



**ÂNDERSON DE MORAIS SOARES**

**IDENTIFICAÇÃO DE ESTADOS SEGUROS EM  
SIMULAÇÃO HÍBRIDA NO *DISTRIBUTED*  
*CO-SIMULATION BACKBONE - DCB***

**LAVRAS - MG  
2011**

**ÂNDERSON DE MORAIS SOARES**

**IDENTIFICAÇÃO DE ESTADOS SEGUROS EM SIMULAÇÃO  
HÍBRIDA NO *DISTRIBUTED CO-SIMULATION BACKBONE* - DCB**

Monografia apresentada ao Colegiado do  
Curso de Ciência da Computação, para  
obtenção do título de Bacharel em  
Ciência da Computação.

Orientador

Dr. Bráulio Adriano de Mello

Coorientador

Dr. Tales Heimfarth

**LAVRAS - MG  
2011**

ÂNDERSON DE MORAIS SOARES

IDENTIFICAÇÃO DE ESTADOS SEGUROS EM SIMULAÇÃO  
HÍBRIDA NO *DISTRIBUTED CO-SIMULATION BACKBONE - DCB*

Monografia apresentada ao Colegiado do  
Curso de Ciência da Computação, para  
obtenção do título de Bacharel em  
Ciência da Computação.

APROVADA em 06 de junho de 2011

Dr. Tales Heimfarth

UFLA



Dr. Luiz Henrique Andrade Correia

UFLA



Dr. Raphael Winckler de Bettio

UFLA



Orientador

Dr. Tales Heimfarth

Coorientador

Dr. Bráulio Adriano de Mello

LAVRAS - MG  
2011

## **AGRADECIMENTOS**

Agradeço ao meu orientador Bráulio Adriano de Mello pelos ensinamentos, pelas avaliações e pela correção que sempre demonstrou ao fazer suas considerações.

Ao Flávio Migowski, que desenvolveu trabalho na mesma linha de pesquisa, e me ajudou a tomar decisões importantes sobre o rumo deste trabalho.

Ao professor Dr. Tales Heimfarth pela co-orientação, aos professores Dr. Luiz Henrique Andrade Correia e Dr. Raphael Winckler de Bettio, por prontamente aceitarem o convite para participação na banca.

Enormemente à minha namorada Gabriella que é uma grande pesquisadora, e sempre teve uma crítica às vezes madura, outras vezes inocente, e por demais coerente em relação a este trabalho.

*Primeiramente aos meu pais que jamais deixaram de confiar na minha capacidade de conclusão da faculdade e deste trabalho, sempre apoiando as minhas decisões. A meu pai Wanomerato dedico pela determinação em mostrar o caminho correto e a firmeza para a tomada de decisões. Jamais deixou de me receber com um sorriso no rosto, um abraço e um aperto de mão de boas vindas nos finais de semana em nossa casa em Itaúna-MG, minha cidade natal. À minha querida mamãe Geralda pela doçura, pela extrema confiança, pela dedicação em saber se estava tudo bem comigo morando longe de casa. À fé que sempre teve em Deus, às orações por mim, e às ligações rápidas ou demoradas para contar casos e perguntar se minhas roupas estavam limpas ou se eu tinha almoçado bem. Enfim, à minha mãe, agradeço simplesmente por ser A MINHA mãe.*

*Aos meus irmãos Alysson e Wesley, simplesmente por fazerem bem o papel de irmão nas horas boas e ruins pelas quais passei. Aos meus eternos amigos da faculdade, Oswaldo Zezé, Éder Lespa, Eduardo Carioca, Ivan Lima, a dupla Carol e Ariana, os irmãos Flávio e Bruno Migowski, Tereza Tete, Gustavo Jojó, Elias, Thiago Vozim, Filipe Shun, Maycow e vários outros que esqueço de citar aqui. Estas pessoas conviveram comigo durante os anos da faculdade, e cada um ao seu jeito, me ensinou melhor o conceito de amizade. Foram trotes, festas, calouradas, rodeios, quartas, quintas e sexta-nejas, onde a diversão sempre prevaleceu. Teve também aquele teréré gelado em casa ou no Ciune, ou um churras em uma república qualquer, uma cachaça, um torresmo e um violão, seja para relaxar, seja pra festejar. Podem ter certeza, de cada um de vocês eu tenho um caso pra contar, e jamais vou esquecer. Aos companheiros de república, Marcus Paulo e Víctor, que como companheiros de república dividiram comigo momentos hilários, loucos, retardados, bacanas e também difíceis. Aos amigos de infância Vinícius Nico, Junin e Saulo. Mesmo de longe sempre se mostraram presentes para trocar aquela idéia bacana.*

*Não posso deixar de agradecer à equipe MDA Pesquisa. Ali trabalhei como entrevistador, e tive oportunidade de realizar meu primeiro estágio e assinar meu primeiro contrato profissional. A meus companheiros de trabalho Luiz, Juliana, Mariana, e claro, ao grande Marcelo, por ser além de chefe, um grande amigo. Esta oportunidade me fará um profissional e um homem mais completo. Agradeço também ao Marcus Couto e ao João Tavares. A oportunidade que me deram de trabalhar, e a “dura” tempos depois por não ter terminado este trabalho, foi fundamental para uma injeção de ânimo.*

*Existe uma pessoa que merecia uma dedicatória de fato, dedicada. Não por ser mais ou menos especial. Mas com minha maravilhosa namorada Gabriella, pude aprender a ser doce, pude aprender a cuidar e ter responsabilidade por uma pessoa, pude ter exemplo de disciplina e responsabilidade com meus estudos, pude ser homem e criança. Matei aula para vê-la, e fui à aula para vê-la. À esta linda pessoa, ainda me fogem palavras, mas devo agradecer à imensa dedicação, amor e carinho por mim, e por ter me ajudado, incondicionalmente, em tudo que precisei. Com ela, pude entender o que é ser namorado. Pude valorizar o fator família, e aprendi a ser companheiro. Após momentos de desânimo com a faculdade, ela insistiu que eu era capaz, e acreditei. Se não fosse isso, provavelmente eu jamais terminaria este trabalho.*

DEDICO

## RESUMO

Em simulação distribuída são necessários mecanismos que garantam a consistência do sistema em situações de *rollback* e *recovery*. O sistema se torna inconsistente a partir da existência de mensagens órfãs ou perdidas, registradas na linha de simulação de seus elementos quando estes executam *rollback*. Para garantia da consistência é necessário que os *checkpoints* (instantes locais) nos processos sejam determinados em instantes seguros. Na execução da simulação, caso elementos assíncronos necessitem retornar no tempo, estes retornam para os *checkpoints* determinados. Este trabalho apresenta um algoritmo para identificação de estados seguros em simulação híbrida no *Distributed Cosimulation Backbone*, DCB. Até o desenvolvimento deste trabalho, o DCB não identificava quais instantes de tempo eram seguros para se determinar um *checkpoint*. Os *checkpoints* no DCB eram não coordenados e portanto cada processo tinha total autonomia para determinar seu instante local. Para o DCB foi implementado um algoritmo de detecção de estados seguros, aliado à uma coordenação realizada por um elemento central, para determinação dos mesmos. Para o desenvolvimento do algoritmo foi adotada a ideia do conceito de *Termination Detection* (Detecção de Terminação). Este conceito prevê que uma computação terminou quando não existem mensagens em trânsito no sistema. Quando um processo é terminado, é possível dizer que um *checkpoint* naquele instante é seguro. No DCB este conceito é aplicado fazendo com que os elementos determinem *checkpoints* em instantes anteriores à execução de uma transição entre estados. A transição entre estados significa que um processo deixou de estar em um estado seguro, e passou para inseguro, e vice-versa. Portanto para cada computação terminada identificada no DCB, *checkpoints* seguros são determinados e um novo estado global consistente é gerado. Após a realização de testes através de um estudo de caso, eliminou-se a presença dos *checkpoints* inúteis. Além disso diminuiu-se o *overhead* de processamento e armazenamento, pois seriam necessários mecanismos de *Garbage Collector* para remoção destes *checkpoints*. Foi verificado neste trabalho que a eliminação de *checkpoints* não úteis para a simulação, a partir da construção de estados globais consistentes, permitiu ao DCB realizar uma simulação completa de elementos heterogêneos distribuídos.

Palavras-chave: *Termination Detection*. *Domino Effect*. *Checkpoint*. *Rollback Recovery*. Simulação Heterogênea Distribuída.

## LISTA DE FIGURAS

Figura 1 Um sistema distribuído: processos P1, P2, P3 e P4. Canais c1, c2, c3 e c4. Mensagem m1 no canal c1, m2 no canal c3, m3 no canal c4 e m4 no canal c2 .....	17
Figura 2 Visão funcional em alto nível de uma federação HLA.....	20
Figura 3 Troca de Mensagens, Mensagem Órfã, Mensagem Perdida, <i>Checkpoint</i> e Efeito Dominó .....	22
Figura 4 Mensagem Órfã e Mensagem Perdida.....	23
Figura 5 Estado Consistente e Estado Inconsistente .....	25
Figura 6 Sistema Conservativo de um <i>token</i> .....	27
Figura 7 Algoritmo Chandy e Lamport.....	28
Figura 8 Visão geral do DCB .....	33
Figura 9 Exemplo de <i>checkpoint</i> inútil.....	42
Figura 10 Mensagem em trânsito no sistema e criação de <i>checkpoints</i> inúteis ..	43
Figura 11 Terminação de computação e reinício da computação .....	48
Figura 12 Buffer que controla mensagens em trânsito no canal de entrada do processo.....	50
Figura 13 Mensagem descrita pelo federado 8, com destino ao federado 4 .....	50
Figura 14 Código do <i>Gateway4</i> que detecta Terminação de Computação.....	51
Figura 15 Novas mensagens de controle do <i>FedGVT</i> .....	52
Figura 16 Atendimento de uma requisição de <i>checkpoint</i> enviada pelo <i>FedGVT</i> .....	53
Figura 17 <i>Array</i> que apresenta ao sistema a situação do mesmo, seguro ou inseguro.....	54
Figura 18 Transição de Estados no DCB .....	55
Figura 19 Envio de mensagem a um federado.....	56
Figura 20 Mensagem recebida no tempo 10000 .....	57



Figura 21 Sistema executando algoritmo de Estados Seguros.....	58
Figura 22 Elementos: EDCB, ApplicationDCB, DCBS, DCBR, DCB .....	60
Figura 23 Elementos: Gateway, EF, Gateway 4 e Chat .....	61

## SUMÁRIO

<b>1 INTRODUÇÃO</b> .....	10
1.1 Contextualização e Motivação .....	10
1.2 Objetivo .....	12
1.3 Estrutura do Trabalho .....	13
<b>2 REFERENCIAL TEÓRICO</b> .....	14
2.1 Simulação.....	14
2.1.1 Simulação Distribuída .....	14
2.1.2 Simulação Heterogênea .....	15
2.1.3 Simulação Heterogênea Distribuída .....	16
2.2 HLA – <i>High Level Architecture</i> .....	18
2.2.1 Quesitos técnicos do HLA .....	19
2.3 Quesitos e Propriedades para Simulação Híbrida.....	20
2.3.1 <i>Checkpoints</i> .....	20
2.3.2 Anti-Mensagens .....	21
2.3.3 Efeito Dominó .....	21
2.3.4 Mensagem Órfã e Mensagem Perdida .....	23
2.4 Identificação de Estados Globais Consistentes .....	24
2.4.1 Algoritmo de Chandy e Lamport .....	25
2.4.2 Algoritmo de Lai e Yang ( <i>Lai-Yang Algorithm</i> ).....	29
2.4.3 Detecção de Deadlock .....	30
2.4.4 Detecção de Terminação.....	31
2.5 Distributed Co-simulation Backbone.....	32
2.5.1 Sincronização no DCB.....	34
2.5.2 Anti-Mensagens .....	36
2.5.3 Rollback Recovery .....	37
2.5.4 <i>Checkpoint</i> .....	37
<b>3 METODOLOGIA</b> .....	39
<b>4 RESULTADOS E DISCUSSÃO</b> .....	40
4.1 Problema dos Estados Incosistentes no DCB.....	41
4.2 O Impacto dos <i>Checkpoints</i> Não Coordenados para Determinação de Estados Seguros.....	44
4.3 <i>Checkpoints</i> Coordenados e Identificação de Estados Seguros no DCB.....	45
4.3.1 O algoritmo de <i>Termination Detection</i> na Identificação de Estados Seguros .....	46

<b>4.3.2 Adaptações para alcance dos objetivos .....</b>	<b>47</b>
<b>4.3.3 Gateway 4, o Termination Analyzer .....</b>	<b>49</b>
<b>4.3.4 <i>FedGVT</i> .....</b>	<b>52</b>
<b>4.4 Estudo de Caso .....</b>	<b>54</b>
<b>4.5 Estrutura do DCB .....</b>	<b>59</b>
<b>5 CONCLUSÃO .....</b>	<b>62</b>
<b>5.1 Trabalhos Futuros.....</b>	<b>63</b>
<b>REFERÊNCIAS .....</b>	<b>64</b>

## 1 INTRODUÇÃO

### 1.1 Contextualização e Motivação

Simulação é o conceito que se dá para a representação da realidade. No contexto deste trabalho simular significa representar um comportamento, que pode ser real ou virtual, com o propósito de se analisar eficácia, eficiência e falhas, reduzindo custos de produção, construção de protótipos e homologação (Fujimoto & Weatherly, 1996).

A simulação computacional possibilita que a execução de um projeto passe por testes mais eficazes, analisando o comportamento do mesmo quando submetido a situações que possam causar falhas, impasse, perda de sincronia, quebra, necessidade de re-execução de uma tarefa dentre outros. Qualquer área do conhecimento que seja passível de ser modelada computacionalmente pode ser também simulada computacionalmente.

Em uma simulação pode ser necessário que dois ou mais modelos computacionais sejam executados em cooperação, ou seja, pode ser que dois equipamentos distintos precisem trabalhar juntos efetuando troca de informações. Um exemplo de cooperação entre elementos distintos pode ser verificada quando o componente a ser simulado seja uma pessoa, havendo interação direta com uma máquina, que, como passível de modelagem computacional, seja simulada a fim de se representar o comportamento dos elementos.

Na simulação computacional a representação dos modelos a serem simulados podem se diferir quanto arquitetura, linguagem de especificação, controle interno de tempo, distribuição geográfica dentre outros fatores.

Tendo como base os diversos aspectos da simulação, são definidos dois tipos principais de simulação: homogênea e heterogênea.

Simulação onde os elementos são semelhantes quanto aos seus atributos, é conhecida como Simulação Homogênea. Este tipo de simulação é eficiente sobre o ponto de vista ao qual se propõe, porém muitas vezes a simulação precisa representar elementos completamente diferentes, definindo assim a Simulação Heterogênea (Reynolds Jr, 1988).

A simulação de elementos, sejam eles homogêneos ou heterogêneos, pode ser realizada sobre um aspecto físico diferente. Dois elementos podem não estar no mesmo local geográfico, e a interação destes elementos pode ser realizada através de um sistema de simulação que suporte simulação distribuída (Mello & Wagner, 2001). O objetivo da simulação se limita portanto a representar com fidelidade o comportamento dos sistemas reais. A realidade no que tange os sistemas computacionais, é considerar que qualquer elemento, em qualquer instante ou circunstância pode vir a falhar, reproduzir comportamento inesperado, ter sua execução interrompida (mesmo que por fatores externos), ou até mesmo sua execução finalizada. Tratando fatores como estes, o objetivo de simular com coerência a realidade pode ser alcançado, já por outro lado, nota-se a dificuldade de se representar fielmente um modelo real.

Existem na literatura diversos trabalhos que apresentam sistemas de suporte a simulação distribuída. Dentre os vários podemos citar Vithayathil (1974) que apresentou um modelo interessante de simulação de correntes marítimas utilizando os conceitos de simulação híbrida, Mello e Caimi (2008) apresentam um estudo de caso para validação de sistemas computacionais de apoio à agricultura de precisão, Labella et al. (2008) apresentam um sistema de simulação de sensores utilizando a plataforma de simulação distribuída denominada BARAKA e Borshchev et al. (2002) apresentam um sistema para simulação do comportamento de tanques de água e um operador humano.

Este trabalho foi desenvolvido tomando como base a arquitetura de co-simulação heterogênea denominada *Distributed Cosimulation Backbone* (Mello,

2005), ou simplesmente DCB. O DCB foi inspirado na *High Level Architecture* (HLA). A HLA é um padrão IEEE para arquitetura de sistemas de simulação distribuídos (Fujimoto & Weatherly, 1996; Kuhl, 2000). O padrão HLA define uma interface de comunicação e ainda define que uma simulação é baseada em componentes, denominados federados. Um conjunto de federados formam uma federação. O DCB trabalha sob os principais tópicos sobre simulação heterogênea e distribuída.

O DCB como um projeto em desenvolvimento possui suporte a simulação distribuída, uma vez que permite que os modelos estejam distribuídos fisicamente e/ou logicamente, e também suporte simulação heterogênea, pois permite que elementos desenvolvidos em linguagens, arquitetura e controle temporal diferentes, possam ser simulados em cooperação.

O DCB possui uma solução para determinação de *checkpoints*. *Checkpoints* são marcas no tempo que permitem que a linha de simulação retorne no tempo por diversos motivos, dentre os mais citados estão falhas de comunicação ou então necessidade intrínseca de cada elemento, sem a perda de informações gravadas anteriormente a esta marca de tempo. O DCB possuía até este trabalho, *checkpoints* não coordenados (*uncoordinated checkpoints*) (Elnozahy et al, 2002; Khunteta & Kumar, 2010), garantindo autonomia para cada elemento na determinação de seus *checkpoints*.

## 1.2 Objetivo

O trabalho teve como objetivo desenvolver uma solução para a identificação de estados seguros na linha de simulação do DCB, permitindo definir quais os momentos mais apropriados para o estabelecimento dos *checkpoints* em situações de *rollback* e *recovery* (Venkitakrishnan, 2002).

### **1.3 Estrutura do Trabalho**

O trabalho foi organizado da seguinte forma: na Seção 2 foram apresentados os principais aspectos para apresentação da simulação, e os elementos que compõem a arquitetura do DCB. Na Seção 3 é apresentada a metodologia do trabalho. Na Seção 4 são apresentados os resultados. Nesta seção foram apresentados os problemas encontrados para determinação de estados seguros, e algoritmos e técnicas utilizados. Na Seção 5 é apresentado um estudo de caso para demonstração da aplicação do algoritmo, e o objetivo alcançado. Na Seção 6 é apresentada a bibliografia utilizada no trabalho.

## 2 REFERENCIAL TEÓRICO

Este capítulo apresenta o funcionamento da simulação híbrida através do DCB (*Distributed Cosimulation Backbone*). São apresentados trabalhos anteriores que contribuíram para estudos em simulação, abordagens distintas de técnicas de interação entre simuladores e ainda formas de identificação de estados seguros para determinação de *checkpoints* úteis, apresentados por outros autores.

### 2.1 Simulação

A simulação de sistemas se define basicamente no desenvolvimento de técnicas que permitem imitar o comportamento dos sistemas reais, permitindo execução de atividades a partir da construção de modelos que os representem (Law & Kelton, 1991). O propósito geral da simulação computacional é reduzir o esforço dos desenvolvedores no que tange a criação de protótipos e validação dos sistemas (Mello 2005).

#### 2.1.1 Simulação Distribuída

Na simulação distribuída os elementos podem estar em diferentes localizações geográficas, sobre uma LAN (*Local Área Network*) ou WAN (*Worldwide Área Network*).

Os benefícios apresentados por esta abordagem são: (i) descentralização do projeto, (ii) o design e validação do projeto pode ser realizada por mais de uma equipe, (iii) gerenciamento de propriedade intelectual sobre o software, sem a necessidade de abertura do código fonte, (iv) gerenciamento de licenças, uma vez que os simuladores podem ser instalados em poucas máquinas, e (v) permite



o compartilhamento de recursos, já que quanto maiores os recursos computacionais, mais rápida pode ser uma simulação (Ferscha, 1995; Hessel et al., 1998).

Por outro lado, a principal desvantagem dessa abordagem é o tempo requerido para simulação, que pode oferecer alto *overhead* de comunicação.

A simulação local não oferece *overhead*, em contrapartida não oferece também as vantagens da simulação distribuída, sobre o ponto de vista de conectividade e robustez (Fujimoto & Weatherly, 1996).

A simulação distribuída é dividida em duas abordagens: SRIP (*Single Replication in Parallel*) ou uma replicação em paralelo, e MRIP (*Multiple Replication in Parallel*) ou várias replicações em paralelo. Essas duas abordagens definem como o sistema é particionado.

Na abordagem SRIP parte-se o sistema em vários processos lógicos, sendo que cada um é mapeado em um processador. Para garantir a sincronia dos eventos (trocas de mensagens) é necessária a implementação de um protocolo de sincronização. Esta medida é tomada a fim de se sincronizar os tempos de simulação dos processos lógicos, cujos tempos de eventos podem depender de outros processos lógicos.

Na abordagem MRIP o sistema não é particionado. Nesse caso o sistema é alocado independentemente através de replicações do mesmo, e são executados em paralelo. O analisador global recebe os resultados individuais e calcula a média. Quando a média está dentro da faixa esperada, a simulação é terminada (Ferscha, 1995).

### **2.1.2 Simulação Heterogênea**

A modelagem de um sistema de simulação pode ser realizada de forma homogênea ou heterogênea.

Na primeira abordagem, é necessário somente um simulador, sendo que a linguagem dos módulos simulados é transcrita em uma linguagem denominada intermediária, chamada *formato intermediário*, que descreve todo o sistema. Na segunda abordagem, é necessário um módulo em *backplane* para cada simulador, que será responsável pela interpretação e tradução das informações transmitidas pelos diversos módulos, para o módulo central (Liem et al., 1997).

Na abordagem heterogênea tem-se a vantagem da cooperação de elementos distintos, já que podem se diferir quanto à linguagem de especificação, arquitetura e tempo de simulação (Ferscha, 1995; Fujimoto, 1995; Mello, 2005). As vantagens da abordagem heterogênea trazem consigo os maiores desafios de sua modelagem, (i) como decidir o paradigma mais apropriado para cada parte que compõe a especificação global, (ii) como definir as especificações do sub-sistema em um formato unificado para *design*, e (iii) como verificar se as especificações e o formato de cada sub-sistema corresponde aos requerimentos da especificação global (Hessel et al., 1998). Um quesito importante na simulação de sistemas heterogêneos é o tempo interno de funcionamento de cada parte.

### **2.1.3 Simulação Heterogênea Distribuída**

Um sistema distribuído é formado por um conjunto de processos que trocam informações através de canais de comunicação. O envio de mensagens é representado por um grafo, onde as linhas horizontais representam os processos, e as linhas contínuas entre os processos representam a comunicação entre eles, formando um canal de comunicação (Randell, 1975; Higaki et al., 1997), conforme pode ser observado pela Figura 1.

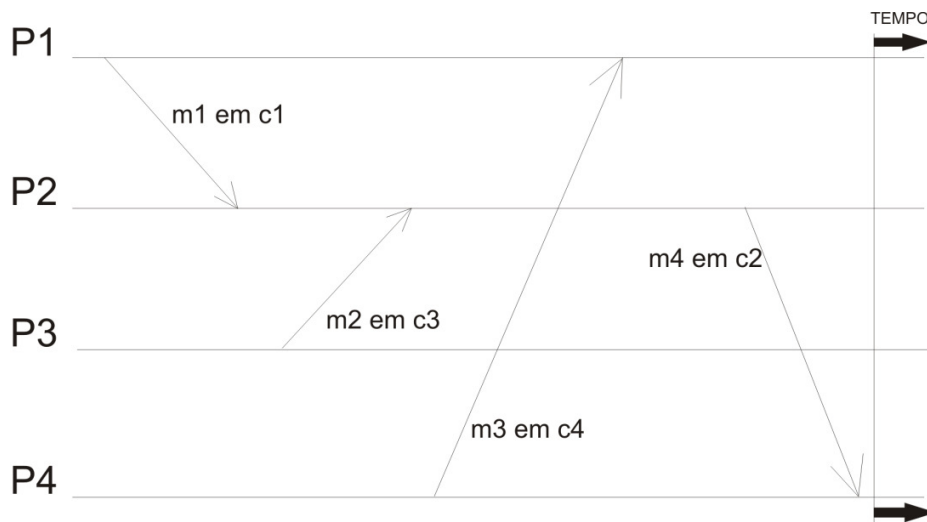


Figura 1 Um sistema distribuído: processos P1, P2, P3 e P4. Canais c1, c2, c3 e c4. Mensagem m1 no canal c1, m2 no canal c3, m3 no canal c4 e m4 no canal c2

A Figura 1 apresenta um sistema distribuído composto de quatro processos. Cada processo possui um canal de comunicação. Veja que na figura, foi transmitida a mensagem m1 através do canal c1 do processo P1, em direção ao processo P2. Similarmente, m2 foi transmitida pelo processo P3, através do canal de comunicação c3, no sentido de P2.

Um modelo distribuído deve garantir a sincronização da simulação, fazendo com que os eventos disparados por cada Processo Lógico (modelo) ocorram nos seus devidos tempos de evento (*timestamp*). É obrigatório também considerar que os tempos de atraso das mensagens são arbitrários, porém finitos.

O modelo híbrido (modelo de simulação de elementos heterogêneos e distribuídos) deve considerar então três formas de abordagem sobre o tempo de simulação: síncrono, assíncrono e *no-timed*. O modelo síncrono é outrora denominado conservador. O modelo assíncrono pode também ser chamado de otimista.

O simulador cuja abordagem é *no-timed* não possui coordenação sobre tempo, logo os eventos que ocorrem sobre sua linha de simulação são registrados conforme eles são executados, obedecendo necessariamente à ordem de chegada das mensagens (Mello, 2005).

Na abordagem conservadora, a ordem da ocorrência dos eventos deve ser assegurada por um protocolo que gerencia as trocas de mensagens. Este protocolo deve funcionar em conjunto com um relógio global para realizar a sincronização das mensagens, o GVT (*Global Virtual Time*) (Ferscha, 1995). Esta abordagem trabalha sobre o conceito de que a linha de simulação do elemento é sempre progressiva, ou seja, o elemento não pode voltar no tempo de simulação. Um problema deste tipo de abordagem é a dificuldade de controlar o tempo de diferentes simuladores. Neste tipo de modelagem, caso o elemento identifique um fator de inconsistência, ou uma falha, a linha de simulação deve ser interrompida e recomeçada do zero, ou então simplesmente barrada sua execução (Mello, 2005).

No oposto da abordagem conservadora, elementos assíncronos (otimistas) podem retornar no tempo, o que é chamado de *rollback*, que significa retorno no tempo de simulação para depois refazer a simulação (Higaki et al., 1997). Este tipo de modelo possui maior flexibilidade quanto ao gerenciamento do tempo, e portanto é mais difícil de ser implementado. Considerando este tipo de abordagem em uma simulação distribuída, o controle sobre o tempo é ainda mais complexo, sendo dependente do *overhead* gerado pela comunicação na rede (Mello, 2005).

## **2.2 HLA – High Level Architecture**

O HLA é um padrão para o IEEE atualmente. Em 1995 o HLA foi desenvolvido pelo Departamento de Defesa dos Estados Unidos (DoD –

*Department of Defense*) como resultado de uma pesquisa realizada entre o governo, o ambiente acadêmico e a indústria nos termos de simulação interativa e distribuída. O objetivo do HLA era aperfeiçoar os treinamentos militares na época (Calvin & Weatherly, 2003).

O HLA define um padrão no IEEE, entretanto não exige um padrão de implementação, nem uma linguagem específica. O que ele propõe é uma metodologia de estudo para ambientes de simulação distribuído, respeitando as características específicas do ambiente onde estiver sendo utilizada (Wagner, 2003).

A arquitetura HLA fornece uma interface padronizada para simulações distribuídas e oferece suporte para o desenvolvimento de simulações baseadas em componentes, chamados de federados. Um conjunto de federados forma uma federação (Wagner, 2003; Mello, 2005).

### **2.2.1 Quesitos técnicos do HLA**

O HLA trabalha com o conceito de federações. Um conjunto de federados que combinados com a RTI (*RunTime Infrastructure Services* – Infraestrutura de Serviços em Tempo de Execução), formam a federação (Amory et al., 2002).

A RTI é responsável pelo controle de operações de troca de mensagens entre os federados e federações (Fujimoto & Weatherly, 1996; Amory et al., 2002; Wagner, 2003; Mello, 2005). Como dito o HLA não impõe regras específicas quanto arquitetura dos federados, porém existe um conjunto de especificações que devem ser respeitadas para um federado participar da federação (Mello, 2005).

Na figura 2 pode ser observada a estrutura do HLA. É importante destacar a interface que existe entre a RTI e os federados.

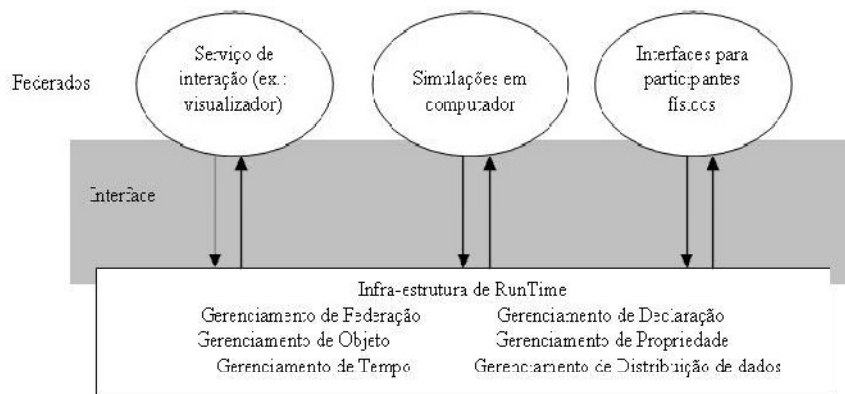


Figura 2 Visão funcional em alto nível de uma federação HLA

Fonte: Dahmann (1998)

## 2.3 Quesitos e Propriedades para Simulação Híbrida

### 2.3.1 Checkpoints

*Checkpoints* são marcas no tempo da simulação de cada elemento, para onde o mesmo elemento pode retornar caso seja necessário. Esta é uma técnica na simulação que permite que um simulador retorne a um tempo passado, determinado pelo *checkpoint* (Higaki et al., 1997; Elnozahy et al., 2002). Esta característica torna o sistema *failure-free*, ou seja, tolerante a falhas (Johnson & Zwaenepoel, 1990; Baldoni et al., 1995).

É imprescindível que a determinação de *checkpoints* em uma linha de simulação seja feita de forma eficiente, ou seja, é importante considerar um número menor possível de *checkpoints*, pois assim diminui-se a quantidade de informações armazenadas/trocadas entre os elementos. É importante também

considerar que os *checkpoints* sejam úteis para linha de simulação, ou seja, ele não deve salvar mensagens que possam se tornar órfãs em situações de *rollback*, evitando assim o efeito-dominó (*domino-effect*), contribuindo para a menor perda possível de informações.

Existem na literatura diversos algoritmos para determinação de *checkpoints*. Elnozahy et al. (2002) mostra de forma detalhada e abrangente vários desses algoritmos.

### **2.3.2 Anti-Mensagens**

Anti-Mensagens são mensagens semelhantes a outras, que são enviadas a outros federados quando um ou mais federados necessitam retornar no tempo de simulação, uma vez que houve uma violação de tempo (Ferscha, 1995). A Anti-Mensagem possui um código que a difere. Quando um federado recebe uma mensagem desse tipo, ele retorna no tempo de simulação até um *checkpoint* anteriormente determinado. Esta mensagem serve somente para “avisar” ao federado que ele precisa retornar no tempo, pois ele precisa executar um evento em um tempo passado ao seu atual (Ferscha 1995, Elnozahy et al., 2002; Mello, 2005).

### **2.3.3 Efeito Dominó**

O Efeito Dominó é uma propagação incontrolada de cada processo (Baldoni et al., 1995), quando ele retorna no tempo. Este acontecimento deve ser evitado, pois um processo ao necessitar realizar *rollback* pode fazer com que outros processos também voltem no tempo. No fim todos os processos podem ter de voltar ao seu estado inicial, necessitando assim realizar toda simulação

novamente (Randell, 1975; Higaki et al., 1997). A Figura 2 representa o Efeito Dominó na simulação.

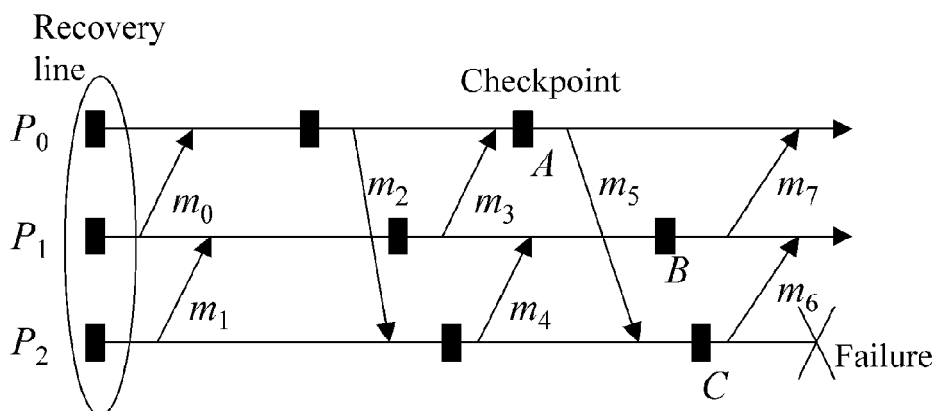


Figura 3 Troca de Mensagens, Mensagem Órfã, Mensagem Perdida, *Checkpoint* e Efeito Dominó

Fonte: Elnozahy et al. (2002)

A Figura 3 representa um sistema distribuído constituído de três processos,  $P_1$ ,  $P_2$  e  $P_3$ , que são representados pelas retas horizontais. Os retângulos pretos localizados acima das linhas dos processos representam os *checkpoints* locais. As setas entre as retas dos processos, representam as mensagens trocadas.

Observa-se pela Figura 3 que o processo  $P_2$  falha e retorna ao *checkpoint*  $C$ . Ao retornar,  $P_2$  dispara uma Anti-Mensagem ao  $P_1$  e o informa que a mensagem  $m_6$  será invalidada, uma vez que ela se tornaria uma mensagem órfã, e por isso ele deve retornar no tempo. Quando  $P_1$  retorna, é necessário remover  $m_7$  pelo mesmo motivo anterior, e  $P_1$  então requer que  $P_0$  também retorne no tempo, nesse caso, até o *checkpoint*  $A$ . O leitor pode observar que a propagação de retornos levará o sistema ao seu estado inicial, ou seja, aos



primeiros *checkpoints* de cada processo, no tempo zero. Esta situação é extremamente indesejada, e deve ser contornada quando se trata de simulação distribuída.

### 2.3.4 Mensagem Órfã e Mensagem Perdida

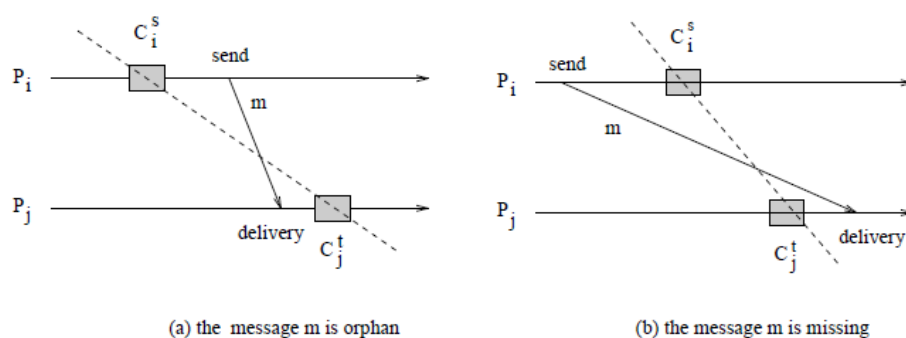


Figura 4 Mensagem Órfã e Mensagem Perdida

Fonte: Baldoni et al. (1995)

Uma mensagem órfã é aquela em que em determinado tempo da simulação, um *checkpoint* no processo de destino registra seu recebimento, porém nenhum outro *checkpoint* no processo de origem registra seu envio. A Figura 4 (a) representa uma mensagem órfã no sistema de simulação.

Uma mensagem perdida surge pelo mesmo problema que gerou uma mensagem órfã, entretanto, uma mensagem perdida é aquela em que um *checkpoint* no processo de origem registra seu envio, porém nenhum *checkpoint* no destino registra seu recebimento. Observando a Figura 4 (b), o processo  $P_i$  registra o envio da mensagem  $m$ , entretanto, o processo  $P_j$  não registra o recebimento da mesma.

Estes dois tipos de mensagens são altamente indesejáveis, uma vez que ao necessitar realizar um *rollback* a presença das mesmas tornará o sistema inconsistente com a linha de simulação. Para evitar estes tipos de mensagens um estudo profundo na determinação dos *checkpoints* deve ser realizado a fim de encontrar instantes de tempo consistentes ou seguros, que não registrem estes tipos de mensagens. Ao encontrar um instante de tempo seguro, um *checkpoint* deve ser determinado, deixando o sistema sempre consistente mesmo em situações de *rollback*.

#### **2.4 Identificação de Estados Globais Consistentes**

Mensagens Órfãs e Mensagens Perdidas são conseqüências diretas do *rollback*. Um dos desafios de se determinar quando um estado é consistente é analisando se existe algum desses tipos de mensagens no processo.

Estados consistentes são uma exigência para tornar o sistema recuperável. Um sistema é chamado recuperável se e somente se todos estados dos processos são estáveis e o sistema resultante é consistente. Isto garante que caso o sistema necessite retornar no tempo, a re-execução das tarefas descartadas será realizada de forma idêntica à que tinha ocorrido antes do *rollback* (Johnson & Zwaenepoel, 1990).

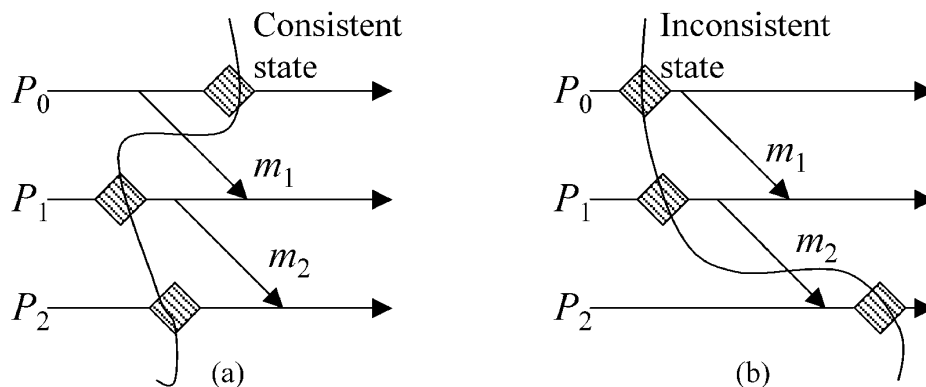


Figura 5 Estado Consistente e Estado Inconsistente

Fonte: Elnozahy et al. (2002)

Observa-se pela Figura 5 (a) que o estado de  $P_0$  mostra que  $m_1$  foi enviada, e ela continua em trânsito. Este estado é consistente. Na Figura 5(b) o estado de  $P_1$  é inconsistente, pois  $P_2$  registra o recebimento de  $m_2$ , no entanto  $P_1$  não registra seu envio. É nítido que se o processo  $P_1$  falhar depois do envio de  $m_2$ , ele retornará ao estado salvo na figura, e a mensagem enviada por ele, quando descartada, se tornará órfã em  $P_2$ . Na Figura 5(a),  $P_1$  pode se tornar inconsistente, pois caso ele falhe após o recebimento de  $m_1$ , a mensagem se tornará perdida em  $P_0$ .

Pode-se observar que um algoritmo para determinação de estados consistentes não é algo trivial.

#### 2.4.1 Algoritmo de Chandy e Lamport

Chandy e Lamport (1985) foram pioneiros ao construir um algoritmo para detecção de estados globais consistentes. A partir de um estado global é possível definir se ele é estável para uma simulação distribuída. Uma propriedade estável é uma que persiste da seguinte definição: "uma vez que uma

*propriedade se torna estável, ela se mantém sempre estável*". São exemplos de propriedades estáveis: fim da computação (*Termination Detection*) e *deadlock*.

O algoritmo proposto por eles deve cumprir os seguintes requisitos:

- Um processo pode salvar seu próprio estado e as mensagens enviadas ou recebidas por ele.
- Deve obedecer a relação de precedência, por eles chamado de "*Happens Before*". Um evento  $e1$  precede (*Happens Before*)  $e2$ , denotado por  $e1 \rightarrow e2$  se, (1)  $e1$  ocorre antes de  $e2$  no mesmo processo, ou (2)  $e1$  é o evento de envio da mensagem e  $e2$  é o evento de recebimento da mensagem, ou ainda (3)  $e1$  precede  $e'$  e  $e'$  precede  $e2$ . Esta condição atende à *Causal Order Delivery*, e sendo assim, o canal de comunicação é FIFO – *First In First Out*.

Seja um sistema de dois processos  $p$  e  $q$ , e um canal  $C$  de envio, e outro  $C'$  de recebimento representado pela Figura 6.

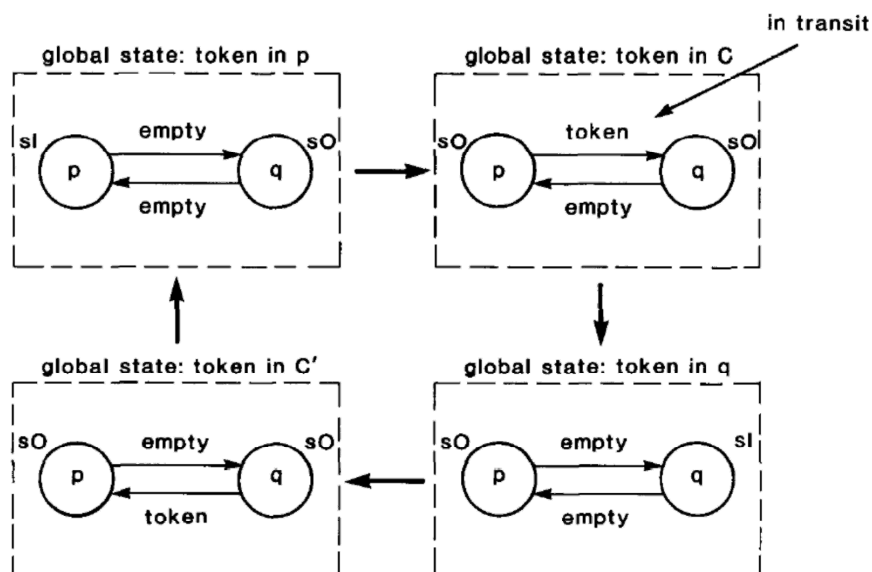


Figura 6 Sistema Conservativo de um *token*

Fonte: Chandy &amp; Lamport. (1985)

É entre esses canais onde será realizada a comunicação entre os processos. O cenário demonstra o que é chamado de *single-token conservation*, ou sistema conservativo de somente um token.

**p** salva seu estado antes do envio de uma mensagem, logo o *token* (mensagem) é salvo no estado global *in-p*. Considere que o estado global dos processos transita para *in-c*, ou seja, a mensagem foi enviada por **p** e o canal C encontra-se sobre posse do *token*. Logo o estado global registra que o *token* está em C. Neste exemplo, o estado **p** e C registram que estão sobre posse do *token* e os estados **C'** e **q** não o possuem. Com este cenário nosso estado global apresenta dois *tokens* ( $N=2$ ), sendo que na verdade só foi emitido um. Um cenário de um *token* jamais pode apresentar dois *tokens* em seu estado global. Isto acontece pois o estado **p** salvou seu estado antes do envio do *token* e o estado de C foi salvo após o envio do mesmo. Este exemplo sugere que o sistema será inconsistente se  $N < N'$ , ou seja, não se pode receber mais mensagens do que já foram enviadas.

Agora consideremos um cenário alternativo. Salva-se o estado de C no estado global *in-p*. O sistema ao transitar para *in-c* salva o estado de **C'**, **p** e **q** em *in-c*. Sendo assim, o estado global registra zero *tokens*. Isso mostra que o estado global é inconsistente se C salva seu estado antes de **p** enviar a mensagem, e se **p** salva seu estado depois de ele mesmo enviar uma mensagem, ou seja, se  $N > N'$ . Conclui-se então que um estado global deve registrar que  $N = N'$ .

O algoritmo utiliza-se de duas sub-rotinas para controle de envio e recebimento de mensagens, o *Marker-Sending Rule* aqui chamado de MSR e *Marker-Receiving Rule* aqui chamado de MRR. A primeira rotina é executada

sempre que um marcador de controle é enviado pelo processo P1, e a segunda, sempre que P2 receber um marcador. Para o MSR: P1 envia um marcador através de C depois de P1 salvar seu próprio estado e antes de P1 enviar outras mensagens

Para o marcador de envio, realiza-se a seguinte rotina: para cada canal C incidente em P1 e direcionado no sentido de saída de P1: P1 envia um marcador através de C depois de P1 salvar seu próprio estado e antes de P1 enviar outras mensagens.

Para o segundo marcador, ao receber o marcador proveniente do canal C, realiza-se a seguinte rotina conforme Figura 7.

```

if  $q$  has not recorded its state then
  begin  $q$  records its state;
         $q$  records the state  $c$  as the empty sequence
  end
else  $q$  records the state of  $c$  as the sequence of messages received along  $c$  after  $q$ 's state
      was recorded and before  $q$  received the marker along  $c$ .

```

Figura 7 Algoritmo Chandy e Lamport

Fonte: Chandy e Lamport (1985)

Alguns algoritmos tomam o algoritmo de Chandy & Lamport (1985) como base para sua definição, porém no DCB o canal não é FIFO, uma vez que uma mensagem  $m1$  enviada antes de  $m2$  não precisa necessariamente ser recebida antes de  $m2$ . Sejam  $m1$  e  $m2$  mensagens enviadas pelo mesmo processo. No tempo 10,  $m1$  pode ser enviada por P1 para ser executada no P2 no tempo 30, e  $m2$  pode ser enviada por P1 no tempo 20 para ser executada por P2 no tempo 25. Logo o recebimento de  $m2$  ocorrerá antes do recebimento de  $m1$ .

No DCB, a ocorrência das mensagens no processo destino é determinada por seu *timestamp*. Outro quesito importante é que no DCB os *checkpoints* são

tomados de forma não coordenada, e no algoritmo de Chandy e Lamport os *checkpoints* devem ter alguma coordenação.

#### 2.4.2 Algoritmo de Lai e Yang (*Lai-Yang Algorithm*)

O algoritmo de Lai e Yang (1987) é um que se baseia na idéia inicial de Chandy e Lamport. Porém em sua implementação, uma pergunta foi gerada: pode um algoritmo não coordenado de *checkpoints* gerar um estado global estável? Em (1987) Lai e Yang provam que sim.

São duas as diferenças primordiais entre o algoritmo de Chandy e Lamport para o algoritmo de Lai e Yang. Lai e Yang assumem que os *checkpoints* no sistema são definidos de forma não coordenada, e os canais de comunicação não são necessariamente *first-in first-out* (mensagens são entregues na ordem de envio) (Lai & Yang, 1987).

O algoritmo de Lai e Yang funciona basicamente dividido em três passos.

1 – Todo processo é inicialmente branco e se torna vermelho até determinar um *checkpoint* local.

2 – Toda mensagem enviada por um processo branco, é colorida de vermelho.

3 – Todo processo branco determina um *checkpoint* local de acordo com sua conveniência, mas jamais antes de uma mensagem vermelha ser possivelmente recebida. Sendo assim, a chegada de uma mensagem vermelha em um processo branco vai fazer com que o processo tome um *checkpoint* antes de receber a mensagem.

Após cada processo determinar seus instantes locais, então eles são utilizados para formar um *checkpoint* global. É visível que este algoritmo se torna mais caro no momento em que exige trocas de mensagens entre processos,

aumentando assim o *overhead* de comunicação, mas por outro lado não exige que o canal de comunicação seja FIFO. Esta representação se torna assim uma importante solução para aplicações como *Termination e Deadlock Detection*, onde o número de mensagens enviadas e recebidas é motivo de atenção. É importante notar que o algoritmo é forte em manter a independência dos processos em determinar seus *checkpoints* locais. Ele garante ainda que cada processo possa iniciar o algoritmo independentemente, ou o algoritmo ser iniciado por um único processo.

Como pode ser observado, uma aplicação irá determinar N estados globais, para descobrir que algum deles será consistente. Por outro lado permite que cada processo ao determinar um novo instante local, resete seu conjunto de *checkpoints* locais para o conjunto vazio, diminuindo assim a quantidade de informações armazenadas (Lai & Yang, 1987), e contribuindo diretamente para *Garbage Collection*.

### **2.4.3 Detecção de Deadlock**

Detecção de *Deadlocks* é uma forma de determinar quando processos estão aguardando por informações, sejam porque requisitaram a informação, ou porque enviaram a informação, e aguardam uma resposta. Quando a detecção de *deadlock* é feita por um único processo chamado agente central, e ainda os processos se intercomunicam diretamente e livremente, a detecção é mais difícil. Se a comunicação entre os processos é feita de forma instantânea, ou seja, não existem atrasos de entrega causados por *delays* na rede, a detecção, por outro lado, é trivial. No entanto, em simulação distribuída, é impossível abdicar deste problema, considerando que os atrasos são arbitrários, porém finitos, e os canais de comunicação sejam FIFO.



Existem dois tipos de *deadlocks*: o de recurso, e o de comunicação.

*Deadlock* de recurso é aquele em que um processo pode ficar aguardando todos recursos requisitados, **a**, **b** e **c**, que devem ser transmitidos por outros processos, e podem jamais chegar. A continuidade da execução desse processo é dependente da chegada deste recurso.

*Deadlock* de comunicação é mais abstrato e mais genérico. Este é analisado conforme portas lógicas AND e OR. Um processo pode ficar aguardando um recurso  $a$  AND ( $b$  OR  $c$ ). Se o processo receber o recurso  $a$  ele pode ficar permanentemente aguardando o recurso  $b$  OR  $c$ .

Analogamente, se um processo aguarda por um recurso qualquer, este recurso pode ser comparado à uma mensagem a ser recebida ou transmitida. Se há um recurso a ser recebido, logo há uma mensagem em trânsito no sistema. Como citado nos tópicos anteriores, uma mensagem em trânsito no sistema é suficiente para identificar que o sistema é inseguro para a determinação de um *checkpoint* (Chandy & Haas, 1983). O algoritmo para encontrar *deadlocks* foge do escopo do trabalho, uma vez que foi utilizado o algoritmo de *Termination Detection* para determinação de estados seguros.

#### **2.4.4 Detecção de Terminação**

Um sistema distribuído pode ser representado simplesmente por um conjunto de processos que se comunicam através de mensagens. Detectar que uma computação terminou não é trivial, e deve se considerar os seguintes fatores:

- 1 – Os atrasos (*delays*) são arbitrários, porém finitos.
- 2 – Os canais de comunicação adotam um modelo assíncrono de troca de mensagens

3 – As mensagens não são necessariamente entregues na ordem em que foram enviadas, ou seja, os canais de comunicação não necessitam ser FIFO.

4 – O *buffer* de envio e recebimento de mensagens é considerado ser infinito

5 – Um processo nunca espera que uma mensagem enviada seja recebida, para assim poder enviar outra mensagem.

O fato de este algoritmo trabalhar com canais não FIFO, além das outras características mostradas acima, torna este algoritmo muito importante para detecção de predicados globais em sistemas distribuídos que implementam este tipo de canal.

O conceito básico de Terminação de Computação é no entanto simples: “uma computação é globalmente finalizada se para cada processo, ele está localmente finalizado, e não existe mensagem em trânsito” (Dhamdhere et al., 1997). Um processo está localmente terminado se ele terminou sua computação e não a reiniciará até que receba uma mensagem. Este comportamento é intrínseco a este algoritmo. Ele será utilizado na implementação do algoritmo para o DCB, porém com alguma adaptação que será apresentada no Capítulo 4.

Detecção de Terminação é utilizada para determinar quando um sistema é globalmente finalizado (Mattern, 1987; Dhamdhere et al., 1997).

## **2.5 Distributed Co-simulation Backbone**

O DCB é uma arquitetura de simulação proposta em (Mello, 2005). O propósito geral do DCB é fornecer uma estrutura que permita realizar simulação híbrida de modelos heterogêneos.

No DCB os federados comunicam com a federação através do seu *Gateway*. Como dito anteriormente, o *Gateway* é o responsável pela comunicação entre o federado e a federação, ou seja, ele é a interface entre eles.

Cada federação pode ser composta por N elementos, porém cada elemento só pode fazer parte de uma federação (Wagner, 2003). O fato de existir um *gateway* para cada federado destaca um fator importante de distribuição do DCB: não são violados quesitos de propriedade intelectual, não é necessário alterar o código fonte do federado, e nenhum federado acessa informações internas do DCB (Wagner, 2003). Sendo assim, são necessários somente alguns ajustes entre o *gateway* e o federado para que possa ser realizada a comunicação do federado com o DCB.

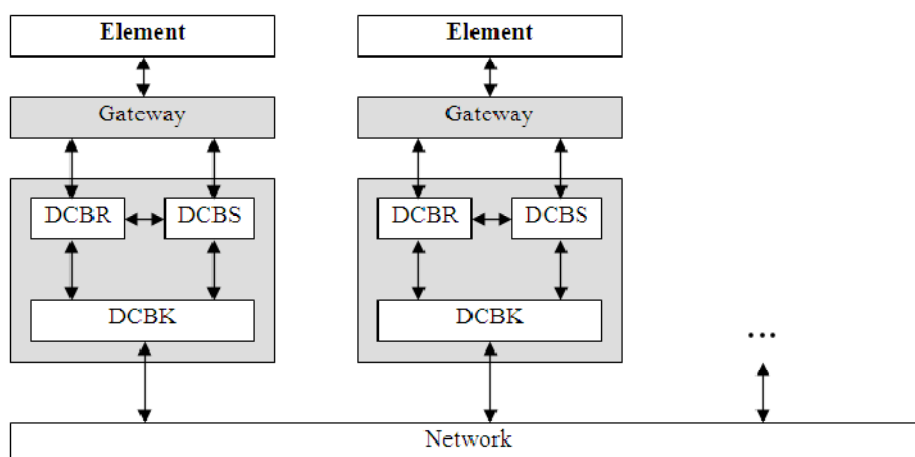


Figura 8 Visão geral do DCB

Fonte: Mello (2005)

Na Figura 8 pode ser observada a estrutura do DCB de forma geral. Ainda na Figura 8, observa-se que o DCB é constituído por três módulos: o *DCB Sender* - DCBS, o *DCB Receiver* - DCBR e o *DCB Kernel* - DCBK, ou DCB Expedidor, DCB Receptor e Núcleo do DCB, respectivamente.

O DCBR gerencia mensagens recebidas de outros federados, remotos ou locais. O DCBR decodifica os pacotes recebidos e participa das atividades de

gerenciamento do tempo de simulação. O DCBS gerencia as mensagens emitidas pelo federado que ele representa. O DCBS e o DCBR mantêm o gerenciamento do tempo virtual local (LVT – *Local Virtual Time*) e do modo de sincronização utilizado por cada elemento para cooperar com a federação (Wagner, 2003).

Cada elemento participante da simulação no DCB é representado por uma interface gráfica denominada *Chat*. Cada um dos seis elementos possui uma interface própria. Cada interface, para cada elemento, é denominada *ChatX*, onde X é um número que identifica o elemento. Cada elemento possui um *Gateway*, que é a interface de comunicação do elemento com o DCB. Um *Gateway* de um elemento, é denominado *GatewayX*, similarmente ao nome dado aos *chats*.

O DCB possui seis elementos, *Chat4*, *Chat5*, *Chat6*, *Chat7*, *Chat8* e *Chat9*, cada um representado pelo seu *Gateway*. Os elementos *Chat6*, *Chat7* e *Chat8* são elementos assíncronos. O *Chat5* é um elemento síncrono e os elementos *Chat4* e *Chat9* são *notimed* (Carvalho, 2009).

Cada *Gateway* no DCB é representado pelo seu Federado na Federação. Cada Federado é uma *thread*, ou executa sobre uma *thread*.

No EF - Embaixador do Federado há um *array* de mensagens, denominado *InputAttributeQueue*. Esta estrutura é responsável por armazenar as mensagens que chegam ao federado e que serão executadas no seu devido tempo.

O elemento *Chat4* não existia até o desenvolvimento deste trabalho, e será apresentado com mais detalhes na Seção 4.

### **2.5.1 Sincronização no DCB**

O DCB faz a sincronização das mensagens através do tempo de evento

(*timestamp*), ou seja, cada mensagem deve ocorrer exatamente no tempo demarcado. A ordem dos eventos externos é controlada por um tempo global (GVT – *Global Virtual Time*). Por exemplo, um PL(A) com LVT igual a 10 deseja executar um evento interno com tempo de evento igual a 12. Contudo, um PL(B) está preparando uma solicitação ao PL(A) para que ele execute um evento no tempo 11. Um algoritmo de sincronização conservadora deve garantir que o PL(A) execute o evento no tempo 12 somente depois que existam garantias de que nenhum outro PL irá solicitar ao PL(A) a execução de eventos num tempo menor que 12, como pretendido por PL(B) neste exemplo (Mello, 2005).

No DCB atualmente é possível cooperar elementos síncronos e assíncronos. Porém existem algumas regras para sincronização desses elementos. Um elemento cuja abordagem sobre o tempo é conservadora só pode enviar mensagem a outro elemento, caso este tempo (*timestamp*) seja maior do que o GVT. Caso contrário ocorreria violação de Causalidade Local (LLC – *Local Causality Constraint*). Uma LLC ocorre exatamente quando um elemento deseja executar uma tarefa num tempo anterior ao GVT. Na abordagem assíncrona no DCB, é permitido a um federado enviar e receber mensagens num tempo futuro ou passado ao GVT, podendo inclusive ser trocadas mensagens em tempos passados ao seu LVT. Na abordagem *no-timed* a ordem de execução das mensagens é organizada em consequência das chegada delas, ou seja, a mensagem é executada no momento em que ela é recebida (Mello, 2005).

Para realizar a sincronização dos elementos no DCB, considerando a simulação de elementos otimistas, foram implementadas soluções que aprimoram sua execução. Estas soluções são: Salvamento de *Checkpoints*, Mensagens Nulas, Anti-Mensagens e *Rollback*. No corpo deste trabalho será apresentada uma nova solução, que determinará instantes de tempo consistentes para a determinação de *checkpoints*.

### 2.5.2 Anti-Mensagens

O DCB é dotado de um sistema de anti-mensagens que é utilizado nos processos de *rollback* e *recovery*. No DCB uma anti-mensagem possui estrutura semelhante a outras mensagens quaisquer, com algumas propriedades que a diferem. Uma anti-mensagem no DCB possui a seguinte estrutura:

- *FederationSource*: representa o número da federação de onde a anti-mensagem saiu;
- *FederateSource*: representa o número do elemento que disparou a anti-mensagem;
- *FederationDestination*: representa a federação destino;
- *FederateDestination*: representa o elemento destino;
- *AttributeID*: representa o número do atributo de destino, que em todos elementos otimistas corresponde a 444.3;
- *Value*: valor da mensagem, ou seja o corpo do texto;
- LVT: *timestamp* para o qual o federado deve retornar;
- *Operation*: vem intitulada *AntiMessage*;

Uma anti-mensagem é montada de acordo com esta estrutura e é enviada, quando necessário, para a fila de mensagens e colocada na primeira posição. Sendo assim, assim que a mensagem entra na fila, ela já é retirada e enviada aos elementos destinos através do *Gateway* de cada elemento.

O *Gateway* recebe esta mensagem, identificada pelo valor do seu atributo, 444.3. Este valor foi escolhido para todos elementos otimistas no DCB, uma vez que só eles podem retornar no tempo. Logo ao receber a mensagem com este atributo, é iniciado o processo de *rollback* (Carvalho, 2009).

### 2.5.3 Rollback Recovery

No DCB o processo de *rollback* é realizado pelos elementos DCBR (DCBReceiver) e DCBS (DCBSender). Os métodos destes elementos são acessados pelo *Gateway*. Nos módulos DCBR e DCBS são armazenadas as mensagens enviadas e recebidas pelos elementos, além de que o DCBS é quem identifica as mensagens órfãs e realiza o disparo de anti-mensagens.

O processo de *rollback* é iniciado pausando a evolução do tempo nos elementos que participarão do processo. É identificado para cada elemento qual o *checkpoint* mais atual. Este é escolhido e o processo de *rollback* é realizado até o tempo (LVT) deste *checkpoint*. Do tempo atual do início do processo de *rollback* até o tempo deste *checkpoint* são descartadas, caso existam, todas mensagens recebidas por este processo. Em (Carvalho, 2009) é apresentado com detalhes como acontece o processo de *rollback* no DCB.

### 2.5.4 Checkpoint

Até o desenvolvimento deste trabalho havia no DCB uma solução para *checkpoints* não coordenados (*uncoordinated checkpoints*) (Carvalho, 2009). Entendeu-se que seria mais fácil a implementação do algoritmo, além de produzir menor *overhead* por troca de mensagens.

O checkpoint não coordenado torna o sistema passível a situações de efeito-dominó. Porém uma grande vantagem dessa abordagem é exatamente o baixo custo operacional, no que tange ao tempo da simulação (Elnozah et al., 2002).

Por questões de sincronização, no DCB todos os elementos devem determinar um *checkpoint* logo no início da simulação, no tempo 0. Este *checkpoint* serve para demarcar o início da simulação, e impede que a linha de

simulação precise ser reiniciada, inclusive com as propriedades de sincronização, caso o pior caso aconteça, um efeito dominó.

No DCB, como os *checkpoints* foram implementados de forma não coordenada, cada elemento implementava duas regras para determinação de seus *checkpoints*: a cada 5000 de tempo sem execução de um evento, ou a cada 10 mensagens trocadas, um *checkpoint* era determinado.

Este algoritmo funcionava perfeitamente para a determinação dos *checkpoints*. No entanto estes *checkpoints* poderiam se tornar inúteis para a linha de simulação do DCB uma vez que não eram consideradas as existências de mensagens órfãs e mensagens perdidas nos canais do DCB.

Um prejuízo destes *checkpoints* no DCB é a geração de vários deles inúteis. Caso algum algoritmo de *Garbage Collection* seja invocado para a eliminação destes *checkpoints*, um alto *overhead* seria gerado, tornando lenta a linha de simulação.



### 3 METODOLOGIA

A pesquisa deste trabalho quanto à sua natureza é classificada como tecnológica, uma vez que o conhecimento adquirido durante seu estudo foi utilizado para a implementação de um novo elemento, aprimorando o funcionamento do DCB. Sobre aspecto semelhante, ao se criar um novo elemento, o trabalho possui como objetivo, uma pesquisa aplicada, uma vez que buscou a solução para um problema inerente à simulação distribuída, identificando estados seguros para a simulação. Quanto aos procedimentos, o trabalho é uma pesquisa experimental, ao ser desenvolvido em laboratório, provendo a descoberta de novos métodos, tais como adaptação de algoritmos que permitiram alcançar uma nova tecnologia. Como laboratório, entende-se sendo um ambiente onde foi possível experimentar, testar e alterar variáveis que faziam parte do sistema com o fim de alcançar o objetivo proposto.

Inicialmente foi feito o levantamento bibliográfico sobre simulação, a partir de leitura de artigos e textos que faziam referência à simulação, e suas variantes. Em seqüência foram estudados algoritmos para detecção de estados consistentes em simulação. Após levantamento bibliográfico, identificou-se o problema, e iniciou-se a implementação do mesmo. A escrita do algoritmo se baseou em estudos de outros autores, que tratavam de *Termination Detection* para identificação de estados consistentes em simulação distribuída. Por fim, foi realizado um estudo de caso para apresentar o sistema identificando os estados seguros conforme proposto.

## 4 RESULTADOS E DISCUSSÃO

Neste capítulo serão apresentados os resultados do trabalho. Foi realizada análise e implementação de um algoritmo que permitiu a Identificação de Estados Seguros em Simulação Distribuída no DCB.

Para alcance dos objetivos foi realizado um estudo sobre o comportamento dos *checkpoints* no DCB, e verificado qual impacto seria causado ao aplicar um algoritmo de detecção de término de computação para identificar estados seguros na linha de simulação.

A Seção 4.1 apresenta um estudo realizado para verificar quais problemas os *checkpoints* determinados em estados inconsistentes geravam para o DCB.

A Seção 4.2 apresenta o impacto dos *checkpoints* não coordenados para a identificação dos estados seguros nos DCB.

A Seção 4.3 explica a necessidade de implementação de um novo sistema de *checkpoints* coordenados, em detrimento da manutenção do algoritmo anterior, que era não coordenado. Este novo algoritmo deu ao DCB um comportamento diferente quanto à determinação dos seus *checkpoints*. A maior vantagem deste novo algoritmo é a determinação de *checkpoints* somente em tempos estipulados pela saída do algoritmo. Eliminou-se aqueles *checkpoints* que poderiam se tornar inúteis. Evitou-se um excesso de *overhead* a nível de processamento e comunicação, uma vez que os elementos não estão mais livres para determinar *checkpoints* conforme sua própria especificação, e menos mensagens inúteis serão enviadas pelos elementos.

A Sub-Seção 4.3.1 apresenta o algoritmo que foi base para implementação do algoritmo de identificação de estados seguros no DCB.

A Sub-Seção 4.3.2 mostra as adaptações que foram realizadas nos algoritmos estudados, para o propósito do DCB.

Em seqüência, na Sub-Seção 4.4.4 entendeu-se como relevante a criação de um novo elemento, a partir da concepção de um novo *Gateway*. Este novo *Gateway*, representado pelo seu *chat* (interface de comunicação no estudo de caso), trabalha de forma semelhante a todos outros elementos que já existiam no DCB. O tipo deste elemento foi escolhido ser *notimed* por ser um elemento de fácil criação, com isso pode-se focar mais tempo na concepção do algoritmo. A diferença principal deste elemento para os outros, é que este implementa as regras para determinação dos estados seguros, a partir de um algoritmo específico. A implementação deste algoritmo em um elemento separado foi feito por se desejar um desacoplamento dos elementos quanto a suas propriedades de simulação, como sincronização e controle de tempo. Sendo assim, o DCB passa a requerer dois elementos como obrigatórios para o sistema de simulação: o *Gateway5*, que trata da sincronização, e o *Gateway4* que trata da identificação dos estados seguros.

Na Sub-Seção 4.3.4 é definida a nova funcionalidade do *FedGVT*, que é um federado do DCB que implementa o GVT e auxilia na comunicação entre os elementos, tratando as mensagens trocadas entre eles.

Na Seção 4.4 é apresentado o estudo de caso, onde os simuladores são representados por um sistema de *Chat*. Cada *Chat* no DCB representa um simulador, que possui suas próprias características.

#### **4.1 Problema dos Estados Inconsistentes no DCB**

Um estado consistente é um instante de tempo em que se pode determinar um *checkpoint* seguro, ou seja, este *checkpoint* deixará a linha de simulação segura para situações de *rollback* e *recovery*.

Determinar *checkpoints*, apesar de ser uma tarefa trivial, não é suficiente para garantir que a linha de simulação continuará consistente em casos de

ocorrência de falhas, perdas de informações e conseqüente necessidade de retorno no tempo. Portanto simplesmente demarcar *checkpoints* sem um estudo de consistência pode ser considerada uma tarefa inútil em sistemas distribuídos.

Como passo inicial para aprimoramento da política de *checkpoints* no DCB foi necessário um estudo profundo sobre a forma como eles são determinados. A partir deste estudo foram verificados quesitos como importância dos *checkpoints* para o DCB, *overhead* de processamento, comunicação e espaço de armazenamento que os *checkpoints* geravam, complexidade dos mesmos no que tange inclusão de novos elementos futuros, e a manutenção da consistência do sistema.

Os *checkpoints* não coordenados davam alta flexibilidade para cada elemento definir sua política de salvamento de *checkpoints*. Esta qualidade gerava, no entanto, uma grande quantidade de *checkpoints* inúteis. A Figura 9 apresenta um *checkpoint* inútil em um elemento. Como demonstrado, o *checkpoint* em cor vermelha, determinado pelo terceiro processo (de cima para baixo), é um *checkpoint* inútil. Neste caso o *checkpoint* não foi útil para a simulação, a ponto de salvar um comportamento que impedisse a perda da mensagem *m4*. Este *checkpoint* em nada contribuiu para a simulação, e ele deve ser evitado.

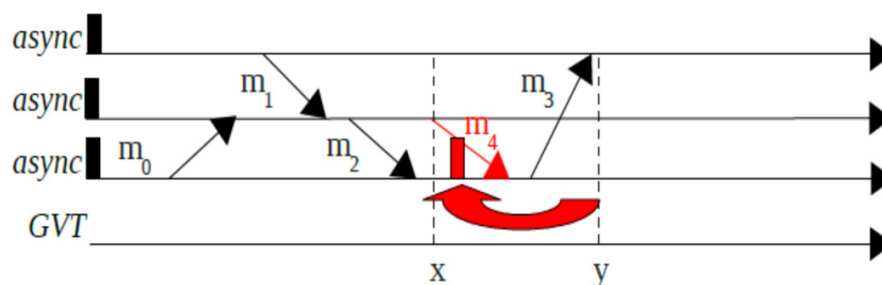


Figura 9 Exemplo de *checkpoint* inútil

Fonte: Carvalho (2009)

A existência de mensagens em trânsito é um fato comum em simulação distribuída, exigindo, portanto, uma grande atenção sobre seu comportamento. Esta atenção especial se reflete no fato de que salvar *checkpoints* quando existem mensagens em trânsito no sistema pode gerar mensagens órfãs e/ou perdidas. A Figura 10 apresenta duas situações onde o salvamento de *checkpoints* ocorreu quando uma mensagem estava em trânsito. No exemplo, o primeiro *checkpoint* foi determinado antes do processo P2 receber uma mensagem. Este *checkpoint* pouco contribui para consistência do sistema, já que ele salva uma mensagem que poderá se tornar perdida caso o processo P2 retorne no tempo. Já o segundo *checkpoint* salva uma mensagem que poderá se tornar órfã, caso o processo P3 retorne no tempo até o início da simulação.

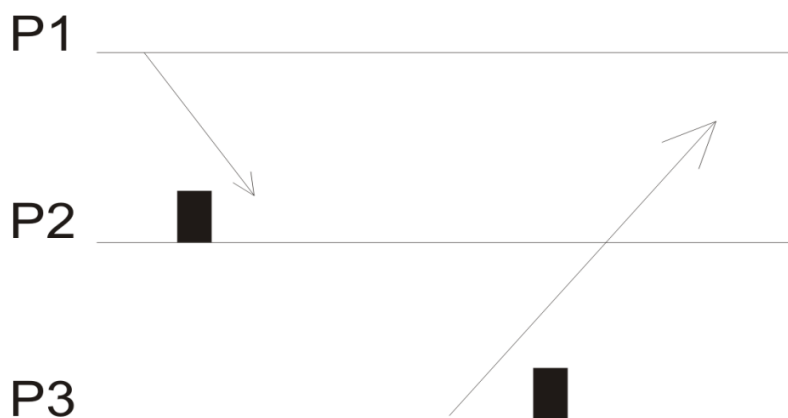


Figura 10 Mensagem em trânsito no sistema e criação de *checkpoints* inúteis

Definida esta relação de causa e consequência, o problema se limitou em determinar os *checkpoints* somente quando não existissem mensagens em trânsito no sistema. Esta regra se torna extremamente válida a partir do momento

em que não seriam necessárias novas mensagens, ou verificações para confirmar a não existência de mensagens órfãs/perdidas.

#### **4.2 O Impacto dos *Checkpoints* Não Coordenados para Determinação de Estados Seguros**

Os *checkpoints* não coordenados causavam um grande impacto no estudo de estados seguros, uma vez que a não coordenação dificultava o controle sobre a existência de mensagens em trânsito nos canais de comunicação do DCB.

Como cada elemento era responsável por sua política de determinação de *checkpoints*, um algoritmo seria necessário para validação de tal situação, em cada elemento. Mesmo que cada elemento identificasse que seu canal de comunicação era livre de mensagens em trânsito, globalmente não se poderia garantir que aquele *checkpoint* era seguro para a linha de simulação como um todo.

Pode-se concluir que mesmo que um *checkpoint* local era “seguro” para o processo, globalmente aquele *checkpoint* poderia continuar inútil, sobre o ponto de vista que os elementos interagem diretamente, e uma operação de *rollback* envolve indistintamente todos elementos da simulação.

Esta execução poderia ter um custo alto, uma vez que o *delay* na rede de comunicação causaria certa ineficiência sobre o ponto de vista de que no momento da análise de consistência, novas mensagens poderiam ser enviadas e recebidas pelos canais de comunicação, forçando a execução do algoritmo inúmeras vezes. Uma pausa na linha de execução poderia contornar este problema, porém em um sistema distribuído onde se busca representar a realidade isto não é desejável.

Como um módulo centralizador, que controlasse as mensagens nos canais de entrada de cada elemento, não existia, seria necessária a implementação de um algoritmo que verificasse a todo instante a presença destas mensagens.

O objetivo do trabalho se limitou então em identificar quando, globalmente, cada instante de tempo, em cada elemento, era seguro para a simulação global.

### **4.3 Checkpoints Coordenados e Identificação de Estados Seguros no DCB**

As vantagens do *checkpoint* não coordenado são conhecidas, porém no desenvolvimento deste trabalho foi identificada uma solução mais eficaz de *checkpoints* coordenados, que permitiam um melhor desempenho na identificação de instantes de tempo seguros na linha de simulação, e melhor aproveitamento dos *checkpoints*.

Os *checkpoints* coordenados, em sua definição básica, exigem a existência de um módulo central que identifica quais instantes de tempo são consistentes para a determinação dos *checkpoints*.

A primeira vantagem óbvia que o DCB ganhou com este tipo de *checkpoint* foi a determinação de somente *checkpoints* que seriam úteis para a linha de simulação. O *overhead* gasto com *Garbage Collection* dos *checkpoints* inúteis seria eliminado em sua totalidade.

A segunda e não menos importante vantagem do *checkpoint* coordenado, é a eliminação do *efeito-dominó*.

Agregando estas duas vantagens deste tipo de *checkpoint* ao DCB, ficou relativamente simples determinar quais instantes de tempo seriam seguros para determinar um *checkpoint* em cada elemento, e automaticamente verificar que este conjunto de *checkpoints* formariam um estado global consistente.

Uma desvantagem prevista nos *checkpoints* coordenados é o *overhead* causado nos canais de comunicação, uma vez que cada elemento, individualmente, informa ao módulo centralizador se ele possui ou não mensagens em trânsito em seu canal de entrada. Adiante será exposto qual estratégia foi tomada para minimizar este impacto na execução do algoritmo.

#### **4.3.1 O algoritmo de *Termination Detection* na Identificação de Estados Seguros**

O algoritmo de *Termination Detection* baseado na idéia de Lai e Yang (1987) foi base para o desenvolvimento do algoritmo de determinação de estados seguros no DCB. Na verdade, o algoritmo do DCB é uma síntese de conceitos, e procura absorver o que de melhor há nos algoritmos de Lai e Yang (1987) e Chandy & Lamport (1985), além dos algoritmos derivados destes, como o próprio conceito de *Termination Detection*.

O conceito de *Termination Detection* define que uma computação é globalmente finalizada, se para cada processo contido na computação, ele é localmente finalizado e não existe mensagem em trânsito (Dhamdhere et al., 1997).

Para o DCB, cada interação entre processos (conjunto de canais de entrada e saída entre eles) é considerada uma computação a parte. Cada processo comunica com outro numa relação de 1 para N, sendo que cada processo possui um canal de entrada de onde todas mensagens são recebidas, e N canais de saída, um para cada destinatário, por onde as mensagens são enviadas. Veja que esta definição é algo que podemos chamar de virtual, uma vez que um canal de saída de A para B, é o mesmo canal de saída de C para B, uma vez que o canal de entrada de B é único. Conclui-se que cada um desses canais é tido como único em cada processo, uma vez que nenhum sabe da existência do outro.



Se determinado processo não possui mensagens no seu canal de entrada, logo todos outros processos não possuem mensagens no seu canal de saída, com destino a este elemento. Assim sendo, ele está ocioso, e garante para o módulo central não haver mensagens em trânsito em seu canal de entrada. Neste momento a computação terminou para este elemento, e conforme determinado pelo conceito de *Termination Detection*, ele não pode mais enviar mensagens até que receba uma notificação externa solicitando seu reinício na computação.

Generalizando, um conjunto de  $N$  processos, sendo  $N > 1$  até  $K$  ( $N_0, N_1, \dots, N_k$ ), cada processo envia ao módulo central informando a situação do seu canal de entrada.

Em certo momento onde todos elementos não possuem mensagens nos seus canais de entrada, conclui-se que o sistema é livre de mensagens em trânsito. Este momento é seguro para os elementos determinarem *checkpoints* locais, e uma vez que todos *checkpoints* são determinados seguramente, pode-se formar então um estado global denominado consistente.

#### 4.3.2 Adaptações para alcance dos objetivos

A primeira adaptação do conceito de *Termination Detection* é que o sistema possui infinitas computações, uma vez que é desejado que a simulação possa ser executada em um intervalo finito de tempo, porém infinita quanto ao número de computações realizadas.

Para o algoritmo funcionar, é definido no DCB que quando uma computação é terminada, um estado global consistente é determinado, e subseqüentemente, uma nova computação é iniciada, sem a perda de quaisquer informações. Neste momento o sistema pode ser finalizado, se assim for desejado, a simulação poderá ser paralisada, e todos os elementos poderão ter

seus históricos preservados. A Figura 10 representa um sistema com N computações terminadas.

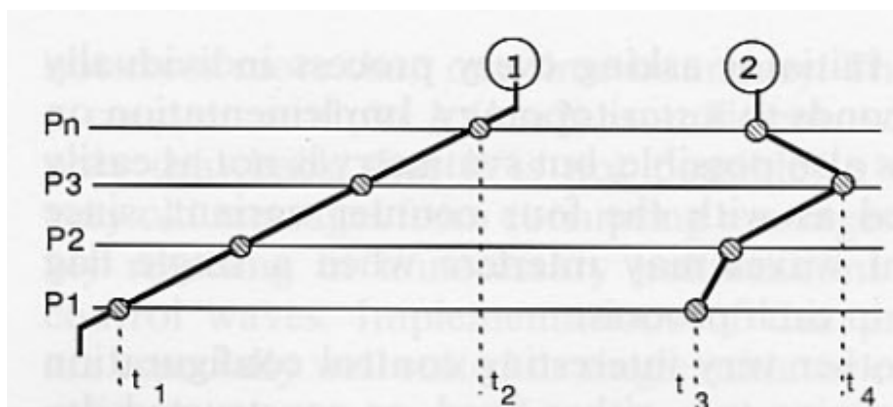


Figura 11 Terminação de computação e reinício da computação

Fonte: Mattern (1987)

Na Figura 10 pode-se observar quatro instantes de tempo,  $t_1$ ,  $t_2$ ,  $t_3$  e  $t_4$ .  $t_1$  representa o início de uma computação, que foi finalizada em  $t_2$ . No instante de tempo de  $t_2$  todos os processos determinam seu *checkpoint* local. A computação em  $t_3$  é representada como um novo início, e  $t_4$  representa seu término. Em  $t_2$  e em  $t_4$  tem-se dois instantes de tempo onde não existem mensagens em trânsito, logo conclui-se que nestes mesmos instantes, o sistema é globalmente consistente, e todos elementos podem determinar instantes locais de forma segura.

Outra adaptação realizada foi no que tange a coordenação dos *checkpoints*. Lai e Yang definem que os *checkpoints* podem ser não coordenados, já para o DCB foi escolhido que os *checkpoints* são agora coordenados.

A otimização realizada sobre o aspecto de trocas de mensagens entre os processos descentralizados e o processo central coordenados foi feita da seguinte

maneira. Cada processo envia uma, e somente uma mensagem ao módulo analisador centralizado para informar seu status atual. Esta mensagem é enviada a cada alteração de status no seu canal de entrada. Um elemento que está no tempo 1000 seu canal de entrada possui tamanho igual zero. Se no instante de tempo igual a 2000 ele passa a ter uma mensagem no seu canal de entrada, outra mensagem caracterizada é enviada ao módulo centralizado informando conter mensagens em trânsito no seu canal de entrada. A mensagem por outro lado só será executada no tempo igual a 5000. Quando o processo alcançar o tempo 3000, ele recebe outra mensagem de outro processo que deverá ser executada no tempo 7000. Ao alcançar o tempo 5000 a primeira mensagem será executada, porém seu status não mudou para vazio, uma vez que o tamanho do canal de entrada é maior que zero. Ao alcançar o tempo 7000, a mensagem é executada e seu canal de entrada possui zero mensagens.

Pode-se ver que o status é informado ao processo centralizado somente uma vez a cada alteração de status. Esta é uma otimização com o objetivo de minimizar a troca de mensagens entre os processos e o processo analisador central.

#### **4.3.3 Gateway 4, o Termination Analyzer**

O *Gateway4*, cuja interface de comunicação é o *Chat4*, foi desenvolvido neste trabalho com o propósito de colaborar na identificação de estados seguros no DCB. *Gateway4* e *Chat4* representam o mesmo federado e são tratados de forma semelhante e direta no texto que segue.

Toda mensagem de status dos processos é encaminhada ao *Gateway4*, que fica responsável por verificar quando ou não existem mensagens em trânsito no sistema.

O *array* de mensagens, denominado *InputAttributeQueue*, é uma *array* ordenado pelos *timestamps* das mensagens, é único de cada elemento, e representa o canal de entrada de cada processo. Para determinar se há mensagens em trânsito no seu canal de entrada, basta verificar se o tamanho deste *array* é maior que zero (Figura 12). Se num instante de tempo se verificar que uma mensagem deve ser executada no seu federado, ela é encaminhada para execução, e o *InputAttributeQueue* é decrementado.

```

1 //contElemIAQ = número de mensagens no canal de entrada do elemento, a serem executadas
2 //enviou = booleana que controla se o elemento já enviou mensagem de status
3
4 if (contElemIAQ == 0 && !enviou){
5     sendTerminationControlMessage("444.11", "FALSE", "0");
6     enviou = true;
7 }else if (contElemIAQ > 0 && enviou){
8     sendTerminationControlMessage("444.11", "TRUE", "0");
9     enviou = false;
10 }
11

```

Figura 12 Buffer que controla mensagens em trânsito no canal de entrada do processo

A *thread* descrita na Figura 12 representa o algoritmo que envia uma mensagem ao elemento centralizador informando haver ou não mensagens em trânsito em seu canal de entrada. Esta *thread* é implementada por todos federados. Na Figura 13 pode ser observada a implementação desta mensagem por um federado qualquer, com destino ao federado 4.

A variável denominada *enviou* controla o envio das mensagens, para que somente a cada alteração de estado, uma mensagem seja enviada.

```

<ATTRIBUTE id="444.11" name="termination_send" type="String">
  <DESTINATION federationid="1" federateid="4" attribute="444.12" />
</ATTRIBUTE>

```

Figura 13 Mensagem descrita pelo federado 8, com destino ao federado 4

```

1  A0 = getAttributeReceived("444.12");
2  if (A0 != null) {
3      String msg = "";
4      //array que controla mensagem em transito no sistema
5      controlTransitMessages(A0.Source, ToBoolean(A0.Value));
6
7      /*envia mensagem informando que todos elementos podem
8      determinar checkpoint no GVT informado*/
9      sendTerminationControlMessage("444.13", "checkpoint", GVT);
10 }

```

Figura 14 Código do *Gateway4* que detecta Terminação de Computação

Quando o sistema possui mensagens em trânsito, o *Gateway4* permanece em análise, até definir que não há mensagens em trânsito no sistema. Neste momento uma mensagem é encaminhada ao EDCB que cria uma mensagem padrão, envia através do DCBSender, e pela codificação informada, encaminha aos elementos (que a recebe através do DCBReceiver) uma requisição de *checkpoint*.

A chamada ao método do EDCB, *sendTerminationControlMessage*, apresentada na imagem anterior, representa a identificação de uma terminação, ou uma *Termination Detection*. Nele, o *Gateway4* envia uma mensagem com atributo igual a *444.13*. Este atributo identifica uma mensagem que será enviada ao *FedGVT*, que a codificará, e redirecionará a todos elementos participantes da simulação, informando a necessidade de determinar um *checkpoint*. Este pedaço de código é crucial para a identificação dos estados seguros do DCB.

#### 4.3.4 FedGVT

O FedGVT tinha como foco principal gerenciar o GVT da Federação e sincronização dos elementos através de mensagens. Além dessas atribuições, o envio de anti-mensagens era obrigação deste federado.

Devido às novas necessidades, ficou no encargo do *FedGVT* o repasse da mensagem de status dos processos ao *Gateway 4*, além de remeter aos processos a mensagem originada deste *gateway*, informando da possibilidade de se definir um *checkpoint*.

Dois tipos de mensagens (Figura 15) foram criadas na descrição do FedGVT.

1. Mensagem 444.12, que identifica “*termination-detection*”, representada pelo atributo 444.11
2. Mensagem 444.14, que identifica “*checkpoint-requisition*”, representada pelo atributo 444.13

```
15 <ATTRIBUTE id="444.11" name="port" type="String">
16   <DESTINATION federationid="1" federateid="4" attribute="444.12" />
17 </ATTRIBUTE>
18
19 <ATTRIBUTE id="444.13" name="port" type="String">
20   <DESTINATION federationid="1" federateid="4" attribute="444.14" />
21   <DESTINATION federationid="1" federateid="5" attribute="444.14" />
22   <DESTINATION federationid="1" federateid="6" attribute="444.14" />
23   <DESTINATION federationid="1" federateid="7" attribute="444.14" />
24   <DESTINATION federationid="1" federateid="8" attribute="444.14" />
25   <DESTINATION federationid="1" federateid="9" attribute="444.14" />
26 </ATTRIBUTE>
```

Figura 15 Novas mensagens de controle do *FedGVT*

A partir desta definição uma mensagem de status originada de qualquer processo com destino ao processo centralizador (*Gateway 4*), possui *AttributeID* igual *444.11*, e representa uma mensagem de controle de mensagem em trânsito. O *FedGVT* intermédia esta troca de mensagens, repassando-a ao *Gateway 4* com o *AttributeID* igual a *444.12*. O valor desta mensagem é por padrão “*TRUE*” ou “*FALSE*”. Caso o *Gateway 4* receba um valor igual a “*TRUE*” significa que algum processo remeteu um status informando que ele possui mensagens em trânsito. Por outro lado, um valor igual a “*FALSE*” algum processo informou que não possui mensagens em trânsito.

A mensagem cujo *Attribute ID = 444.13* representa uma mensagem de requisição de *checkpoint*, feita pelo *Gateway 4*, com destino a todos elementos da federação. Novamente o *FedGVT* intermédia a requisição, e repassa uma mensagem a cada federado, com *Attribute ID = 444.14*. Cada federado identifica a presença desta mensagem, e a repassa ao seu *Gateway*, para então determinar o *checkpoint*.

```

1 void attendCheckpointRequisition() {
2     switch (gVal) {
3         case 6: {
4             /*Se o array de checkpoints do elemento não
5              já possui um checkpoint naquele tempo informado*/
6             if (checkpointsArray.contains(LVT) == false)
7                 setCheckpoint(LVT);
8             break;
9         }
10    }

```

Figura 16 Atendimento de uma requisição de *checkpoint* enviada pelo *FedGVT*

O *Gateway 4* possui um *array* que controla cada status de cada processo. Se o valor do status é “*FALSE*”, ele retira o elemento do seu *array*, caso contrário acrescenta.

```
113
114 □ public static void controlTransitMessages (String name, boolean status){
115     if (status){
116         Gateway4.arrayTransit.add(name);
117     }else if (!status){
118         Gateway4.arrayTransit.remove(name);
119     }
120 }
```

Figura 17 Array que apresenta ao sistema a situação do mesmo, seguro ou inseguro

Este *array* serve somente para o *gateway* informar via interface gráfica quando, ou não, existem mensagens em trânsito.

Veja que a implementação desta regra em um novo elemento, no caso, o elemento 4, permite ao DCB a manutenção de uma propriedade muito importante na simulação distribuída: o desacoplamento. O desacoplamento no DCB permite que  $N$  elementos participem da simulação sem influenciar na identificação dos estados seguros, e já que o *Gateway4* não faz distinção de quem é o remetente da mensagem “*termination-detection*”, a identificação de uma computação terminada pode ser realizada sobre estes  $N$  elementos com garantia.

#### 4.4 Estudo de Caso

Para realização do estudo de caso para identificação de instantes de tempos seguros, e a determinação de um *checkpoint* seguro, foi utilizado o próprio ChatDCB.

O ChatDCB é fruto de um estudo realizado para implementação de *checkpoints* não coordenados e *rollback* no DCB (Carvalho, 2009). O chat é composto de elementos síncronos, assíncronos e *no-timed*. Os elementos assíncronos são representados pelos chats denominados Chat 6, Chat 7 e Chat 8.



O Chat 5 representa o elemento síncrono, e os chats Chat 9, e o Chat 4 (fruto do desenvolvimento deste trabalho) representam os elementos *no-timed*.

O elemento denominado Chat4 foi desenvolvido no escopo deste trabalho. Cada elemento ChatX é descrito pelo seu Gateway, que possui a denominação GatewayX, onde 'X' é o número que compõe o nome do elemento. Para este novo elemento foi determinado seu tipo como sendo *notimed*.

Foi realizado um estudo de caso onde identificou-se a alteração de status dos sistema, realizando a transição entre:

***Estado Seguro -> Estado Não Seguro -> Estado Seguro***

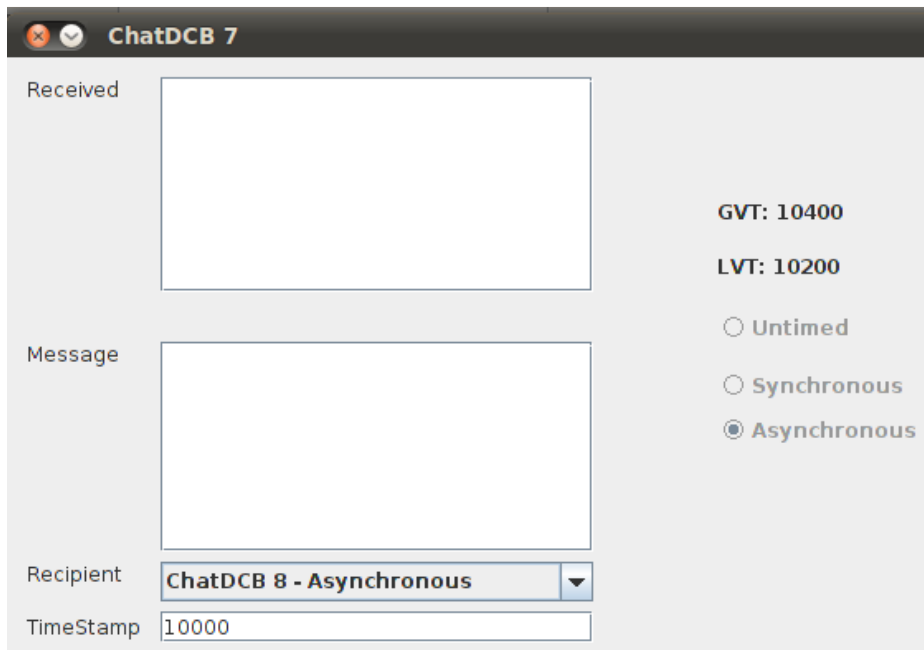
Na linha de tempo até o instante de tempo 9800 não existia mensagem em trânsito, como pode ser observado pela Figura 18.

```
Msg Transito = FALSE
Msg Transito = FALSE
Msg Transito = FALSE
Msg Transito = FALSE
Msg Transito = FALSE
TIMESTAMP DA MSG: 0 GVT
Redirecionou 4 para: 44
checkpoint: 9800
Msg Transito = TRUE
Msg Transito = TRUE
TIMESTAMP DA MSG: 0 GVT
Redirecionou 4 para: 44
checkpoint: 10000
Msg Transito = FALSE
Msg Transito = FALSE
Msg Transito = FALSE
```

Figura 18 Transição de Estados no DCB

O elemento da simulação representado pelo nome *Chat7* (Figura 19) envia uma mensagem, no seu LVT = 9800, ao elemento representado pelo *Chat8*

(Figura 20). Esta mensagem deve ser executada no tempo 10000 por este elemento conforme solicitado pelo *Chat7*.



The screenshot shows a window titled "ChatDCB 7" with a dark header bar. The main area is divided into several sections:

- Received:** A large empty rectangular box.
- Message:** A large empty rectangular box.
- Recipient:** A dropdown menu currently displaying "ChatDCB 8 - Asynchronous".
- TimeStamp:** A text input field containing the value "10000".
- Configuration:** On the right side, there are two labels: "GVT: 10400" and "LVT: 10200". Below these are three radio button options: "Untimed", "Synchronous", and "Asynchronous". The "Asynchronous" option is selected, indicated by a filled circle.

Figura 19 Envio de mensagem a um federado

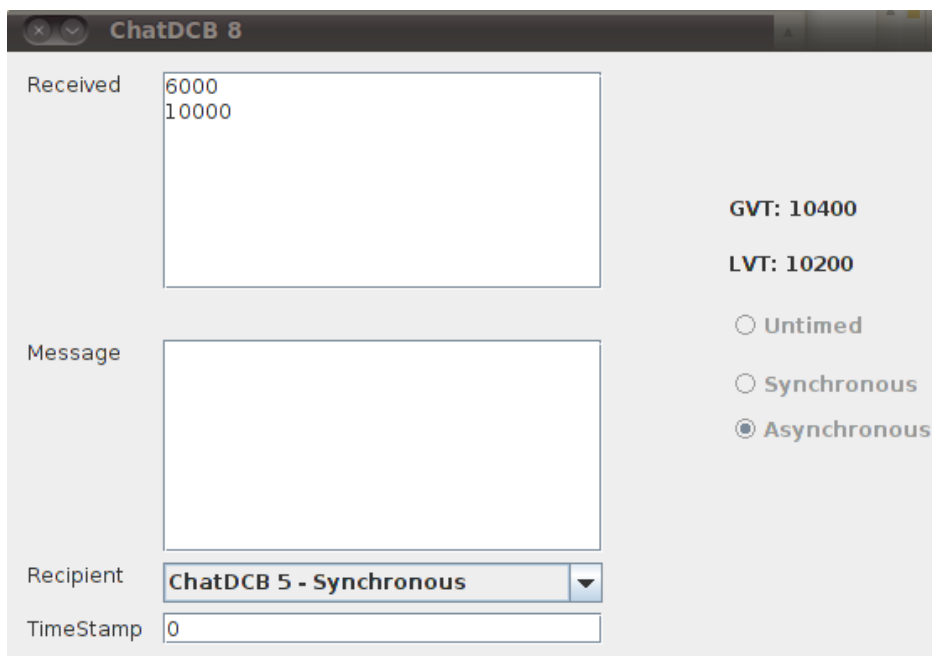


Figura 20 Mensagem recebida no tempo 10000

Como pode ser visto pelas figuras 19 e 20, o sistema realiza a transição entre estados, e identifica quando no tempo da simulação o sistema é seguro, informando se há mensagem em trânsito ou não, através das *flags TRUE* e *FALSE*. Um checkpoint para cada elemento é então determinado nos instantes de tempo seguro exatamente anterior ao envio da mensagem, e posterior ao recebimento da mesma.

Na Figura 21 um grafo apresenta o comportamento do sistema descrito anteriormente. Os *checkpoints* só são determinados de acordo com a transição entre estados pelos quais o sistema passa. Na mesma imagem, os quadrados azuis representam os *checkpoints*, as retas horizontais os processos, e as setas entre estas retas representam as trocas de mensagens.

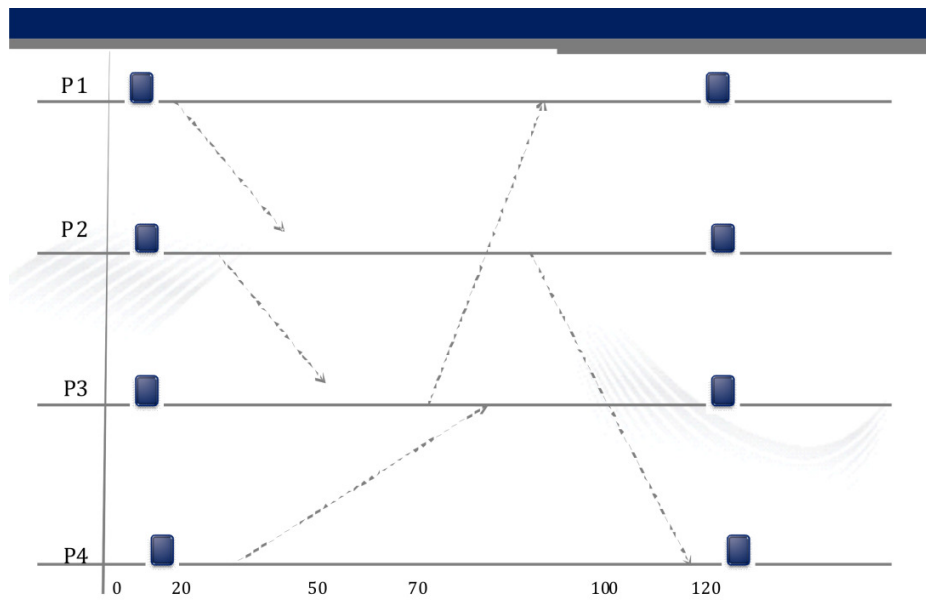


Figura 21 Sistema ejecutando algoritmo de Estados Seguros

#### 4.5 Estrutura do DCB

Com o desenvolvimento do trabalho, o DCB sofreu poucas alterações. Para explicar esta nova estrutura, foi criado um diagrama simplificado representado pela Figura 22 e Figura 23. Este diagrama foi simplificado para representar as classes mais importantes no desenvolvimento deste trabalho.

A classe *ApplicationDCB* centraliza as operações do DCB, e é responsável por criar instâncias de todas as outras classes. É a partir desta classe que todas as outras se comunicam.

O DCB através da instância *newDCB*, o objeto DCBS através da instância *NewDCBS* e o objeto DCBR através da instância *NewDCBR* são responsáveis pela comunicação entre os elementos da federação, e comunicação via rede ethernet ou localmente.

O EDCB representado pela instância *newEDCB*, faz envio de mensagens de controle, como anti-mensagens e requisição de *checkpoints*. Faz também armazenamento de mensagens enviadas, codificação da mensagem a ser repassada ao DCB para envio para os elementos e atualização do tempo dos elementos (LVT), de acordo com o tipo de cada destinatário.

A classe *EF* foi alterada neste trabalho, e possui uma *thread* denominada *ControleBufferInputMessage* que realiza controle das mensagens em trânsito.

A classe *Gateway* faz o redirecionamento de informações para os *Gateways* destinatários, além de receber requisições dos mesmos.

A classe *Gateway 4* trata do recebimento da identificação de estados seguros, e o envio da requisição de *checkpoint* para todos os elementos.

A classe *Chat* representa a interface de comunicação de cada elemento participante da simulação.

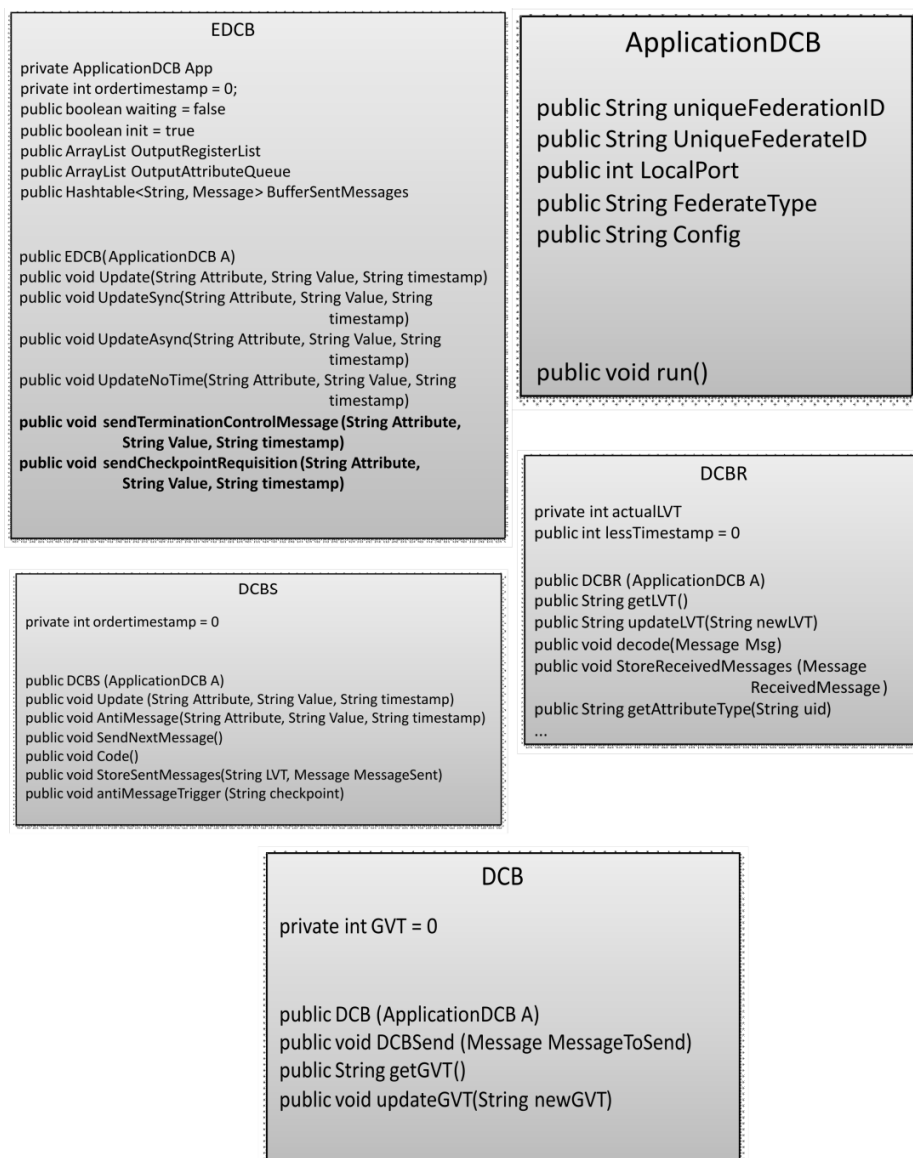


Figura 22 Elementos: EDCB, ApplicationDCB, DCBS, DCBR, DCB

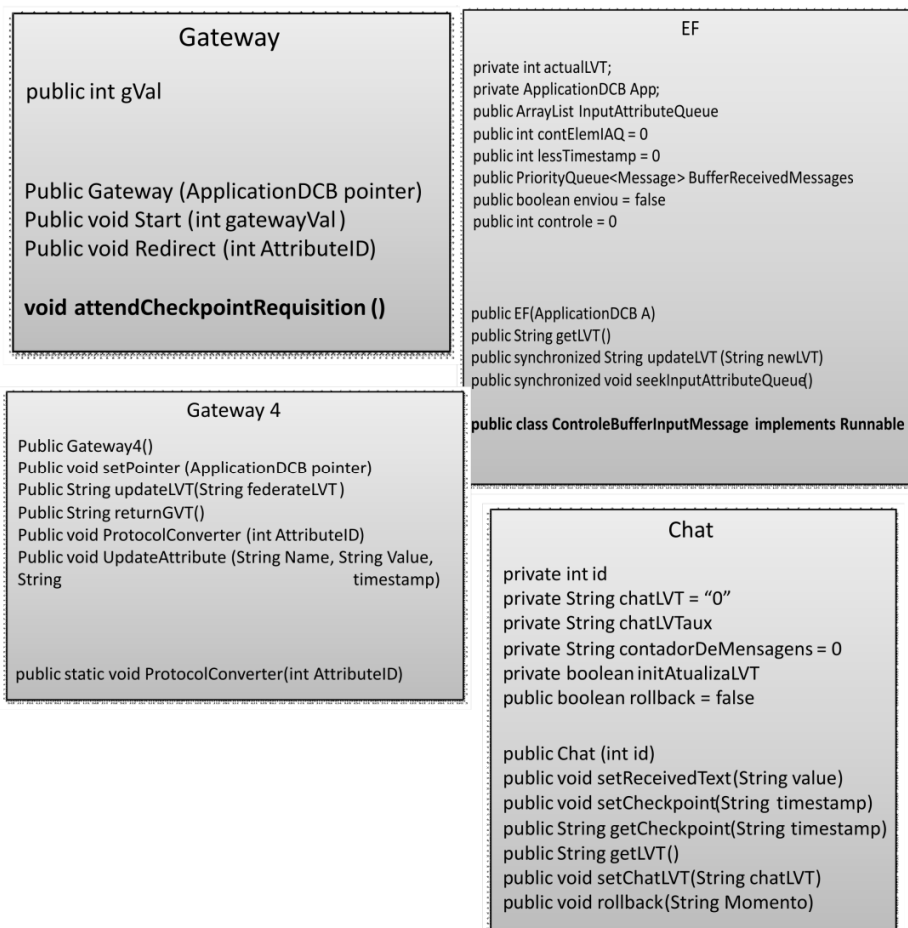


Figura 23 Elementos: Gateway, EF, Gateway 4 e Chat

## 5 CONCLUSÃO

Neste trabalho foi desenvolvido um mecanismo de detecção de estados globais seguros/consistentes para o DCB.

A Identificação de Estados Seguros é fundamental no escopo de simulação distribuída. A determinação de *checkpoints* só é completa se houver um estudo que garanta a consistência do sistema, mesmo que este sistema falhe, pare, seja interrompido por qualquer motivo, e necessite retornar no tempo, desfazendo ações, e posteriormente as refazendo.

Algumas questões importantes foram observadas durante a execução deste trabalho. Houve uma grande preocupação em evitar o *overhead* de mensagens nos canais de comunicação do DCB. Este problema foi sanado controlando a quantidade de mensagens de controle enviadas por cada elemento, ao elemento centralizador. Este elemento centralizador, aqui chamado de *Gateway4*, que se comunica através do *Chat4*, implementa a recepção das mensagens de controle, e conforme a verificação tráfego de mensagens no sistema, apresenta ao EDCB o *Termination Detection*, ou Fim da Computação. Com várias computações terminadas em certas faixas de tempo, o DCB pode identificar quando, globalmente, o sistema está em um estado global consistente.

Uma alteração importante neste trabalho foi a alteração da forma de determinação de *checkpoints*. Antes o DCB possuía um algoritmo de *checkpoints* não coordenados. Porém pelos motivos apresentados neste trabalho, foi necessária a alteração desta regra, para que o sistema passasse a identificar os estados consistentes, para posteriormente determinar os *checkpoints*.

Após realização de testes, tal como o estudo de caso apresentado, identificou-se que o sistema se comportou conforme especificado e passou a identificar estados seguros eficientemente em um sistema de simulação híbrido e distribuído.



### 5.1 Trabalhos Futuros

Um mecanismo interessante que poderia ser implementado no DCB, é o *Garbage Collector*. Mesmo que este trabalho tenha apresentado uma solução para redução de mensagens armazenadas pelos processos, em uma linha de simulação extensa, pode ser que uma grande quantidade de mensagens seja trocada, aqueles *checkpoints* antigos possam ser removidos. Um mecanismo de limpeza portanto auxiliaria na otimização do sistema quanto à quantidade de armazenamento utilizada.

Além do mecanismo de *Garbage Collector*, seria interessante a implementação de um mecanismo que calculasse o desempenho da linha de simulação. Os resultados seriam interessantes para medir tempos médios de simulações de acordo com diversos fatores, como quantidade de elementos em cooperação, distância geográfica, poder computacional dentre outros.

## REFERÊNCIAS

AMORY, A.; MORAES, F.; OLIVEIRA, L.; CALAZANS, N.; HESSEL, F. A Heterogeneous and Distributed Co-Simulation Environment, In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 15. 2002.

**Proceedings...** [S.l.:s.n.], 2002, p. 115-120.

BALDONI, R.; BREZINSKY, J.; HELARY, J.M.; MOSTEFAOUI, A.; RAYNAL, M. On Modeling Consistent *Checkpoints* and the Domino Effect in Distributed Systems. In: IEEE INFORMATICS CONFERENCE ON FUTURE TRENDS IN DISTRIBUTED COMPUTING SYSTEMS. France, 1995.

**Proceedings...** France: [s.n.] 1995. p. 314-323.

BALDONI, R.; HELARY, J. M.; RAYNAL, M. About State Recording in Asynchronous Computations. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 15., 1996. Philadelphia. **Proceedings...**

Philadelphia: [s.n.], 1996. p. 55.

BORSHCHEV, A.; KARPOV, Y.; KHARITONOV, V. Distributed simulation of hybrid systems with anylogic and HLA. **Future Generation Computer Systems**, v.18, n.6, p. 829-839, may. 2002.

BRUSCHI, S. M. **ASDA – Um Ambiente de Simulação Distribuída Automático**. 2002. 228 p. Tese (Doutorado em Ciências - Ciências de Computação e Matemática Computacional) – Universidade de São Paulo, São Carlos.

CALVIN, J. O.; WEATHERLY, R. The Introduction on High Level Architecture (HLA) Runtime Infrastructure (RTI). In: SICE 2003 ANNUAL CONFERENCE, 3., 2003, Fukui, Japan. **Proceedings ...** Fukui, Japan: [s.n.], 2003. p. i - lxix.

CARVALHO, F. M. M. **Controle de Checkpoints para Execução Otimista de Modelos Heterogêneos no DCB**. 2009. 44 p. Monografia (Graduação em Ciência da Computação) – Universidade Federal de Lavras, Lavras.

CHANDY, K. M.; HAAS, L. M. Distributed Deadlock Detection. **ACM Transactions on Computer Systems**, v. 1, n. 2, p. 144-156, may. 1983.

CHANDY, K. M.; LAMPORT, L. Distributed Snapshots Determining Global States of Distributed Systems. **ACM Transactions on Computer Systems**, v. 3, n. 1, p. 63-75, feb. 1985.

DAHMAN, J. S.; FUJIMOTO, R. M.; WEATHERLY, R. M. The Department of Defense High Level Architecture. In: WINTER SIMULATION CONFERENCE, 1997, Atlanta, GE, USA. **Proceedings...** Atlanta: WSC Foundation, 1997. p.142-149.

DHAMDHARE, D.M.; IYER, S.; REDDY, E.K.K. Distributed Termination Detection for Dynamic Systems. **Journal Parallel Computing**, v. 22, n. 14, p. 2025-2045, mar. 1997.

ELNOZAHY, E. N.; ALVISI, L.; WANG, Y.-M.; JOHNSON, D. B. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. **ACM Computing Surveys**, v. 34, n. 3, p. 375-408, sep. 2002.

FERSCHA, A. Parallel and Distributed of Discrete Event Systems. In: ZOMAYA, A. Y. H. **Parallel an Distributed Computing Handbook**. New York: McGraw Hill, 1995. p. 1003-1041.

FUJIMOTO, R. M. Parallel and Distributed Simulation. In: WINTER SIMULATION CONFERENCE, 1995, Arlington, USA. **Proceedings...** [S.l.:s.n.], 1995. p.118-125.

FUJIMOTO, R. M.; WEATHERLY, R. M. Time Management in the DoD High Level Architecture. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 10, 1996, Philadelphia, Pennsylvania. **Proceedings...** Philadelphia, Pennsylvania: [s.n.], 1996. p. 60-67.

HESSEL, F.; LE MARREC, P.; VALDERRAMA, C. A.; ROMDHANI, M.; JERRAYA A. A. Multilanguage Distributed Cosimulação Tool. In: WORKSHOP ON DISTRIBUTED AND PARALLEL EMBEDDED SYSTEMS-DIPES, 1998, Paderborn, Germany. **Proceedings...** [S.l.:s.n.], 1998. p. 192-199.

HIGAKI, H.; SHIMA, K.; TACHIKAWA, T.; TAKIZAWA, M. Checkpoint and Rollback in Asynchronous Distributed Systems. In: ANNUAL JOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES, 16., 1997. **Proceedings ...** [S.l.:s.n.], 1997. p. 998-1005.

JOHNSON, D. B.; ZWAENEPOEL, W. Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing. **Journal of Algorithms**, v. 11, n. 3, p. 462-491, sep. 1990.

KHUNTETA, A.; KUMAR, P. A survey of checkpointing algorithms for Distributed Mobile Systems. **International Journal of Research and Reviews in Computer Science**, New York, v. 1, n. 2, p. 127-133, jun. 2010.

KUHL, F.; WEATHERLY, R.; DAHMANN, J. **Creating Computer Simulation Systems: An Introduction to the High Level Architecture**. Prentice Hall PTR, MITRE Corporation, 2000. 212 p.

LABELLA T. H.; DIETRICH, I.; DRESSLER, F. Hybrid simulation of Sensor and Actor Networks with BARAKA. **Wireless networks**, [S.l.]: v. 16, n. 6, p. 1525-1539, Amsterdam, sep. 2008.

LAI, T. H.; YANG, T. H. On Distributed Snapshots. **Information Processing Letters**, v. 25, n. 3, p. 153-158, may. 1987.

LAW, A. M.; KELTON, W. D. **Simulation Modeling & Analysis**. 2. ed. New York: McGraw-Hill, 1991. 759 p.

LE MARREC, P.; VALDERRAMA, C. A.; HESSEL, F.; JERRAYA, A. A. Hardware, Software and Mechanical Cosimulation for Automotive Applications. IN: PROCEEDINGS OF 9TH INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, 9, 1998, Leuven, Belgium. **Proceedings...** Leuven, Belgium: [s.n.], 1998. P. 202-206.

LIEM, C.; NACABAL, F.; VALDERRAMA, C.; PAULIN, P.; JERRAYA, A. System-on-a-chip cosimulation and compilation. **Design & Test of Computers, IEEE**, [S.l.], v.14, n.2, p.16-25, jun. 1997.

LIU, J.; LEE, E. I. A Component-Based Approach to Modeling and Simulating Mixed-Signal and Hybrid Systems. **ACM Transactions on Modeling and Computer Simulation**, v. 12, n. 4, p. 343-368, oct. 2002.

MATTERN, F. Algorithms for distributed termination detection. **Distributed Computing**, v. 2, n. 3, p. 161-175, 1987.

MELLO, B. A.; CAIMI, L. L. Simulação na validação de sistemas computacionais para a agricultura de precisão. **Revista Brasileira de Engenharia Agrícola e Ambiental**, Campina Grande, v. 12, n. 6, p. 666-675, abr. 2008.

MELLO, B. A. **Co-Simulação Distribuída de Sistemas Heterogêneos**. 2005. 145 p. Tese (Doutorado em Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre.

MELLO, B. A.; WAGNER, F. R. A Standard Co-Simulation Backbone. In: FORUM DE ESTUDANTES DE MICROELETRÔNICA, 2001, Pirenópolis, GO. **Proceedings ...** Pirenópolis, GO: [s.n.], 2001.

ORAW, B.; CHOUDHARY, V.; AYYANAR, R. A Cosimulation Approach to Model-Based Design for Complex Power Electronics and Digital Control Systems. In: SUMMER COMPUTER SIMULATION CONFERENCE, 2007. **Proceedings...** [S.l.:s.n.], 2007. p. 157-164.

PELZ, G.; BIELEFELD, J.; HESS, G.; ZIMMER, G. Hardware/Software-Cosimulation for Mechatronic System Design. In: CONFERENCE ON EUROPEAN DESIGN AUTOMATION. 1996. **Proceedings...** [S.l.:s.n.], 1996. p. 246-251.

RANDELL, B. System Structure for Software Fault Tolerance. **IEEE Transactions on Software Engineering**, [S.l.], v. 1, n. 2, p. 220-232, jun. 1975.

REYNOLDS JR, P. E. Heterogeneous Distributed Simulation. In: WINTER SIMULATION CONFERENCE, 1988, San Diego, CA. **Proceedings ...** San Diego, CA: [s.n.], 1988.

SACCHI, R. P. S. **ETW: Um Núcleo para Simulação Distribuída Otimista**. 2005. 155 p. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Mato Grosso do Sul.

SULISTIO, A.; SHIN YEO, C.; BUYYA, R. Simulation of Parallel and Distributed Systems: A Taxonomy and Survey of Tools. Disponível em: <<http://www.cs.mu.oz.au/~raj/papers/simtools.pdf>>. Acesso em: 14 feb. 2011.

VENKITAKRISHNAN, P. Rollback and Recovery Mechanisms In Distributed Systems, [S.l.:s.n.], 2002.

VITHAYATHIL, F. Hybrid computer simulation of wind-driven ocean currents. In: ENGINEERING IN THE OCEAN ENVIRONMENT, OCEAN '74 - IEEE INTERNATIONAL CONFERENCE, 1974, Halifax NS, Canada. **Proceedings...** Halifax NS, Canada: [s.n.], 1974. p. 308-313.

WAGNER, F. R. **Geração de Modelos de Co-Simulação Distribuída para a Arquitetura DCB**. 2003. 103 p. Dissertação (Mestrado em Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre.