em uma das seguintes situações:

- está parado, aguardando a chegada de uma mensagem;

- já transmitiu uma mensagem para o processo **C**, a qual irá quebrar o *deadlock*, logo o árbitro interrompe o processo de resolução de *deadlock* solicitado por **C**; ou

- está processando e irá enviar logo uma mensagem, o que possibilita ao árbitro também interromper o processo de resolução de *deadlock* solicitado por **C**, visto que uma mensagem será enviada.

Caso **E** também esteja parado, este informará ao árbitro esta situação, o qual fará novamente o processo de quebra de *deadlock*.

O árbitro à medida que vai avançando acaba por descobrir qual o *deadlock* que está ativo. No caso da figura 7.2 temos quatro possíveis situações de *deadlock*.

O processo escravo, ao detectar que a sua fila de menor *timestamp* está vazia, não deve logo começar um processo de quebra de *deadlock*, pois poderá simplesmente estar acontecendo uma demora na chegada da mensagem.

O escravo deverá ter, para cada canal de entrada, um tempo de espera mínimo. Esta fatia de tempo de espera de uma mensagem pelo canal irá variar conforme o desempenho da rede e a simulação. O tempo de espera deve ser diferenciado por canal a fim de se obter um desempenho maior no sistema, pois um tempo idêntico poderá causar em certos processos uma espera desnecessária.

Inicialmente todos os processos e todos os canais possuem o mesmo tempo de espera α.

O árbitro ao receber uma solicitação de quebra de *deadlock* tenta resolvê-lo. Uma de duas situações poderá ser detectada:

1. o processo pode estar realmente em *deadlock*, ou

2. a solicitação é um falso *deadlock*.

O árbitro possui junto à estrutura de agências o tempo de espera que cada uma está atualmente utilizando. Com base na situação resultante da solicitação feita pelo processo (este está ou não em *deadlock*), o árbitro irá recalcular o tempo de espera para o canal que provocou a solicitação de quebra de *deadlock*.

Caso o árbitro encontre uma situação real wazzu de *deadlock* ele irá pegar o maior *timestamp* entre os processos envolvidos e o transmitirá para os mesmos a fim de quebrar o *deadlock*. Estes processos então incrementam seus tempos locais de simulação para este tempo comunicado pelo árbitro.

O árbitro poderá receber mais de uma solicitação de quebra de deadlock. Ele irá verificar se o processo que está solicitando não faz parte de um possível deadlock que está sendo processado. Caso este venha a fazer parte, o árbitro guarda a solicitação e aguarda o término do processo de quebra do deadlock. Caso o processo realmente esteja dentro do deadlock a mensagem é descartada, caso contrário um novo processo é disparado.

A comunicação entre o árbitro e os processos é feita através de mensagens especialmente criadas para esta finalidade.

Durante o processo de solicitação de quebra de deadlock poderemos ter quatro tipos de mensagens, as quais estão divididas em dois grupos:

1. Grupo Agência solicitante / Árbitro

• mensagem agência solicitante para o Árbitro: A agência que está com problemas para continuar a simulação, devido ao fato de que seu canal de entrada de menor *timestamp* está vazio, envia uma solicitação de detecção/resolução de *deadlock* ao árbitro.

• mensagem Árbitro para agência solicitante: após o árbitro ter resolvido o problema relativo ao possível *deadlock*, este poderá ou não enviar uma mensagem para que a agência que provocou a solicitação de resolução de *deadlock* possa continuar a simulação e/ou atualizar o tempo de espera do canal desta agência, conforme mostra a figura 7.3.

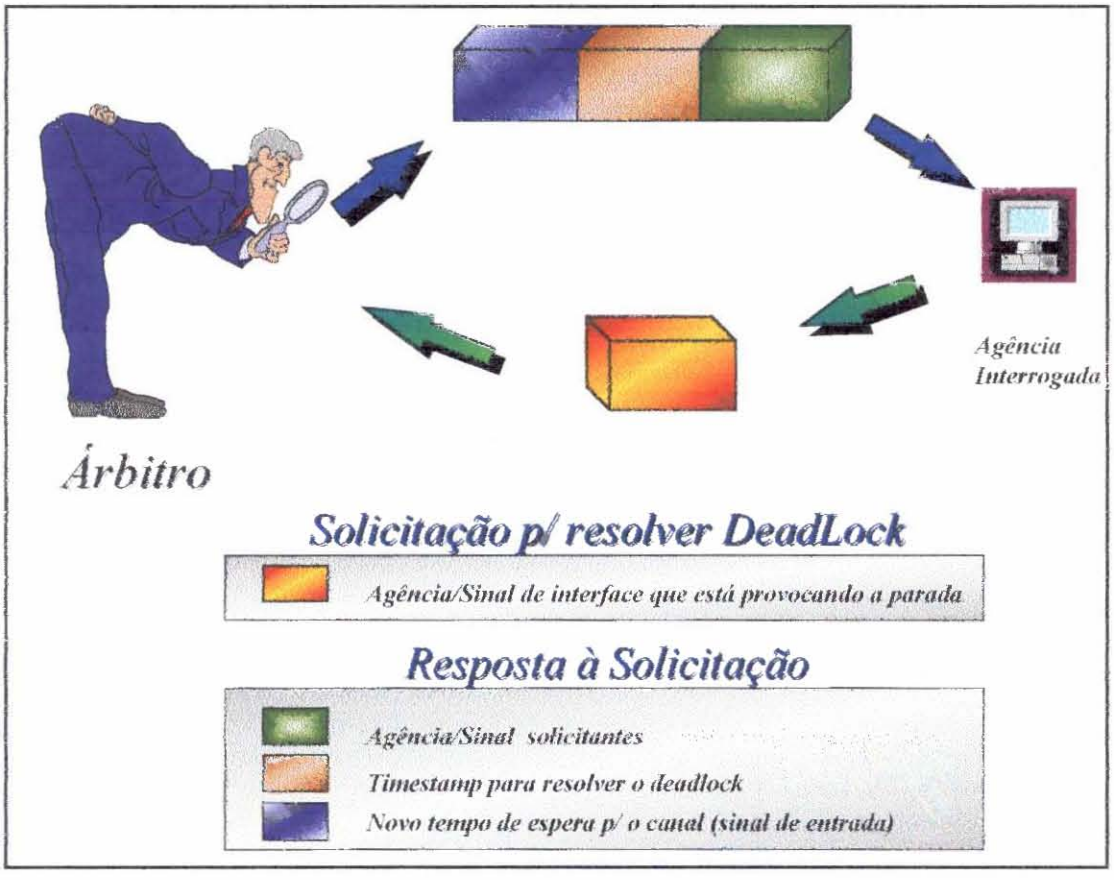


FIGURA 7.3 - Mensagens agência solicitante / árbitro e árbitro / agência solicitante

2. Grupo Árbitro / Agência interrogada

• mensagem Árbitro para agência interrogada: o árbitro ao receber uma mensagem de solicitação de detecção/resolução de *deadlock*, irá iniciar o processo para resolvê-lo. De posse de uma lista que contém os possíveis *deadlocks* (figura 7.4) em que a agência solicitante poderá participar, este enviará uma mensagem para a agência antecessora à agência solicitante conforme o ciclo que está sendo avaliado no momento.

•mensagem agência interrogada para Árbitro: a agência que receberá a mensagem do árbitro solicitando a sua situação atual deverá retornar uma mensagem com a informação solicitada. A figura 7.5 mostra o processo e as estruturas das mensagens utilizadas.

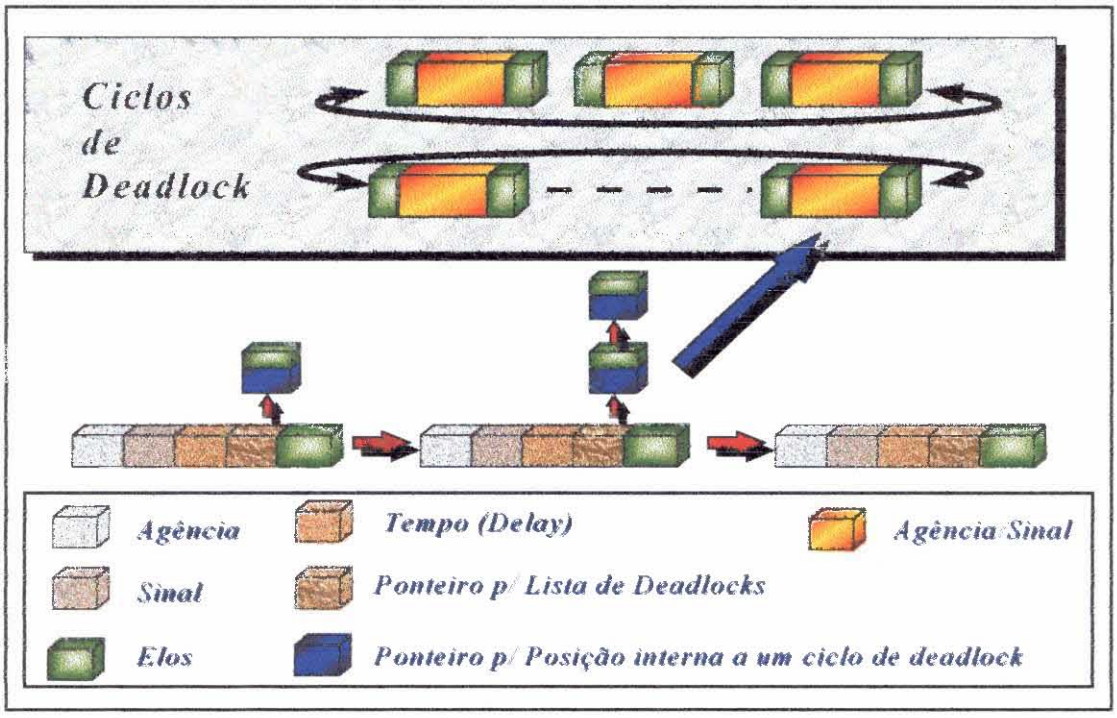


FIGURA 7.4 - Estruturas de dados para comunicação árbrito/agência interrogada

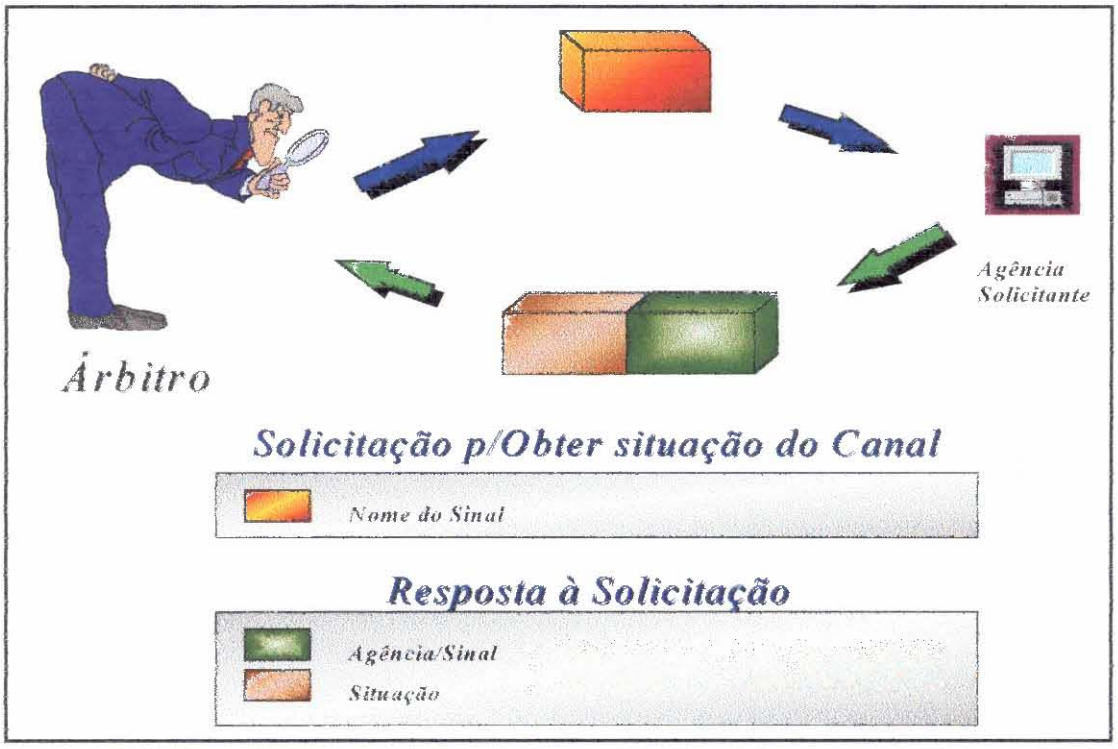


FIGURA 7.5 - Mensagens árbitro/agência interrogada e agência interrogada/árbitro

7.2 Proposta de distribuição usando paradigma otimista

O paradigma otimista prevê a independência total dos processos quanto ao avanço do tempo de simulação, ou seja, cada processo irá avançar a simulação por conta própria.

A instalação dos processos que irão auxiliar na simulação é feita da mesma forma que na versão conservativa, ou seja, instala-se primeiro o ambiente e logo após os processos escravos que irão compor o ambiente distribuído.

Os escravos contêm uma estrutura de dados para guardar as informações relativas à evolução da simulação dentro da agência. Conforme o nível em que esta se encontra descrita, os dados armazenados serão diferentes, porém para todos níveis de descrição será guardado o nome e o valor do sinal, bem como o valores das variáveis da agência.

A técnica aqui desenvolvida consiste em guardar os dados não para retrocesso, como seria o caso de um *rollback*, mas sim para registrar o avanço da simulação da agência. Caso ocorra uma discrepância no tempo de chegada de uma das mensagens para os sinais de interface da agência, bastará o escravo começar com a sua simulação a partir do tempo em que a nova mensagem ocorreu, ignorando desta forma todas as outras mensagens enviadas para as agências vizinhas, as quais estão conectadas a um ou mais canais de saída.

Dois novos tipos de mensagens foram criados nesta versão, para que seja possível manter a consistência da simulação e também para manter a quantidade dos dados de avanço da simulação dentro da agência dentro de um limite aceitável, ou seja, para que a lista de estados da agência não se torne demasiadamente grande e para evitar que o cálculo do tempo virtual global seja um problema para a performance da simulação.

Os tipos de mensagens que podemos ter nesta proposta de simulação que utiliza o paradigma otimista são os seguintes:

- mensagem segura - indica que é uma mensagem gerada por uma agência que tem a situação de seus canais de entrada em um estado confiável,

ou seja, não é possível que chegue uma mensagem com *timestamp* inferior ao da mensagem segura na agência que a produziu, e

• mensagem insegura - informa a agência receptora que poderá futuramente chegar uma mensagem com *timestamp* inferior.

A figura 7.6 mostra o formato das mensagens que irão trafegar pelo sistema.

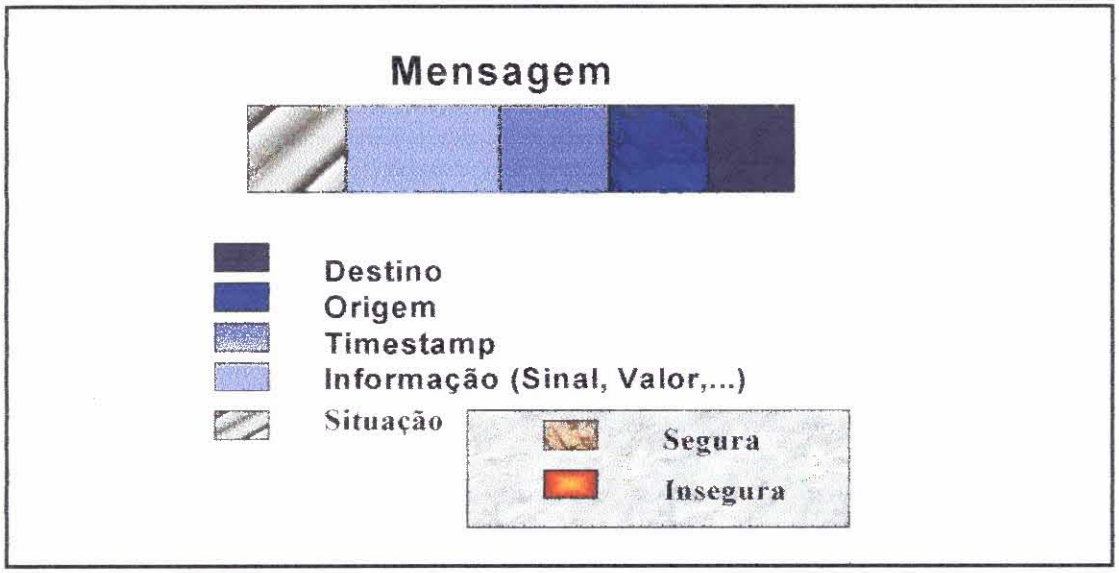


FIGURA 7.6 - Formato da mensagem para versão otimista

As mensagens inseguras poderão ser geradas de duas formas:

1.pela ausência de mensagens no canal de menor timestamp, com a chegada de uma mensagem segura por qualquer outro canal de entrada.

2. pela chegada de uma mensagem insegura - a partir do ponto em que a agência vá processar uma mensagem recebida e esta é do tipo insegura, a agência irá propagar junto com as próximas mensagens a situação de insegurança, e

O salvamento dos estados da agência está baseado na chegada das mensagens seguras e mensagens inseguras.

A agência ao processar uma mensagem verifica a situação dos canais de comunicação. A situação de cada canal é controlada através da última mensagem processada para este.

Um canal poderá estar numa situação de segurança se a última mensagem processada para este foi uma mensagem segura, caso contrário este estará em uma situação de insegurança.

Caso todos os canais estejam na situação de segurança, a agência descarta todos os dados guardados até o menor timestamp entre seus canais de entrada. A estrutura a ser usada para guardar os dados de cada agência pode ser vista na figura 7.7.

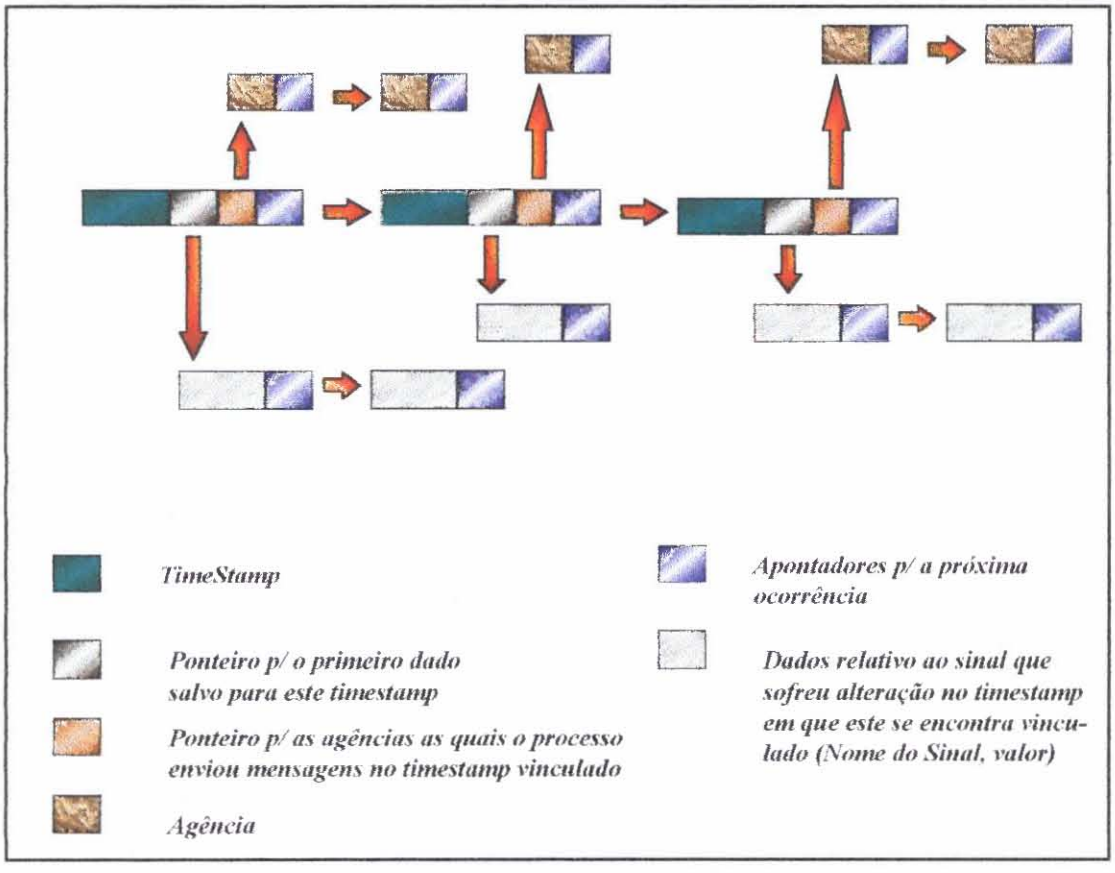


FIGURA 7.7 - Estrutura para salvar estados da agência

A chegada de uma mensagem de correção, ou seja, uma mensagem com timestamp inferior ao processado até o momento pela agência, provocará a liberação dos dados armazenados na estrutura de estados da agência, sendo que todos os estados

com timestamp maior do que o timestamp da mensagem de correção deverão ser eliminados. Os dados armazenados no tempo em que ocorreu a correção deverão ser carregados para a agência, bem como deverá ser providenciada a transmissão de uma mensagem de correção para todas as agências que obtiveram mensagens enviadas por esta agência. Para que seja possível transmitir as mensagens de correção o processo deve, cada vez que enviar uma mensagem para uma agência em um determinado tempo de simulação, guardar o nome desta na sua estrutura de dados de segurança. Caso seja enviada mais de uma mensagem para uma mesma agência no mesmo tempo de simulação, basta armazenar uma única vez o nome da agência a quem se destinam as mensagens.

O tamanho da estrutura será basicamente controlado pela chegada das mensagens seguras e pelas de correção, porém pode-se ter uma situação onde a estrutura não suporte mais dados. Caso isto venha a ocorrer, a agência provocará o envio de uma mensagem ao ambiente o qual acionará a rotina que fará o cálculo do tempo virtual global. Após o ambiente ter calculado este tempo, este irá enviar a todos os processos uma mensagem informando o novo tempo virtual global, o que garantirá um ponto de segurança aos processos que estejam com insegurança quanto à evolução da sua simulação. Desta forma as agências poderão eliminar os dados armazenados que possuam timestamp inferior ao tempo virtual global.

7.3 Processo de Término da Simulação

A simulação poderá ser finalizada por uma das seguintes situações:

1. um processo ter alcançado o tempo final de simulação;

2. uma condição de *breakpoint* ter ocorrido com sucesso, ou

3. ter ocorrido um erro fatal.

Quando qualquer uma destas situações acontecer, o escravo deverá enviar uma mensagem para o ambiente notificando-o, como mostra a figura 7.8. Mas não basta comunicar ao ambiente. O processo deverá colocar-se em estado de bloqueio, ou seja, este não poderá mais receber mensagens de outros processos, até que o

ambiente envie uma nova mensagem solicitando a continuação da simulação ou a destruição do processo escravo. A agência que está finalizando suas tarefas, seja qual for o motivo, deverá iniciar um processo de notificação desta situação a todas as agências que este possui ligação.

Conforme ilustrado na figura 7.8, uma agência controlada pelo processo C está finalizando suas tarefas devido a uma das três possíveis situações de término de simulação, descritas acima.

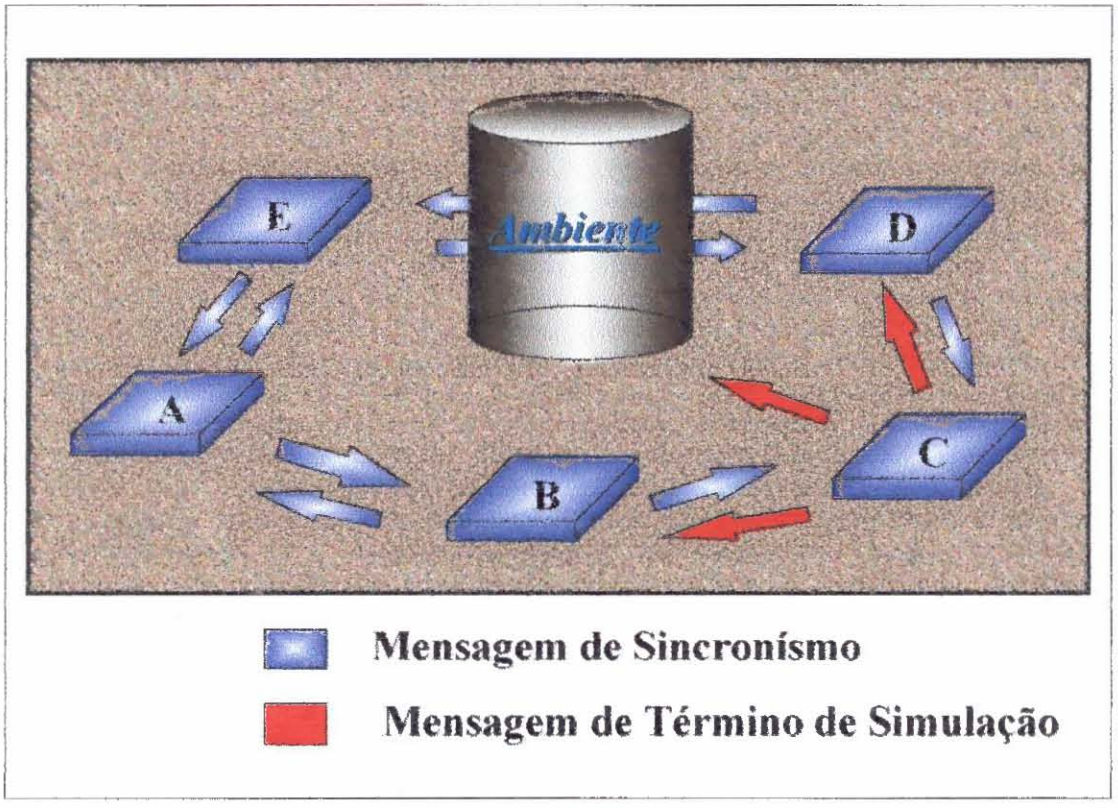


FIGURA 7.8 - Envio de mensagem de Término de simulação

Supondo que a simulação esteja sendo feita sobre uma rede de agências conforme a figura 7.9, e que a agência C1 que está executando no processo C tenha encontrado um situação de término de simulação. Ao detectar o término da simulação, a agência C1 deverá enviar a todas as agências que possuem ligações com suas entradas uma mensagem informando a estas a sua condição de parada, bem como o timestamp local a esta. O processo B, ao receber a mensagem de rompimento de comunicação com a agência C1 do processo C, deverá colocar os canais de comunicação com esta agência em uma estrutura que será identificada como guardiã. Toda vez que há uma

transmissão de mensagens o processo consulta a estrutura guardiã. Caso a agência a quem esta mensagem se destina esteja inclusa na estrutura, o processo deverá enviar uma mensagem para o ambiente, a fim de notificar a este que houve uma tentativa de enviar uma mensagem para a agência que está temporariamente desativada. Torna-se necessário informar ao ambiente as mensagens produzidas para a agência bloqueada após o seu timestamp de bloqueio, para que o ambiente possa posteriormente continuar a simulação. Outra situação que poderá acontecer é a chegada de uma mensagem atrasada para a agência que está em situação de término de simulação, logo o ambiente deverá tratá-la de forma adequada.

Caso o usuário resolva prosseguir com a simulação o ambiente deverá transmitir todas as mensagens recebidas que estavam destinadas à agência momentaneamente parada.

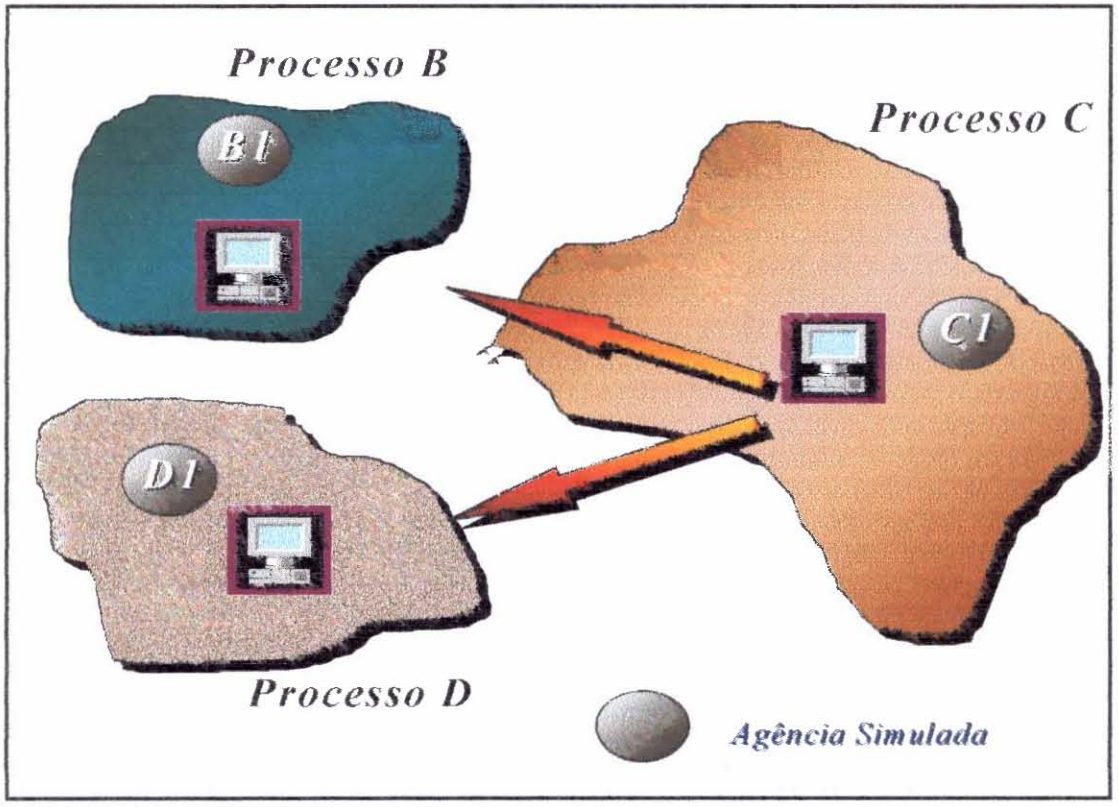


FIGURA 7.9 - Agência iniciando processo de término de simulação

Caso o usuário resolva desativar o ambiente a fim de interromper por completo a simulação, será necessário enviar uma mensagem para cada processo ativo, solicitando a estes que finalizem por completo suas atividades. A comunicação entre o ambiente e os escravos é mostrada na figura 7.10.

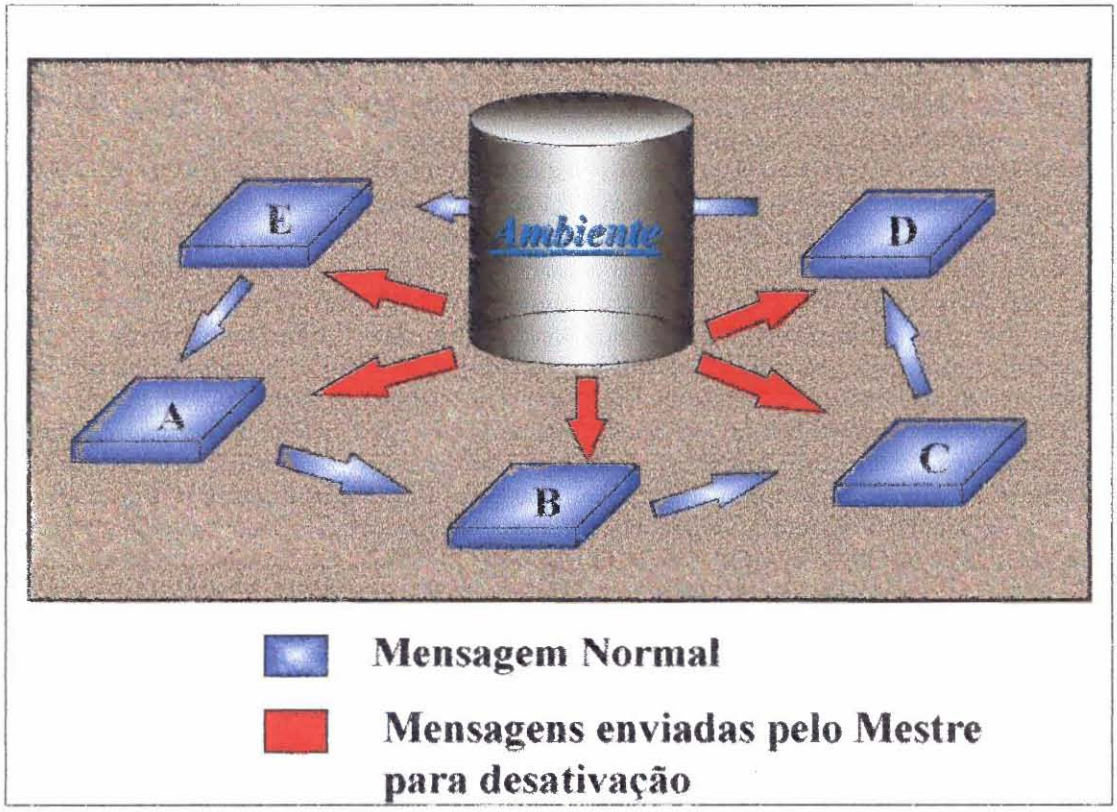


FIGURA 7.10 - Envio de Mensagens do mestre para desativação do ambiente

8 Avaliações da Simulação Seqüencial e da sSimulação Distribuída

Durante o processo de teste dos simuladores foram utilizadas agências com a mesma função (um flip-flop D), bem como o mesmo atraso de propagação, porém com níveis de abstração diferentes, sempre sendo uma descrita em NILO e outra em LAÇO.

Os testes dos simuladores seqüenciais foram feitos em um equipamento PC, com processador Pentium 166, com 32 Mb de RAM, sendo estes executados em módulo de emulação DOS, ou seja, seus códigos são de 16 bits.

8.1 Desempenho da versão seqüencial

Dois tipos de testes foram aplicados na versão seqüencial em relação ao desempenho individual da simulação de uma agência (um flip-flop D), a fim de ser possível efetuar a coleta dos dados:

1.quanto ao número de elementos dentro de uma agência

- o simulador LAÇO mostrou-se mais eficiente que o simulador NILO, quando a complexidade da agência é maior. Quando a agência possui uma complexidade muito pequena o simulador NILO tem um desempenho melhor que o simulador LAÇO.

2. quanto ao número de interações com a agência

- quando disparadas várias solicitações de simulação para um Flip-Flop do tipo D, obtivemos os resultados mostrados nas tabelas 8.1 e 8.2.

TABELA 8.1 - Dados relativos à simulação de um flip-flop D descrito em NILO

Número de interações consecutivas	Número de portas simuladas	Tempo de resposta total (Milissegundos)
300	1.500	65
3.000	15.000	670
5.000	25.000	1002
30.000	150.000	6042

TABELA 8.2 - Dados relativos à simulação de um flip-flop D descrito em LAÇO

Número de interações consecutivas	Tempo de resposta total (Milissegundos)
300	53
3.000	540
5.000	953
30.000	5476

8.2 Desempenho das versões distribuídas

Os testes das versões distribuídas foram realizados em uma rede de computadores com as seguintes características:

- um computador PC Pentium 166 Mhz, 32 MB de RAM;
- três computadores PC, processador 486 DX2- 66 Mhz, 8 MB de RAM;
- um computador PC, processador 486 DX - 50 Mhz, 4 MB de RAM;

- um computador PC, processador 486 SX - 33 Mhz, 8 MB de RAM e
- dois Computadores PC, processador 386 DX, 33 Mhz , 4 MB de RAM

No computador PC que utiliza o processador Pentium 166 foi instalado o ambiente, e nos demais os escravos, conforme é demonstrado na figura 8.1. O meio físico de comunicação utilizado entre os computadores foi cabo coaxial de 50 Ohms e placas NE2000 padrão compatível com NetWare Novell. O tipo de topologia utilizada foi a de barramento.

No equipamento PC que utiliza o processador Pentium 166 foi utilizado o Windows 95 e nos demais, Windows 3.11.

O ambiente e o árbitro foram gerados em código de 32 bits, enquanto os escravos têm código de 16 bits. Caso haja mais um computador que suporte um sistema de 32 bits basta instalar a versão do escravo de 32 bits.

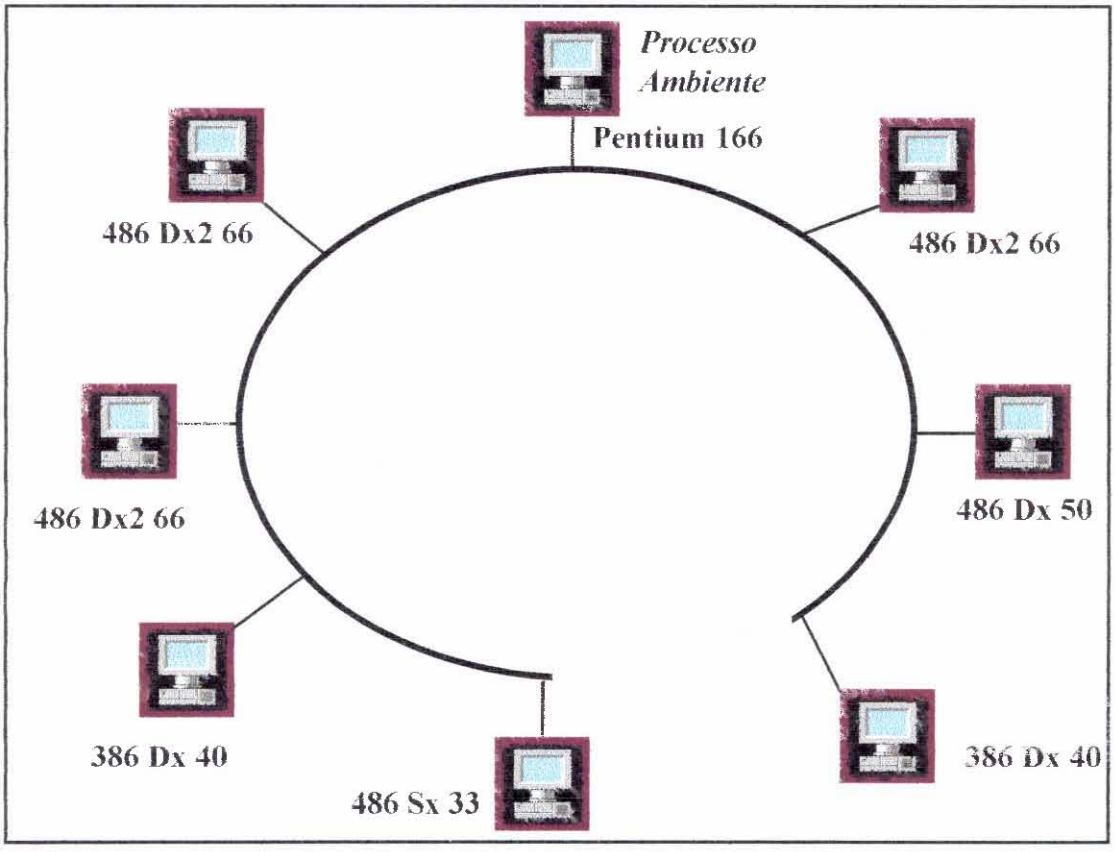


FIGURA 8.1 - Rede de teste

Na versão conservativa, juntamente com o ambiente foi instalado o árbitro no computador PC com processador Pentium de 166 Mhz, para que a resposta fosse mais rápida quando uma possível situação de deadlock fosse detectada.

Na versão otimista foi inserido juntamente com o ambiente o módulo de cálculo do tempo virtual global.

Todos os testes foram feitos sem o compartilhamento da rede, ou seja, os escravos e o ambiente estavam sozinhos na rede.

Como o desempenho dos escravos não afetará o desempenho dos algoritmos de controle do avanço do tempo, o código dos escravos não foi inserido no corpo do escravo geral, visto que os simuladores escravos, na forma seqüencial, algumas vezes ainda apresentam problemas de funcionamento. Como o objetivo deste trabalho não era a de corrigir o código dos escravos, optou-se pela criação de um escravo de teste, o qual simplesmente espera o tempo de atraso.

Quando um escravo era ativado, além do endereço do ambiente, lhe era passada a configuração do equipamento em que este estava sendo instalado. Estes dados foram utilizados para que a distribuição fosse feita com um pouco mais de homogeneidade, visto os diversos tipos de processadores que havia na rede.

Vários testes foram realizados tomando-se como base os seguintes fatores:

- número de agências;
- interrelacionamento entre agências, e
- atividade interna das agências.

Os valores apresentados nas tabelas que demonstram os resultados obtidos durante as simulações de teste estão representados em uma forma simbólica de unidade de tempo.

A figura 8.2 ilustra a rede de agências primitivas utilizada para o primeiro teste. Ao ser efetuada a simulação constatou-se que a versão seqüencial teve um melhor desempenho em relação às versões distribuídas, visto o tamanho da rede de agências e a interconectividade, conforme se verifica nas tabelas 8.1 à 8.3.

Em relação ao desempenho entre as versões distribuídas observou-se que a versão otimista teve um desempenho extremamente superior ao da conservativa, pois a conservativa por várias vezes enviou mensagens ao árbitro, referenciado na seção 7.1, a fim de desfazer um possível deadlock.

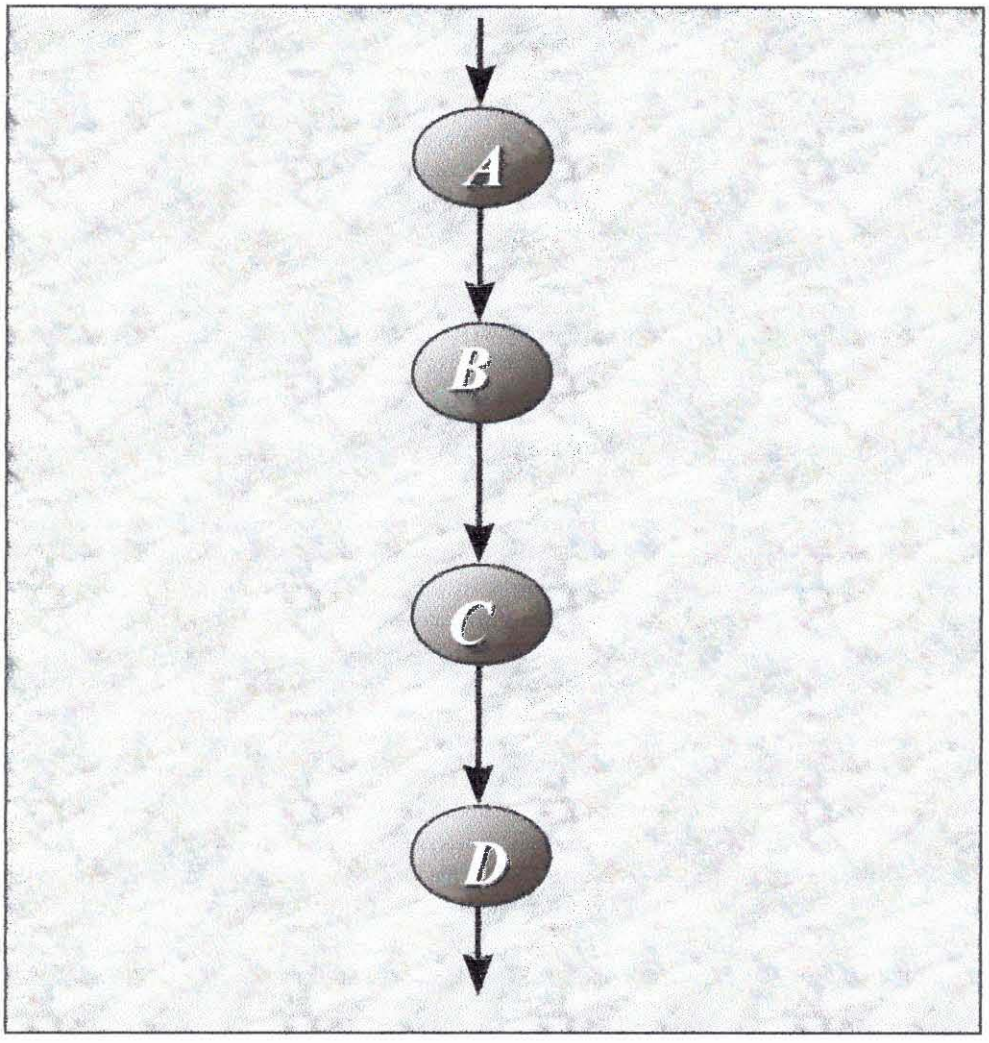


FIGURA 8.2 - Rede de agências para teste de simulação 1

TABELA 8.1 - Dados relativos a simulação de teste 1 - simulação seqüencial

Experimentos	Tempo Simulação	Tempo de Sincronização
1	135	-
2	467	-
3	780	-
4	915	-
5	1070	-
6	1110	-
7	1250	-

TABELA 8.2 - Dados relativos a simulação de teste 1 - paradigma otimista

Experimentos	Tempo Simulação	Tempo de Sincronização
1	173	47
2	585	49
3	863	57
4	1005	87
5	1197	95
6	1245	103
7	1355	147

TABELA 8.3 - Dados relativos a simulação de teste 1 - paradigma conservativo

Experimentos	Tempo Simulação	Tempo de Sincronização
1	215	89
2	680	144
3	973	167
4	1215	297
5	1415	313
6	1460	318
7	1541	333

Na simulação da rede de agências representada na figura 8.3 a versão seqüencial mostrou-se novamente superior em relação às versões distribuídas. A versão otimista teve um desempenho superior ao da conservativa, conforme as tabelas 8.4 à 8.6. Tal desempenho foi devido ao fato de que a chegada de mensagens atrasadas na agência D não afeta outras agências, logo o processo de envio de mensagens de

correção não teve efeito no desempenho da simulação, porém a versão conservativa constantemente solicitou a intervenção do árbitro.

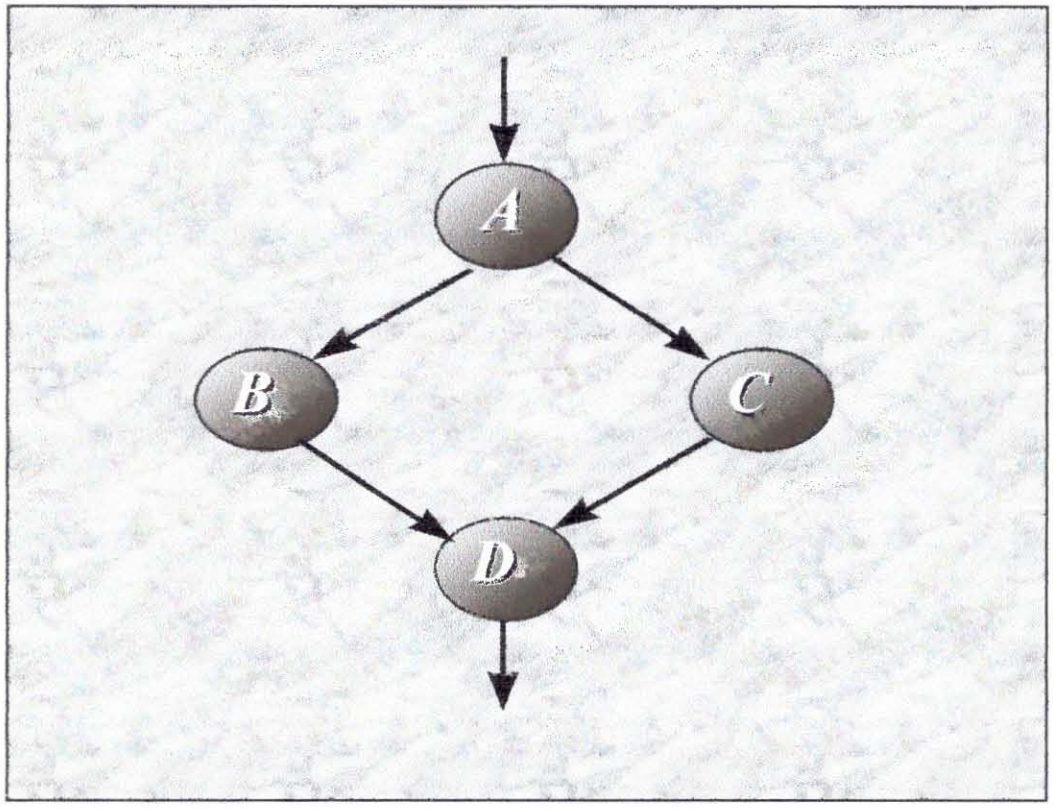


FIGURA 8.3 - Rede de agências para teste de simulação 2

TABELA 8.4 - Dados relativos a simulação de teste 2 - simulação seqüencial

Experimentos	Tempo Simulação	Tempo de Sincronização
1	142	-
2	475	-
3	792	-
4	923	-
5	1094	-
6	1127	-
7	1268	-

TABELA 8.5 - Dados relativos a simulação de teste 2 - paradigma otimista

Experimentos	Tempo Simulação	Tempo de Sincronização
1	167	42
2	582	46
3	958	52
4	996	78
5	1188	86
6	1243	101
7	1347	139

TABELA 8.6 - Dados relativos a simulação de teste 2 - paradigma conservativo

Experimentos	Tempo Simulação	Tempo de Sincronização
1	215	89
2	680	144
3	973	167
4	1215	297
5	1415	313
6	1460	318
7	1541	333

A figura 8.4 demonstra a mesma rede de agências que a figura 8.3, porém a interconectividade entre estas aumentou.

Nesta simulação a versão otimista, também mostrou ser a melhor, conforme as tabelas 8.7 à 8.9, apesar de terem ocorrido várias transmissões de mensagens de correção. O tempo geral de simulação nas diversas situações testadas com esta rede de agências, na versão otimista, foi menor que nas versões seqüencial e conservativa. Fazendo-se a comparação de desempenho entre as duas últimas constatou-se que a versão seqüencial foi melhor que a conservativa, pois apesar da versão conservativa ser distribuída esta gastou muito tempo esperando a chegada de mensagens e resolvendo os falsos deadlocks.

Após um certo número de interações que o sistema sofreu, na versão conservativa ajustava-se cada vez mais os tempos de espera dos canais de entrada de cada agência, fazendo com que o tempo de simulação cada vez diminuísse mais.

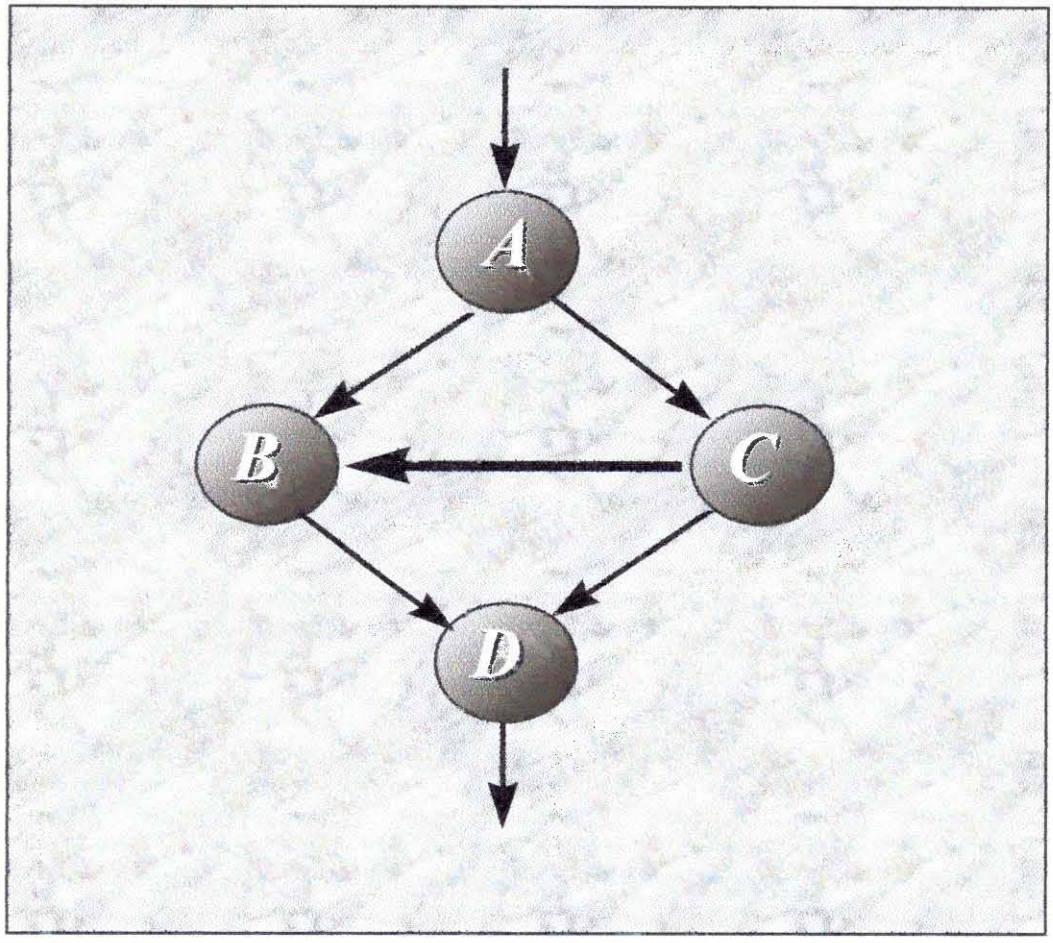


FIGURA 8.4 -Rede de agências para teste de simulação 3

TABELA 8.7 - Dados relativos a simulação de teste 3 - simulação seqüencial

Experimentos	Tempo Simulação	Tempo de Sincronização
1	189	-
2	617	-
3	915	-
4	1038	-
5	1317	-
6	1395	-
7	1402	-

TABELA 8.8 - Dados relativos a simulação de teste 3 - paradigma otimista

Experimentos	Tempo Simulação	Tempo de Sincronização
1	175	49
2	597	53
3	875	62
4	1027	93
5	1215	104
6	1263	117
7	1372	155

TABELA 8.9 - Dados relativos a simulação de teste 3 - paradigma conservativo

Experimentos	Tempo Simulação	Tempo de Sincronização
1	223	97
2	695	151
3	988	175
4	1320	386
5	1442	331
6	1489	343
7	1582	365

Ao simular a rede de agências representada pela figura 8.5, a versão seqüencial foi a que apresentou o pior desempenho, conforme as tabelas 8.10 à 8.12.

Dentre as versões distribuídas a otimista obteve o melhor desempenho. Durante a simulação, com a versão conservativa, constatou-se novamente um alto grau de intervenção do árbitro.

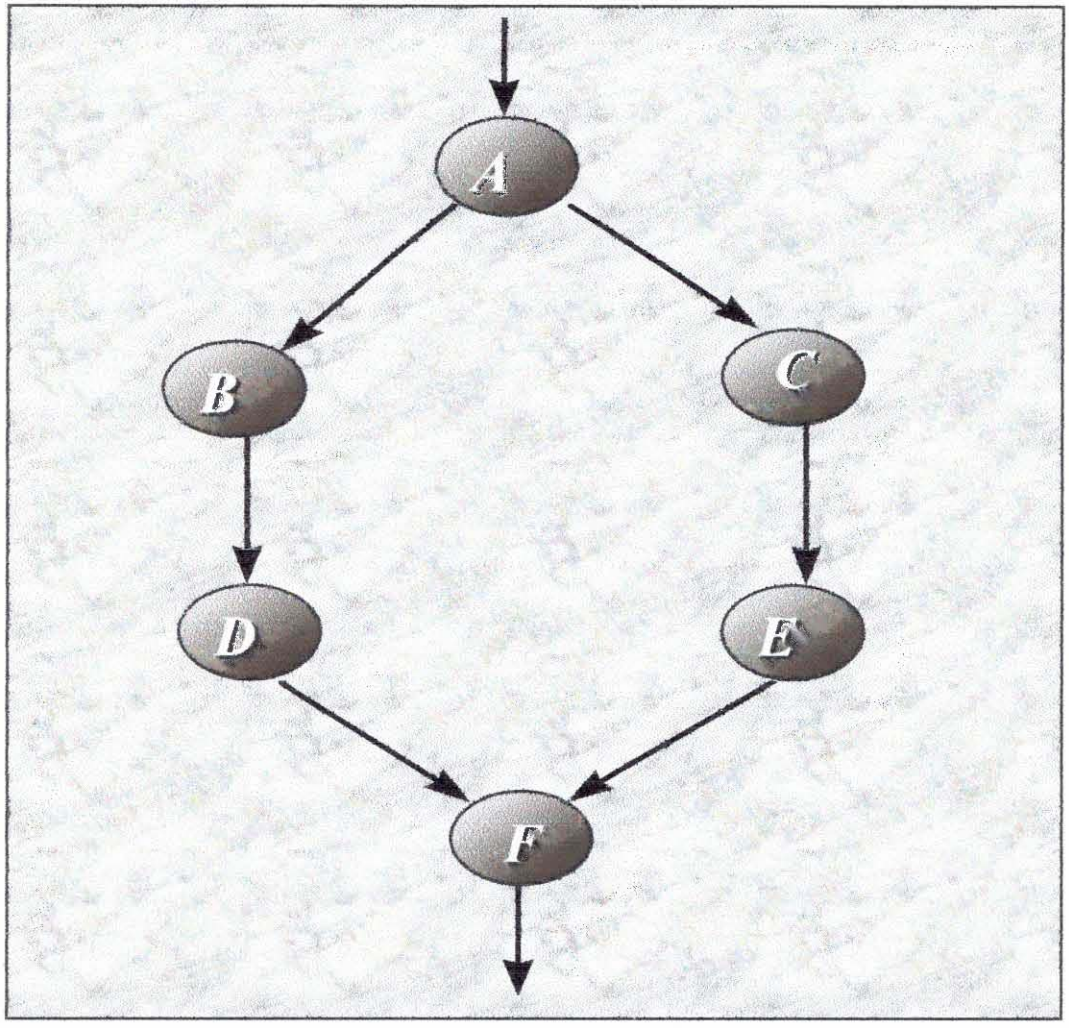


FIGURA 8.5 - Rede de agências para teste de simulação 4

TABELA 8.10 - Dados relativos a simulação de teste 4 - simulação seqüencial

Experimentos	Tempo Simulação	Tempo de Sincronização
1	217	-
2	586	-
3	877	-
4	1097	-
5	1318	-
6	1572	-
7	1721	-

TABELA 8.11 - Dados relativos a simulação de teste 4 - paradigma otimista

Experimentos	Tempo Simulação	Tempo de Sincronização
1	155	52
2	438	58
3	727	65
4	875	103
5	1113	118
6	1315	133
7	1417	103

TABELA 8.12 - Dados relativos a simulação de teste 4 - paradigma conservativo

Experimentos	Tempo Simulação	Tempo de Sincronização
1	187	84
2	485	105
3	783	121
4	919	147
5	1164	169
6	1348	166
7	1492	178

A rede de agências da figura 8.6 foi baseada na figura 8.5, porém foram inseridas duas outras linhas de comunicação entre os processos B/C e D/C, as quais possibilitaram a formação de deadlock.

A versão otimista obteve o melhor desempenho durante as primeiras interações, porém durante a evolução da simulação a conservativa foi ajustando os valores dos tempos de espera dos canais de entrada de cada agência, obtendo a partir de um determinado tempo uma performance melhor do que a versão otimista, como se vê nas tabelas 8.13 à 8.15, onde as simulações com menor tempo de execução possuem o tempo de sincronização maior do que as simulações com maior tempo de execução.

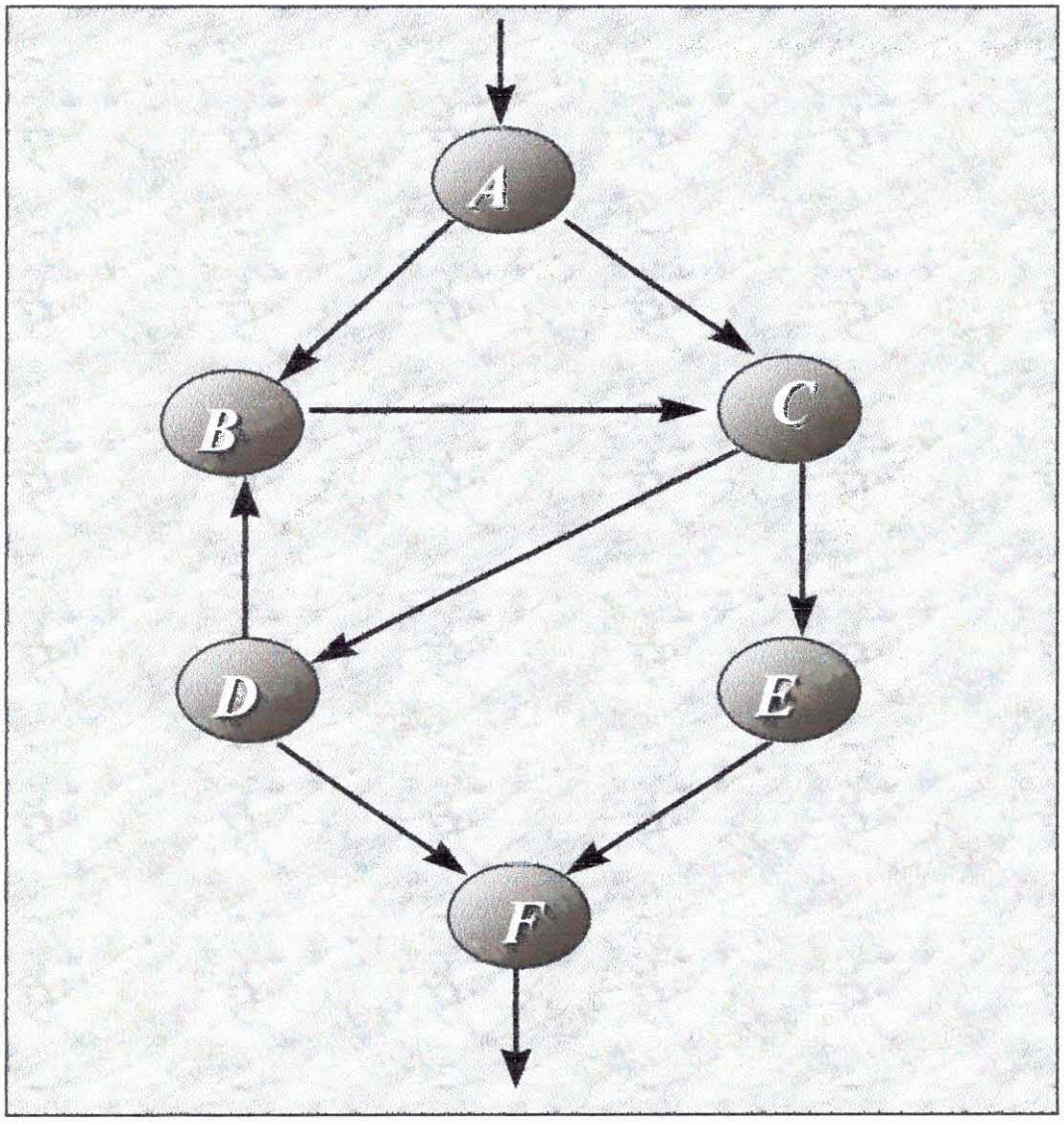


FIGURA 8.6 - Rede de agências para teste de simulação 5

TABELA 8.13 - Dados relativos a simulação de teste 5 - simulação seqüencial

Experimentos	Tempo Simulação	Tempo de Sincronização
1	383	-
2	731	-
3	1002	-
4	1342	-
5	1635	-
6	2042	-
7	2517	-

TABELA 8.14 - Dados relativos a simulação de teste 5 - paradigma otimista

Experimentos	Tempo Simulação	Tempo de Sincronização
1	175	75
2	478	90
3	752	79
4	1071	220
5	1345	296
6	1567	309
7	1721	288

TABELA 8.15 - Dados relativos a simulação de teste 5 - paradigma conservativo

Experimentos	Tempo Simulação	Tempo de Sincronização
1	193	93
2	495	107
3	792	119
4	983	132
5	1182	133
6	1393	135
7	1571	138

A figura 8.7 está representando a rede de agências de maior complexidade utilizada durante os testes. A complexidade está na interconectividade entre as agências.

A simulação otimista mostrou-se ótima durante as primeiras evoluções da simulação, porém com a atualização dos tempos de espera dos canais de entrada a versão conservativa tornou-se extremamente mais rápida, como se vê nas tabelas 8.16 à 8.18.

A versão otimista provocou uma série de retransmissões de mensagens, causando um grande overhead de mensagens na rede.

A versão seqüencial mostrou-se muito lenta, visto que para este teste aumentou-se a atividade interna das agências.

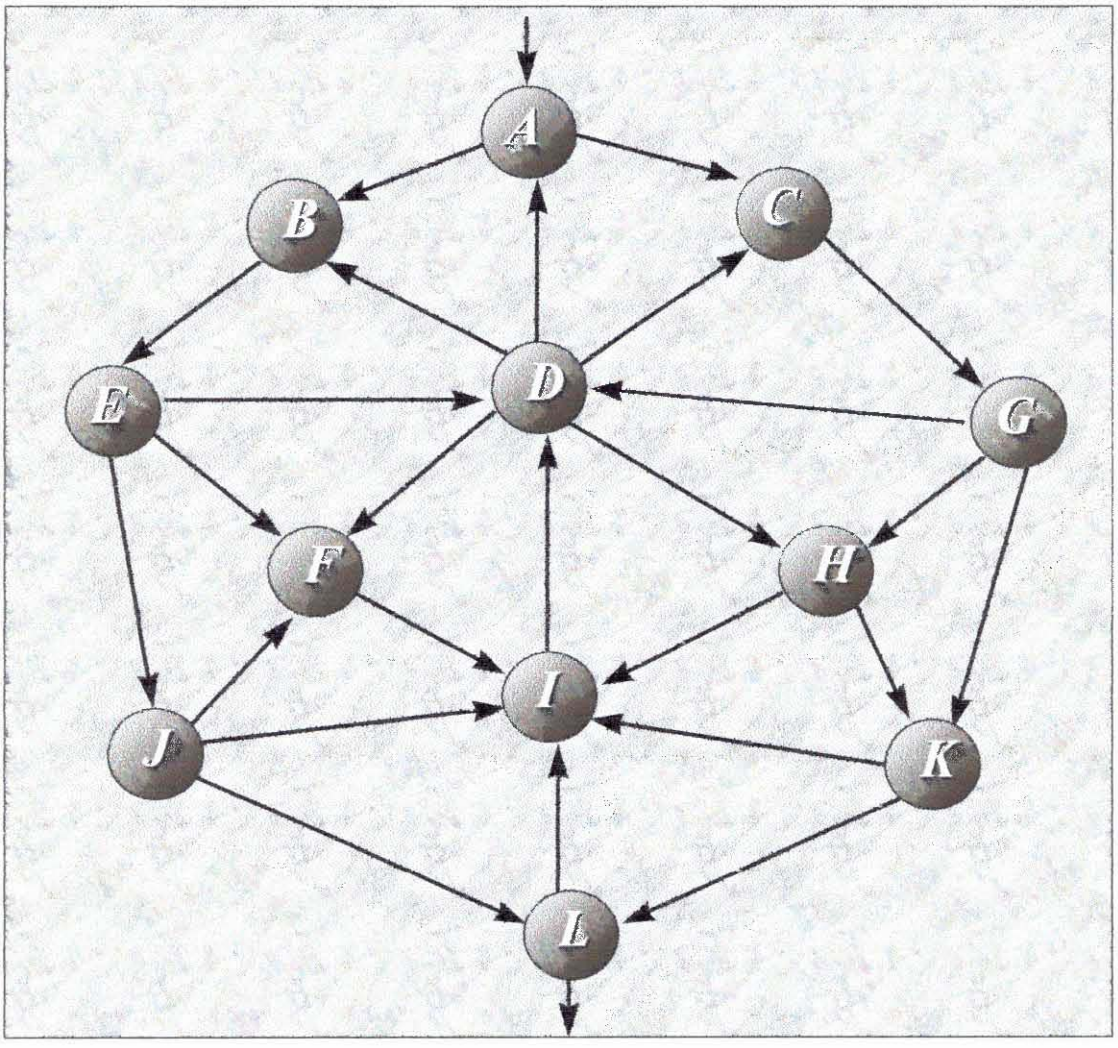


FIGURA 8.7 - Rede de agências para teste de simulação 6

TABELA 8.16 - Dados relativos a simulação de teste 6 - simulação seqüencial

Experimentos	Tempo Simulação	Tempo de Sincronização
1	7417	-
2	27127	-
3	39891	-
4	41025	-
5	49787	-
6	56565	-
7	72876	-

TABELA 8.17 - Dados relativos a simulação de teste 6 - paradigma otimista

Experimentos	Tempo Simulação	Tempo de Sincronização
1	1172	799
2	1763	1248
3	2235	1579
4	7114	5196
5	9813	5774
6	12575	7060
7	18752	12040

TABELA 8.18 - Dados relativos a simulação de teste 6 - paradigma conservativo

Experimentos	Tempo Simulação	Tempo de Sincronização
1	560	187
2	720	205
3	873	217
4	2234	316
5	4389	350
6	5876	361
7	7078	366

Através da avaliação do desempenho das três versões podemos chegar a algumas conclusões:

1. quando a rede de agências é pequena e há uma certa seqüência entre as execuções destas, o simulador seqüencial apresenta uma performance melhor do que os simuladores distribuídos;

2. quando há pouca interação entre as agências e o número de agências é elevado, o simulador utilizando o paradigma otimista mostra-se como a melhor alternativa; e

3. em uma rede de agências onde a interação é muito grande e principalmente quando há muitos grafos cíclicos de processos, o simulador desenvolvido utilizando-se do paradigma conservativo obtém melhor desempenho, principalmente quando o tempo de simulação aumenta, visto que este ajusta cada vez mais os tempos dos canais de comunicação entre os processos. Já o simulador

utilizando-se do paradigma otimista provoca a transmissão de muitas mensagens de correção, principalmente devido aos grafos cíclicos entre os processos.

9 Conclusões e trabalhos futuros

A possibilidade de se fazer uso do tempo de processamento de "estações de trabalho" interligadas por uma "rede local" que estejam sub-utilizadas torna atrativo o estudo de técnicas que possibilitem tirar proveito de tal situação. É baseado neste contexto que os sistemas distribuídos estão cada vez mais populares.

A simulação multinível distribuída, a qual combina as facilidades da simulação multinível com os benefícios fornecidos pelos sistemas distribuídos, busca reduzir o tempo gasto a fim de se obter resultados da simulação de sistemas discretos. Porém a utilização de uma plataforma distribuída traz à tona um problema de difícil resolução, o qual é o controle do tempo global de simulação, e que deve ser controlado de forma a garantir a coerência no avanço da simulação do sistema.

Neste trabalho utilizou-se a teoria básica dos paradigmas otimista e conservativo de simulação distribuída, a fim de propor soluções para controlar o sincronismo da simulação não somente para o projeto AMPLO, mas para qualquer outro tipo de sistema de simulação discreta.

Durante os testes de simulação feitos para avaliar o desempenho dos simuladores pode-se concluir que:

1. quando a rede de agências é pequena e há uma certa seqüência entre as execuções destas, o simulador seqüencial apresenta uma performance melhor do que os simuladores distribuídos;

2. quando há pouca interação entre as agências e o número de agências é elevado, o simulador utilizando o paradigma otimista mostra-se como a melhor alternativa; e

3. em uma rede de agências onde a interação é muito grande e principalmente quando há muitos grafos cíclicos de processos, o simulador desenvolvido utilizando-se do paradigma conservativo obteve melhor desempenho, principalmente quando o tempo de simulação aumenta, visto que este ajusta cada vez mais os tempos dos canais de comunicação entre os processos. Já o simulador utilizando-se do paradigma otimista mostra-se inadequado pois provoca a transmissão

de muitas mensagens de correção, principalmente devido aos grafos cíclicos entre os processos.

Para ambas versões constatou-se a possibilidade de se introduzir melhorias futuras a fim de aumentar a velocidade da simulação.

Para a versão conservativa verificou-se que muitas vezes uma agência acionava o processo de resolução de *deadlock* para um determinado sinal de entrada, porém este sinal nem mesmo participava de um grupo de *deadlocks*, ou seja este jamais estaria nesta situação. Para resolver este problema podemos, ao fazer a distribuição das agências no início da simulação, enviar para cada canal a informação se o próprio poderá ou não participar de um possível *deadlock*.

Neste trabalho foi feita a aplicação da técnica a que se refere o paradigma otimista, apenas na comunicação entre interfaces das agências, ou seja, a propagação de mensagens seguras ou inseguras só era decidida no nível dos sinais de interface. Porém, a mesma poderia ser aplicada junto ao algoritmo de simulação dos escravos a fim de tornar a transmissão das mensagens dependente da estrutura interna das agências. Para esclarecer um pouco mais sobre esta possibilidade de aplicação desta técnica pode-se avaliar a figura 9.1 que demonstra uma agência tendo-se a visão externa (somente os sinais de interface) e a figura 9.2, a qual demonstra como as mensagens poderiam ser propagadas internamente na agência.

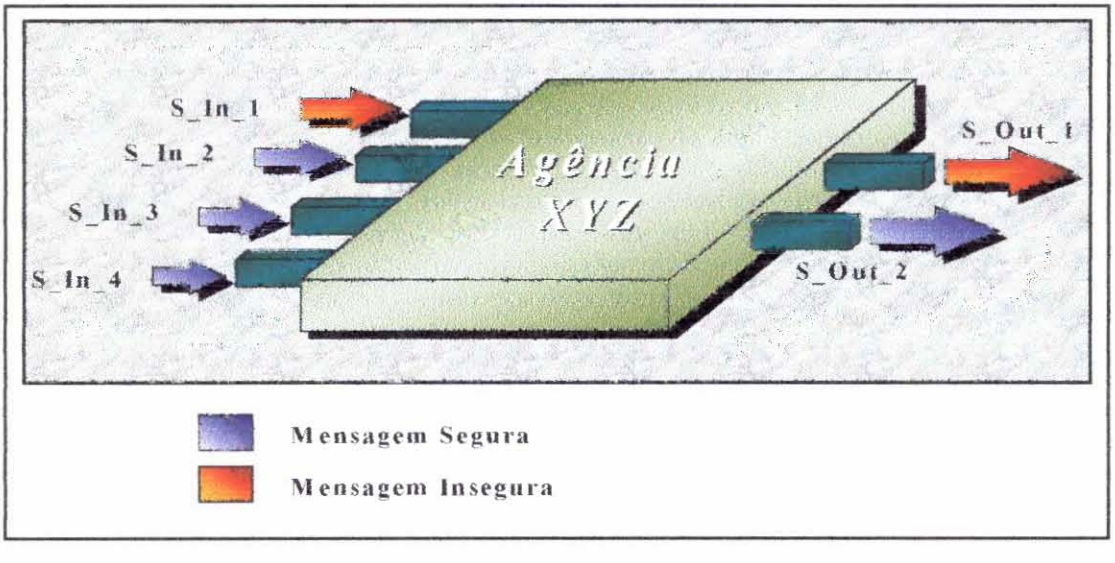


FIGURA 9.1 - Exemplo de agência

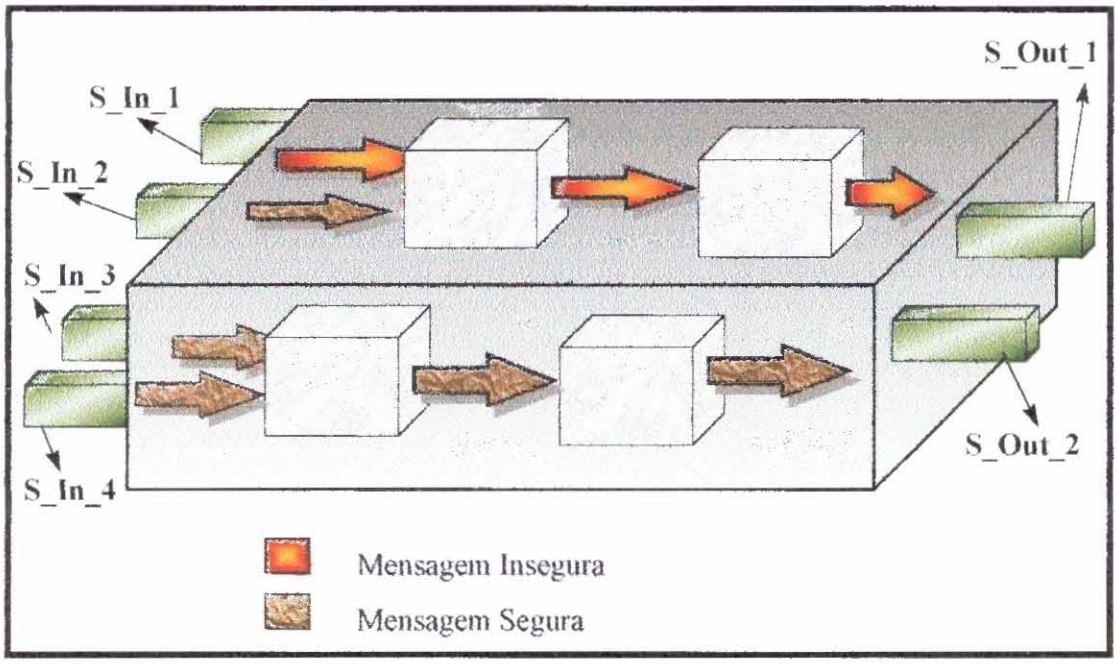


FIGURA 9.2 - Propagação de mensagens dentro da agência XYZ

Conforme a figura 9.2 pode-se perceber que a chegada de uma mensagem insegura nos sinais de entrada S_In_1 e S_In_2 não afetaria totalmente o processo de simulação, pois a agência poderia gerar mensagens inseguras por apenas um canal de saída (S_Out1_1), enquanto que o outro canal (S_Out_2) continuaria a transmitir mensagens seguras aos seus vizinhos. A conseqüência deste aperfeiçoamento fica clara ao se analisar a figura 9.3, onde a agência A estaria gerando duas mensagens, uma insegura, a qual é transmitida à agência B, e outra mensagem segura, que é transmitida a agência C. Deve-se notar que o processo B irá propagar mensagens inseguras e o processo C mensagens seguras.

Na implementação feita neste trabalho, uma agência que receba mensagens inseguras em qualquer canal de entrada irá gerar mensagens inseguras em todos os seus canais de saída.

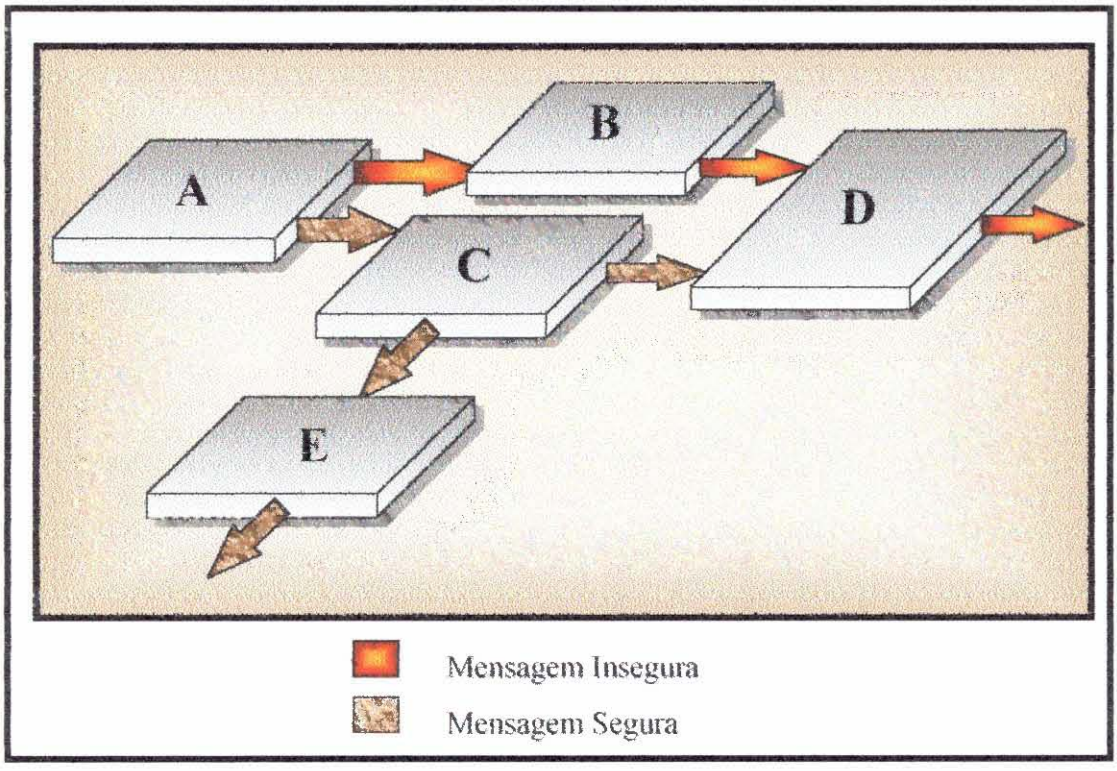


FIGURA 9.3 - Propagação inteligente de mensagens

Outro fator que merece um estudo mais aprofundado diz respeito à distribuição das agências nos processos participantes da simulação. Durante os testes notou-se que a distribuição das agências entre os processos afetou de forma significativa o desempenho geral da simulação.

Bibliografia

[BAR 92] BARCELOS, Antônio Marinho Pilla. **O Sistema Operacional de Rede Heterogêneo HetNOS**. Porto Alegre: CPGCC da UFRGS, 1992.

[CHA 79] CHANDY, K.Mani; MISRA, J. Distributed Simulation: a case study in design and verification of distributed programs. **IEEE Transactions on Software Engineering**, New York, v. SE-5, n.5, p. 440-452, Sept. 1979.

[FIG 94] FIGUEIRÓ, Joice Pavek. **Análise da Distribuição de um Simulador Multinível**. Porto Alegre: CPGCC da UFRGS, 1984.

[FUJ 90] FUJIMOTO, Richard M. Parallel. Discrete-event simulation. **Comunications of the ACM**, New York, v. 33, n. 10, p. 3-53, Oct. 1990.

[GRO 89] GROSELJ, B. A deadlock resolution scheme for distributed simulation. In: SCS MULTICONFERENCE ON DISTRIBUTED SIMULATION, 1989, Florida, US. **Proceedings...** San Diego: SCS, 1989 . p.108-112. (Simulation series, v.21, n.2).

[LIU 90] LIU, L.Z. Local deadlock detection in distributed simulation. In: SCS MULTICONFERENCE ON DISTRIBUTED SIMULATION, 1990, San Diego, US. **Proceedings...** San Diego: SCS, 1990. p.137-143.

[LUB 29] LUBRACHEVSKY, B.D. Efficient distributed event-driven simulations of mutiple-loop networks. **Communications of the ACM**, New York, v.32, n.1, p.111-123, Jan. 1989.

[SIL 88] SILVA FILHO, J.; WAGNER, F.R.; LE FAOU,C. **LAÇO - Uma linguagem para descrição de hardware no nível de sistema**. Porto Alegre: PGCC da UFRGS, 1988. (RP-94).

[SU 89] SU, Wen-King; SEITZ, Charles. Variants of the Chandy-Misra-Bryant distributed discrete-event simulation algorithm. In: SCS MULTICONFERENCE ON DISTRIBUTED SIMULATION, 1989, Florida, US. **Proceedings...** San Diego: SCS, 1989. p. 38-43. (Simulation series, v.21, n.2).

[WAG 87] WAGNER, Fávio Rech; FREITAS, C.M.D.S. **NILO - Uma linguagem de descrição de hardware no nível de portas lógicas**. Porto Alegre: PGCC da UFRGS, 1987. (RP-066).

[WAG 87a] WAGNER, Flávio Rech; FREITAS, C.M.D.S.; GOLENDZINER, L.G. **Linguagens de descrição de hardware para suporte à integração do projeto em AMPLO**. Porto Alegre: PGCC da UFRGS, 1987.(RP-065).

[WAG 87b] WAGNER, F.R.; FREITAS, C.H.D.S.; GOLENDZINER, L.G. A digital system design methodology based on nets os agencies. In: IFI WG10.2 INTERNATIONAL CONFERENCE ON COMPUTER HARDWARE DESCRIPTION LANGUAGES AND THEIR APPLICATIONS, 8., 1987, Amsterdam, **Proceedings...** Amsterdam: North-Holland, 1987. p.213-224.

[WAG 89] WAGNER, F.R. Um ambiente integrado para simulação de sistemas digitais. In: SIMPÓSIO BRASILEIRO DE CONCEPÇÃO DE CIRCUITOS INTEGRADOS, 4.,1989, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1989. p. 74-82.

[WAG 91] WAGNER, P.R. **Um ambiente integrado de simulação de sistemas digitais**. Porto Alegre, CPGCC da UFRGS, 1991. 121p. Dissertação de mestrado.

Informática  UFRGS

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Projeto e Implementação da Distribuição de um Simulador Multinível.

por

Luís Fernando da Silva

Dissertação apresentada aos Senhores:

Prof. Dr. Celso Maciel da Costa (PUCRS)

Profa. Dra. Ingrid Eleonora Schreiber Jansch Pôrto

Prof. Dr. Luigi Carro (IEE/UFRGS)

Vista e permitida a impressão.
Porto Alegre, 31/07/97.

Prof. Dr. Flávio Rech Wagner,
Orientador.

Profa. Carla Maria Pal Sasso Freitas
Coordenadora Substituta do Curso de
Pós-Graduação em Ciência da Computação
Instituto de Informática - UFRGS