FERNANDO SANCHES VARDÂNEGA

UM MECANISMO GERENCIADOR DE ROLLBACK EM SIMULAÇÕES DISTRIBUÍDAS OTIMISTAS NA ARQUITETURA HLA

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Pontifícia Universidade Católica do Paraná como parte dos requisitos para a obtenção do título de Mestre em Ciências.

Área de concentração:
Sistemas Distribuídos

Orientador:
Prof. Dr. Carlos Alberto Maziero

Curitiba
1999

Vardânega, Fernando Sanches

Um Mecanismo Gerenciador de Rollback em Simulações Distribuídas Otimistas na Arquitetura HLA. Curitiba, 1999.

122p.

Dissertação (Mestrado) – Pontifícia Universidade Católica do Paraná. Departamento de Informática.

1.Simulações Distribuídas 2.Arquitetura HLA I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Departamento de Informática. II.t

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
Pró-Reitoria de Pós-Graduação, Pesquisa e Extensão

ATA DA SESSÃO PÚBLICA DE EXAME DE DISSERTAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA DA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ.

Exame de dissertação nº 013

Aos 22 dias do mês de novembro de 1999, realizou-se a sessão pública de defesa de dissertação "**UM MECANISMO GERENCIADOR DE ROLLBACK EM SIMULAÇÕES DISTRIBUÍDAS OTIMISTAS NA ARQUITETURA HLA**", apresentada por Fernando Sanches Vardânega, ano de ingresso 1996, para obtenção do título de Mestre em Ciências. A Banca Examinadora foi composta pelos seguintes professores:

MEMBROS DA BANCA	ASSINATURA
Presidente:. **Prof. Dr. Carlos Maziero (PUCPR)**	
Prof. Dr. Alcides Calsavara (PUCPR)	
Prof. Dr. Elias Procópio Duarte Jr (UFPR)	
Prof. Dr. Rogério Drummond (UNICAMP)	

De acordo com as normas regimentais a Banca Examinadora deliberou sobre os conceitos a serem atribuídos e que foram os seguintes:

MEMBROS DA BANCA	CONCEITOS
Presidente: **Prof. Dr. Carlos Maziero (PUCPR)**	APROVADO
Prof. Dr. Alcides Calsavara (PUCPR)	APROVADO
Prof. Dr. Elias Procópio Duarte Jr (UFPR)	APROVADO
Prof. Dr. Rogério Drummond (UNICAMP)	APROVADO
Conceito Final	APROVADO

Observações da Banca Examinadora

Profº Júlio Cesar Nievola
Coordenador do Programa de Pós-Graduação em Informática Aplicada–PUC-PR

Rua Imaculada Conceição, 1155 – Prado Velho – CEP 80215-901 – Telefone: +55 41 330-1669 / + 55 41 330-1654
Fax: + 55 41 330- 1669 – Curitiba – Paraná – Brasil - E-mail:ppgia@rla01.pucpr.br

Ao meu irmão Guto...

AGRADECIMENTOS

À minha família que sempre me incentivou e deu exemplo de perseverança e caráter.

Ao meu orientador Maziero por sempre achar tempo, na sua sempre lotada agenda, para me auxiliar durante o desenvolvimento desta dissertação.

Agradeço a todos os funcionários e professores da Pontifícia Universidade Católica do Paraná que, direta ou indiretamente, ajudaram na conclusão deste trabalho.

Aos meus amigos e colegas de turma que nunca deixaram eu desanimar, em especial ao Délcio por dividir comigo os inúmeros trabalhos durante o curso.

E, em especial, à minha esposa pelo carinho e compreensão nas muitas madrugadas de estudo.

Sumário:

Índice de Figuras:

Lista de Abreviaturas:

API - *Application Programming Interface*

AHVPS - *AttributeHandleValuePairSet*

CORBA – *Common Object Request Broker Architecture*

DARPA - *Defense Advanced Research Projects*

DIS – *Distributed Interactive Simulation*

DMSO – *Defense Modeling & Simulation Office*

DoD – *Department of Defense (USA)*

FedExec - *Federation Executive process*

FIFO - *First-In First-Out*

FOM - *Federation Object Model*

GVT – *Global Virtual Time*

HLA – *High Level Architecture*

IEEE - *Institute of Electrical and Electronics Engineers*

I/F Spec - *HLA Interface Specification*

IP - *Internet Protocol*

LAN - *Local Area Network*

LBTS - *Lower Bound on Time Stamp*

LLC - *Local Causality Constraint*

LRC - *Local RTI Component*

LP – *Logical Process*

LVT – *Local Virtual Time*

NER - *Next Event Request*

NERA - *Next Event Request Available*

NTP - *Network Time Protocol*

OMG – *Object Management Group*

OMT - *Object Model Templates*

PADS - *Parallel And Distributed Simulation*

PDU - *Protocol Data Unit*

PHVPS - *ParameterHandleValuePairSet*

RAMP - *Reliable Adaptive Multicast Protocol*

RO –*Receive Ordered Message*

RTI - *Runtime Infrastructure*

RtiExec - *RTI Executive process*

SIMNET - *SIMulator NETworking*

SOM – *Simulation Object Model*

TAG – *Time Advance Grant*

TAR – *Time Advance Request*

TARA - *Time Advance Request Available (TARA)*

TCP - *Transmission Control Protocol*

TSO –*Time Stamped Ordered Message*

UDP - *User Datagram Protocol*

WAN - *Wide Area Network*

Resumo

A utilização de recursos computacionais para a simulação de sistemas físicos já vem sendo explorada há décadas, primeiramente para fins acadêmicos e militares e, agora, cada vez mais para fins civis. Primeiramente, as simulações eram realizadas em um único computador, multiprocessado, de grande porte. Porém, com a evolução da complexidade dos modelos de simulação tornou-se evidente a necessidade cada vez maior de recursos computacionais para o desenvolvimento e execução de modelos para simulações complexas envolvendo um número cada vez mais elevado de variáveis. A solução encontrada foi dividir a carga de processamento entre diversos processadores interligados via rede de computadores, o que deu origem à simulação distribuída.

Os primeiros estudos envolvendo aspectos da simulação distribuída começaram há mais de duas décadas, nos quais os problemas de sincronismo entre as simulações começaram a ser pesquisados. Hoje, o foco dos estudos está voltado para uma nova proposta de arquitetura, chamada HLA, que visa uma padronização para o ambiente de desenvolvimento e execução de simulações distribuídas.

A arquitetura HLA (*High Level Architecture*) é formada, basicamente, por três componentes que visam garantir um alto grau de interoperabilidade entre as simulações e também permitir a reutilização dos componentes das simulações. O objetivo deste trabalho é estudar aspectos referentes ao gerenciamento de tempo dentro da arquitetura HLA. Este é um campo interessante pois o gerenciamento de tempo é fundamental para sincronizar a execução de simulações distribuídas dentro da arquitetura HLA. Este serviço é responsável pela coordenação do avanço do tempo lógico das simulações e seu relacionamento com o tempo físico. Este trabalho propõe uma extensão aos serviços de gerenciamento de tempo oferecidos pela HLA, com o objetivo de auxiliar na realização de operações de *rollback*. As operações de *rollback* são necessárias para restaurar o requisito de causalidade que pode ser violado quando são utilizados mecanismos otimistas para sincronização de eventos. O trabalho apresenta um gerente de *rollback* que utiliza técnicas de reflexão computacional para criar um meta objeto responsável pela detecção da necessidade de *rollback*, bem como pela atuação para restaurar o requisito da causalidade dentro da simulação.

Resumo

A utilização de recursos computacionais para a simulação de sistemas físicos já vem sendo explorada há décadas, primeiramente para fins acadêmicos e militares e, agora, cada vez mais para fins civis. Primeiramente, as simulações eram realizadas em um único computador, multiprocessado, de grande porte. Porém, com a evolução da complexidade dos modelos de simulação tornou-se evidente a necessidade cada vez maior de recursos computacionais para o desenvolvimento e execução de modelos para simulações complexas envolvendo um número cada vez mais elevado de variáveis. A solução encontrada foi dividir a carga de processamento entre diversos processadores interligados via rede de computadores, o que deu origem à simulação distribuída.

Os primeiros estudos envolvendo aspectos da simulação distribuída começaram há mais de duas décadas, nos quais os problemas de sincronismo entre as simulações começaram a ser pesquisados. Hoje, o foco dos estudos está voltado para uma nova proposta de arquitetura, chamada HLA, que visa uma padronização para o ambiente de desenvolvimento e execução de simulações distribuídas.

A arquitetura HLA (*High Level Architecture*) é formada, basicamente, por três componentes que visam garantir um alto grau de interoperabilidade entre as simulações e também permitir a reutilização dos componentes das simulações. O objetivo deste trabalho é estudar aspectos referentes ao gerenciamento de tempo dentro da arquitetura HLA. Este é um campo interessante pois o gerenciamento de tempo é fundamental para sincronizar a execução de simulações distribuídas dentro da arquitetura HLA. Este serviço é responsável pela coordenação do avanço do tempo lógico das simulações e seu relacionamento com o tempo físico. Este trabalho propõe uma extensão aos serviços de gerenciamento de tempo oferecidos pela HLA, com o objetivo de auxiliar na realização de operações de *rollback*. As operações de *rollback* são necessárias para restaurar o requisito de causalidade que pode ser violado quando são utilizados mecanismos otimistas para sincronização de eventos. O trabalho apresenta um gerente de *rollback* que utiliza técnicas de reflexão computacional para criar um meta objeto responsável pela detecção da necessidade de *rollback*, bem como pela atuação para restaurar o requisito da causalidade dentro da simulação.

Abstract

Computer-based simulations have been used to simulate the behavior of physical systems for the past two decades, initially for academic and military proposes, and currently for civil proposes. The first systems were able to run simulations in a single large multiprocessor computer.

The evolution on the complexity of the simulation models showed that the amount of computer resources would always increase to allow the development and execution of complex simulations with a large number of variables. The solution to this problem was to divide the processing load among many processors interconnected via a high-speed computer network. This approach is called *distributed simulation*. The first studies involving distributed simulation began two decades ago, and the focus was the synchronization issues among simulations. Today, research focus is on a new architecture, named *HLA*, which proposes a standard environment to develop and execute distributed simulations.

The HLA architecture consists of three main components designed to ensure a high level of interoperability among simulations, and to allow maximum component reusability. The HLA provides several services that can be used by the simulation entities during the simulation exercise. This work aims to study time management aspects in the HLA architecture. This field is particularly because time management is fundamental to synchronize the execution of events in distributed simulations. This service is responsible for coordinating the advance of logical time and its relationship to real time. This work proposes an extension to the standard time management services provided by HLA. This extension helps the simulation entities to perform rollback. The rollback operations are necessary to ensure the non-violation of the causality constraint. This work presents a rollback manager mechanism based on computational reflection techniques. The rollback manager will detect the causality violation and help simulation entities perform rollback procedures when needed.

Abstract

Computer-based simulations have been used to simulate the behavior of physical systems for the past two decades, initially for academic and military proposes, and currently for civil proposes. The first systems were able to run simulations in a single large multiprocessor computer.

The evolution on the complexity of the simulation models showed that the amount of computer resources would always increase to allow the development and execution of complex simulations with a large number of variables. The solution to this problem was to divide the processing load among many processors interconnected via a high-speed computer network. This approach is called *distributed simulation*. The first studies involving distributed simulation began two decades ago, and the focus was the synchronization issues among simulations. Today, research focus is on a new architecture, named *HLA*, which proposes a standard environment to develop and execute distributed simulations.

The HLA architecture consists of three main components designed to ensure a high level of interoperability among simulations, and to allow maximum component reusability. The HLA provides several services that can be used by the simulation entities during the simulation exercise. This work aims to study time management aspects in the HLA architecture. This field is particularly because time management is fundamental to synchronize the execution of events in distributed simulations. This service is responsible for coordinating the advance of logical time and its relationship to real time. This work proposes an extension to the standard time management services provided by HLA. This extension helps the simulation entities to perform rollback. The rollback operations are necessary to ensure the non-violation of the causality constraint. This work presents a rollback manager mechanism based on computational reflection techniques. The rollback manager will detect the causality violation and help simulation entities perform rollback procedures when needed.

1 Introdução

A idéia de se desenvolver sistemas capazes de simular situações reais em um ambiente computacional não é nova. Há mais de duas décadas são desenvolvidos sistemas cada vez mais complexos capazes de simular comportamentos que vão desde treinamentos de bombeiros até sofisticados aviões de combate.

A realização de simulações de sistemas complexos não é uma tarefa fácil pois a confiabilidade dos seus resultados depende muito do grau de detalhamento do modelo que irá representar o sistema físico a ser simulado. O projeto deste modelo requer o uso de sofisticadas ferramentas que utilizam bibliotecas de modelos padronizadas e interfaces gráficas que ajudam a minimizar o tempo de desenvolvimento. Outro aspecto importante é a execução propriamente dita do modelo que, dependendo da complexidade, pode consumir muitas horas de processamento em computadores multiprocessados de última geração.

A evolução da complexidade dos modelos de simulação tornou evidente a necessidade cada vez maior de recursos computacionais para o desenvolvimento e execução de modelos para simulações complexas envolvendo um número cada vez mais elevado de variáveis. A solução adotada foi utilizar vários computadores interligados em rede, de forma a se obter o maior poder computacional possível. Assim surgiram os primeiros sistemas de simulação distribuídos entre diferentes computadores. Em 1985, surgiu a primeira proposta de padrão para a simulação distribuída, o SIMNET (*SIMulator NETworking*) [KAN91][LOC95], patrocinado pela DARPA (*Defense Advanced Research Projects Agency*), ligada ao Departamento de Defesa dos Estados Unidos (DoD). O SIMNET foi uma proposta ambiciosa e visava a simulação realística envolvendo milhares de entidades, controle de entrada de dados para as entidades vindas de operadores humanos, do próprio sistema simulador ou de outras entidades de forma dinâmica. Por se tratar de uma iniciativa da área militar, os termos e entidades usadas se referiam a simulações de batalhas militares. Este projeto tinha como objetivo principal permitir a interoperabilidade entre diferentes simuladores conectados via rede. Desta forma, o poder computacional foi drasticamente aumentado, pois o processamento da simulação pode ser distribuído em diversos computadores interconectados. Outro ponto importante foi o aparecimento da interação dos operadores

humanos de forma dinâmica com o sistema de simulação, o que permite maior flexibilidade e realidade. A combinação de técnicas de simulação distribuídas via rede e da interação humana, deu origem a um novo e promissor ramo de pesquisa, a Simulação Interativa Distribuída (DIS – *Distributed Interactive Simulation*).

O uso da simulação distribuída vem provando ser uma técnica com ótima relação custo versus benefício para o estudo e entendimento de sistemas complexos reais. Através de DIS é possível combinar inúmeros processadores localizados remotamente ou localmente de forma a conseguir o poder computacional necessário.

A simulação distribuída trouxe consigo várias questões relativas à comunicação, via rede LAN ou WAN, entre os sistemas responsáveis pela simulação. Desta forma, a parte básica de DIS é um conjunto de protocolos que transportam mensagens entre entidades e eventos através de uma rede de computadores.

O ambiente DIS, proposto pelo DoD, é formado basicamente por um conjunto de PDUs (*Protocol Data Units*) que são enviadas via *broadcast* através de um protocolo padrão UDP/IP [COM88]. O uso de um protocolo padrão de mercado sem dúvidas traz uma série de benefícios, entre os quais pode ser destacado a interoperabilidade desta solução, uma vez que numerosos sistemas utilizam este padrão *de facto*. Contudo, o uso deste protocolo traz os mesmos problemas enfrentados no desenvolvimento de aplicações distribuídas no que se refere ao estabelecimento da comunicação entre os diversos computadores, através da atribuição de endereços IP [COM88] e rotas de acesso.

A modelagem e simulação de sistemas, já estudada há muito tempo pela comunidade científica, recebeu recentemente um novo impulso. Este impulso foi dado pelo DoD através de um esforço de padronização de uma interface para desenvolvimento e execução de simulações distribuídas e interativas, chamada *High Level Architecture* (HLA) [HLA97] em 1995.

Com as entidades do sistema de simulação sendo processadas em diversos computadores, torna-se fundamental um mecanismo seguro de comunicação entre elas. Este mecanismo é disponibilizado na arquitetura HLA através da infra-estrutura de comunicação, chamada RTI (*Runtime Infrastrucutre*). Esta arquitetura está rapidamente se tornando um padrão com diversas empresas desenvolvendo produtos seguindo as suas especificações. Um dos componentes mais importantes desta arquitetura é a RTI. A infra-estrutura RTI provê uma série de serviços que podem ser utilizados pelas

entidades que interagem entre si dentro da simulação. Estes serviços compreendem o gerenciamento das simulações, dos objetos envolvidos e seus atributos e também um serviço para a gerência do tempo lógico. Todos estes serviços provêm apenas os recursos básicos para as entidades e seguem as especificações da arquitetura HLA. No caso do gerenciamento de tempo, os serviços disponibilizados atendem apenas aos requisitos mínimos definidos na especificação da arquitetura deixando a cargo das entidades, chamadas federados, todos os mecanismos adicionais necessários para o desenvolvimento e execução das simulações.

A combinação de toda a potencialidade das técnicas de simulação interativa distribuída (DIS) sobre uma arquitetura padrão (HLA) torna possível o desenvolvimento de sistemas de simulação muito mais eficientes, complexos e mais próximos da realidade, que poderão auxiliar muito no desenvolvimento e testes não só de aplicações militares, mas também de qualquer outra aplicação complexa de uso civil.

Um dos objetivos deste trabalho será estudar aspectos referentes ao gerenciamento de tempo dentro da arquitetura HLA. Existe um campo promissor de pesquisa no que se refere ao gerenciamento de tempo dentro da arquitetura HLA. A arquitetura define apenas as funcionalidades necessárias, porém não especifica como elas serão implementadas. Uma grande parte do suporte ao gerenciamento de tempo precisa ser incorporado dentro das simulações ou interfaces [REY95]. Cabe à RTI, que é um componente da HLA, o suporte a estampilhas (instante de tempo no qual a mensagem foi gerada), entrega de mensagens e gerenciamento de filas. Este serviço de gerenciamento de tempo é responsável pela coordenação do avanço do tempo lógico (simulado) e seu relacionamento com o tempo físico. Cada entidade participante da simulação possui seu próprio tempo lógico que precisa estar sincronizado com as demais entidades que fazem parte da mesma simulação. O tempo lógico é fundamental para que os eventos trocados entre as entidades, chamadas *federados* dentro da HLA, possam ser processados na ordem correta, garantindo assim a integridade da simulação. Dentro da notação usada em HLA, um conjunto de federados que interagem em uma simulação é chamado *federação*. O sincronismo entre as entidades pode ser atingido através de diferentes abordagens como a conservadora, onde nenhuma entidade pode receber um evento no seu passado, ou a otimista onde o recebimento de eventos no passado implica em desfazer os eventos com estampilhas posteriores ao do evento recebido (*rollback*). Este trabalho irá estudar estas alternativas e propor um mecanismo

para o gerenciamento de tempo em federados otimistas que atenda às especificações da HLA. Este mecanismo estende os recursos já oferecidos dentro da arquitetura HLA de forma a tornar o procedimento de *rollback* mais transparente para os federados.

No capítulo 2 será feita uma revisão das principais técnicas utilizadas para o desenvolvimento e execução de simulações distribuídas. A primeira a ser apresentada será a PADS (*Parallel And Distributed Simulation*), seguida pela DIS (*Distributed Interactive Simulation*). Por fim, será apresentada a arquitetura HLA sobre a qual todo o trabalho foi desenvolvido. No capítulo 3 serão discutidos aspectos referentes ao gerenciamento de tempo em HLA, incluindo a ordenação das mensagens e avanço de tempo lógico. A seguir, no capítulo 4, será apresentado o mecanismo proposto neste trabalho para auxiliar os federados nas tarefas de *rollback*. No capítulo 5, será apresentado o procedimento efetuado para a validação do mecanismo proposto. Finalmente, no capítulo 6 serão apresentadas as considerações finais referentes ao trabalho e as perspectivas dos futuros desenvolvimentos.

2 Simulação Distribuída

A Simulação Distribuída consiste da execução de uma simulação cujas entidades participantes podem estar sendo processadas em diferentes processadores geograficamente distribuídos. Dentro da área científica da Simulação Distribuída existem diferentes ramos de pesquisa e desenvolvimento. Estes ramos de pesquisa adotaram abordagens diferentes assim como os objetivos a serem alcançados. Notadamente, dois ramos importantes de pesquisa foram iniciados, um no ambiente acadêmico e outro incentivado pelo DoD.

A idéia de se desenvolver um ambiente acadêmico de alto desempenho para simulação de eventos discretos de forma paralela e distribuída deu origem ao termo PADS (*Parallel And Distributed Simulation*). Este ramo de pesquisa é caracterizado pela ênfase em se garantir a causalidade dos eventos, ou seja, assegurar que os eventos a serem processados estão cronologicamente ordenados. Outro ramo importante surgiu impulsionado pela necessidade do DoD em desenvolver programas sofisticados para o treinamento de pessoal militar em situações que refletissem o melhor possível a realidade. Este tipo de programa envolve o desenvolvimento de ambientes de combate virtuais onde pessoas possam interagir mesmo estando fisicamente em lugares distantes. Surgiu neste contexto o termo DIS (*Distributed Interactive Simulation*), cujas pesquisas levaram à definição de uma infra-estrutura para simulação que envolve entidades distribuídas que interagem entre si.

Estes dois ramos principais de pesquisa em simulação distribuída se desenvolveram de forma quase que independente durante as últimas duas décadas, porém os seus domínios de atuação cresceram de tal forma que tornaram-se concorrentes em várias áreas de pesquisa. Áreas de pesquisa antigamente limitadas à comunidade acadêmica PADS, como por exemplo a sincronização de tempo, tornaram-se foco de pesquisas da comunidade DIS. Da mesma forma, as pesquisas PADS avançaram sobre territórios até então exclusivos da comunidade DIS.

Ambas as linhas de pesquisas conseguiram atingir ganhos expressivos dentro das suas áreas de atuação. Contudo, vários pontos ainda não foram totalmente solucionados, especialmente no que se refere o desempenho, utilização da rede e

construção de ambientes de simulação heterogêneos. Para tentar resolver estas questões que muitas vezes inviabilizavam o desenvolvimento rápido de simulações distribuídas, o DoD lançou em 1995 uma iniciativa para o estabelecimento de uma arquitetura única de alto nível para simulações, chamada de HLA (*High Level Architecture*) [HLA97]. Esta arquitetura tem como objetivo principal permitir que entidades heterogêneas possam interagir e ser reutilizadas dentro de simulações distribuídas, criando um padrão de interface para o desenvolvimento de novas simulações. Desta forma o tempo necessário para o desenvolvimento de simulações complexas pode ser drasticamente reduzido.

Todos estes cenários envolvidos dentro do contexto de simulações distribuídas envolvem diferentes tecnologias que foram surgindo e sendo aprimoradas com o passar do tempo. Na sequência deste texto, cada uma destas tecnologias será detalhada de forma a proporcionar uma visão do estado da arte no campo da simulação distribuída interativa.

2.1 PADS (Parallel And Distributed Simulation)

A simulação paralela e distribuída a eventos discretos teve sua origem no meio acadêmico com o objetivo principal de desenvolver tecnologias e ferramentas que permitissem a execução de grandes programas de simulação. Porém, antes do programa estar pronto para execução, são necessárias técnicas e utilitários que possam minimizar o tempo de desenvolvimento de tais programas, bem como permitir uma execução rápida do programa. O desenvolvimento destas técnicas e utilitários é um dos focos mais importantes dentro da comunidade PADS [FUJ95].

Dentro de um modelo de simulação, o comportamento de um sistema físico é descrito através de um conjunto de estados e eventos. Desta forma, quando uma simulação é feita, são simulados eventos em diferentes momentos de tempo e os seus efeitos são propagados através de alterações de estado. Alterações nos estados podem gerar novos eventos, que são agendados para serem efetivados no futuro de forma a garantir a correta execução do modelo.

Com o uso de simulações, o comportamento de sistemas complexos pode ser analisado de forma minuciosa, uma vez que a passagem do tempo pode ser controlada. O controle do tempo pode ser feito quando o estado do sistema muda

apenas em momentos discretos de tempo, de acordo com a ocorrência de eventos (*event driven*). A contagem do tempo passa a ser feita de forma virtual, independentemente da passagem do tempo real, caracterizando assim a existência de um tempo virtual (*virtual time*) ou tempo lógico. Todo o estudo de PADS está ligado a simulações paralelas e distribuídas de eventos discretos [FUJ90] [MIS86].

Simulações paralelas e distribuídas, dentro do contexto PADS, envolvem geralmente a execução de um único grande modelo do sistema real que está sendo simulado. Este modelo é composto por módulos desenvolvidos em um mesmo ambiente (linguagem de programação) de forma totalmente integrada. Não há preocupação em se obter interoperabilidade dentre diferentes modelos, cada modelo é projetado para uma determinada simulação. Para suprir esta deficiência, foi proposto um protocolo de sincronização PADS, o ALSP (*Aggregate Level Simulation Protocol*) [WIL94] que tem como objetivo facilitar a interoperabilidade entre diferentes modelos de simulação.

Dentro do contexto de PADS, as simulações são geralmente realizadas dentro de sistemas multiprocessados altamente acoplados através de uso de memória compartilhada ou troca de mensagens. Através da utilização de técnicas de paralelismo, o objetivo a ser alcançado é um desempenho satisfatório. Outro ambiente muito utilizado para simulações consiste de diversas estações de trabalho conectadas através de uma rede local (LAN). Desta forma o poder computacional pode ser substancialmente aumentado, porém entram em cena questões cruciais no que se refere a confiabilidade e latência da comunicação entre os ambientes computacionais. Dentro da linha de pesquisa PADS, geralmente, assume-se que a comunicação é confiável e com latência arbitrária, mas relativamente baixa.

Em PADS, o importante é garantir que os eventos sejam ordenados de acordo com suas datas de ocorrência. Apesar de buscar sempre uma execução rápida, dentro da PADS não há um compromisso com execuções em tempo real, sincronizadas com o tempo físico (real). Este é um fator que limita a área de atuação da PADS, uma vez que em muitas simulações interativas, como por exemplo treinamento virtual, a noção de tempo real é vital para a obtenção de resultados realísticos. Contudo, o uso de algoritmos PADS em ambientes de tempo real é um campo de pesquisa promissor [GHO94]. Porém em muitos casos este requisito não é fundamental, nestes casos o importante é se obter uma simulação

das funcionalidades do sistema, como em uma simulação de um sistema de telecomunicações.

O modelo de simulação PADS é basicamente composto por um conjunto de eventos gerados a partir da execução de processos lógicos (LPs) que representam o modelo e variáveis $S = (s_1, s_2, ..., s_n)$ que representam o estado do sistema [FER94]. Os eventos são agendados para ocorrerem em momentos definidos de tempo, estampilhados com suas datas de ocorrência (*timestamped*) e ordenados em uma lista de eventos. Existe ainda a necessidade da existência de um relógio global para indicar o instante de tempo atual. Este relógio representa o tempo lógico global (*Global Virtual Time* – GVT) que serve como referência única para a evolução do tempo dentro da simulação. A simulação ocorre através de mecanismos, conhecidos como escalonadores (*simulation engines)*, que controlam a execução dos processos lógicos (LPs) do modelo. Estes mecanismos irão continuamente retirar o primeiro evento da lista, ou seja, aquele cujo instante de ocorrência for menor, e irá simular os seus efeitos no estado do sistema, o que poderá gerar novos eventos a serem colocados na lista de eventos. Este processo é repetido continuamente até que o tempo virtual atinja um valor pré-estabelecido ou que não haja mais eventos na lista.

Cada processo lógico gerencia uma cópia local do relógio da simulação (GVT) que evolui em função do estado local do processo. Esta cópia local do relógio global é chamada tempo virtual local *(Local Virtual Time* – LVT).

É necessário estabelecer um sincronismo entre os eventos distribuídos, de forma que a causalidade e o determinismo sejam respeitados. Estes dois princípios básicos dos sistemas físicos determinam que eventos futuros não podem influenciar o comportamento do sistema no presente, porém o comportamento futuro pode ser determinado com base nos estados passados e no estado presente do sistema.

Para garantir o processamento de eventos discretos na ordem correta, de acordo com a data de ocorrência, foram desenvolvidos vários algoritmos de sincronismo. Esta área de pesquisa envolvendo algoritmos de sincronismo é sem dúvida uma das áreas mais difundidas dentro da PADS. O objetivo destes algoritmos é garantir que os eventos processados no ambiente distribuído estejam na ordem correta, de acordo com suas estampilhas (*timestamps*), o que irá garantir que a causalidade seja respeitada. Os algoritmos utilizam duas abordagens de sincronismo: conservadora

(pessimista) e otimista. Ambos, geralmente, pressupõem que um modelo de simulação é composto por uma coleção de processos lógicos (LP_i) que se comunicam através de mensagens ou eventos [FER94] [FUJ95]. Estes processos podem ser executados de forma paralela e distribuída para se obter um melhor desempenho da simulação. O modelo de simulação através de processos lógicos é composto [FER94], basicamente, por um conjunto de elementos, conforme é mostrado na figura 1.

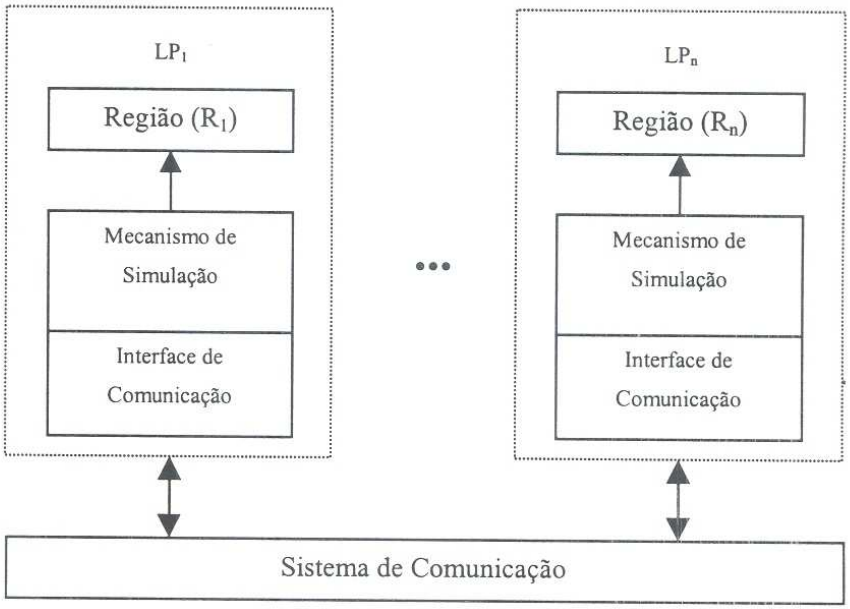

Figura 1: Modelo de simulação com LP

Os elementos do modelo apresentados na figura 1 são:

- Um conjunto de processos lógicos (LP_i) encarregados de processar sincronamente ou assincronamente eventos em paralelo.
- Um sistema encarregado da comunicação entre os processos lógicos, que possibilita a troca de mensagens entre eles.
- Para cada processo lógico (LP_i) existe uma região (R_i) na qual o escalonador (*simulation engine*) processa os eventos locais e gera eventos locais e remotos, adiantando o tempo virtual local (LVT).
- Dado que S representa o conjunto de todas as variáveis de estado, cada processo lógico (LP_i) tem acesso apenas a um sub-conjunto estático de variáveis de estado $S_i \subset S$. S_i e S_j são distintos se $i \neq j$.

- Dois tipos de eventos são processados em cada LP_i : eventos internos que afetam apenas $S_i \subset S$ e externos que também afetam variáveis de estado de outros LPs, ou seja, $S_j \subset S$ onde $i \neq j$.
- A Interface de Comunicação é responsável pela propagação dos efeitos do processamento local de eventos para os demais processos lógicos. Da mesma forma, é responsável por transmitir os efeitos gerados pelo processamento de eventos em processos remotos para a simulação local. A propagação ocorre através do envio, recebimento e processamento de mensagens, que contém eventos, com cópias do LVT do remetente no instante de envio.

2.1.1 Gerenciamento de Tempo em PADS

Segundo o modelo PADS apresentado, os processos lógicos se comunicam através de mensagens que contêm sempre uma estampilha com o LVT do remetente. Cabe aos algoritmos de sincronismo, conservadores ou otimistas, garantir que os processos lógicos tratem os eventos de acordo com a ordem das suas estampilhas, o que irá garantir o requisito da causalidade (*local causality constraint* - LCC). Segundo este princípio, o evento que causou o processamento deve ser processado antes de qualquer outro evento que represente um efeito deste processamento. Para respeitar este requisito, cada processo lógico deve tratar os seus eventos locais (interno ou externos recebidos através da Interface de Comunicação) na ordem das suas estampilhas no tempo virtual ou simulado. Existem muitas publicações [RIB98] [FER94] [FUJ95] [RIG89] que descrevem detalhadamente as abordagens de sincronismo conservadora e otimista. Neste trabalho, as abordagens de sincronismo são descritas de forma simplificada.

2.1.1.1 Abordagem Conservadora ou Pessimista

A abordagem de sincronismo conservadora foi historicamente a primeira a ser pesquisada [CHA79] e está baseada na determinação do momento seguro (*safe*)

para processar cada evento. Esta abordagem é também conhecida como *protocolo CMB* em homenagem aos seus primeiros pesquisadores: Chandy, Misra e Bryant.

O princípio básico desta abordagem é evitar de toda forma que o requisito de causalidade local (LCC) seja violado. A causalidade dos eventos entre os processos lógicos é obtida através do envio de mensagens com eventos e estampilhas (*timestamp*), que são cópias dos LVT's dos processos remetentes das mensagens [MIS86]. Logo, cada mensagem contém um evento e uma estampilha. A abordagem conservadora consiste em garantir que todos os eventos com menor estampilha que a o evento externo recebido tenham sido processados pelo LP. Isto garante que os eventos internos e externos sejam processados em uma ordem cronológica.

Os processos lógicos se comunicam através de canais FIFO (*first-in first-out*) confiáveis. As mensagens recebidas por um processo em um canal também são ordenadas de acordo com suas estampilhas. Isto garante que a estampilha da última mensagem recebida em um canal será o limite inferior de tempo para qualquer outra mensagem recebida posteriormente neste mesmo canal. Desta forma não será permitido receber mensagens (com eventos externos) com estampilhas maiores do que o LVT atual do processo. Cada canal possui um relógio cujo valor é igual à estampilha da primeira mensagem da fila, se existir alguma mensagem, senão o relógio é igual a estampilha da última mensagem recebida neste canal. O processo lógico continuamente seleciona o canal com menor valor de relógio e verifica se ele possui alguma mensagem, se sim processa o evento contido na mensagem. Caso não haja nenhuma mensagem na fila deste canal, o processo lógico bloqueia.

Como a abordagem conservadora ocasiona bloqueio de processos, ela está sujeita ao aparecimento de bloqueios perpétuos (*deadlocks*) [CHA83]. A prevenção contra a ocorrência de *deadlocks* é usualmente feita através da utilização de mensagens nulas (*null messages*) [MIS86][FUJ95][MAZ94][PRE91]. Os processos lógicos enviam mensagens nulas em cada canal de comunicação após o processamento de cada evento. Este procedimento é utilizado para avisar aos demais processos que o LP, emissor da mensagem nula, não vai mais mandar nenhuma mensagem com estampilha menor do que a contida na mensagem nula. Ao enviar uma mensagem nula, um processo comunica ao outro uma previsão (*lookahead*) sobre o seu comportamento futuro. Embora resolva o problema de

deadlocks, a utilização de mensagens nulas causa uma sobrecarga na rede de comunicação (*overhead*). Existem outros métodos utilizados para o tratamento de *deadlocks*, especialmente no que se refere a detecção e recuperação de *deadlocks*, muitos dos quais baseados em um gerenciamento centralizado de *deadlocks* [FUJ90][FUJ92].

2.1.1.2 Abordagem Otimista

A abordagem otimista difere da conservadora no que se refere a obrigatoriedade do respeito à causalidade dos eventos. No caso da abordagem conservadora, os eventos só são processados quando é seguro, de acordo com suas estampilhas e com o LVT, o que garante o respeito ao requisito de causalidade local (LCC). Já no caso da abordagem otimista, a violação do requisito de causalidade local é tolerada, mesmo que posteriormente isto implique em reprocessamento de eventos. A abordagem otimista possui mecanismos que possibilitam a detecção de violações de causalidade, bem como a recuperação destas situações.

Dentro da abordagem otimista, os eventos internos e externos são processados sem garantia de que a causalidade local está sendo respeitada. Isto evita que haja bloqueio de processos como no caso da abordagem conservadora, uma vez que os eventos disponíveis nos canais de comunicação vão sendo processados e o LVT avança. Se alguma mensagem, evento externo, com menor estampilha chegar em um canal, será preciso desfazer (*rollback*) o processamento dos eventos com estampilhas maiores já processados. Assim todos os eventos são processados, ao contrário dos algoritmos conservadores onde há a necessidade de se determinar o momento seguro para cada evento. Na abordagem otimista, os eventos que não violam a causalidade (LCC) são confirmados, enquanto os outros são cancelados através do mecanismo de *rollback*. Da mesma forma, os efeitos gerados por estes eventos cancelados também precisam ser anulados.

As primeiras pesquisas envolvendo abordagens otimistas foram feitas por Jefferson e Sowizral [JEF85] e deram origem ao método otimista mais conhecido, o *Time Warp* [SOW85]. O *Time Warp* usa um mecanismo de *rollback* para restaurar o requisito de causalidade sempre que alguma violação for detectada.

Quando um processo lógico em T_3 recebe, em um dos seus canais, um evento (mensagem) com estampilha (T_2) menor do que algum evento já processado, este processamento precisa ser desfeito. Esta situação está representada na figura 2. Após o processo voltar para um estado seguro (último *checkpoint* em T_1), anterior ao processamento destes eventos, é preciso processar novamente os eventos na ordem das suas estampilhas (*coast forward*) [FUJ90] [LIN93].

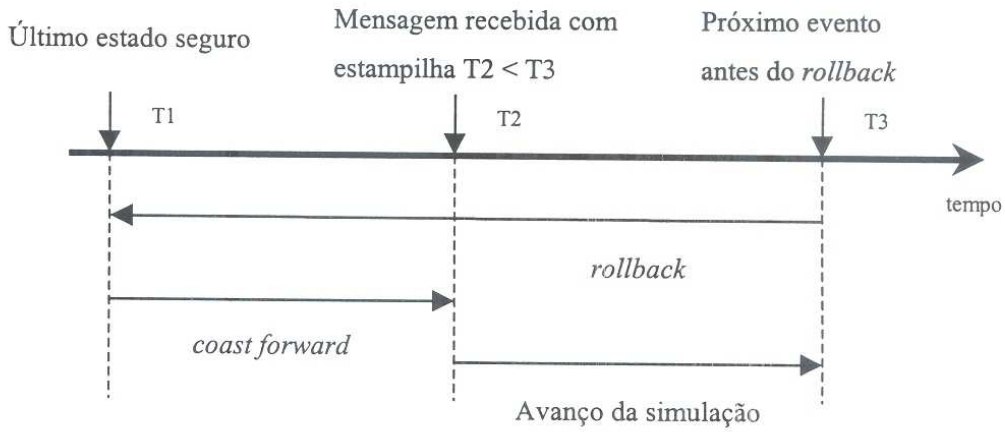

Figura 2 : Rollback

Porém, além do reprocessamento dos eventos, o processo lógico precisa avisar os demais processos para os quais ele mandou mensagens indevidamente; estas mensagens incorretas são também chamadas de mensagens falsas. Os demais processos precisam também desfazer todo o processamento gerado pelas mensagens falsas recebidas do processo que sofreu *rollback*. Para realizar esta tarefa, o *Time Warp* precisa enviar mensagens aos demais processos para que eles desfaçam todo o processamento gerado pelos eventos cancelados. Um mecanismo muito utilizado para este fim é o envio de anti-mensagens (*anti-messages*) [FER94]. Uma anti-mensagem é uma cópia da mensagem previamente enviada, ou seja, da mensagem falsa. Sempre que uma anti-mensagem for enfileirada em um canal de comunicação juntamente com a mensagem original, as duas são eliminadas da fila. Caso a mensagem original já tenha sido processada, o processo lógico precisa desfazer (*rollback*) o processamento e possivelmente enviar também novas anti-mensagens.

Apesar do método *Time Warp* resolver os problemas referentes à manutenção da causalidade, ele apresenta alguns pontos deficientes. Este mecanismo otimista necessita de um controle maior do estado do processo, que deve ser feito através de pontos de controle (*check points*) [PRE93] que indicam o último estado seguro para o qual o processo precisa retornar caso haja uma violação da causalidade (LCC). A manutenção destes controles consume recursos do sistema, principalmente memória. Outro ponto importante é a existência de certas operações, como I/O, que não podem ser desfeitas. Uma alternativa viável para a solução destes problemas é o uso do conceito de tempo virtual global (*Global Virtual Time* - GVT). O GVT é o limite inferior de estampilhas para qualquer futuro *rollback*. O valor do GVT é obtido através da menor estampilha entre todos os eventos (mensagens) ainda não processados ou parcialmente processados. O GVT atua como um delimitador inferior de tempo para a ocorrência de *deadlocks*. Na prática, a determinação do GVT é complicada, podendo ser feita por algoritmos [BEL90] centralizados ou distribuídos, gerando alta degradação da rede de comunicação.

Cancelamento de Mensagens

Quando o protocolo otimista *Time Warp* foi proposto em [JEF85] o mecanismo adotado para o tratamento do *rollback* era chamado de cancelamento agressivo (*aggressive cancellation*). Neste tipo de mecanismo quando um processo lógico recebe uma mensagem (m_2) com estampilha (T) menor do que alguma mensagem (m_1) recebida e já processada, este processamento precisa ser desfeito e as possíveis mensagens enviadas (falsas) precisam ser imediatamente anuladas. Se $T_{m1} > T_{m2}$, a mensagem m_1 sofre preempção (*message preemption*) [LIN93] e o seu processamento é desfeito. A preempção de mensagens é um requisito do mecanismo *Time Warp*.

Em [REI90] e [GAF88] é discutida uma nova abordagem para o cancelamento de mensagens (falsas) que foram indevidamente enviadas. Esta abordagem ficou conhecida como cancelamento lento (*lazy cancellation*) das mensagens enviadas. Quando um processo lógico recebe uma mensagem (m_2) com estampilha (T) menor do que alguma mensagem (m_1) recebida e já processada, este processamento é desfeito e as mensagens enviadas (falsas) não são imediatamente anuladas através

de anti-mensagens. O processo lógico volta a ser executado assim que o processamento de m_1 é desfeito. Quando a mensagem m_1 é reprocessada, o conteúdo das mensagens geradas pela sua execução é comparado com o conteúdo das mensagens (falsas) previamente enviadas. Se o conteúdo for diferente, então é necessário o envio de anti-mensagens para cancelar as mensagens falsas já enviadas. Após o envio das anti-mensagens é preciso enviar as novas mensagens corretas assim como acontece com a abordagem agressiva. Por outro lado, se não houver diferenças no conteúdo, nada precisa ser feito e o envio de anti-mensagens e novas mensagens foi evitado, o que garante um desempenho melhor do que a abordagem baseada no cancelamento agressivo de falsas mensagens enviadas.

2.1.1.3 Comparativo

Existem diversos estudos [LIP90] [LIN90] que fazem uma análise comparativa entre as abordagens conservadora e otimista, especialmente no que se refere ao desempenho e ao consumo de memória. Cada abordagem possui o seu nicho de aplicação de acordo com o modelo de simulação que está sendo elaborado. Não é o escopo deste trabalho fazer uma comparação profunda entre as abordagens, porém é importante ressaltar alguns aspectos que são fundamentais para o entendimento do mecanismo proposto no capítulo 4.

As principais características dos algoritmos de sincronismo conservadores e otimistas são:

Conservadores:

- Os processos lógicos precisam enviar mensagens nulas (*null messages*) em cada canal de comunicação após o processamento de cada evento para evitar a ocorrência de *deadlocks*. Este procedimento também serve para avisar aos demais processos que o processo lógico não vai mais mandar nenhuma mensagem com estampilha menor do que a contido na mensagem nula. A utilização de mensagens nulas causa uma sobrecarga na rede de comunicação (*overhead*) e também aumenta a carga de processamento do processo lógico.

- Os algoritmos conservadores restringem muito o desempenho de simulações paralelas, pois frequentemente os processos lógicos ficam bloqueados a espera de mensagens nos seus canais de comunicação. O processo lógico continuamente seleciona o canal com menor valor de relógio e verifica se ele possui alguma mensagem, se sim processa o evento contido na mensagem. Caso não haja nenhuma mensagem na fila deste canal, o processo lógico bloqueia.

- Requerem uma quantidade pequena de memória, pois precisam apenas armazenar os eventos pendentes e uma cópia do vetor de estado do simulador.

Otimistas:

- Os algoritmos otimistas oferecem um potencial maior para a exploração de paralelismo em simulações a eventos discretos [DAS97].

- Proporcionam uma maior transparência dos mecanismos de sincronismo para os desenvolvedores de simulações paralelas a eventos discretos [DAS97].

- Em muitos modelos de simulações de combate [WIE89], redes de comunicação [CAR94], entre outras, apresentam uma melhoria no desempenho, reduzindo o tempo necessário para a simulação.

- Os algoritmos otimistas necessitam do cálculo do GVT para estabelecer um tempo virtual mínimo para todos os futuros *rollbacks*. Este cálculo pode demandar recursos de processamento e, dependendo do algoritmo usado, sobrecarregar ainda mais a rede de comunicação.

- Nos algoritmos otimistas é importante limitar o avanço do tempo dos processos lógicos. Caso um processo avance muito no tempo, ele estará sujeito a um número maior de *rollbacks,* o que irá consumir mais recursos computacionais.

- A quantidade de memória a ser consumida durante a simulação é maior em relação aos algoritmos conservadores. É preciso armazenar um volume maior de informações sobre o histórico dos processos para o caso de ocorrer a necessidade de *rollbacks*.

2.2 DIS (Distributed Interactive Simulation)

A simulação distribuída interativa (DIS) teve sua origem na projeto SIMNET [POP91] em 1985, patrocinado pelo DoD. O projeto SIMNET tinha como objetivo permitir simulações realísticas envolvendo milhares de entidades que podiam estar sendo executadas em diferentes sistemas separados fisicamente. Basicamente o SIMNET era um protocolo de rede com diversas PDUs (*Protocol Data Units*) [LOC95] com as quais os programas que faziam parte da simulação podiam se comunicar. As simulações implementavam estas PDUs ou parte delas de acordo com a complexidade do modelo.

A SIMNET não requisitava nenhum hardware de rede específico nem protocolo de baixo nível, conforme mostrado na figura 3. Devido a grande utilização do protocolo *Ethernet*, a SIMNET recomendava a sua utilização como infra-estrutura de comunicação. Era usado um endereço *Ethernet* de 48 bits como endereço *multicast*, com certos campos identificando o grupo *multicast* e o exercício de simulação. Desta forma a SIMNET usava uma codificação não padrão para a comunicação sobre a rede *Ethernet*. Em 1990, a SIMNET era capaz de integrar 250 simuladores localizados em 11 localizações [FUJ95].

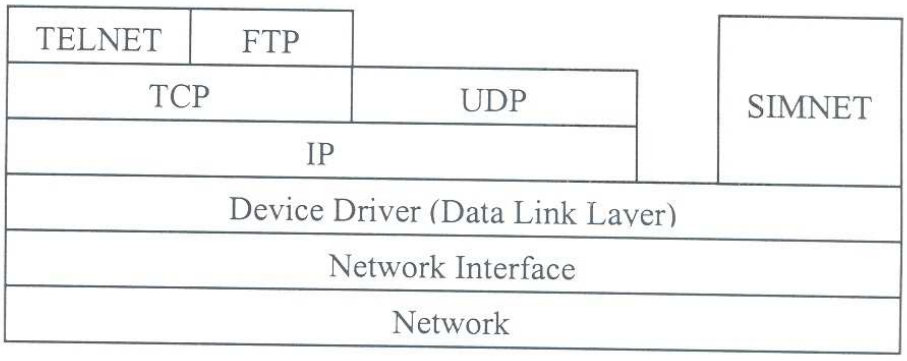

Figura 3 : SIMNET

O projeto SIMNET serviu de base para uma nova proposta para padronização de uma infra-estrutura para simulações distribuídas interativas, que ficou conhecida como DIS. O objetivo principal do DIS é interconectar e interoperar diversos simuladores distantes geograficamente, de forma que eles possam compartilhar um único ambiente virtual de simulação. As simulações podem estar distribuídas, porém interconectadas através de uma infra-estrutura comum de comunicação. Além de distribuídas, as simulações podem ser interativas, o que significa que as entidades da simulação podem influenciar outros elementos, inclusive com intervenção de operadores humanos. Dentro do mundo virtual criado pelo DIS, as entidades interagem entre si através de eventos. Um evento pode afetar outras entidades e se propagar pela simulação ou não. Como o DIS foi incentivado pelo DoD, as entidades definidas envolvem elementos de campos de batalhas, ou seja, tanques, soldados, armas, etc.

No documento [VIS93] a missão primária do DIS é descrita como sendo prover uma infra-estrutura capaz de interoperar simulações de vários tipos geograficamente distribuídas para criar mundos virtuais, realísticos e complexos para a simulação de atividades altamente interativas. Esta infra-estrutura une sistemas construídos para diferentes propósitos, de diferentes tecnologias, de vários fornecedores e plataformas, de forma que todos eles possam interoperar. O DIS provê padrões de interface, arquiteturas de comunicação, estruturas de gerenciamento e outros mecanismos que tornam possível a integração de sistemas de simulação heterogêneos.

Assim como na SIMNET, o DIS está baseado em uma série de protocolos que viabilizam a troca de mensagens entre as entidades da simulação através de uma rede de computadores. Estes protocolos foram padronizados em 1993 através do padrão IEEE 1278 DIS [IEE93] na sua versão 1.0. Novas versões do DIS definem novas PDUs capazes de suportar novas características , porém existem incompatibilidades entre as versões. Atualmente existem cerca de 27 diferentes PDUs definidas que são utilizadas para troca de informações entre as entidades envolvidas na simulação.

2.2.1 PDUs DIS

Uma PDU (*Protocol Data Unit*) define um conjunto de informações sobre a entidade que é agrupado na forma de um pacote enviado via rede. Dentro de uma simulação DIS, as entidades interagem entre si através de PDUs. Para garantir uma interoperabilidade entre simuladores desenvolvidos de forma independente por diferentes organizações, foram definidos padrões de formato e conteúdo para as PDUs [IEE93]. As PDUs padronizadas são apresentadas na figura 4 a seguir.

Acknowledge PDU	Receiver PDU
Action Request PDU	Remove Entity PDU
Action Response PDU	Repair Complete PDU
Collision PDU	Repair Response PDU
Comment PDU	Resupply Cancel PDU
Create Entity PDU	Resupply Offer PDU
Data PDU	Resupply Received PDU
Data Query PDU	Service Request PDU
Designator PDU	Set Data PDU
Detonation PDU	Signal PDU
Electromagnetic Emission PDU	Start/Resume PDU
Entity State PDU	Stop/Freeze PDU
Event Report PDU	Transmitter PDU
Fire PDU	

Figura 4 : PDUs DIS

Estas PDUs estão detalhadas e explicadas em [IEE93]. As PDUs assim como toda a especificação DIS estão fortemente ligadas a simulação de ambientes de batalha. Cada PDU tem uma função específica dentro da simulação, porém o formato básico de uma PDU é:

- **Cabeçalho da PDU**: é a primeira parte da PDU e contém um número de identificação associado com o exercício DIS, a versão do protocolo, o tipo da PDU, uma estampilha e o tamanho da PDU.
- **Informação da Entidade**: informação associada com a aparência e localização de uma entidade.

Os padrões DIS não especificam uma linguagem para a sua implementação. Muitos produtos desenvolvidos utilizando a tecnologia DIS foram feitos em linguagem C ou C++. Na sua essência, uma PDU é uma estrutura de dados que pode ser descrita em linguagem C, como no exemplo a seguir, onde é mostrada a *Entity State PDU* v.2.0 [IST91].

```
typedef struct {
      PDUheader          entity_state_header;
      EntityID           entity_id;
      ForceID            force_id;
      Unsigned char      num_articulat_params;
      EntityType         entity_type;
      EntityType         alt_entity_type;
      VelocityVector     entity_velocity;
      EntityLocation     entity_location;
      EntityOrientation  entity_orientation;
      Unsigned int       entity_appearance;
      DeadReckonParams   dead_reckon_params;
      EntityMarking      entity_marking;
      Unsigend int       capabilities;
      ArticulatParams    articulat_params(MAX_ARTICULAT_PARAMS);
      } EntityStatePDU;
```

As PDUs são transmitidas via rede para todos os simuladores através de *broadcast* de mensagens usando o conjunto de protocolos de comunicação UDP/IP (*User Datagram Protocol/Internet Protocol*) [COM88], na maioria das suas implementações. A figura 5 mostra o relacionamento do DIS com os demais protocolos de comunicação. Os simuladores não precisam garantir que os outros simuladores recebam as PDUs, bem como não precisam especificar quais simuladores devem recebê-las, o que justifica o uso do UDP. As PDUs são enviadas

via *broadcast* para todos os simuladores, cabendo a eles determinar se as PDUs são relevantes ou não para o seu exercício, o que gera um grande tráfego de rede no caso de simulações de grande porte.

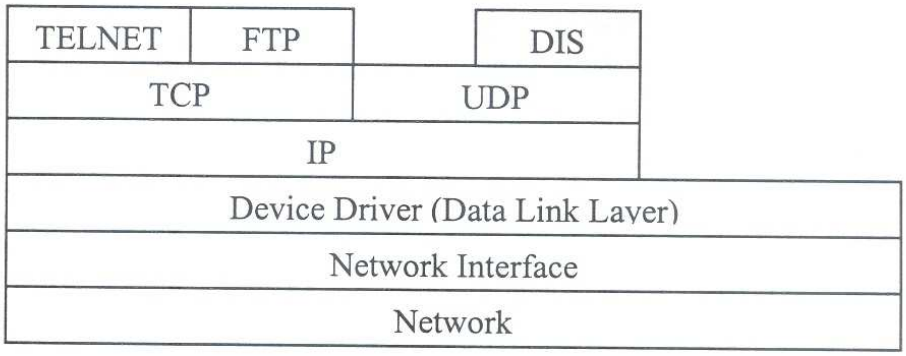

Figura 5 : DIS

Contudo, algumas implementações mais recentes, como a NPSNET [NPS] da *Naval Postgraduate School*, utilizam o modo *multicast* para tentar minimizar o tráfego de rede gerado durante a simulação distribuída. Assim as PDU's são enviadas apenas para um grupo de simuladores, para os quais elas são relevantes. Uma outra linha de pesquisa estuda o uso de um protocolo de transporte para enviar mensagens no modo de transmissão *multicast*. Neste caso é usado o protocolo RAMP (*Reliable Adaptive Multicast Protocol*) [KOI96][BRA93], que além de evitar o envio de mensagens não relevantes para simuladores, evita também a perda e consequente retransmissão de mensagens [SMI96].

A troca intensa de PDUs necessária para as interações ocasiona uma carga considerável na rede de comunicação, mesmo usando-se *multicast*. Cada entidade deve enviar uma mensagem, PDU, com seu estado atual (*Entity State PDU* v.2.0) no máximo a cada 30 segundos podendo chegar a 5 vezes por segundo [FUL96]. Cada PDU deste tipo possui 140 bytes (v.2.0.4), o que dependendo do número de entidades pode gerar um tráfego elevado na rede. Além do envio das mensagens, outro problema é gerado no momento da recepção das mensagens ou PDUs. Todos os simuladores recebem todas as PDUs e precisam processar todas. Dependendo do tamanho do exercício DIS, o volume de PDUs pode ser muito grande, onde apenas uma pequena parte delas é relevante para cada simulador.

Além do utilização do modo *multicast* de transmissão, existem outras técnicas que são utilizadas para a diminuição do tráfego gerado por uma simulação

distribuída na rede de comunicação. Dentro do princípio DIS, uma técnica empregada é a transmissão de mensagens entre os simuladores somente quando as entidades mudam de estado. Quando o estado da entidade não muda, como por exemplo um veículo continua se movimentando em linha reta, a frequência com que o seu estado é transmitido diminui. Este procedimento economiza recursos de rede, porém os simuladores precisam enviar mensagens periódicas com o estado das entidades para que novos simuladores que venham a participar do exercício possam incluir as entidades nos seus ambientes.

Outra forma de diminuir o consumo de rede, causado pela intensa troca de PDUs, é a utilização de algoritmos chamados *dead reckoning* [FUL96][FUJ95]. As PDUs carregam informações sobre as entidades da simulação, como por exemplo, o seu tipo, localização, velocidade, aceleração, orientação e velocidade angular. Estas informações são utilizadas pelos algoritmos de *dead reckoning,* em todos os simuladores, para estimar a localização de entidades que estão sendo executadas em outros simuladores. Estes algoritmos evitam que a cada movimento de uma entidade seja necessário o envio de uma PDU, ao invés disto, cada simulador pode estimar este movimento localmente. Cada entidade deve informar aos simuladores que fazem parte deste exercício DIS como esta estimação deve ser feita. Existem modelos definidos em [IEE95] para *dead reckoning* (DRM - *Dead Reckoning Models).* A notação destes modelos consiste de três elementos [FUL96]:

DRM (F or R, P or V, W or B)

Nesta notação, o primeiro elemento indica se o sistema de coordenadas da entidade é fixo (F) ou rotacional (R). O segundo elemento especifica se a taxa é constante (P) ou variável (V). Finalmente, o terceiro elemento indica qual o sistema de coordenadas que deve ser utilizado, se o global (W) ou local (B). Na maioria dos casos, é usado o algoritmo DRM (F, P,W).

Apesar de todas estas técnicas aqui apresentadas para diminuir o tráfego de PDUs DIS na rede, o problema da utilização de banda de rede ainda persiste. Em 1996, a comunidade científica começou a desenvolver uma variação do protocolo DIS para reduzir a utilização de banda de rede, chamado DIS-Lite [TAY96]. Este protocolo elimina a transmissão de informações redundantes, o que reduz entre 30-

70% [TAY96] o tráfego gerado pela transmissão de PDUs, conforme mostrado na figura 6.

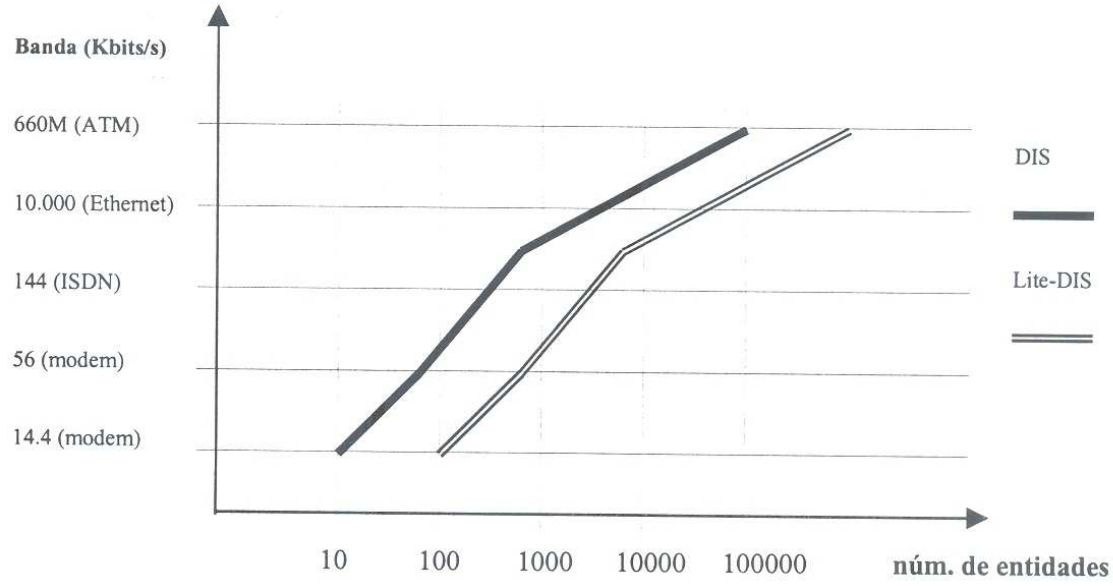

Figura 6 : Diagrama de utilização de rede DIS e Lite-DIS

Para a construção do gráfico, no caso do protocolo DIS, o tráfego considerado foi 100 bytes por entidade a cada segundo. Já no Lite-DIS, são transmitidos 20 bytes por entidade a cada segundo.

2.2.2 Gerenciamento de Tempo em DIS

Cada PDU que é enviada leva consigo uma estampilha que indica a hora exata vigente no simulador que a enviou, que é obtida a partir do relógio de tempo local (LVT). Devido a atrasos na entrega das mensagens ocasionados pela latência da rede, as PDUs podem chegar no destinatário atrasadas. Tipicamente a latência máxima da rede de comunicação deve oscilar entre 100 e 300 ms. Os simuladores recebem PDUs de várias origens e com estampilhas que podem não estar ordenadas cronologicamente, o que pode causar problemas. Os simuladores precisam possuir mecanismos que possibilitem compensar as diferenças entre os relógios dos simuladores [GOL94].

Dentro do PADS, cada processo lógico possui seu próprio tempo virtual ou simulado. Ao contrário do que acontece em simulações PADS, em um exercício DIS

todos os simuladores estão sincronizados em uma única referência de tempo real. O problema neste caso está em garantir uma sincronização entre os relógios de tempo real distribuídos. Caso esta sincronização não seja mantida, os aspectos temporais da simulação ficam comprometidos, não refletindo mais o comportamento do mundo real. Dentro do contexto DIS, uma área de pesquisa para evitar problemas de correlação temporal é a utilização de um tempo de coordenação universal (*Coordinated Universal Time* – UTC). Os relógios de tempo real dos simuladores são sincronizados com o relógio padrão UTC [CHE94] que pode ser consultado através de diferentes implementações, como por exemplo, sinal de rádio, linha discada para um *bureau* de tempo ou através do protocolo NTP (*Network Time Protocol*) [MIL92].

2.3 HLA (High Level Architecture)

Em março de 1995, o DoD iniciou a implantação de um plano ambicioso para padronizar uma arquitetura para modelagem e simulação (*Modeling & Simulation* – M&S) de sistemas. A base deste plano, conhecido como "*Modeling and Simulation Master Plan*" [MAS95], é o estabelecimento de uma arquitetura comum de alto nível para facilitar a interoperabilidade entre todos os tipos de modelos e simulações e também facilitar a reusabilidade dos componentes. Esta arquitetura de alto nível ficou conhecida como HLA (*High Level Architecture*) [HLA97] e tem como finalidade atingir o objetivo 1 do plano mestre.

Resumidamente o plano mestre do DoD é formado por seis objetivos que visam a elaboração de uma arquitetura única para modelagem e simulação.

Os objetivos são:

- Objetivo 1: Desenvolver um *framework* técnico comum para modelagem e simulação. Dentro deste objetivo existem três sub-objetivos:
  - 1-1 : Arquitetura de Alto Nível (HLA)
  - 1-2 : Modelos conceituais (*mission space*)
  - 1-3 : Padronização dos dados
- Objetivo 2 : Prover representações definitivas e completas para o meio ambiente. Dentro deste objetivo existem quatro sub-objetivos:

- 2-1 : Terrenos
- 2-2 : Oceanos
- 2-3 : Atmosfera
- 2-4 : Espaço

- Objetivo 3 : Prover representações definitivas dos sistemas simulados

- Objetivo 4 : Prover representações definitivas e completas para o comportamento humano. Dentro deste objetivo existem dois sub-objetivos:
  - 4-1 : Indivíduos
  - 4-2 : Grupos e organizações

- Objetivo 5 : Estabelecer uma infra-estrutura para modelagem e simulação (M&S) que atenda as necessidades dos desenvolvedores e usuários finais. Dentro deste objetivo existem cinco sub-objetivos:
  - 5-1 : Levantamento dos requisitos e custos de M&S
  - 5-2 : Desenvolvimento de metodologias e padrões para verificação, validação e aceite de modelos e simulações (*Verification, Validation & Acceptance* - VV&A)
  - 5-3 : Repositórios para facilitar o acesso pelos usuários aos recursos de M&S
  - 5-4 : Desenvolvimento de uma infra-estrutura de comunicação
  - 5-5 : Centro de Coordenação e suporte operacional para os desenvolvedores

- Objetivo 6 : Compartilhar os benefícios da modelagem e simulação. Dentro deste objetivo existem três sub-objetivos:
  - 6-1 : Quantificar o impacto de M&S
  - 6-2 : Educação de usuários potenciais de M&S
  - 6-3 : Suporte a transferência bi-direcional de tecnologia com outras agências governamentais, industrias e nações

Desta forma, a HLA é apenas uma parte da estratégia do DoD para a modelagem e simulação de sistemas. A partir de setembro de 1996, a HLA tornou-se a arquitetura técnica padrão para todas as simulações do DoD. Atualmente está em processo de padronização pelo IEEE sob o número 1516 [IEE99].

A arquitetura HLA usa uma terminologia própria para designar os seus componentes. Dentro desta arquitetura existem dois elementos principais: *federado* e *federação* [GLO96].

- **Federação**: é um conjunto, provido de um nome, contendo: entidades que interagem entre si, um modelo de objetos comum e uma infra-estrutura comum de comunicação. Estes componentes são usados de forma conjunta para se atingir algum objetivo específico. A criação da federação ocorre através da solicitação feita pela primeira entidade que deseja participar da simulação. As demais entidades irão apenas se juntar a esta federação já criada.

- **Federado**: é um membro de uma federação HLA. Todas as aplicações participantes de uma federação são chamadas federados. Um federado pode ser uma entidade ativa, coletor de dados, gerente de federação ou observador passivo.

Além destes dois conceitos fundamentais, federado e federação, existem ainda quatro termos essenciais para o correto entendimento da arquitetura HLA:

- **Objeto**: é uma entidade de um domínio que está sendo simulada em uma federação. Esta entidade deve ser de interesse de mais de um processo. O processo que implementa um determinado objeto deve publicá-lo, ou seja, divulgar a sua existência e de seus atributos para os demais federados que fazem parte da federação.

- **Atributo**: é um atributo definido no modelo de objetos da federação (*Federation Object Model- FOM*) associado com cada instância de uma classe de objetos.

- **Interação**: é um evento não persistente gerado por um federado e recebido pelos outros federados através de uma infra-estrutura de comunicação.

- **Parâmetro**: é um parâmetro definido no modelo de objetos da federação (FOM) associado com cada instância de uma classe de interações (métodos).

Uma federação é formada por um conjunto de federados que possuem objetos com atributos. Os federados registram os seus objetos e publicam os respectivos atributos. Da mesma forma, os federados podem descobrir os objetos dos demais federados e subscrever os respectivos atributos. Esta interação entre os federados e a federação é mostrada na figura 7 [PRO98].

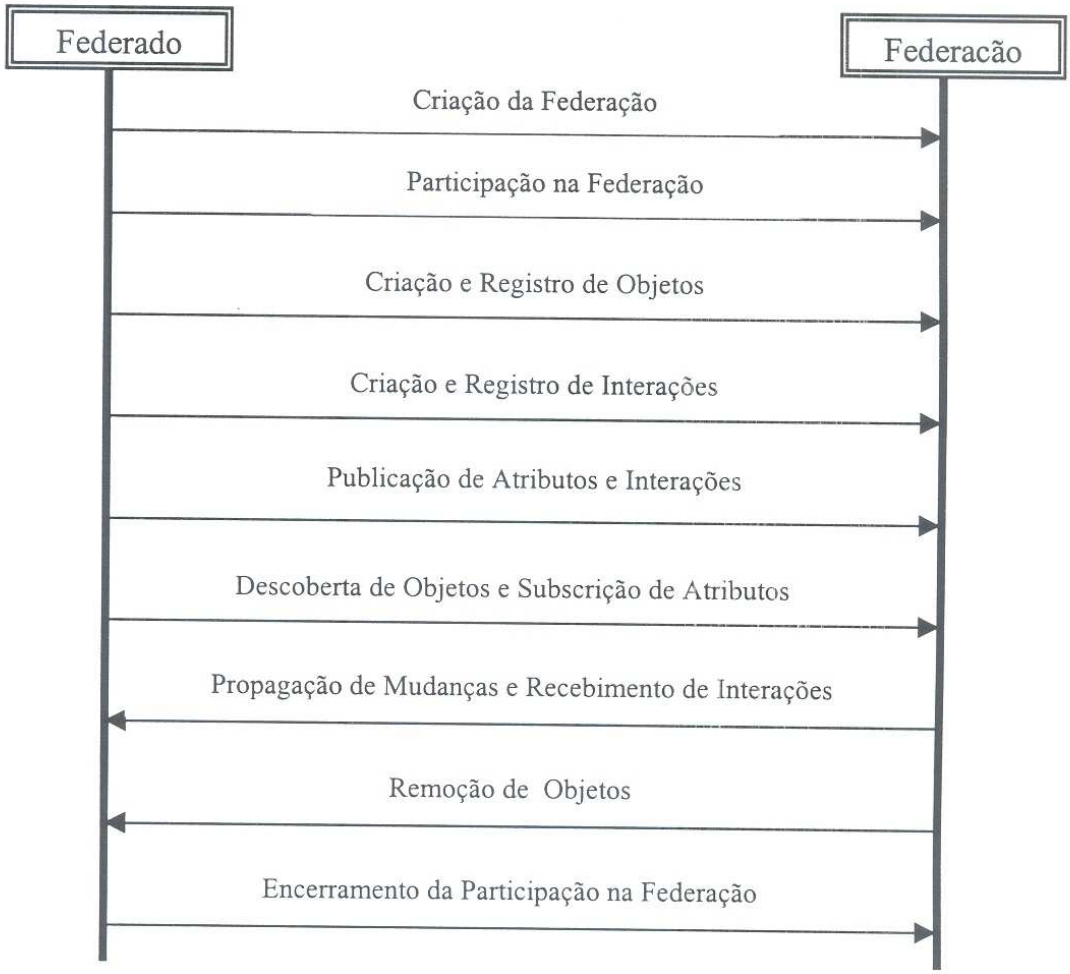

Figura 7 : Roteiro da Interação entre os Federados e a Federação

Cada vez que ocorre uma alteração em um atributo de uma instância de um objeto, esta alteração é difundida (propagada) para todos os federados que subscrevem este atributo.

Além da comunicação através das alterações de atributos de objetos, os federados podem enviar e receber interações que não são persistentes, conforme é mostrado na figura 7.

A HLA provê uma arquitetura para modelagem e simulação que garante a interoperabilidade entre as simulações e o reuso dos seus componentes. A HLA é composta por três componentes básicos:

- *Object Model Templates* (OMT) [OMT96]
- *HLA Compliance Rules* [COM96]
- *Runtime Infrastructure* (RTI) [RTI96]

Cada grupo de federados que interagem entre si forma uma federação. Cada federação é composta ainda por um modelo de objetos comum (OMT) e uma infra-estrutura de suporte de execução (RTI). Dos três componentes da HLA, a RTI e as regras de conformidade (*Compliance Rules*) não mudam em qualquer simulação HLA, contudo cada federação deve definir como será feita a troca de dados e eventos entre as simulações [CAL96].

2.3.1 Object Model Templates (OMT)

Um modelo de objetos consiste em uma especificação de objetos relativos a um determinado sistema, incluindo uma descrição dos atributos dos objetos e uma descrição dos relacionamentos estáticos e dinâmicos entre os objetos. Cada federação define o formato e o conteúdo dos dados e eventos que serão trocados pelos federados. Esta definição é feita através de *Object Model Templates* (OMT). O OMT é usado para descrever os objetos que compõem a federação, bem como seus atributos e relacionamentos.

Dentro da arquitetura HLA, cada federação deve possuir um modelo de objetos chamado *Federation Object Model* (FOM). O FOM contém, basicamente, uma descrição de todas as informações compartilhadas (objetos - atributos, interações - parâmetros) que são essenciais para uma determinada federação. Todas as informações trocadas entre os federados são parametrizadas para a RTI. Os objetos e interações são descritos em um arquivo de configuração (*Federation Execution Data* – FED) que é lido em tempo de execução pela RTI. Para cada objeto são descritos aos seus respectivos atributos, protocolo de transporte (confiável ou não) a ser usado e tipo de mensagem (com ou sem estampilha de tempo).

Além do FOM, existe ainda outro modelo de objetos chamado *Simulation Object Model* (SOM) que descreve os objetos, atributos e interações em uma determinada simulação que podem ser usados externamente em uma federação.

O OMT é formado pelos seguintes componentes [HLA97]:

- **Tabela de identificação do modelo de objetos**: contém informações que identificam o nome, versão, data de criação, propósito, domínio e criador do OMT.

- **Tabela de estruturas de classes de objetos**: contém informações sobre todas as classes de objetos definidas no modelo, bem como a hierarquia entre elas.

- **Tabela de estruturas de classes de interações**: contém informações sobre todas as classes de interações definidas no modelo, bem como a hierarquia entre elas.

- **Tabela de atributos**: contém informações sobre todos os atributos das classes de objetos definidas no modelo. Para cada atributo é especificado o seu nome, tipo de dado, cardinalidade, unidade, resolução, frequência de atualização e outras informações que definem completamente os atributos.

- **Tabela de parâmetros**: contém informações sobre todos os parâmetros das classes de interações definidas no modelo. Para cada parâmetro é

especificado o seu nome, tipo de dado, cardinalidade, unidade e resolução outras informações que definem completamente os parâmetros.

- **Tabela de espaço de roteamento**: contém informações sobre todos os espaços de roteamento definidos no modelo. Para cada espaço é especificado o seu nome, dimensão, tipo, abrangência e demais informações que caracterizam um espaço.

- **Vocabulário FOM e SOM**

Para tornar mais claro o entendimento do papel dos componentes do OMT dentro do desenvolvimento e execução de federações HLA, a seguir será apresentado um pequeno exemplo fornecido pelo *DMSO (Defense Modeling & Simulation Office*) ligado ao DoD [DAH98].

- **Tabela de identificação do modelo de objetos**:

Tabela de identificação do modelo de objetos	
Categoria	**Informação**
Nome	
Versão	
Data	
Propósito	
Domínio de Aplicação	
Patrocinador	
POC	
Organização POC	
Telefone POC	
Email POC	
Neste exemplo	
Nome	Simulação de Ataque SOM
Versão	1.0
Data	01 Julho 1999
Propósito	Servir de exemplo de modelo de objetos para uma simulação de ataque

- **Tabela de estruturas de classes de objetos:**

Tabela de estruturas de classes de objetos			
<classe>(<ps>)	[<classe>(<ps>)]	[<classe>(<ps>)]	[<classe>(<ps>)] | [<ref>]
		...	...
	[<classe>(<ps>)]	[<classe>(<ps>)]	[<classe>(<ps>)] | [<ref>]
		...	...
	...	...	...
Neste exemplo			
Veículo aéreo (S)	Asa fixa (S)	Caça de ataque (S)	F-14 (PS)
			F-16 (PS)
		Bombardeiro (S)	B-1 (PS)
			B-2 (PS)

- **Tabela de estruturas de classes de interações:**

Tabela de estruturas de classes de interações			
<classe>(<isr>)	[<classe>(<isr>)]	[<classe>(<isr>)]	[<classe>(<isr>)] | [<ref>]
		...	...
	[<classe>(<isr>)]	[<classe>(<isr>)]	[<classe>(<isr>)] | [<ref>]
		...	...
	...	...	...
Neste exemplo			
Detonação de bomba (S)	Detonação de bomba em alvo marítimo (R)	Em navio (R)	Detonação em cruzador (IR)
			Detonação em destroyer (IR)
		Em submarino (IR)	Detonação em sub. nuclear (IR)
			...
	Detonação de bomba em alvo terrestre (R)	Em Tanque (IR)	...
		Em Caminhão (IR)	...

- **Tabela de atributos:**

Tabela de atributos												
Objeto	Atributo	Tipo de dado	Card.	Un.	Res.	Pre.	Pre.C.	At.	At.C.	T/A	U/R	Rot
<classe>	<atributo>	<tipodata>										
		...										
	...	<tipodata>										
		...										
Neste exemplo												
Tanque	Velocidade	Double	1	m/seg	0.01	0.01	none	Per.	10 Hz	TA	UR	n/a
	Posição	Rec_type	1	n/a	n/a	n/a	n/a	Per.	10 Hz	TA	UR	Lo.

Onde:

Card. = Cardinalidade
Un. = Unidade
At.C. = Condição de Atualização
Lo. = Localização
Res. = Resolução
Pre. = Precisão
Rot. = Espaço de Roteamento
N/a = Não disponível
Pre.C. = Condição de Precisão
At. = Atualização
Per. = Periódica

- **Tabela de parâmetros:**

Tabela de parâmetros								
Interação	Parâmetro	Tipo de dado	Card.	Un.	Res.	Pre.	Pre.C.	Rot.
< interação >	< parâmetro >	<tipodata>						
		...						
	...	<tipodata>						
		...						
Neste exemplo								
Detonação de bomba	Localização	Rec_type	1	n/a	n/a	n/a	n/a	n/a
	Tamanho	Short int	1	lbs	1.0	perfeito	sempre	n/a

Onde:

Card. = Cardinalidade
Un. = Unidade
Rot. = Espaço de Roteamento
N/a = Não disponível
Res. = Resolução
Pre. = Precisão
Lo. = Localização
Pre.C. = Condição de Precisão
Per. = Periódica

- **Tabela de espaço de roteamento:**

Tabela de espaço de roteamento					
Espaço	Dimensão	Tipo de dimensão	Faixa/Conjunto	Unidade	Função de Normalização
<r_espaço	< dimensão >	<tipoa>			
		...			
	...	<tipo>			
		...			
Neste exemplo					
Localização	X_dim	float	(0-100)	km	Linear (X)
	Y_dim	float	(0-100)	km	Linear (Y)

2.3.2 HLA Compliance Rules

As regras de conformidade definem dez condições básicas que devem ser seguidas para que a simulação esteja de acordo com a especificação HLA. As regras definem as responsabilidades e relacionamentos entre os componentes de uma federação HLA, incluindo a própria federação, os federados e a RTI. Das dez regras, cinco são aplicáveis as federações e as outras cinco são aplicáveis aos federados.

As regras são [HLA97]:

- **Regra 1**: As federações devem ter um modelo de objetos (*Federation Object Model* – FOM) documentado de acordo com o HLA *Object Model Template* (OMT).

- **Regra 2**: Em uma federação, todas as representações de objetos devem estar nos federados e não na infra-estrutura de execução (RTI).

- **Regra 3**: Durante a execução de uma federação, toda troca de dados (alterações de atributos e interações) entre os federados, definida no FOM, deve ocorrer via RTI.

- **Regra 4**: Durante a execução de uma federação, os federados interagem com a RTI de acordo com a especificação da interface HLA.

- **Regra 5**: Durante a execução de uma federação, um atributo de uma instância de objeto deve pertencer a apenas um federado de cada vez.

- **Regra 6**: Os federados devem ter um modelo de simulação de objetos (*Simulation Object Model* – SOM) documentado de acordo com o HLA *Object Model Template* (OMT).

- **Regra 7**: Os federados devem poder atualizar e/ou propagar qualquer alteração de atributos dos objetos do seu modelo (SOM). Devem, também, poder enviar e/ou receber interações externas dos objetos SOM, conforme especificado no seu modelo SOM.

- **Regra 8**: Os federados devem poder transferir e/ou aceitar a posse de atributos dinamicamente durante a execução da federação, conforme especificado no seu modelo SOM.

- **Regra 9**: Os federados devem poder variar as condições nas quais eles atualizam os atributos dos objetos, conforme especificado no seu modelo SOM.

- **Regra 10**: Gerenciamento de Tempo. Os federados devem poder gerenciar o seu tempo local de forma que eles possam coordenar a troca de dados com outros membros da federação. As simulações dentro de uma federação devem gerenciar o tempo de forma que aparente existir um relógio único. Internamente, a simulação gerencia o tempo com quiser, porém deve manter os compromissos temporais com outras simulações dentro da federação.

2.3.3 Runtime Infrastructure (RTI)

As simulações interagem com a infra-estrutura de execução *(Runtime Infrastructure* - RTI) através de interfaces bem definidas, conforme mostra a figura 8. De uma maneira geral, a RTI pode ser vista como um sistema operacional distribuído genérico que provê serviços de interface comuns durante a execução de uma federação.

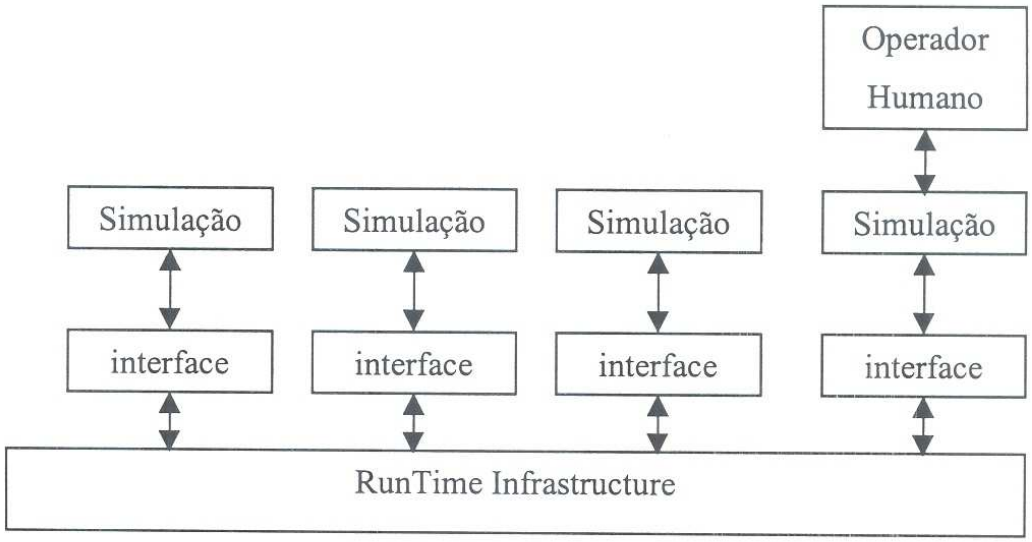

Figura 8 : Visão Conceitual do HLA RTI

O acesso aos serviços oferecidos pela RTI é definido através da especificação de interfaces HLA.

A infra-estrutura de execução (RTI) provê serviços de comunicação e coordenação aos federados. Toda a comunicação na federação deve ser feita via RTI. Desta forma os federados podem estar localizados em qualquer computador conectado via rede, cabendo à RTI prover a comunicação entre eles.

O desenvolvimento da RTI segue um planejamento que prevê duas fases distintas. A primeira consiste no desenvolvimento da versão RTI 1.0 incentivada pelo DoD para garantir a viabilidade técnica da proposta da HLA. Este esforço de desenvolvimento começou em 1996 e a primeira versão disponibilizada apareceu em 1997. Esta versão inclui a maioria dos serviços definidos na especificação de interfaces HLA. A versão RTI 1.0 corresponde a versão 1.1 da especificação de interfaces HLA (*HLA Interface Specification – I/F Spec*). A próxima versão a ficar

pronta foi a RTI 1.3, disponível primeiramente para a plataforma UNIX/Solaris e que corresponde a versão 1.3 desta mesma especificação.

A segunda fase do desenvolvimento da RTI tem como objetivo lançar uma versão 2.0 desenvolvida pela iniciativa privada, incluindo todos os serviços especificados. Contudo, a versão RTI 2.0 também irá implementar a versão 1.3 da especificação de interfaces HLA (*I/F Spec*).

2.3.3.1 Componentes da RTI:

A implementação da RTI 1.3 é feita através de três módulos distintos [PRO98]:

- RTI *Executive process (RtiExec)*
- *Federation Executive process (FedExec)*
- Biblioteca libRTI

Estes módulos podem ser executados em uma única estação isolada (*standalone*) ou podem ser distribuídos por diferentes estações conectadas via rede TCP/IP. Cada módulo desempenha uma função distinta dentro do contexto da RTI que serão sumarizadas a seguir. A figura 9 ilustra as interações entre os módulos da RTI 1.3.

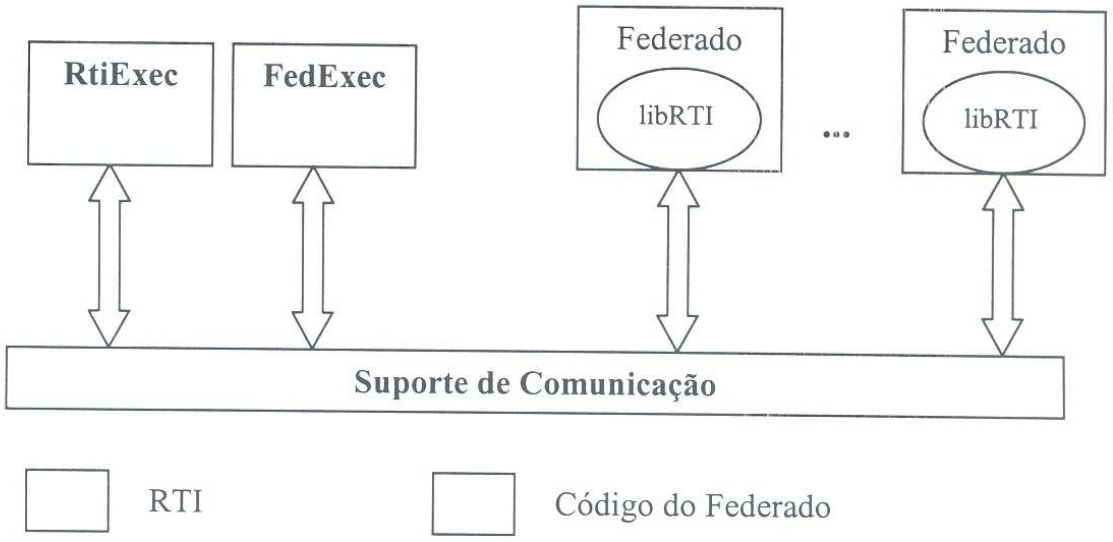

Figura 9 : Componentes da RTI

Processo RtiExec:

É responsável pela criação, gerenciamento e destruição da execução das federações. O processo *RtiExec* é executado em apenas uma estação da rede. Este processo global monitora uma porta lógica do protocolo de comunicação definida para ser o canal de comunicação entre a RTI e os federados. Cada federado se comunica com o *RtiExec* para inicializar os componentes da RTI.

Além de gerenciar a criação e destruição das execuções de federações (*FedExecs*), o *RtiExec* proporciona a execução de várias federações ao mesmo tempo, desde que elas possuam nomes lógicos diferentes.

O *RtiExec* possui, ainda, uma interface (console) que permite a interação do operador via comandos. Esta interface permite que todas as federações ativas sejam listadas ou removidas, entre outras funcionalidades.

Processo FedExec:

Cada execução de uma federação implica na execução de um processo *FedExec*, que contém as definições globais de uma determinada federação. Este processo (*FedExec*) é responsável pela inclusão e exclusão de federados em uma determinada federação. Cada *FedExec* gerencia uma única federação. Dentro de cada federação podem existir muitos federados que podem trocar dados via *FedExec*. O processo *FedExec* é criado pelo primeiro federado que irá participar da federação, com o nome lógico fornecido como parâmetro. Cada federado, ao participar de uma federação possui um identificador único pelo qual é referenciado.

Biblioteca libRTI:

O terceiro componente da RTI é uma biblioteca que estende seus serviços para os federados. Os desenvolvedores de federados utilizam esta biblioteca para ter acesso os serviços disponibilizados pela RTI. A biblioteca *libRTI* é uma biblioteca C++ que disponibiliza os serviços especificados na HLA *I/F Spec* para os federados.

Os federados usam a *libRTI* para se comunicar com o *RtiExec*, com um *FedExec* e com os outros federados. Dentro da *libRTI* existem classes que possibilitam esta interação. Existem duas classes principais: *RTIambassador* e *FederateAmbassador*. A classe *RTIambassador* contém os serviços providos pela RTI. Desta forma, todos os pedidos feitos pelo federado à RTI são feitos via chamada de métodos do *RTIambassador*.

Em contra-partida, a classe *FederateAmbassador* é uma classe abstrata que identifica as funções de retorno (*callback*) que cada federado é obrigado a fornecer à RTI. Assim, cabe a cada federado implementar a funcionalidade declarada na classe *FederateAmbassador*. Uma instância desta classe é necessária para que o federado participe de uma execução de federação. As funções de retorno atuam como um mecanismo para a federação sinalizar o federado em resposta a uma requisição do federado.

A HLA *I/F Spec* divide os inter-relacionamentos entre o federado e a federação em seis áreas ou categorias de gerenciamento. Cada categoria apresenta uma série de serviços que podem ser invocados pelos federados, pela federação ou pela RTI durante o ciclo de vida da federação, conforme mostrado na figura 10.

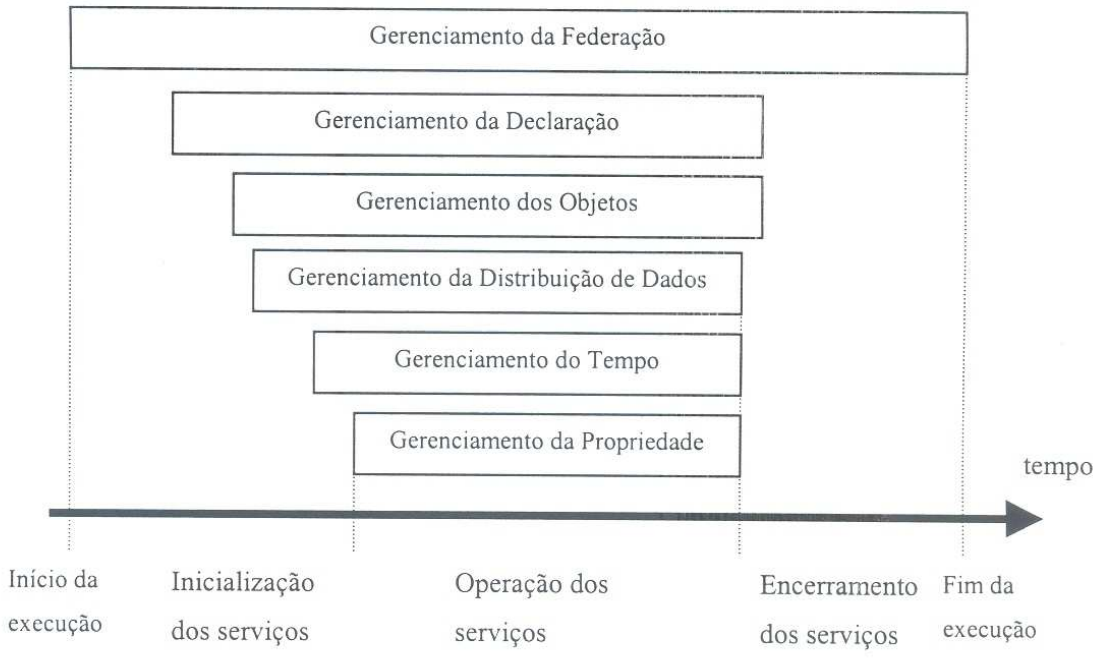

Figura 10 : Ciclo de Vida - FedExec

Os serviços comuns oferecidos pela RTI [HLA97] mostrados na figura 9 são detalhados a seguir:

- Gerenciamento de Federação (*Federation Management*)

  Contém 20 serviços que englobam:

  - criação e encerramento de execuções de federações
  - participação e separação de federados em execuções de federações
  - pontos de controle, sincronização e re-inicialização

- Gerenciamento de Declaração (*Declaration Management*)

  Contém 12 serviços que englobam:

  - declaração de intenção de receber ou gerar informações sobre o estado de objetos, bem como interações.

- Gerenciamento de Objeto (*Object Management*)

  Contém 17 serviços que englobam:

  - Criação, descoberta e encerramento de instâncias de objetos
  - Envio de atualizações em atributos de objetos e envio de interações
  - Recebimento de atualizações em atributos de objetos e recebimento de interações

- Gerenciamento de Propriedade (*Ownership Management*)

  Contém 16 serviços que englobam:

  - transferência de propriedade sobre atributos de objetos

- Gerenciamento de Tempo (*Time Management*)

  Contém 23 serviços que englobam:

  - coordenação do avanço do tempo lógico
  - relacionamento do tempo lógico com o real

- Gerenciamento de Distribuição de Dados (*Data Distribution Management*)

  Contém 13 serviços que englobam:

- suporte ao gerenciamento eficiente da distribuição dos dados através da criação e manutenção de espaços de roteamento.

O foco deste trabalho é a categoria referente ao gerenciamento de tempo. Os serviços que fazem parte desta categoria têm como objetivo gerenciar o tempo lógico através do estabelecimento de políticas que regem o avanço do tempo dos federados. O gerenciamento de tempo é fundamental dentro do contexto de simulações distribuídas, onde o processamento dos eventos na ordem correta é vital para a garantia dos aspectos temporais e causais da simulação.

3 Gerenciamento de Tempo em HLA

O serviço de gerenciamento de tempo da arquitetura HLA está intrinsecamente relacionado a outro serviço provido pela RTI, o serviço de gerência de objetos. Antes de discutirmos os aspectos referentes ao gerenciamento de tempo é importante apresentar uma pequena descrição da a gerência de objetos.

3.1 Gerenciamento de Objetos

Quando um federado requisita o avanço do seu tempo lógico à RTI, ele irá receber como retorno uma série de mensagens. Após o término do envio das mensagens disponíveis naquele determinado momento de tempo, a RTI irá autorizar o avanço do tempo lógico do federado. Esta troca de mensagens contendo eventos entre o federado e a RTI é importante, pois pode envolver mensagens que irão alterar objetos e seus atributos que poderão, no futuro, ser alvo de um procedimento de *rollback*.

Cada federado possui uma série de objetos com atributos que são registrados na RTI e consequentemente tornam-se públicos dentro da federação. Para atualizar um ou mais atributos associados com uma instância registrada de um objeto, o federado precisa preparar uma estrutura de dados chamada *AttributeHandleValuePairSet* (AHVPS). Esta estrutura contém um conjunto de tamanho variável contendo atributos e seus respectivos valores. É através desta estrutura (classe abstrata) que os dados são trocados entre os federados.

Outra forma de comunicação entre os federados é através de interações (*interactions*). As interações são similares aos objetos, sendo que a diferença fundamental entre eles é a persistência. Os objetos são persistentes e as interações não, cabendo ao analista escolher a melhor forma de representar um elemento dentro do modelo de simulação. No caso das interações, os dados são passados de um federado para outro através de parâmetros e não atributos. Logo, existe uma estrutura de dados chamada *ParameterHandleValuePairSet* (PHVPS) que desempenha um papel similar ao AHVPS.

3.2 Arquitetura de Gerenciamento de Tempo

Apesar de ser um serviço opcional oferecido pela arquitetura HLA, o gerenciamento de tempo é, sem dúvida, um dos mais importantes serviços envolvidos em uma simulação distribuída. Dentro de uma federação, os federados trocam eventos que podem, freqüentemente, estar associados com o tempo através de estampilhas. Como os federados evoluem de forma assíncrona, cabe à RTI garantir o comportamento causal através da ordenação das mensagens e do controle de sua entrega aos federados.

A arquitetura HLA provê serviços de comunicação entre os federados nos quais as mensagens com os eventos são entregues com latência mínima. Quando os eventos são enviados, via rede, de um federado para outro eles estão sujeitos a atrasos que não existem no mundo real que está sendo modelado. Estes atrasos podem fazer com que os eventos cheguem aos federados atrasados ou em ordem incorreta, o que pode comprometer toda a simulação. Uma conseqüência comum deste problema é a perda da causalidade entre os eventos. Neste caso, os efeitos se manifestam antes das suas respectivas causas. Ainda devido ao atrasos ocasionados pela rede de comunicação, outro problema que pode acontecer é o recebimento de eventos em uma ordem incorreta. Se estes eventos forem processados nessa ordem pelos federados todo o resultado da simulação estará comprometido. Esta entrega não determinística dos eventos faz com que a execução de uma mesma simulação a partir de um mesmo estado inicial e eventos externos possam apresentar resultados totalmente diferentes.

Para evitar estes tipos de anomalias temporais, a HLA propõe o uso de estampilhas relacionadas a cada evento, que permitam o estabelecimento de uma ordem entre os eventos. Cada vez que um evento é criado, uma estampilha com o tempo simulado deste momento é atribuída a ele. Essas estampilhadas serão usadas para garantir que os eventos serão entregues aos federados na ordem correta, garantindo o princípio da causalidade. Outro ponto muito importante dentro da arquitetura HLA é a garantia de que um federado não irá receber nenhum evento no seu passado. Em outras palavras, o federado não irá receber nenhum evento com estampilha menor que o seu valor de tempo atual. A atribuição de estampilhas aos

eventos, a sua ordenação e entrega aos federados respeitando o tempo lógico de cada federado são atribuições dos serviços de gerenciamento de tempo que por sua vez obedecem à política de cada federado.

Os serviços de gerenciamento de tempo coordenam o avanço do tempo lógico da federação, de forma que os federados possam trocar eventos mesmo que eles estejam seguindo diferentes políticas de tempo. Como "políticas de tempo" entende-se os diferentes relacionamentos entre os federados e o tempo lógico da federação. As políticas de gerenciamento de tempo serão descritas e analisadas no transcorrer deste capítulo.

Segundo [FUJ98] um dos principais problemas quando da discussão e análise dos aspectos de gerenciamento de tempo é o correto entendimento do que é tempo. Dentro do contexto de simulações distribuídas existem basicamente três tipos de tempo que devem ser compreendidos e distinguidos:

- **Tempo físico**: corresponde ao tempo real do sistema físico que está sendo modelado pela simulação.

- **Tempo simulado ou lógico**: corresponde à representação do tempo dentro da simulação.

- **Tempo do relógio de parede (*wallclock*):** corresponde ao tempo necessário para a execução da simulação. O tempo do relógio de parede começa a ser marcado quando inicia a execução da simulação e dura o tempo necessário para seu término.

As estampilhas atribuídas aos eventos para garantir o seu correto ordenamento são amostras do tempo simulado. O tempo simulado de um federado representa um ponto dentro do eixo global do tempo da federação [FUJ99]. Em qualquer instante durante a sua execução, o federado estará em um ponto específico do eixo do tempo da federação. Por exemplo, se o tempo simulado do federado for igual a 2.0, isto indica que ele já simulou as primeiras 2 horas do exercício.

Dependendo das características do sistema que está sendo simulado, o avanço do tempo simulado pode estar vinculado ao avanço do tempo do relógio de parede.

Neste tipo de simulação, chamadas *simulações em tempo real* [FUJ98], existe uma relação linear clara entre o tempo simulado e o relógio de parede. Outra abordagem comum dentro das simulações é a sua execução no menor tempo possível independentemente do tempo do relógio de parede. Tipicamente simulações de jogos de guerra são executadas o mais rápido possível de forma a se obter os resultados finais no menor tempo possível. Cada uma destas duas abordagens tem suas aplicações dentro dos modelos de simulações e cabe à arquitetura HLA suportá-las, deixando livre sua escolha, em função das características do sistema que está sendo modelado e simulado.

Dentro da arquitetura HLA, os serviços de gerenciamento de tempo são flexíveis para acomodar as diversas políticas internas usadas para gerenciar o tempo dos federados. Estas políticas foram sendo desenvolvidas durante as últimas décadas para atender as diferentes necessidades das simulações. Um dos pontos principais da arquitetura HLA é permitir que as simulações já desenvolvidas possam ser reusadas, o que inclui, é claro, as suas políticas de gerenciamento de tempo. Assim, para que a arquitetura HLA tenha sucesso é preciso que ela proporcione uma transparência com relação ao tipo de gerenciamento de tempo que está sendo usado dentro do federado, bem como permitir que federados com diferentes políticas possam interagir de forma confiável dentro de uma mesma federação. Desta forma, a HLA não somente suporta diferentes políticas de tempo mas também garante a interoperabilidade entre os federados com diferentes políticas. As políticas de tempo serão descritas na seção 3.3.1.

Na arquitetura HLA, o gerenciamento do tempo dentro de uma federação é realizado pelos federados e pela RTI. Cada uma das partes desempenha um papel importante dentro do esquema de gerenciamento de tempo e a forma do seu relacionamento depende da política de tempo adotada em cada federado.

Cada federado possui a sua própria noção de tempo, ou seja, cada um pode e geralmente tem um tempo lógico distinto. Contudo, os federados que compõe uma federação específica precisam interagir entre si, o que torna fundamental a existência de serviços gerenciadores do avanço do tempo dos federados para garantir a causalidade dos eventos trocados entre eles.

O tempo lógico dos federados avança a medida que a RTI entrega mensagens, ordenadas de acordo com suas estampilhas, e estas são processadas pelos federados.

Dentro de uma federação, o tempo lógico dos federados sempre avança, o que varia são as situações que podem ocasionar este avanço. Quando a RTI termina a entrega das mensagens, os federados recebem uma autorização para avançar a sua referência de tempo local. Para que os requisitos causais e temporais sejam mantidos é fundamental que haja em correto ordenamento das mensagens.

Assim sendo, os dois principais componentes do gerenciamento de tempo em HLA são: ordenação das mensagens e avanço do tempo lógico. Este dois componentes estão fortemente relacionados e serão analisados a seguir.

3.3 Ordenação das Mensagens

Uma parte fundamental do gerenciamento de tempo é feita através da correta ordenação das mensagens vindas dos federados e armazenadas em filas dentro da RTI. As mensagens são enfileiradas de acordo com a existência de estampilhas e com a políticas de tempo utilizadas pelo federado que as enviou e pelo federado que vai receber. As políticas de tempo serão explicadas após a introdução dos conceitos de mensagens RO e TSO.

Para cada federado existem duas filas:

- **Fila FIFO** (*FIFO queue*): nesta fila são armazenadas as mensagens que não precisam ser ordenadas (e consequentemente processadas) de acordo com as suas estampilhas. Este tipo de mensagem é conhecido como *mensagens ordenadas por recepção* (*Receive ordered*, ou simplesmente RO). Esta fila possui um esquema FIFO (*First-In First-Out*), que significa que as mensagens serão entregues ao federado na mesma ordem com que elas foram recebidas pela RTI.

- **Fila TSO**: Fila ordenada pelas estampilhas (*Timestamp ordered queue*): nesta fila as mensagens, com os eventos, são ordenadas e armazenadas de acordo com as estampilhas (*Time Stamp ordered*, ou simplesmente TSO), não importando a ordem do seu recebimento.

A figura 11 [FUJ98] mostra uma visão lógica destas duas filas dentro da RTI e seu relacionamento com o avanço do tempo lógico.

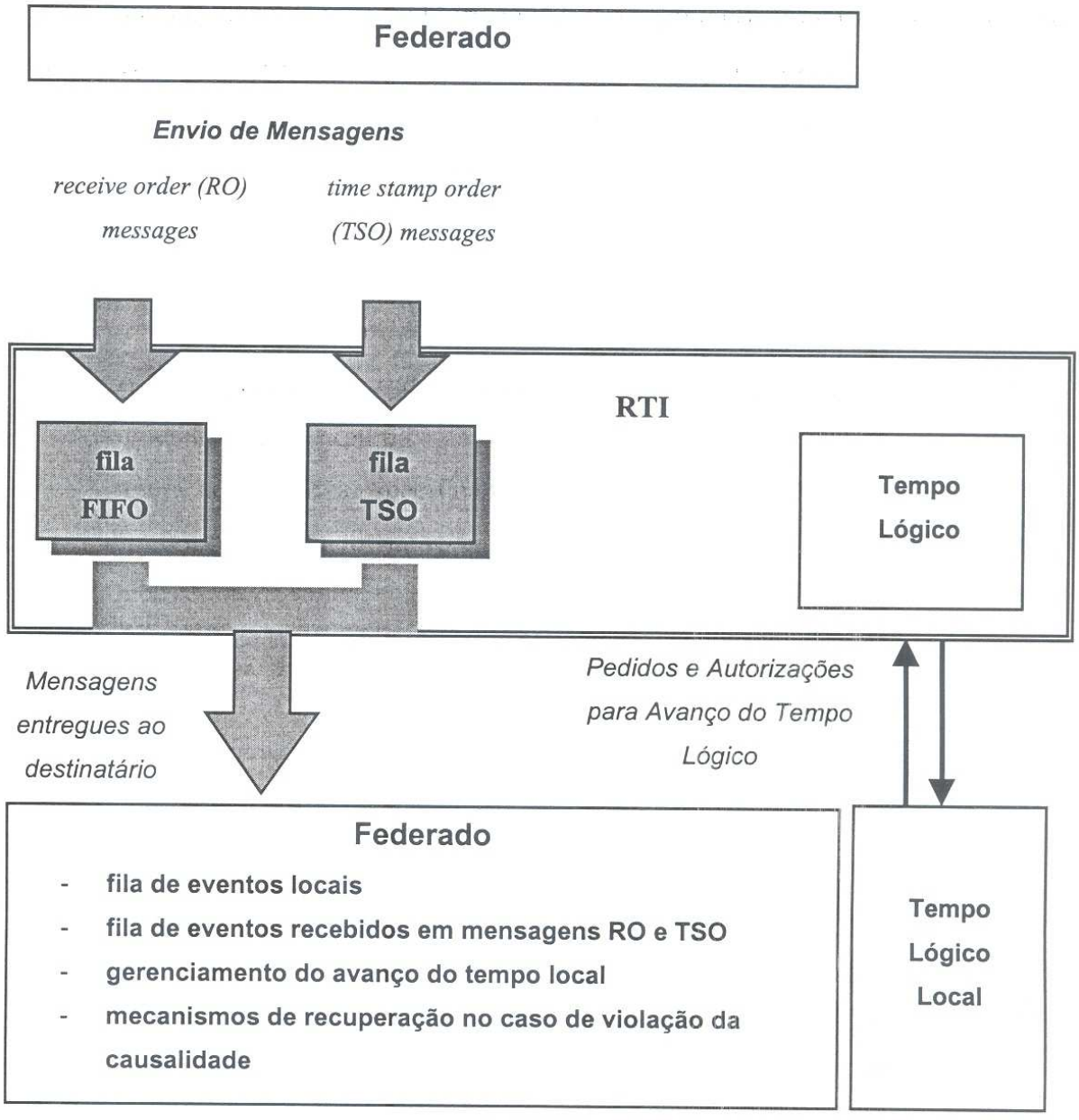

Figura 11 : Gerenciamento de Tempo em HLA

A diferenciação destes dois tipos de mensagens, RO e TSO, é fundamental para o entendimento da gerência de tempo dentro da HLA. Cada vez que uma mensagem RO é recebida pela RTI ela é simplesmente colocada na fila FIFO do federado destinatário. Esta mensagem é imediatamente disponibilizada para o federado. Logo, esta fila FIFO pode ser inteiramente drenada caso o federado disponha de tempo suficiente. Já as mensagens TSO possuem uma estampilha gerada pelo federado

remetente e são armazenadas na fila TSO do federado destinatário. Estas mensagens são entregues aos federados em ordem não decrescente de estampilhas. Porém estas mensagens não serão entregues aos federados até que a RTI possa garantir que não há nenhuma outra mensagem com menor estampilha para este federado e também que nenhum outro federado irá enviar-lhe mensagens com menores estampilhas no futuro. No caso de existirem mensagens com estampilhas idênticas, elas serão entregues ao federado em uma ordem arbitrária, o que pode influenciar no resultado da simulação com um todo. Existem várias técnicas que podem ser utilizadas para resolver este problema [FUJ98] [MEH92] e que fogem ao escopo deste trabalho.

3.3.1 Políticas de Tempo dos Federados

Conforme já mencionado, os federados podem adotar certas políticas em relação ao seu comportamento dentro do esquema de gerenciamento de tempo. Os federados geralmente interagem entre si e com a RTI dentro da federação. Em muitos casos é preciso haver um sincronismo entre a execução dos federados de forma a se obter o resultado desejado na simulação.

Em uma federação, os federados podem assumir comportamentos distintos em relação à sincronização das execuções:

- Comportamento Regulador (*time regulating*):
  Produzem eventos datados (com estampilhas), enviados a outros federados através de mensagens TSO. Federados agindo com esse comportamento podem exercer controle sobre o avanço de tempo de outros federados.

- Comportamento Restrito (*time constrained*):
  Consomem eventos datados, ou seja, consideram as datas presentes nas estampilhas dos eventos recebidos em mensagens TSO. Federados agindo dessa forma têm o avanço de seu tempo local dependente das estampilhas dos eventos recebidos.

Não existe conflito entre esses dois comportamentos. Logo, um federado pode ser regulador, restrito, regulador e restrito e nem regulador e nem restrito. Inicialmente todos os federados não são nem reguladores e nem restritos. Dentro de uma mesma federação podem haver federados que adotem qualquer uma destas quatro políticas de tempo. Dentro da arquitetura HLA, um federado pode adotar uma das políticas simplesmente mudando o estado de duas *flags* lógicas. Existe uma *flag* chamada *time regulating* e outra chamada *time constrained* que indicam qual a política de tempo que está sendo adotada pelo federado. Sempre que um federado quiser alterar a sua política de tempo, ele deve explicitamente solicitar esta alteração à RTI, conforme mostrado na figura 12 [PRO98].

Nesta figura podem ser vistas as classes *RTIambassador* e *FederateAmbassador* (da libRTI) que possibilitam a interação do federado com a RTI. Dentro destas classes existem os métodos que possibilitam que o federados determine o seu comportamento em relação à sincronização das execuções:

RTIambassador	***FederateAmbassador***
enableTimeRegulation	*timeRegulationEnabled*
disableTimeRegulation	
enableTimeConstrained	*timeConstrainedEnabled*
disableTimeConstrained	

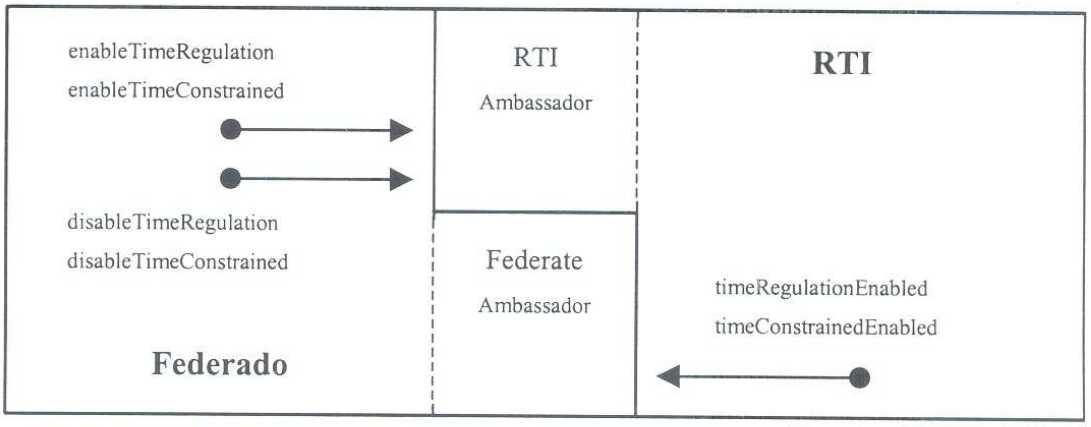

Figura 12 : Políticas de Tempo

A distinção entre estas duas políticas de tempo é fundamental para o esquema de gerenciamento de tempo dentro da arquitetura HLA. Somente um federado regulador pode gerar mensagens TSO e somente um federado restrito pode receber

mensagens TSO. A RTI precisa saber qual é a política de tempo de cada federado para poder gerenciar as filas FIFO (mensagens *Receive Ordered*, RO) e TSO, logo a mudança da política de tempo deve ser controlada pela RTI (*Local RTI Component* – LRC). Se um federado não tem a sua *flag time regulating* assinalada (*set*) quando envia uma mensagem TSO, a RTI irá converter esta mensagem para RO. Da mesma forma, caso um federado cuja *flag time constrained* não esteja assinalada receba uma mensagem TSO, a RTI irá converter esta mensagem para RO.

Desta forma uma mensagem só será tratada como TSO se três condições forem verdadeiras:

- O federado remetente for regulador (*time regulating*)
- O federado destinatário for restrito (*time constrained*)
- A mensagem deve ser identificada como TSO no arquivo de configuração da federação. Por exemplo, as mensagens de atualização de atributo ou de interação seguem a especificação contida no arquivo de configuração (FED), conforme explicado na seção 2.3.1. Neste arquivo está especificado se a mensagem de atualização de atributo ou de interação é uma mensagem RO ou TSO.

Caso uma mensagem não apresente alguma destas três condições, ela será colocada na fila FIFO de mensagens RO do federado destinatário.

3.4 Avanço do Tempo Lógico

O avanço do tempo lógico dos federados é feito de forma explícita, ou seja, o federado solicita à RTI o avanço do seu tempo. Este procedimento é necessário para garantir que o federado não irá receber nenhuma mensagem, evento, com estampilha menor que o seu tempo lógico. A única maneira desta condição ser garantida é através do controle de entrega das mensagens TSO feito pela RTI. Desta forma, o tempo lógico do federado só avança com autorização da RTI.

As operações para avanço do tempo lógico são efetuadas através dos mecanismos apresentados na seção 3.5. Na próxima seção veremos alguns conceitos fundamentais para o entendimento das duas tarefas básicas da RTI [FUJ98] em

relação à entrega das mensagens e consequente avanço do tempo nos federados. As tarefas básicas são:

- Garantir que a entrega das mensagens TSO será feita de acordo com a ordem das estampilhas.
- Garantir que nenhuma mensagem será entregue ao federado com estampilha menor que o tempo lógico atual do federado.

3.4.1 LBTS

Para que a RTI possa liberar o avanço do tempo lógico dos federados, ela precisa calcular o LBTS (*Lower Bound on Time Stamp*) para cada federado. O LBTS indica a menor estampilha que pode ser recebida em uma mensagem TSO por um federado e pode ser vista como uma propriedade monotônica crescente. O LBTS também pode ser analisado como sendo o relógio de entrada do processo lógico [MAZ94], que na terminologia HLA corresponde ao federado.

Depois que o LBTS é calculado para um federado, a RTI pode entregar todas as mensagens TSO contendo estampilhas menores que o LBTS. Desta forma, um federado restrito não pode avançar no tempo além do seu LBTS, ou seja, seu avanço está restrito (*constrained*) ao LBTS. Como a RTI não permite que o federado avance seu tempo lógico além do seu LBTS, a RTI garante que o federado não irá receber nenhuma mensagem no seu passado. O LBTS é mantido dentro da RTI através de um protocolo de sincronização conservador [CHA79], explicado na seção 2.1.1.1.

O cálculo do LBTS para cada federado é feito através de algoritmos [FUJ99] que determinam a menor estampilha de uma mensagem que o federado pode receber agora ou no seu futuro. Para isso a RTI deve verificar, junto aos federados reguladores, qual a menor estampilha possível em uma mensagem TSO baseado no tempo lógico de cada federado. Um federado não pode gerar um mensagem TSO com estampilha menor que o seu tempo lógico. A RTI deve também verificar as estampilhas das mensagens já enviadas e armazenadas nas filas internas do própria RTI e ainda as mensagens em trânsito na rede.

Um federado regulador pode gerar mensagens TSO, logo deve ser considerado no cálculo do LBTS. Já um federado restrito pode receber mensagens TSO, logo ele

precisa do resultado do cálculo do LBTS para que a RTI possam entregar as mensagens com segurança. Todos os federados, restritos ou não, têm um valor de LBTS. Porém, este valor só tem sentido para os federados restritos (*time constrained*) ou aqueles que estão planejando se tornar restritos.

O cálculo do LBTS para cada federado requer significativos recursos computacionais e de rede, logo os federados só devem assinalar as suas *flags* (*time constrained* e *time regulating*) quando realmente necessitarem dos serviços de gerenciamento de tempo da arquitetura HLA.

3.4.2 *Lookahead*

Outro conceito fundamental dentro dos serviços de gerenciamento de tempo é o *lookahead* (L), que é um valor usado para determinar a menor estampilha de uma mensagem TSO que um federado pode gerar no futuro. O conceito do *lookahead* já existe desde a época dos estudos para otimização de simulações paralelas a eventos discretos [NIC94][FER94][FUJ90] sendo então incorporado no serviço de gerenciamento de tempo da HLA.

O federado regulador assume o compromisso de que todas as mensagens TSO geradas por ele irão ocorrer em uma data t, onde $t \geq t_{atual} + L$. Isto significa que o federado deve ser capaz de "olhar adiante" L unidades de tempo para prever quais eventos irá gerar e agendar estes eventos antes. Desta forma, o *lookahead* pode ser visto como uma promessa que o federado regulador não enviará nenhuma mensagem com estampilha menor do que o seu tempo lógico atual mais o seu valor de *lookahead*.

Cada federado regulador estabelece um valor para o *lookahead* (L) no momento em que se torna regulador. Este valor pode, contudo, ser alterado dinamicamente através de chamadas de métodos da RTI. O aumento do valor para o *lookahead* pode ser feito de forma imediata, porém a sua diminuição não pode acontecer de forma imediata. Se o *lookahead* for decrementado de $\underline{n}$ unidades de tempo, o federado deve avançar $\underline{n}$ unidades até que o novo valor do *lookahead* tenha efeito.

A figura 13 mostra primeiramente (a) um gráfico no qual o valor do *lookahead* vale L unidades de tempo. Já no gráfico (b) o valor do *lookahead* foi alterado dinamicamente para L', onde L' = L - **n**.

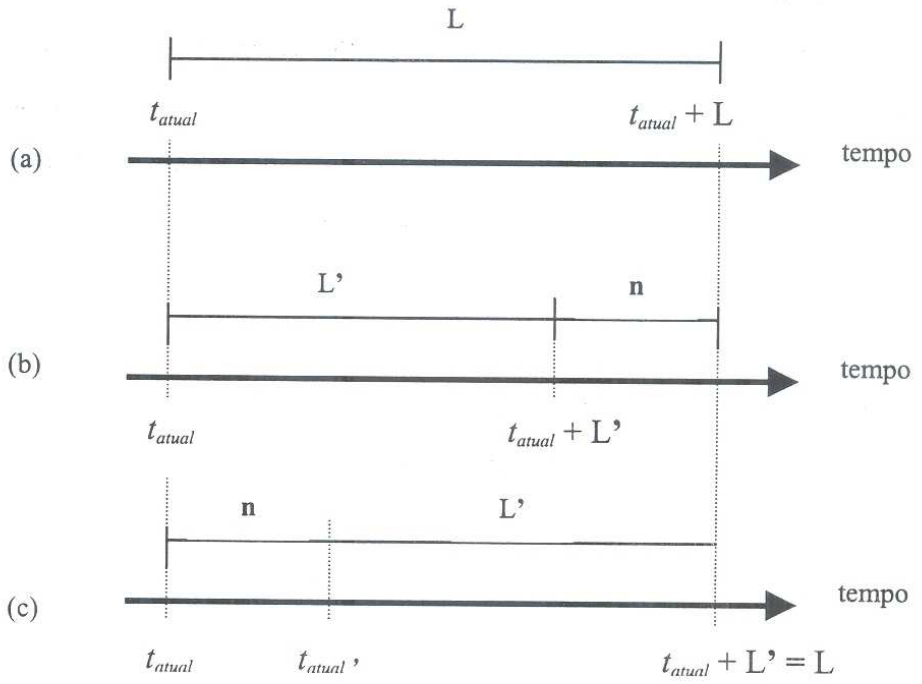

Figura 13 : Redução do valor do *lookahead*

A figura 13 (a) representa o compromisso assumido pelo federado de gerar mensagens TSO com data maior do que t_{atual} + L. Quando o valor do *lookahead* é reduzido para t_{atual} + L', esta alteração só começará a ter efeito depois que o tempo lógico do federado (t_{atual} ') avançar ***n*** unidades de tempo, pois neste caso L' + n = L como mostra a figura 13 (c), o que garante o compromisso anteriormente assumido.

A principal questão envolvendo o *lookahead* é a determinação de um valor ideal para cada tipo de simulação. A atribuição do valor para o *lookahead* é essencial para o desempenho de toda a federação [FUJ89] e está freqüentemente associado a detalhes do modelo de simulação adotado. Muitas vezes o valor do *lookahead* pode ser obtido de limitações físicas do sistema que está sendo modelado e que podem ser representadas dentro da simulação através de valores de *lookahead* [JHA96]. Por exemplo, o tempo em que um evento demora para acontecer no mundo real, ou seja, a partir de uma causa quanto tempo demora para os seus efeitos aparecerem. Se os efeitos demorarem 5 segundos, este seria um bom valor de *lookahead*.

O valor do *lookahead* é um número inteiro a partir do valor zero, porém a utilização de *lookahead* zero implica em uma série de restrições ao federado [PRO98] [FUJ96]. Nos primeiros protótipos da arquitetura HLA não era possível atribuir um valor zero para o *lookahead* pois a sua utilização pode não garantir a recepção e processamento de todas as mensagens com estampilhas iguais ao tempo lógico desejado. Contudo, a proibição do uso de *lookahead* zero restringe muito as simulações sobre HLA segundo experimentos feitos com as primeiras versões da implementação da arquitetura [FUJ98]. A especificação da arquitetura foi então modificada para suportar valores zero para o *lookahead*. Basicamente, dois novos métodos ou serviços foram incorporadas à especificação, o que tornou possível que os federados pudessem criar mensagens TSO com *lookahead* zero. Esses métodos, *Next Event Request Available* (NERA) e *Time Advance Request Available* (TARA), serão apresentados na próxima seção juntamente com os demais métodos utilizados dentro do esquema de avanço de tempo.

A figura 14 [PRO96] apresenta o diagrama de dois eixos que resume os conceitos de federado regulador (*time regulation*), federado restrito (*time constrained*), LBTS e *lookahead* (L) são mostrados graficamente. Observe que os conceitos de LBTS e *lookahead* são válidos para os federados restritos e reguladores respectivamente.

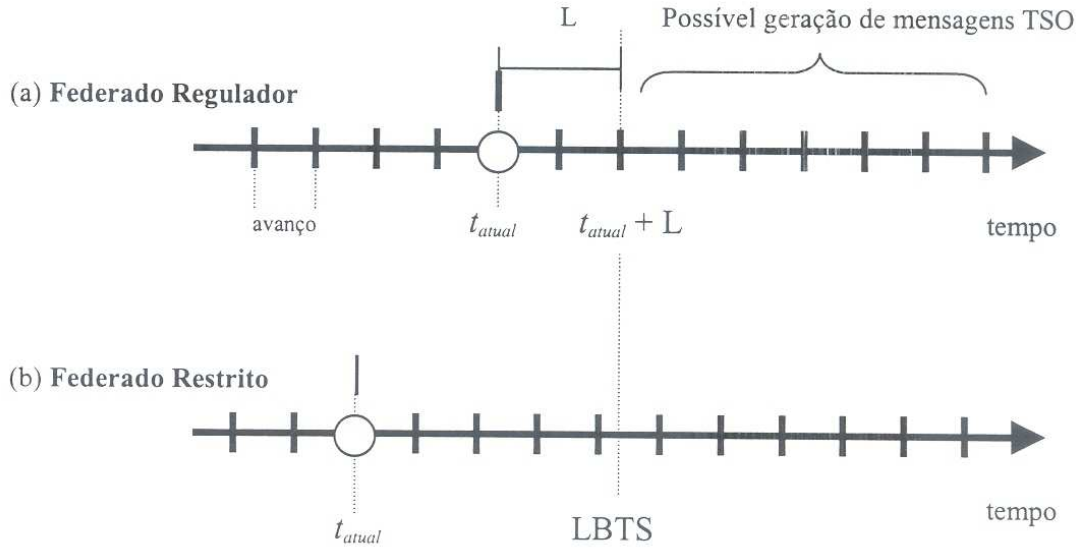

Figura 14 : Diagrama de Dois Eixos

3.5 Mecanismos de Avanço do Tempo

Como já foi citado anteriormente, um federado não pode avançar o seu tempo lógico sem a autorização da RTI. Isto é necessário para que a RTI possa garantir que não irá entregar mensagens ao federado no seu passado, ou seja, garantir o respeito a causalidade.

Dentro da arquitetura HLA, o gerenciamento do tempo de um federado é feito em três etapas distintas, envolvendo o federado (métodos das classes *RTIAmbassador* e *FederateAmbassador*) e o própria RTI. A figura 15 mostra como é feita esta interação:

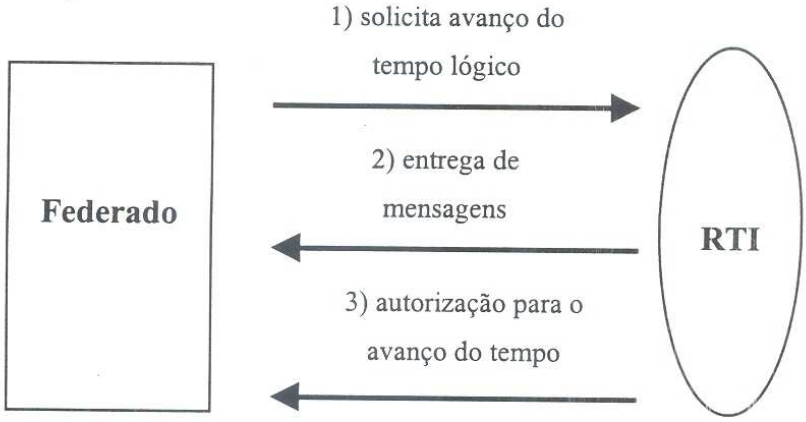

Figura 15 : Etapas envolvidas no gerenciamento de tempo de um federado

Existem dois mecanismos principais para o federado solicitar o avanço do tempo: *Time Advance Request* (TAR) e *Next Event Request* (NER) [PRO98]. Porém a sua utilização depende do mecanismo de gerenciamento de tempo interno de cada federado, como veremos a seguir.

Devido à grande aplicabilidade das simulações, os requisitos de gerência de tempo também podem variar bastante de uma simulação para outra. As simulações podem representar treinamentos, jogos de guerra, tráfego urbano, etc, que implicam claramente em diferentes mecanismos para controle do tempo. Os principais tipos de mecanismos usados para gerenciamento de tempo são brevemente descritos a seguir:

- ***Event driven***: neste mecanismo os federados processam os eventos locais, gerados internamente, e eventos externos gerados pelos demais federados de acordo com a ordem das estampilhas. O evento com a menor estampilha será

processado primeiro e o tempo do federado irá avançar para o tempo desta estampilha e assim sucessivamente até não haver mais eventos para serem processados. Dentro dos serviços de gerenciamento de tempo definidos na arquitetura HLA existem mecanismos que possibilitam este tipo de abordagem. Existe um serviço de entrega de mensagens ordenadas pelas suas estampilhas para os federados, o que garante a ordenação das mensagens. Os federados podem explicitamente solicitar à RTI o avanço no seu tempo e a RTI garante que nenhum evento com menor estampilha será enviado a este federado.

- ***Time stepped***: neste mecanismo o avanço do tempo do federado é feito através de intervalos fixos (passos) do tempo simulado. A simulação avança para o próximo passo de tempo apenas após a conclusão de todo o processamento referente ao passo atual. Esta abordagem também é suportada pelos serviços de gerenciamento de tempo da arquitetura HLA. Neste caso o problema consiste em determinar quando não há mais eventos internos e externos no passo de tempo atual. O federado solicita o avanço de tempo à RTI e da mesma forma a RTI irá garantir que não há mais eventos neste passo, se for o caso, e irá permitir o avanço do tempo do federado.

- ***Wallclock driven***: neste mecanismo a simulação utiliza o próprio tempo do relógio de parede através de algum relacionamento linear, como por exemplo o simulador executa em uma velocidade igual ao triplo do relógio de parede. Neste tipo de mecanismo de gerência de tempo os eventos não precisam ser executados na ordem das suas estampilhas [FUJ98]. Este tipo de gerenciamento de tempo possui requisitos particulares e não se enquadram dentro dos serviços de gerenciamento de tempo disponíveis dentro da arquitetura HLA. Logo, o uso deste mecanismo implica na sua implementação dentro do federado, pelo programador da simulação.

Além destes mecanismos de gerenciamento de tempo existem ainda aqueles utilizados em PADS (*Parallel And Distributed Simulation*) já abordados na seção 2.1.1 deste trabalho. De forma resumida, existem duas estratégias de sincronização: conservadora e otimista. No caso da abordagem conservadora os eventos devem ser

obrigatoriamente processados de acordo com a ordem das suas estampilhas, logo o avanço do tempo está vinculado às estampilhas dos eventos. Esta abordagem é similar ao mecanismo *event driven* dentro da arquitetura HLA. Já no caso da abordagem otimista os eventos podem ser processados fora da ordem das estampilhas; caso haja algum problema de inconsistência existem mecanismos que permitem a sua recuperação. A arquitetura HLA também suporta esta abordagem otimista através de serviços de entrega de todos os eventos para o federado independentemente das suas estampilhas, bem como mecanismos de recuperação (*rollback*) caso necessário. As três abordagens mais comuns para o gerenciamento dos serviços de tempo dentro da arquitetura HLA são:

- *Time stepped*
- *Event driven (conservadora ou pessimista)*
- *Otimista*

A escolha depende dos requisitos de cada federado e do modelo que está sendo utilizado para representar um sistema real. Um federado pode adotar apenas uma abordagem de gerenciamento de cada vez, contudo pode mudar de abordagem no transcorrer da simulação caso seja necessário. Um ponto fundamental dentro da arquitetura HLA é a possibilidade de existirem federados com diferentes mecanismos de gerenciamento de tempo em uma mesma federação. O mecanismo de gerência adotado por cada federado é transparente para os demais federados.

3.5.1 Federados *Time Stepped*

Este tipo de federado executa todos os eventos associados a um determinado passo (*step*) de tempo. O federado solicita o avanço do seu tempo lógico à RTI, que por sua vez envia para o federado uma série de mensagens. Após o processamento dos eventos contidos nas mensagens, a RTI autoriza o avanço de tempo do federado.

Geralmente um federado *time stepped* utiliza a chamada ao método *timeAdvanceRequest* (TAR) [PRO98] para avançar o seu tempo lógico para T. Todas as mensagens RO que estão na fila FIFO da RTI e todas as mensagens TSO com estampilhas menores ou iguais ao tempo corrente mais o passo de tempo são

entregues ao federado. As mensagens são entregues aos federados através das chamadas (*callback*) de métodos do federado pela RTI. Estes métodos são implementados na classe *FederateAmbassador*. Quando não houver mais mensagens TSO, agora e no futuro, que atendam a este critério, a RTI autoriza o avanço do tempo através do método *timeAdvanceGrant* [PRO98] com o parâmetro T. Isto indica que o tempo lógico do federado será avançado para T. A figura 16 mostra os componentes (federado e RTI) envolvidos no avanço de tempo usando o mecanismo *time stepped*.

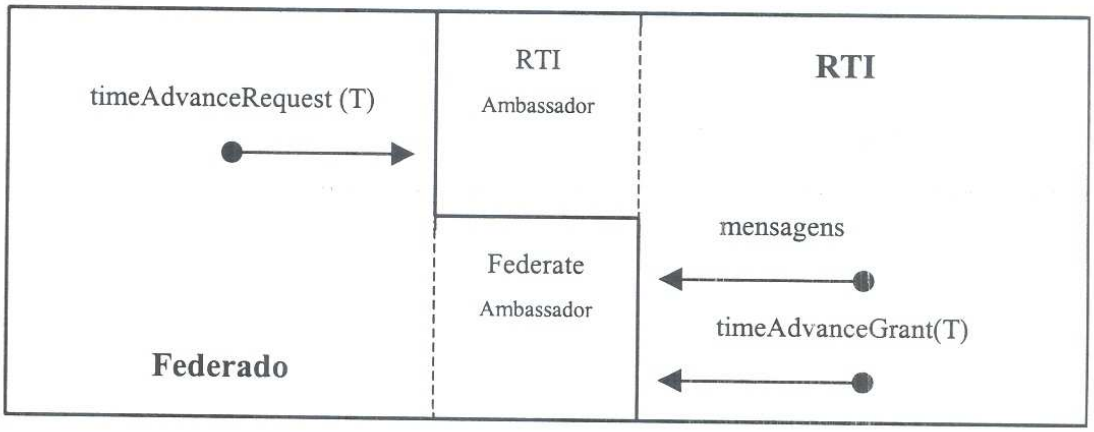

Figura 16 : Avanço de tempo em um federado *time stepped*

Através deste procedimento, a RTI garante que todas as mensagens TSO com estampilhas menores ou iguais a T foram entregues ao federado destinatário. Logo, o federado pode processar os eventos de forma segura já que ele não receberá nenhuma mensagem no seu passado. Contudo, esta afirmação só será verdadeira caso o valor do *lookahead* dos federados for maior do que zero. Se o valor do *lookahead* for zero, o federado que recebeu a mensagem e avançou seu tempo para T, pode agora enviar uma nova mensagem com estampilha igual a T de volta para o federado que mandou a mensagem original. Contudo, o tempo lógico do federado que enviou a mensagem original pode já ter avançado ($T_{atual} > T$), logo a recepção desta mensagem com estampilha igual a T pode representar uma mensagem no seu passado. Para evitar este tipo de problema e ainda permitir a existência de *lookahead* zero foram criados novos serviços dentro da arquitetura HLA: *timeAdvanceRequestAvailable e nextEventRequestAvailable* [PRO98]. No caso de federados *event driven* existe um método chamado *nextEventRequestAvailable* (NERA) que será analisado na próxima seção. No caso de federados *time stepped* existe um método chamado *timeAdvanceRequestAvailable* (TARA). O

funcionamento deste novo serviço é similar ao TAR anteriormente explicado, a diferença ocorre quando a RTI gera o *timeAdvanceGrant*. Neste caso, a RTI não garante que todas as mensagens TSO com estampilhas exatamente iguais a T foram entregues ao federado. O federado pode enviar mensagens com estampilhas iguais a T, ou seja, *lookahead* zero. O serviço TAR foi então modificado para impedir que o federado envie mensagens com estampilhas iguais a T mesmo que o valor do *lookahead* seja zero. Desta forma, cabe ao federado usar o serviço mais adequado para as suas necessidades em função dos valores disponíveis de *lookahead*.

3.5.2 Federados *Event Driven*

Um federado que utiliza o modelo de gerenciamento de tempo *event driven* avança o seu tempo lógico de acordo com os eventos recebidos da RTI. Neste tipo de mecanismo de avanço de tempo é normalmente utilizado o método *nextEventRequest* (NER) [PRO98]. Um federado utiliza este método quando ele tiver terminado todo o processamento relativo ao seu tempo lógico corrente e quer avançar o seu tempo para T, caso haja nenhum outro evento externo com estampilha menor.

O mecanismo de avanço de tempo *event driven* é mostrado na figura 17 juntamente com os componentes do federado e da RTI envolvidos.

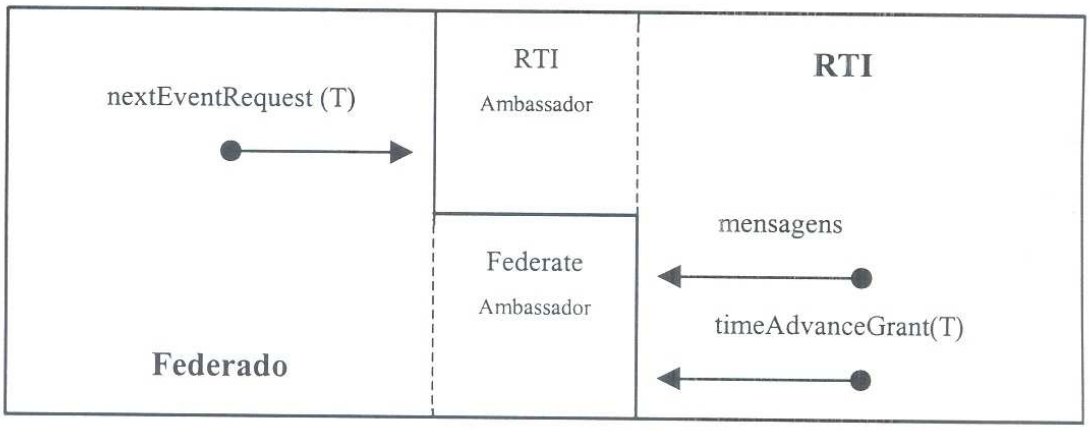

Figura 17 : Avanço de tempo em um federado *event driven*

Na maioria dos casos, o tempo T indica a estampilha do próximo evento do conjunto local de pendências do federado. Após receber o pedido via NER, todas as mensagens RO que estão na fila FIFO da RTI e todas as mensagens TSO com

estampilhas menores ou iguais ao tempo corrente (T) são entregues ao federado pela RTI. Se não houver mensagens TSO com estampilhas menores ou iguais a T, a RTI autoriza o avanço do tempo através do método *timeAdvanceGrant* com o parâmetro T. Isto indica que o tempo lógico do federado será avançado para T. Caso existam mensagens com estampilhas menor ou iguais a T, a RTI irá entregar a próxima mensagem TSO com menor estampilha (T') destinada ao federado. Além desta mensagem com estampilha T' (T'≤ T), todas as outras mensagens destinadas ao federado com estampilhas também iguais a T' serão entregues. Em contra partida, a RTI autoriza o avanço do tempo através do método *timeAdvanceGrant* com o parâmetro T', logo o tempo lógico do federado será T' e não T.

Assim como foi explicado para o serviço TAR, este procedimento garante todas as mensagens TSO com estampilhas menores ou iguais a T foram entregues. Logo, o federado pode processar os eventos de forma segura já que ele não receberá nenhuma mensagem no seu passado. Mais uma vez esta afirmação só será verdadeira caso o valor do *lookahead* seja maior do que zero, caso contrário o mesmo problema que acontece com TAR se repete. Para evitar este tipo de problema e ainda permitir a existência de *lookahead* zero foram criados novos serviços, conforme já mencionado na seção anterior. No caso de federados *event driven* existe um método chamado *nextEventRequestAvailable* (NERA). O funcionamento deste novo serviço é similar ao NER anteriormente explicado, a diferença ocorre quando a RTI gera o *timeAdvanceGrant*. Neste caso, a RTI não garante que todas as mensagens TSO com estampilhas exatamente iguais a T foram entregues ao federado. O federado pode enviar mensagens com estampilhas iguais a T, ou seja, *lookahead* zero. O serviço NER foi, também, modificado para impedir que o federado envie mensagens com estampilhas iguais a T mesmo que o valor do *lookahead* seja zero. O federado deve usar o serviço mais adequado para as sua necessidade de valor de *lookahead*.

3.5.3 Federados Otimistas

Os algoritmos de sincronismo conservadores e otimistas já foram detalhados na seção 2.1 deste trabalho. No caso da abordagem conservadora, o gerenciamento do

avanço do tempo lógico é similar ao modelo *event driven* da HLA. Neste caso, o avanço do tempo está vinculado ao processamento dos eventos internos e externos recebidos em mensagens TSO geradas por federados reguladores (*time regulating*). Já na abordagem otimista, o avanço de tempo dos federados não é estritamente dependente dos federados reguladores. As mensagens com os eventos são entregues pela RTI independentemente da ordem das suas estampilhas. Os eventos são então processados e caso seja necessário, existem mecanismos básicos que permitem a recuperação no caso da recepção de uma mensagem no passado do federado. Dentro da abordagem otimista, os federados podem receber todas as mensagens TSO destinadas a ele antes da RTI poder garantir que ele não receberá nenhuma mensagem no seu passado. O fato da RTI não garantir que não entregará posteriormente outra mensagem com estampilha menor ao federado, pode levar à uma violação dos requisitos causais e temporais da simulação.

Para receber todas as mensagens destinadas a ele não importando a ordem das estampilhas, o federado utiliza o método *flushQueueRequest* [PRO98] da classe *RTIAmbassador* com o parâmetro T. Este método força a RTI a entregar todas as mensagens disponíveis nas suas filas internas para este federado independentemente da ordem das estampilhas e do tempo lógico do federado. Após o envio de todas as mensagens disponíveis para o federado, a RTI autoriza o avanço do tempo lógico do federado para T. A figura 18 mostra a interação deste método.

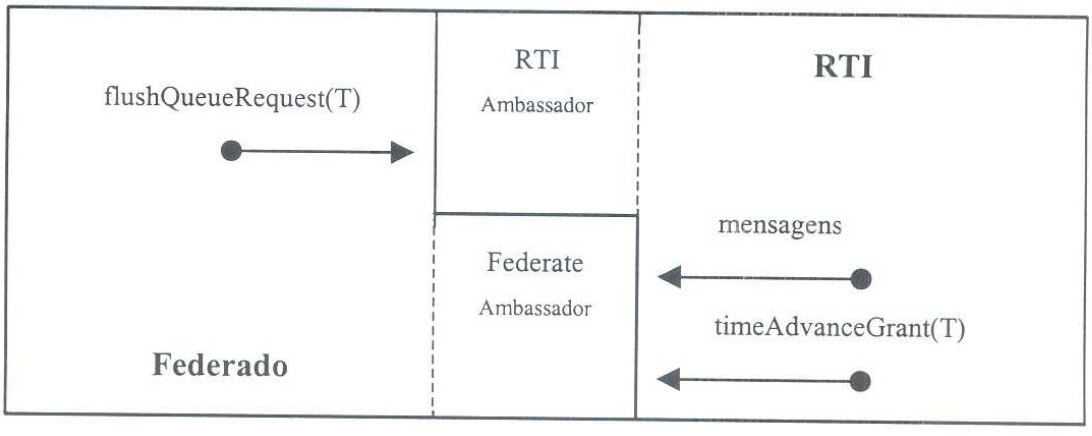

Figura 18 : Avanço de tempo em um federado otimista

Caso o federado receba alguma mensagem com estampilha menor que alguma mensagem já enviada por ele, uma série de procedimentos devem ser executados para anular esta mensagem e todas as outras com estampilhas maiores do que a recebida. Este procedimento de recuperação é chamado *rollback* e inclui a volta ao

estado da simulação até o ponto anterior ao processamento das mensagens com estampilhas maiores que a recebida. Este procedimento pode implicar em reprocessamento de eventos, cancelamento de eventos programados, envio de mensagens para os demais federados para anular o processamento realizado devido as possíveis mensagens erroneamente enviadas. O cancelamento das mensagens é feito através do método *Retract*, conforme é mostrado na figura 19.

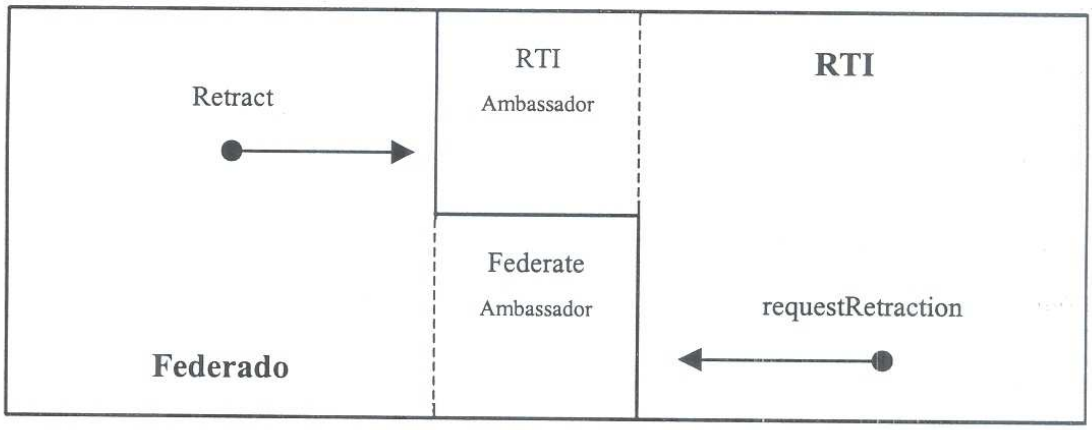

Figura 19 : Método *Retract*

Se a mensagem a ser cancelada ainda não foi entregue ao federado destinatário, ou seja, ainda está na fila interna da RTI, este mensagem é cancelada através do *Retract* [PRO98]. Caso contrário, o pedido de *Retract* será encaminhado para o federado que recebeu a mensagem. Um federado que tenha recebido uma mensagem inválida gerada erroneamente por um federado otimista irá receber um *requestRetraction* [PRO98] através da sua classe *FederateAmbassador*. Cabe ao federado desfazer todo o processamento dos eventos recebidos indevidamente e, se necessário, utilizar o método *Retract* para anular mensagens já enviadas também erroneamente.

O trabalho desenvolvido tem como objetivo a definição e implantação de um mecanismo capaz de detectar a violação da causalidade e temporalidade da simulação. Além de detectar, o mecanismo realiza todos os procedimentos de *rollback* necessários para desfazer todo o processamento errôneo, de forma transparente para o federado.

4 Mecanismo para *Rollback* de Federados Otimistas

Como já vimos na seção 3.5.3, na abordagem de gerenciamento de tempo otimista, um federado pode processar as mensagens fora da ordem das suas estampilhas. Caso mais tarde este federado receba uma mensagem de outro federado com uma estampilha menor do que alguma outra já processada, o federado deve desfazer (*rollback*) todo o processamento gerado por estas mensagens. A arquitetura HLA provê recursos bastante limitados para ajudar o federado nesta tarefa de *rollback*. Através do método *Retract* da classe *RTIAmbassador*, o federado pode solicitar à RTI que uma mensagem já enviada seja anulada. Todavia, não existe nenhum mecanismo para desfazer o processamento já efetuado dentro do federado, ficando esta tarefa a cargo da próprio federado. O mecanismo proposto vem de encontro com esta necessidade, ou seja, auxiliar o federado na manutenção de um estado seguro para onde ele possa voltar caso haja a necessidade de um *rollback*.

A idéia principal do modelo proposto é utilizar as técnicas de reflexão computacional [MAE87] para criar um "meta objeto" entre o federado e a RTI. Este meta objeto, aqui denominado gerente de *rollback*, tem como finalidade realizar todo o controle sobre as operações de *rollback* em federados otimistas, de forma que o código do federado fique mais simples e claro. Todos os procedimentos e controles sobre o *rollback* ficam sob responsabilidade do meta objeto gerente de *rollback*.

4.1 Reflexão Computacional

Reflexão é uma técnica de desenvolvimento que permite que um sistema tenha uma representação de si mesmo. Através desta representação, o sistema computacional pode realizar um controle e atuação sobre seu próprio comportamento. Isto é possível através de um conjunto de estruturas que representam seus próprios aspectos, tanto estruturais quanto computacionais [MAE87][NIS96]. Este conjunto é chamado de auto-representação do sistema.

Segundo [MAE87], um sistema computacional com arquitetura reflexiva deve er constituído por um nível base e um nível meta. O nível base é responsável pela resolução dos problemas pertencentes a um domínio externo, enquanto o nível meta é responsável pelo controle e gerenciamento do nível base. A técnica da reflexão computacional permite uma maior modularidade das aplicações separando o código da aplicação (nível base) e o código de gerência e controle (nível meta). Entre as diversas áreas de aplicação desta técnicas destacam-se: sistemas tolerante a falhas [FAB95], sistemas tempo real [NIS96][HON94] e concorrentes [YON88][RUB98].

Na reflexão computacional, o meta objeto está conectado ao objeto base de forma causal, logo qualquer alteração ocorrida no meta objeto deve ser refletida no respectivo objeto base. Nem todos os objetos de um sistema computacional precisam ser reflexivos. Somente os objetos que necessitam ser gerenciados e controlados precisam ser definidos como reflexivos, sendo então modularizados em objetos base e meta. Porém mesmo que exista um meta objeto, o objeto base pode ser acessado diretamente caso haja interesse.

Quanto um objeto é refletido, os seus métodos também são refletidos no meta objeto. Assim, a chamada a um método do objeto será desviada para o método correspondente do meta objeto. Estes métodos do meta objeto irão fazer o seu processamento e, posteriormente, invocar os métodos respectivos no objeto base, conforme mostrado na figura 20 .

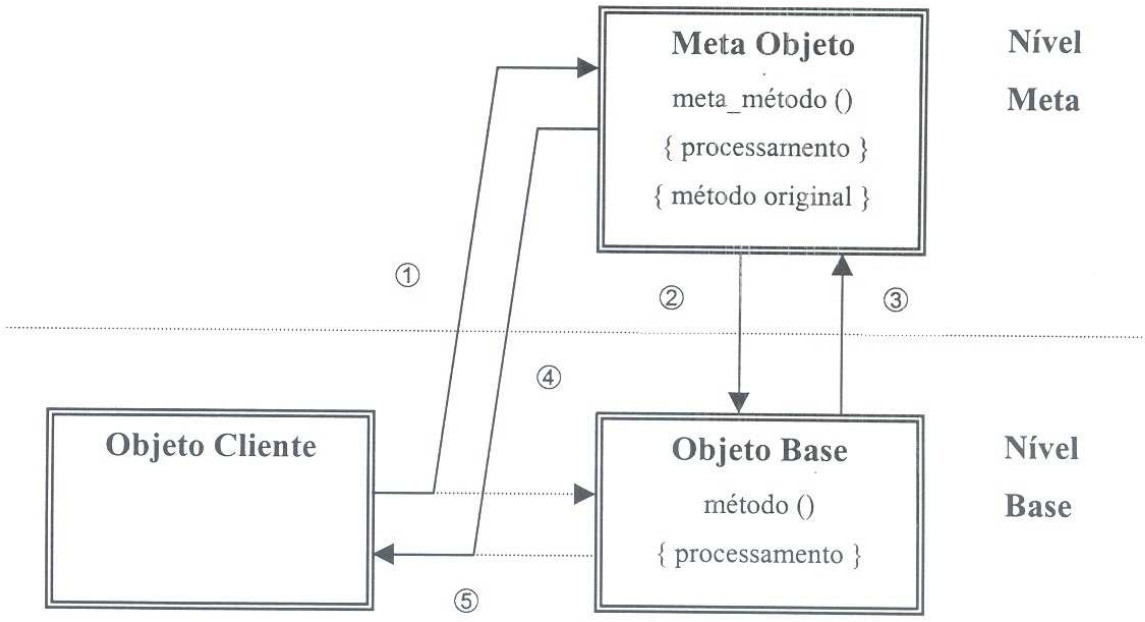

Figura 20 : Reflexão de objetos e métodos

Durante a chamada a um método reflexivo são realizadas 5 etapas distintas, que podem ser feitas por uma linguagem de programação reflexiva ou por uma alteração manual das chamadas (no caso de uma linguagem não reflexiva). As etapas são:

- 1ª etapa: a chamada ao método original no objeto base é interceptada
- 2ª etapa: o método do meta objeto é invocado e faz o processamento necessário para o gerenciamento desejado
- 3ª e 4ª etapas: após concluído o processamento, o método original é chamado pelo meta objeto
- 5ª etapa: após a conclusão do método original, o controle retorna ao objeto que invocou o método

Muitas vezes a reflexão computacional é implementada através de protocolos de meta objetos (*Meta Object Protocols* - MOP) [KIC93] que por sua vez são incorporados a linguagens de programação orientadas a objetos. Estas linguagens são responsáveis pela reflexão dos objetos em tempo de compilação, como por exemplo na linguagem Open-C++ [CHI93], Avalon/C++ [DET88] e CLOS [KIC93].

As técnicas de reflexão computacional podem ser usadas em qualquer aplicação que envolva chamadas a métodos de objetos como no caso da arquitetura HLA. A HLA define um conjunto de métodos que podem ser chamados a partir de duas classes principais conforme veremos a seguir. Contudo, a implementação da HLA fica a cargo de equipes de desenvolvimento que podem utilizar internamente diferentes técnicas e linguagens de programação, uma vez que apenas as interfaces (métodos) são padronizadas. Desta forma, os desenvolvedores de simulações tem acesso apenas aos métodos pré-definidos na especificação HLA, não podendo nem devendo fazer qualquer alteração nos mesmos. O uso de técnicas de reflexão permite que alguns métodos padrões possam ser desviados para meta métodos especialmente desenvolvidos para realizar funções que estendam a especificação HLA. Neste trabalho é proposto o uso de reflexão computacional para agregar um mecanismo de *rollback* com maior funcionalidade ao serviço de gerenciamento de tempo da arquitetura HLA.

4.2 Mecanismo Proposto

Dentro da arquitetura HLA, tanto os federados quanto a RTI são modelados como objetos que interagem entre si. A interação entre os federados e a RTI é feita através da chamada a métodos de duas classes: *RTIAmbassador* e *FederateAmbassador*. Estas classes fazem a ligação entre os federados e a RTI, conforme mostrado na figura 21.

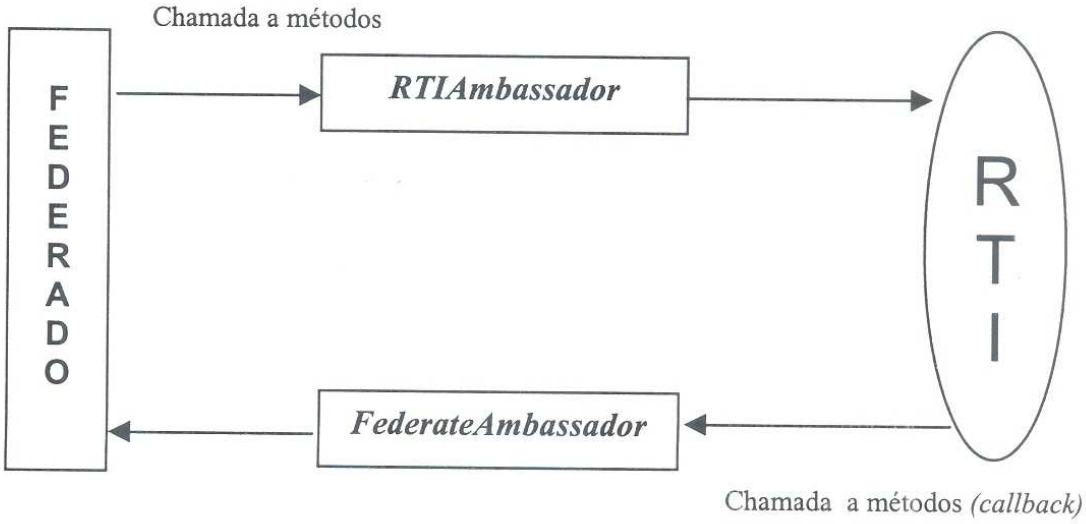

Figura 21 : Classes *RTIAmbassador* e *FederateAmbassador*

A classe *RTIAmbassador* contém os métodos providos pela RTI e que são chamados no código dos federados. Desta forma, todos os pedidos feitos pelo federado à RTI são feitos via chamada de métodos do *RTIAmbassador*. A implementação destas classes, e métodos, está dentro da biblioteca *libRTI* que permite a comunicação entre os federados e a RTI componente *libRTI*. Esta implementação não é diretamente acessível ao programador. Em contrapartida, a classe *FederateAmbassador* é uma classe abstrata que identifica as funções de retorno (*callback*) que cada federado é obrigado a fornecer para que a RTI passe os eventos para os federados. A implementação dos métodos desta classe deve ser feita pelo programador, obedecendo a sintaxe definida na especificação de interfaces HLA (*I/F Spec*).

O mecanismo aqui proposto utiliza técnicas de reflexão computacional [MAE87] para criar um meta objeto (*rollback manager*) encarregado do gerenciamento de recuperação de estado (*rollback*) dos federados otimistas,

complementando os serviços básicos oferecidos pela RTI e assim facilitando o desenvolvimento dos federados. A figura 22 ilustra como é a aplicação da reflexão computacional sobre os métodos das classes *RTIAmbassador* e *FederateAmbassador* .

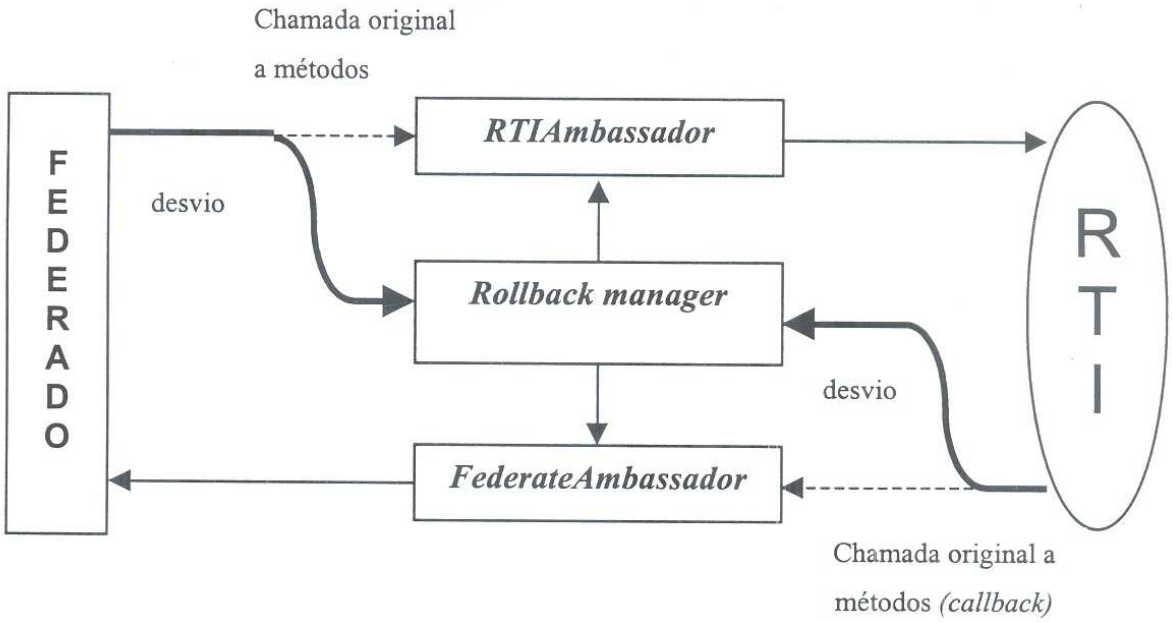

Figura 22 : Reflexão das classes *RTIAmbassador* e *FederateAmbassador*

A reflexão destes métodos é necessária para que o controle sobre o estado do federado e sobre as mensagens recebidas e enviadas seja desvinculado do código do mesmo. O federado continuará chamando os mesmos métodos da classe *RTIAmbassador* para se comunicar com a RTI e também irá receber as mensagens da RTI via *callback* através da classe *FederateAmbassador*. Contudo, algumas chamadas serão interceptadas e direcionadas ao gerente de *rollback* (*rollback manager*). Em nossa proposta somente as chamadas a alguns métodos de gerenciamento de tempo sofrerão reflexão, todos os demais métodos continuarão a ser invocados dos objetos base, no caso *RTIAmbassador* e *FederateAmbassador*.

O mecanismo aqui proposto visa especificamente a intercepção das chamadas aos métodos envolvidos no gerenciamento de tempo de um federado otimista, ou seja, *flushQueueRequest* e *timeAdvanceGrant*. A idéia por trás do mecanismo é permitir que o federado possa adotar uma abordagem para gerenciamento otimista sem contudo ter que se preocupar com eventuais *rollbacks* dentro do seu código. Como já foi dito, a tarefa de controle e execução dos procedimentos envolvidos

durante uma operação de *rollback* ficarão sob responsabilidade do gerente de *rollback*, tornando o código do federado mais simples e claro. O modelo proposto é apresentado na figura 23.

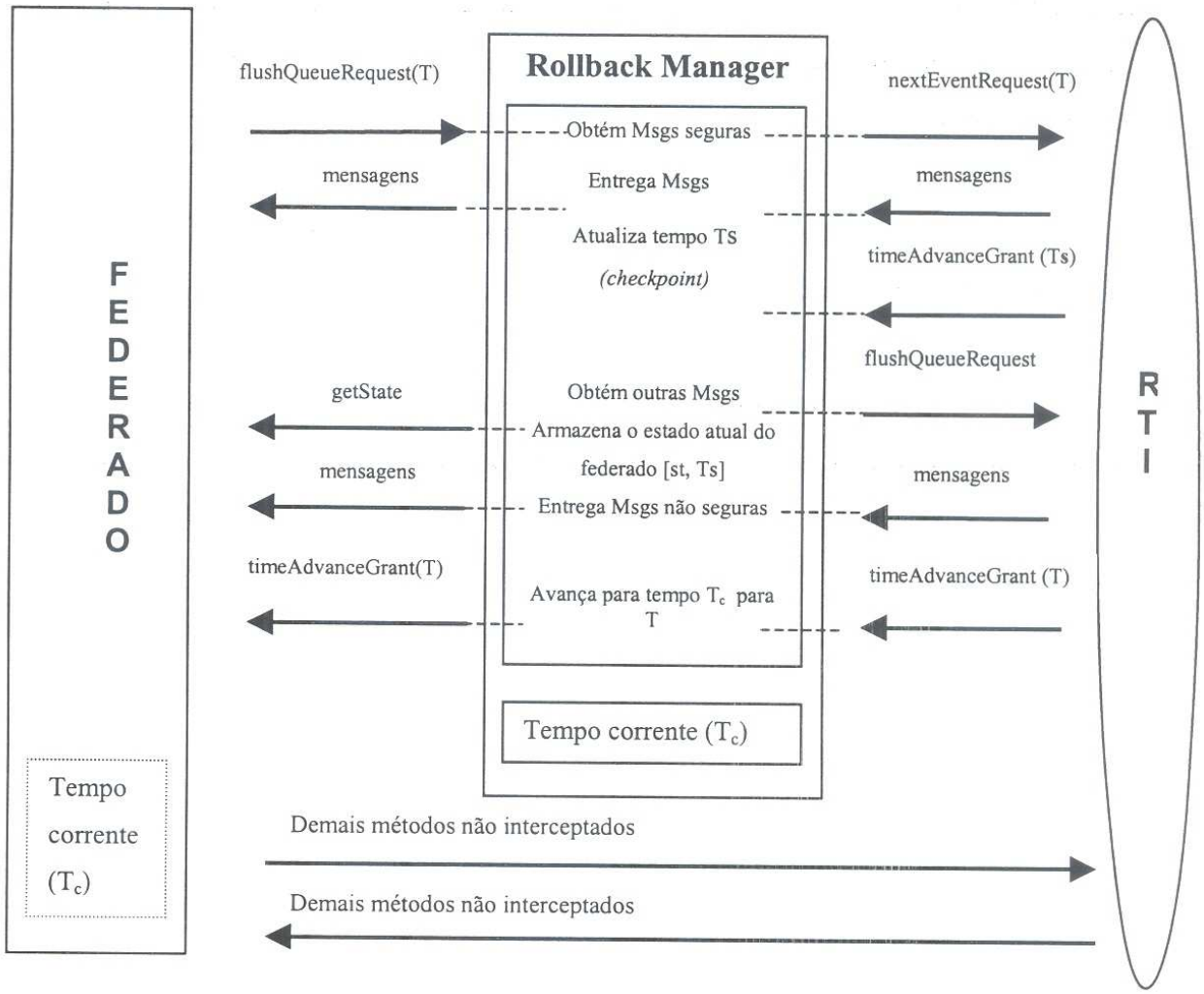

Figura 23 : Interação entre o Gerente de *Rollback*, Federado e RTI

Para que o gerente de *rollback* possa retornar o federado a estados anteriores seguros, é necessário salvar o estado do federado otimista em determinados instantes de tempo (*checkpoints*) nos quais todos os eventos entregues ao federado sejam garantidos. Dentro deste contexto, um evento garantido é aquele que possui uma estampilha de tempo (TSO) e que pode ser processado sem nenhum risco de cancelamento (*rollback*) no futuro, a menos que seja explicitamente requisitada a sua retração (a retração de eventos é utilizada dentro de algumas simulações orientadas a eventos discretos para modelar comportamentos de interrupções e preempções). Para que o gerente possa efetivamente controlar eventuais *rollbacks* sobre o federado de forma transparente, é necessário que ele tenha acesso ao estado interno do federado. O estado de um federado pode ser visto de maneira

simplificada como o conjunto dos valores atuais de suas variáveis locais e sua posição atual em seu código. Para que o gerente de *rollback* possa interagir com o estado do federado que controla, duas operações devem ser implementadas pelo federado: *getState(st)*, que armazena uma cópia do estado atual do federado em um vetor de estado *st*, e *setState(st)*, que usa a informação de estado contida em *st* para restaurar um estado anterior no federado, conforme representado na figura 22. Essas duas operações são similares àquelas utilizadas no sistema *Isis* para o gerenciamento de réplicas em grupos de processos tolerantes a faltas [BIR93]. Com essas operações, o gerente de *rollback* pode manter uma lista de estados anteriores do federado (pares [*st*, *t*], onde *st* corresponde ao estado e *t* ao instante de tempo correspondente), aos quais este pode ser retornado se detectada uma violação de causalidade.

4.3 Determinação de um Estado Seguro

Para que o procedimento de *rollback* seja possível, o gerente deve salvar o estado do federado otimista em um momento de tempo considerado seguro, chamado *checkpoint*. Caso haja necessidade de desfazer o processamento já efetuado, o estado do federado deve voltar ao último estado seguro (*checkpoint*) antes da ocorrência do evento que resultou no *rollback*.

O mecanismo proposto utiliza a reflexão computacional para desviar a chamada ao método *flushQueueRequest* para o gerente de controle. O gerente utiliza primeiramente uma abordagem conservadora para receber as mensagens TSO da RTI. Através do método *nextEventRequest*, o gerente solicita que a RTI entregue todas as mensagens RO que estão na fila FIFO e todas as mensagens TSO com estampilhas menores ou iguais ao tempo corrente mais o passo de tempo, que é passado como parâmetro na chamada do método. Quando não houver mais mensagens TSO, neste instante de tempo e em instantes futuros (mensagens seguras), que atendam a este critério, a RTI autoriza o avanço do tempo através do método *timeAdvanceGrant* com o parâmetro T_s. Isto indica que o tempo lógico do gerente será avançado para T_s. Ao final deste procedimento, o gerente terá recebido todas as mensagens seguras dentro da abordagem conservadora. Através deste procedimento, a RTI garante todas as mensagens TSO com estampilhas menores ou

iguais a T_s foram entregues. Logo, o federado pode processar os eventos de forma segura já que ele não receberá nenhuma mensagem no seu passado. Este tempo T_s é dito como instante de checagem (*checkpoint*) e indica um ponto no tempo simulado no qual o estado corrente do federado está conservadoramente seguro. Ainda neste tempo T_s, o gerente salva o estado [*st*, T_s] do federado através da invocação do método *getState* (*st*) para que seja possível restaurá-lo no caso de um procedimento de *rollback*.

Note que uma chamada ao método *nextEventRequest* e seu respectivo *callback timeAdvanceGrant* são feitas através das classes *RTIAmbassador* e *FederateAmbassador*. A classe *RTIAmbassador* está implementada dentro da *libRTI* no LRC. Já a classe *FederateAmbassador* está implementada dentro do código do gerente de *rollback*. As mensagens conservadoras entregues pela RTI para o gerente são repassadas diretamente para o federado, pois não irão influenciar em um possível *rollback* do federado otimista. Após a entrega das mensagens conservadoras, não sujeitas a rollbacks futuros, o gerente irá salvar o estado seguro do federado. Após este procedimento, o gerente chama o método *flushQueueRequest* da classe *RTIAmbassador* para que a RTI entregue as demais mensagens destinadas ao federado. Este método força a RTI a entregar todas as mensagens disponíveis nas suas filas internas para este federado, não importando a ordem das estampilhas, conforme é definido pela abordagem otimista. Estas mensagens entregues pela RTI para o gerente são aquelas factíveis de sofrerem *rollback*, uma vez que a RTI não garante que no futuro outra mensagem com estampilha menor seja entregue ao federado. Após a entrega do restante das mensagens TSO, o federado recebe via *callback* a autorização da RTI para avanço seu tempo lógico para T. Desta forma, o gerente também tem o seu tempo lógico corrente (T_c) avançado para T, porém ele ainda mantém salvo o estado *st* do federado no tempo T_s, onde $T \geq T_s$.

4.4 Procedimento de Rollback

Caso o federado receba uma mensagem com estampilha menor do que o seu tempo lógico atual T é preciso que o federado retorne a um estado anterior (em T_s) para garantir a correção causal da simulação. A necessidade do procedimento de *rollback* será detectada pelo próprio gerente de *rollback* uma vez que ele recebe

todas as mensagens endereçadas ao federado otimista. Existem basicamente quatro tipos de eventos que alteram objetos e seu atributos, logo precisam ser tratados de forma diferenciada pelo gerente para que este possa manter um controle das alterações feitas. Estes eventos serão descritos em detalhes na seção 4.6 deste trabalho, no momento os eventos recebidos serão tratados de forma genérica.

Ao receber um evento, em uma mensagem TSO, o gerente irá verificar a sua estampilha (T_m) e compará-la com o tempo lógico corrente (T_c). O tempo lógico do gerente é igual ao do respectivo federado. Se $T_m < T_c$ então é detectada a violação da causalidade e o gerente irá automaticamente retornar o federado ao último estado seguro [*st*, T_s] (último instante de *checkpoint*), onde $T_s < T_m$. O gerente utiliza o método *setState* (*st*) para restaurar o estado do federado. A figura 24 mostra o mecanismo de *rollback* implementado no gerente.

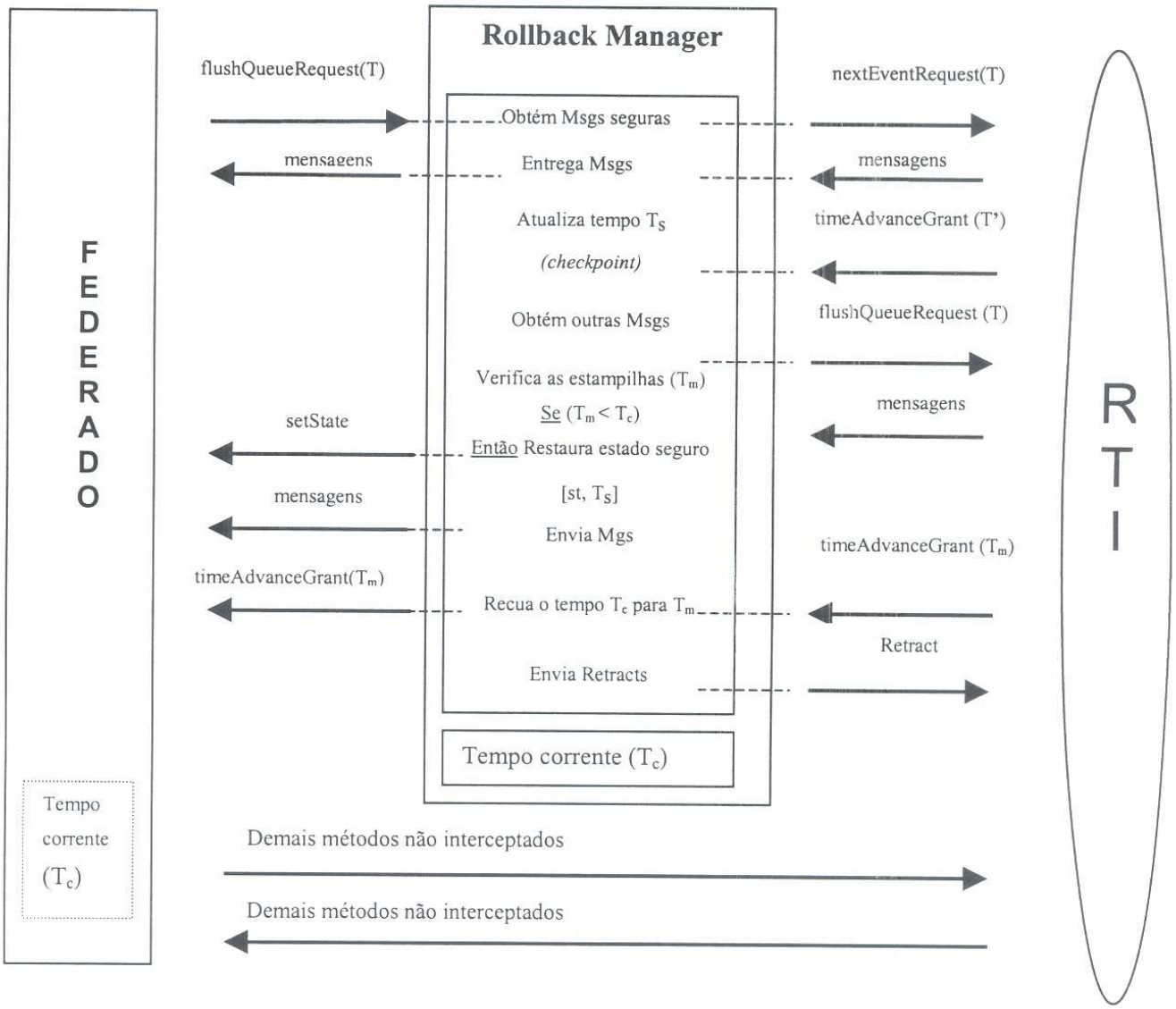

Figura 24 : Procedimento de Rollback no gerente

O gerente de *rollback* tem controle sobre todas as mensagens enviadas pelo federado $T_s < T \leq T_c$, ou seja, entre o tempo corrente (T_c) e o último ponto de controle (*checkpoint* em T_s). Desta forma, o gerente pode invocar o método *Retract* da RTI para cancelar mensagens já enviadas. Para que isto seja possível, o gerente mantém todos os descritores (*EventRetractionHandles*) dos eventos enviados durante este intervalo de tempo. Os eventos enviados, bem como os recebidos, estão relacionados a troca de informações entre os federados que consistem, basicamente, de mudanças no estado dos objetos publicados e subscritos pelos federados, bem como alterações em atributos destes objetos. Da mesma forma, o gerente pode receber pedidos de cancelamento de mensagens enviadas indevidamente por outros federados. A RTI repassa ao destinatário o pedido de cancelamento da mensagens através do método (*callback*) *requestRetraction*. Cabe ao federado implementar os procedimentos necessários para desfazer o processamento ocasionado por esta mensagem. Dentro do modelo proposto cabe ao gerente de *rollback* realizar a recepção e gerenciamento deste tipo de requisição de forma transparente para o federado.

4.5 Serviços de Gerenciamento de Tempo e Objetos

Durante a simulação, os federados trocam mensagens com informações sobre objetos e seus atributos. Esta troca de mensagens contendo eventos entre o federado e a RTI é importante, pois pode envolver mensagens com estampilhas (TSO) que poderão no futuro ser alvo de um procedimento de *rollback*. Desta forma, o modelo proposto precisa identificar estas mensagens, armazenar um histórico dos objetos e atributos alterados por elas para poder realizar o *rollback* caso seja necessário. Para atualizar um ou mais atributos associados com uma instância registrada de um objeto, o federado precisa preparar uma estrutura de dados chamada *AttributeHandleValuePairSet* (AHVPS), conforme descrito na seção 3.1.

Além da troca de mensagens referentes a objetos, outra forma de comunicação entre os federados é através de interações (*interactions*). No caso das interações, os dados são passados de um federado para outro através de parâmetros. Existe uma estrutura de dados chamada *ParameterHandleValuePairSet* (PHVPS) que

desempenha um papel similar ao AHVPS, como também já foi descrito na seção 3.1.

Dentro do modelo proposto estas duas estruturas de dados são importantes pois os eventos recebidos pelo meta objeto gerente precisam ser, posteriormente, repassados ao federado sem nenhuma alteração nos seus conteúdos. Um AHVPS ou um PHVPS recebido precisa ser repassado do gerente para o federado, após ter sido registrado pelo gerente.

4.6 Eventos trocados entre os Federados

Os eventos são trocados entre os federados através do envio de mensagens. Uma mensagem é uma unidade de dados transmitida entre os federados que contém no máximo um evento. Uma mensagem tipicamente contém informação sobre um evento e é usada para notificar outro federado que um evento ocorreu. O federado ao receber esta mensagem irá processá-la de acordo com o seu tipo e conteúdo. Uma mensagem gerada pode ter associada a ela uma estampilha de tempo, dependendo da política de tempo adotada pelo federado. Dentro da abordagem otimista somente os eventos com estampilhas (TSO) são importantes, porque precisarão ser reprocessados ou anulados no caso de *rollback*.

Os federados trocam eventos via RTI, logo quando os federados precisam enviar um evento eles devem solicitar este serviço via métodos da classe *RTIAmbassador*. Existem basicamente quatro métodos que podem ser usados pelo federado para enviar um evento: *UpdateAttributeValues*, *SendInteraction*, *RegisterObjectInstance* e *DeleteObjectInstance* [PRO98]. Estes métodos estão implementados dentro da *libRTI* e utilizam as estruturas AHVPS e PHVPS no caso da alteração de atributos e envio de interações, respectivamente.

A recepção de eventos por parte de um federado acontece através de métodos da classe *FederateAmbassador* que são chamados (*callback*) pela RTI. Os métodos definidos dentro da HLA são: *ReflectAttributeValues*, *ReceiveInteraction*, *DiscoverObjectInstance* e *RemoveObjectInstance* [PRO98]. Estes métodos estão implementados dentro do código do federado e utilizam as estruturas AHVPS e PHVPS no caso da alteração de atributos e recepção de interações, respectivamente.

Quando estes métodos são chamados, o federado é notificado que recebeu um evento gerado por um federado externo através dos métodos de envio de eventos. O relacionamento entre todos estes métodos é mostrado na figura 25 .

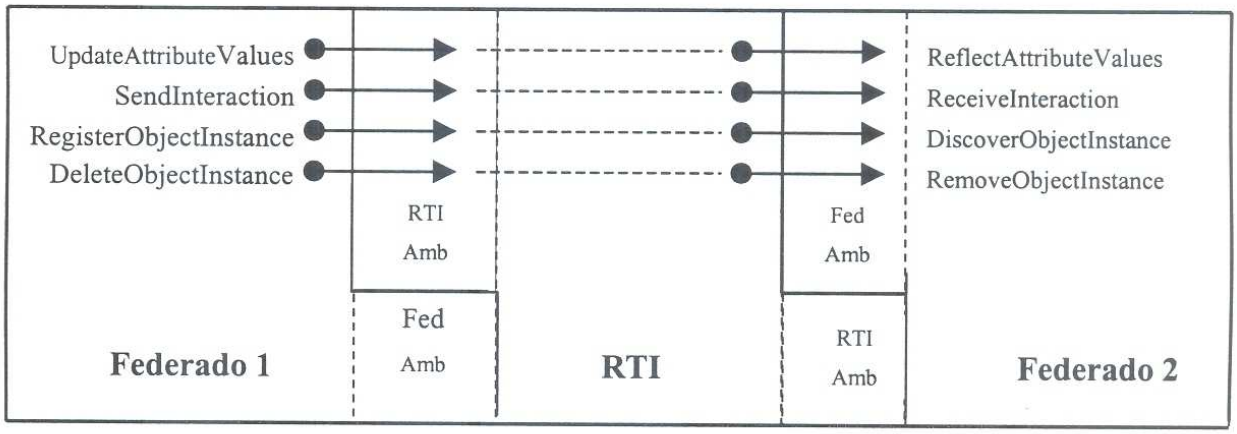

Figura 25 : Envio e recepção de eventos

Todos estes eventos possuem um *EventRetractionHandle* associado que permite o seu cancelamento caso seja necessário. Estes descritores são preservados pelo gerente de *rollback* para futuros cancelamentos (*retract*) caso sejam necessários.

4.7 Roteiro Básico de Funcionamento

O mecanismo proposto combina técnicas de reflexão computacional, abordagem conservadora (recebimento do primeiro lote de mensagens) e abordagem otimista (recebimento do restante das mensagens) para construir um meta objeto gerente responsável pelo controle e execução de *rollback*. Neste modelo as mensagens recebidas são primeiramente analisadas pelo gerente e posteriormente enviadas ao federado otimista. Como já foi visto, as mensagens recebidas com dados (AHVPS ou PHVPS) podem ser basicamente duas: *ReflectAttributeValues (RAV)* e *ReceiveInteraction (RI)*. Neste caso, antes de repassar as mensagens (eventos) ao federado, o gerente irá salvar os atributos antigos para possibilitar um *rollback* futuro.

O funcionamento do mecanismo pode ser resumido através do algoritmo:

Enquanto (simulação não acabou) {

Federado: próximo_evento = estampilha do próximo evento local

Federado: *flushQueueRequest* (próximo_evento)

Meta Objeto: intercepta esta chamada à RTI (*RTIAmbassador*)

Meta Objeto: *nextEventRequest* (proximo_evento) /* conservadora */

RTI: envia mensagens (RAV e RI)

Meta Objeto: recebe as mensagens

RTI: *timeAdvanceGrant*

Meta Objeto: avança o tempo lógico (tempo de *checkpoint*)

Meta Objeto: salva o estado seguro do federado

Meta Objeto: repassa as mensagens para o federado

Federado: recebe as mensagens (RAV e RI) e armazena na sua fila de eventos pendentes

Meta Objeto: *flushQueueRequest* (próximo_evento) /* otimista */

RTI: envia mensagens restantes (RAV e RI)

Meta Objeto: recebe as mensagens

Meta Objeto: salva atributos e descritores de recuperação (*EventRetractionHandles*)

Meta Objeto: realiza o procedimento de *rollback* se necessário

RTI: *timeAdvanceGrant*

Meta Objeto: avança o tempo lógico

Meta Objeto: repassa as mensagens para o federado

Federado: recebe as mensagens (RAV e RI) e armazena na sua fila de eventos pendentes

Meta Objeto: *timeAdvanceGrant*

Federado: avança o tempo lógico

Federado: processa os eventos na sua fila de eventos pendentes

}

4.7.1 Trilha de Eventos

Uma outra forma de representar o mecanismo proposto é através de diagramas de trilhas de eventos. Neste tipo de diagrama é mostrado um mapeamento entre os eventos que são trocados entre o federado, gerente de *rollback* e a RTI. A figura 26 apresenta a trilha de eventos entre estas três entidades que interagem dentro do modelo. No diagrama, as mensagens *ReflectAttributeValues* e *ReceiveInteraction* são representadas pelas siglas RAV e RI, respectivamente.

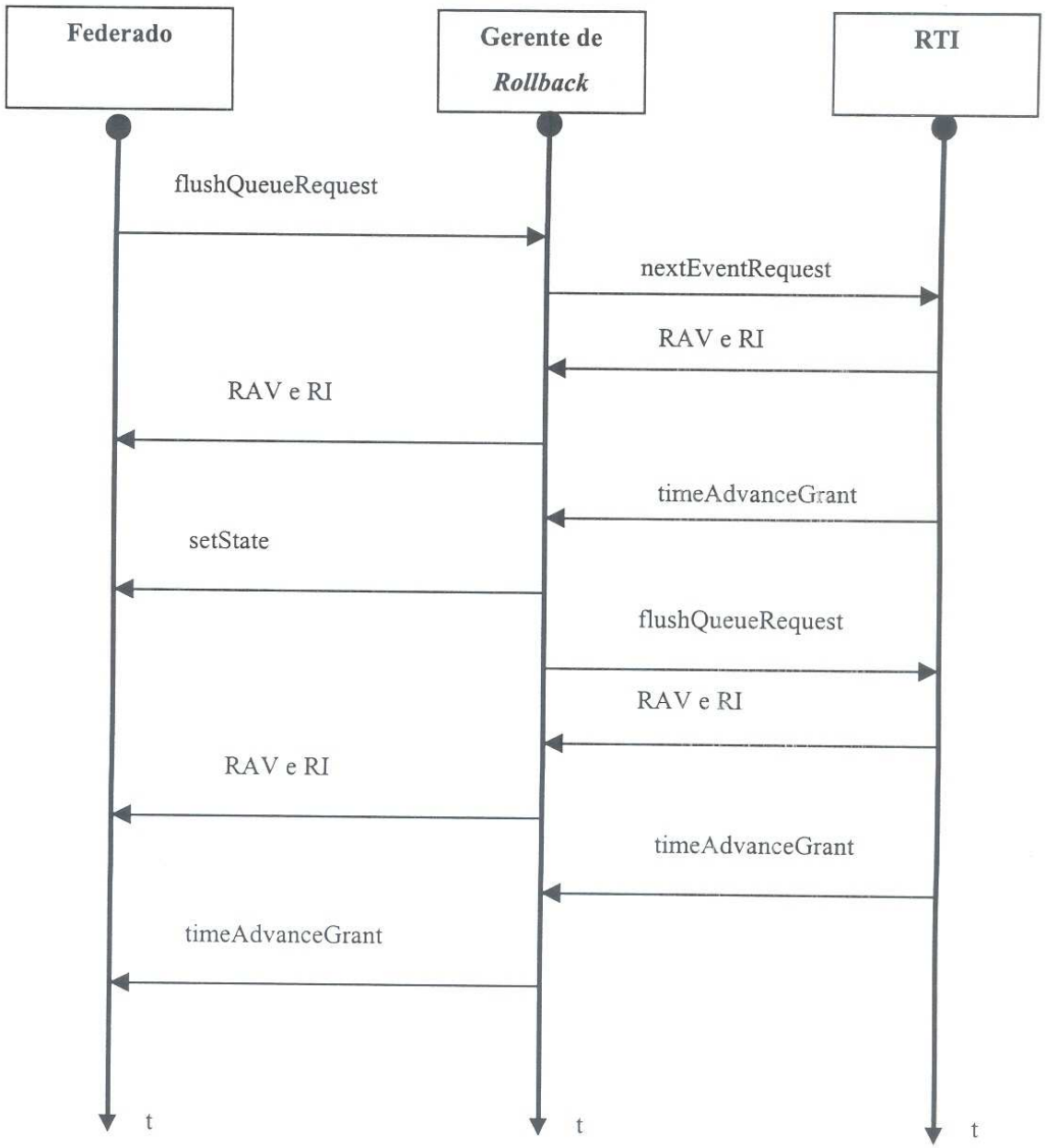

Figura 26 : Trilha de eventos

Esta trilha mostra a troca de eventos entre os três componentes durante o avanço normal de tempo de um federado otimista. Quando o gerente detecta uma mensagem (evento) com estampilha menor que o tempo corrente, ele precisa interagir com o federado e com a RTI para realizar o *rollback*. O gerente restaura o estado do federado em um ponto seguro anterior à violação da causalidade (tempo de *checkpoint*) e utiliza os descritores das mensagens (*EventRetractionHandle*) para anular as mensagens já enviadas.

Quando a mensagem enviada já foi entregue ao federado destino, a RTI simplesmente repassa ao destinatário o pedido de cancelamento da mensagens através do método (*callback*) *requestRetraction*. Como já foi dito, dentro do mecanismo proposto cabe ao gerente realizar a recepção e gerenciamento deste tipo de requisição. A figura 27 mostra o diagrama de trilha de eventos no caso de recepção de uma mensagem de cancelamento de um evento.

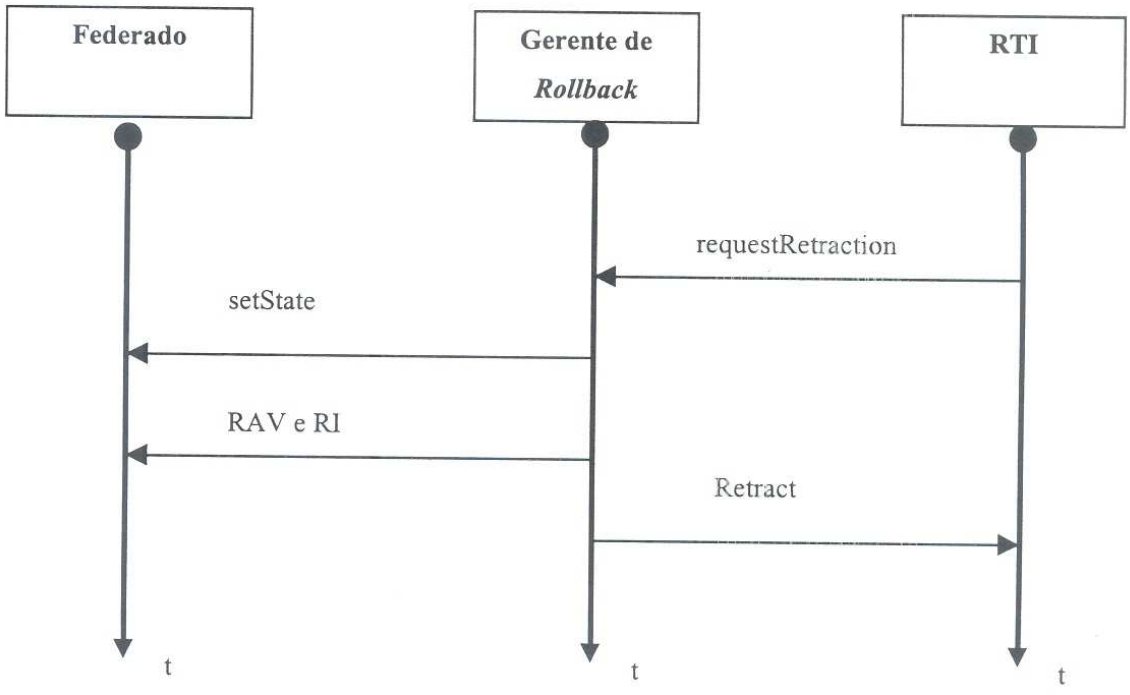

Figura 27 : Trilha de eventos para a recepção de *requestRetraction*

O gerente, ao receber o *requestRetraction*, recebe junto o descritor (*EventRetractionHandle)* do evento que precisa ser cancelado. Através deste descritor, o gerente de *rollback* recupera os valores (atributos ou parâmetros) anteriores ao processamento deste evento. Os valores antigos são passados ao

federado através dos métodos *ReflectAttributeValues* e *ReceiveInteraction*. Desta forma, o federado não precisa se preocupar com o cancelamento deste evento. O estado do federado também é retornado para um ponto seguro anterior ao processamento deste evento. Caso o processamento tenha resultado no envio de novas mensagens a outros federados, o gerente irá enviar o pedido de cancelamento dos mesmos através do método *Retract* do *RTIAmbassador*. O gerente de *rollback* possui controle sobre todas as mensagens enviadas através do seu *EventRetractionHandle* das mensagens *UpdatetAttributeValues* e *SendInteraction*, entre outras.

4.8 Intervalos de Checagem (*Checkpoints*)

O mecanismo proposto neste trabalho está baseado na determinação de estados seguros para os quais o federado retorna no caso da violação da causalidade. Através do método *getState*, o gerente de *rollback* pode salvar o estado corrente do federado em determinados tempos de checagem (*checkpoints*).

Existem diferentes abordagens para a determinação dos pontos de checagem, ou seja, do salvamento do estado atual do federado que visam a otimização do processamento necessário para esta tarefa. A seleção correta do intervalo de checagem e conseqüente salvamento do estado do federado pode influenciar muito no desempenho da simulação otimista conforme é mostrado em [LIN93] [LIN90a].

Em algoritmos de sincronismo otimistas, como o *Time Warp*, normalmente o intervalo de checagem é 1, ou seja, a cada evento processado o estado é salvo. É obvio que este procedimento aumenta muito a carga de processamento necessária para a simulação e também o espaço de memória utilizado para armazenar todos estes estados. Para atenuar este problema diversos mecanismos foram sugeridos e podem ser classificados em dois grupos: salvamento periódico (*periodic state saving*) [BEL92] [LIN89] e salvamento incremental (*incremental state saving*) [BAU93] [UNG93] do estados dos federados.

No salvamento periódico a idéia principal é reduzir o processamento ocasionado pelos *rollbacks* através da redução da freqüência de salvamentos de estados. Os estados são salvos em intervalos maiores, o que ocasiona um aumento no tempo necessário para restaurar o estado do federado no caso de um eventual *rollback*, pois

é preciso ainda reprocessar todas mensagens recebidas a partir do último salvamento de estado. Já nos mecanismos baseados em salvamento incremental, os estados dos federados não são completamente salvos e sim apenas os dados que sofreram modificações desde o último *checkpoint* são salvos. Neste mecanismo, o tempo gasto para salvar o estado do federado é reduzido, porém o tempo de restauração do último estado seguro pode aumentar significativamente, pois para recuperar o estado do federado no caso de um *rollback* é preciso recuperar todas as mudanças incrementais dos estados, uma a uma na ordem inversa das suas gerações.

Existem diversos trabalhos que fazem análises dos desempenhos dos mecanismos baseados em salvamento periódico e incremental [PAL93] [CLE94] em diversos modelos de simulações paralelas otimistas.

No mecanismo proposto foi utilizada a abordagem de salvamento periódico do estado do federado, pois existem pontos de checagem bem definidos o que facilita a implementação desta abordagem.

No caso do gerente de *rollback* cada vez que ele chama o método *nextEventRequest* da classe *RTIAmbassador* e recebe um *timeAdvanceGrant* este instante de tempo representa um *checkpoint* (T_s), pois somente as mensagens conservadoras foram entregues. Neste instante (T_s), o gerente salva o estado do federado [st, T_s] através do método *getState*.

Quando o gerente detecta o recebimento de uma mensagem TSO com estampilha (T_m) menor do que o tempo lógico corrente (T_c) do federado, ele utiliza o método *setState* para retornar o federado ao último estado seguro do vetor de estados [st, T_s], onde $T_s < T_m$, conforme mostrado na figura 28.

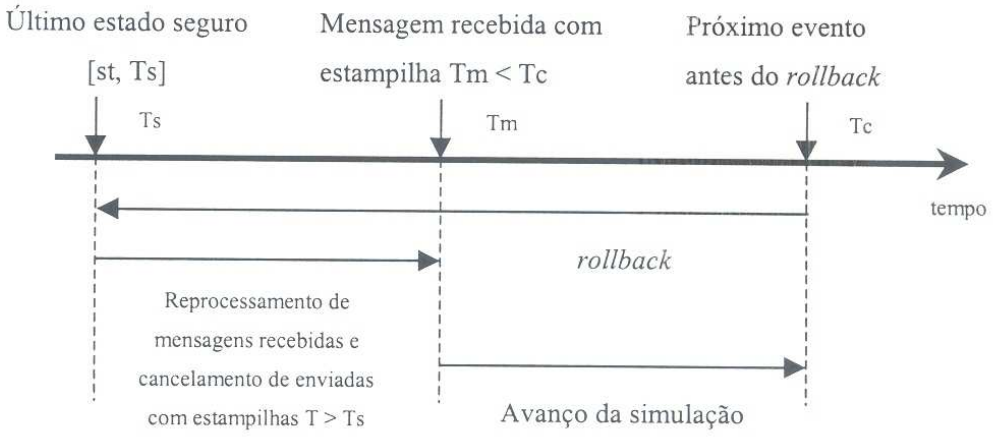

Figura 28 : Restauração de Estado

Dentro deste vetor ficam armazenados os estados seguros do federado, cada estado contém todas as informações relativas a variáveis internas, objetos e interações enviadas necessárias para reconstituir a situação do federado em um dado instante de tempo. Após o federado retornar ao último estado seguro T_s (*checkpoint*) anterior ao *rollback*, é preciso que todas as mensagens recebidas com estampilhas $T > T_s$ sejam reprocessadas na ordem correta. Para obter este efeito, o gerente de *rollback* repassa estas mensagens novamente para o federado, que irá processá-las sem ter noção que se trata de um *rollback*. E ainda, todas as mensagens com estampilhas $T > T_s$ enviadas erroneamente são canceladas pelo gerente através do método *Retract* da classe *RTIAmbassador*, conforme mostrado na figura 24.

Dependendo das características de cada federado, o volume de informações necessárias para salvar o estado, mensagens enviadas e recebidas após o último *checkpoint*, pode variar muito. Na próxima seção serão analisados aspectos referentes ao consumo de memória do mecanismo proposto.

4.9 Consumo de Recursos

Uma questão muito importante dentro de simulações otimistas refere-se a quantidade de memória consumida para armazenar dados históricos sobre os estados de cada federado, além da lista dos eventos pendentes. Para modelos de simulação envolvendo federados otimistas complexos, o espaço de memória necessário para armazenar todos estes dados pode ser um restrição significativa. É fundamental também o armazenamento do menor número possível de estados dentro do vetor por federado para que o consumo de memória minimizado.

O gerente de *rollback* precisa guardar informações sobre três tipos de objetos: vetores de estado, mensagens recebidas e mensagens enviadas [DAS94]. Cada um destes objetos possui uma referência de tempo (T) que será usada para permitir a sua dispensa (liberação de espaço em memória) quando for seguro.

Dentro do mecanismo proposto, os dados sobre os estados são armazenados no vetor até que eles possam ser seguramente dispensados, o que evita o consumo desnecessário de memória. Os federados precisam estabelecer um limite (T_{limite}) de retenção destes dados, ou seja, qualquer informação sobre estados [st, T] e

mensagens enviadas/recebidas, onde $T < T_{limite}$ pode ser seguramente dispensada e os recursos de memória liberados. Este limite inferior de tempo é chamado GVT (*Global Virtual Time*), já descrito na seção 2.1. Como uma situação de *rollback* é causada pela recepção de uma mensagem TSO (ou cancelamento de evento) com sua estampilha T_m, onde $T_m < T_c$ (tempo lógico corrente), o federado pode usar o seu LBTS (seção 3.4.1) como uma boa estimativa para o GVT [FUJ98]. Como no mecanismo proposto o gerente de *rollback* intercepta todas as mensagens, ele pode também através do LBTS do federado calcular o limite inferior para a retenção dos estados do federado dentro do vetor de estados [st, T] . Além dos estados, o gerente precisa manter uma cópia (ou ponteiro no caso de cancelamento direto [FUJ89a]) de todas as mensagens enviadas com estampilhas $T < T_{limite}$ para que possa enviar anti-mensagens no caso de um eventual *rollback*. Todas as mensagens recebidas após o último estado seguro (*checkpoint*) e que estão sujeitas a cancelamento também precisam ser armazenadas. Assim todos os dados sobre estados $[st_i, T_i]$ do federado e mensagens enviadas/recebidas com estampilha T_i, onde $T_i < (GVT + Lookahead)$ e $0 < i <$ (n° máx. de estados), podem ser desconsiderados uma vez que seguramente não estão sujeitos a *rollbacks*. Esta liberação de recursos de memória que estavam sendo utilizados para manter estes dados é conhecida como *fossil collection* [DAS94].

Existem vários outros esquemas propostos para a liberação de recursos de memória como por exemplo: envio de mensagem de *retorno (message sendback protocol* [JEF85] [JEF87] e *Gafni protocol*), protocolo *cancelback* [JEF90], protocolos adaptáveis [DAS94] e *rollback* artificial [LIN91]. Dentro do mecanismo proposto neste trabalho apenas o esquema de *fossil collection* foi implementado, porém outros esquemas podem ser adicionados em futuros desenvolvimentos.

Os mecanismos utilizados para o cancelamento de mensagens, ou seja, envio de anti-mensagens, também afetam o consumo de memória. Na seção 2.1.2 foram descritos os mecanismos de cancelamento agressivo e lento. Segundo [GAF88], o mecanismo de cancelamento agressivo (*aggressive cancellation*) requer uma quantidade menor de memória para armazenar os dados referentes as mensagens enviadas e que podem ser canceladas no futuro. Neste mecanismo as mensagens são imediatamente canceladas assim que uma violação da causalidade seja detectada

pelo gerente de *rollback*. No caso do cancelamento lento (*lazy cancellation*) é preciso reter em memória uma quantidade maior de informações sobre o conteúdo das mensagens para uma posterior comparação e decisão sobre o cancelamento ou não das mensagens já enviadas.

No mecanismo proposto neste trabalho, foi adotada a estratégia de cancelamento agressivo (*aggressive cancellation*) para economizar recursos de memória necessários para armazenar os dados referentes as mensagens enviadas. Outra justificativa para a escolha do cancelamento agressivo é a sua maior simplicidade de implementação e ser a estratégia normalmente utilizada em simulações otimistas, como por exemplo o protocolo *Time Warp*.

4.10 Considerações Adicionais

Uma outra questão envolvendo simulações otimistas é o estabelecimento de um limite para o avanço da execução dos federados. Se um federado avançar muito a sua execução através de um passo muito grande, isto irá ocasionar uma perda de eficiência da abordagem otimista, pois as chances de ocorrerem violações da causalidade irão aumentar muito, gerando um maior número de *rollbacks* grande. Este problema é conhecido como *throttling* e não há nenhuma especificação na arquitetura HLA relativa ao mecanismo a ser usado nesta situação [FUJ99]. Cada federado otimista é responsável por implementar mecanismos para evitar avanços muito grandes na sua execução. Existem vários mecanismos na literatura [REI89] [PRA91] que podem ser usados para limitar o avanço de um federado em relação aos demais.

5 Validação do Mecanismo Proposto

Para validar o mecanismo de *rollback* efetuado por um gerente foi desenvolvida uma simulação envolvendo diversos federados otimistas. Foi criada uma federação composta por um número variável de federados que trocam eventos através de mensagens TSO. Nesta federação, os federados trocam mensagens relativas a alterações nos atributos dos seus objetos e também mensagens referentes a interações entre eles.

Como o objetivo maior deste trabalho é propor um mecanismo para simplificação do processo de *rollback* em simulações otimistas, a federação desenvolvida para o protótipo não teve compromisso com a sua funcionalidade enquanto modelo de simulação. Foi desenvolvida uma simulação simples, baseada em uma simulação pré-existente, com o objetivo de mostrar a viabilidade do mecanismo proposto. Este mecanismo, uma vez validado, pode ser utilizado em qualquer simulação complexa especialmente desenvolvida para representar o modelo físico desejado. Uma das premissas básicas do mecanismo proposto é influenciar o mínimo possível o desenvolvimento das simulações e ainda poder ser utilizado em simulações já existentes. Do lado do federado, o único requisito necessário é a implementação de dois métodos *getState* e *setState* que são chamados pelo gerente de *rollback* quando necessário, conforme descrito na seção 4.2.

O desenvolvimento de uma simulação HLA deve seguir uma metodologia de forma a garantir a sua aderência aos padrões e especificações da arquitetura, assegurando assim a sua interoperabilidade com outras simulações. Os aspectos referentes ao desenvolvimento de simulações HLA não fazem parte do escopo deste trabalho.

5.1 Ambiente de Desenvolvimento e Testes

A infra-estrutura RTI é composta por um conjunto de programas executáveis e bibliotecas que seguem as especificações da arquitetura HLA. A arquitetura define apenas as funcionalidades que devem estar presentes e as suas interfaces, através das

quais os serviços podem ser acessados pelos federados. Não há nenhuma especificação no que se refere a sua forma de implementação nem linguagem de programação a ser utilizada. Cabe aos desenvolvedores escolher a melhor maneira de implementar estes serviços usando a linguagem de programação mais adequada aos seus interesses.

Para dar um impulso maior para a arquitetura HLA recém proposta, o DoD patrocinou a implementação da versão 1 da infra-estrutura RTI. A RTI está disponível atualmente em diversas implementações utilizando-se diferentes ambientes de programação. Neste trabalho foi utilizada a RTI versão 1.3 implementada na linguagem C++. Além deste pacote básico foi utilizada também uma interface de desenvolvimento Java. Esta interface permite que os federados possam ser codificados em linguagem Java ao invés de C++. Esta interface foi escolhida porque na plataforma SPARC SUN é necessário o kit de desenvolvimento SUN *SparcWorks* para a programação em C++. Este kit precisa ser adquirido separadamente do sistema operacional. Já no caso da linguagem Java, o kit é gratuito e distribuído via Internet.

A plataforma escolhida para desenvolvimento e testes da simulação foi:

Hardware:

SPARCStation™ 2 da SUN Microsystems

64 MB de memória RAM

Software:

Sistema Operacional Solaris™ 2.6

Java™ Development Kit (JDK) 1.1.6

RTI 1.3 v.6

5.2 Descrição da Simulação Efetuada

Com o objetivo de validar o mecanismo de *rollback* através de um meta objeto gerente, foi desenvolvida uma simulação na qual um número variável de federados

---

™ Marca registrada da SUN Microsystems

troca eventos e interações. Para simplificar a descrição da simulação, serão considerados apenas 2 federados. Cada federado possui um objeto com dois atributos (nome e população) e existem ainda uma interação com um parâmetro (mensagem texto). Os federados registram (publicam) os seus objetos e atributos, assim como a sua interação. Cada federado também subscreve os objetos e a interação do outro federado, conforme mostrado na figura 29. Desta forma, todos os federados podem enviar e receber atualização de atributos e interações.

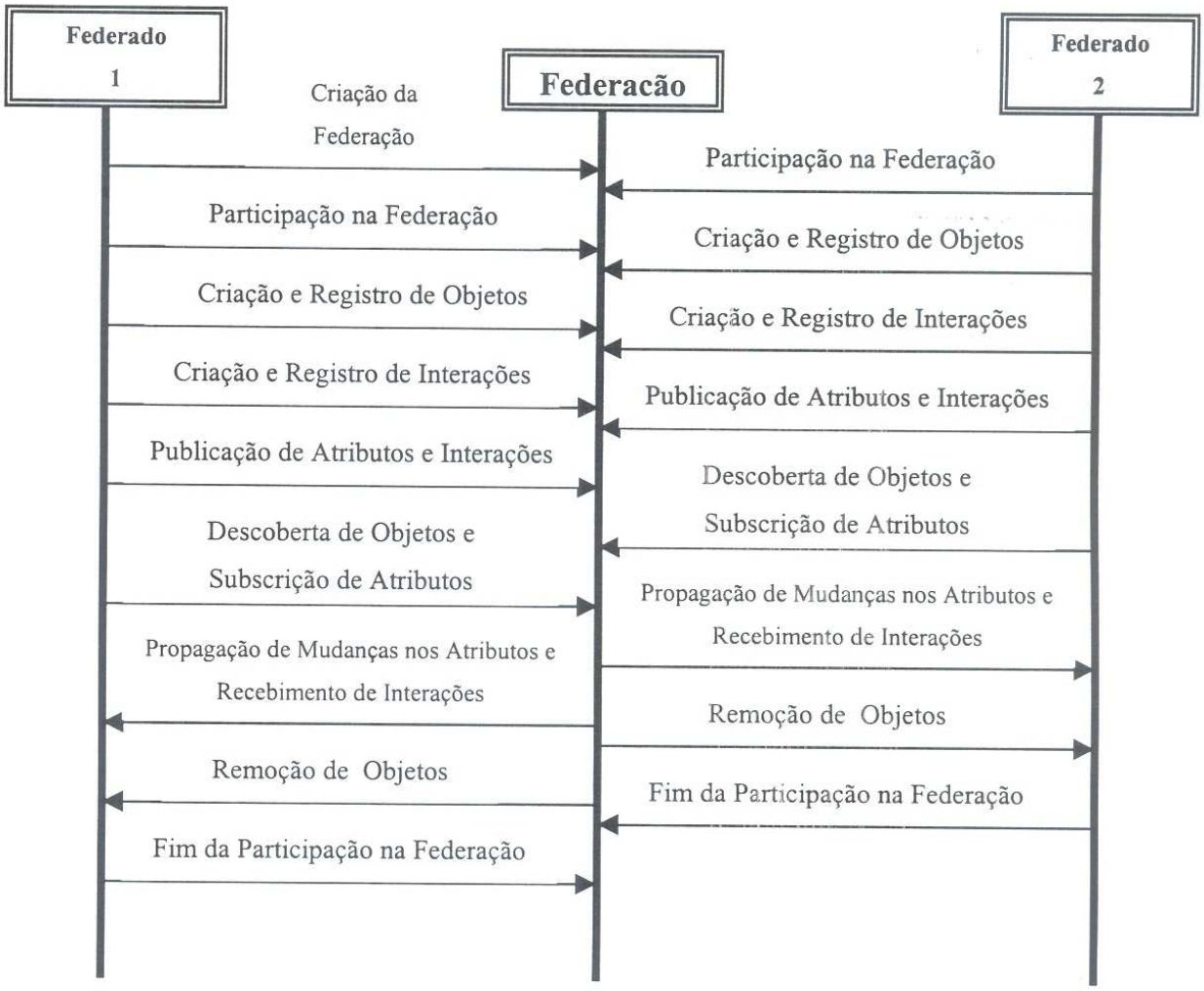

Figura 29 : Cenário da Simulação Efetuada

O primeiro federado que for executado irá criar a federação e logo em seguida, irá se juntar a esta execução da federação. Todas as definições e parâmetros da federação estão contidas em um arquivo de configuração previamente preparado.

Para que o gerente de *rollback* possa ser avaliado é preciso que os federados ativem os serviços de gerenciamento de tempo da arquitetura HLA, conforme foi

descrito na seção 3.3.1. Primeiramente, os federados precisam se tornar restritos (*time constrained*) e reguladores (*time regulating*) para que possam receber e enviar, respectivamente, mensagens TSO. Como já detalhado anteriormente, uma mensagem TSO possui uma estampilha que indica o tempo do federado (LVT) no momento do seu envio. Após a confirmação recebida da RTI, os federados estão prontos para enviar e receber atualizações de atributos e interações através de mensagens TSO.

Os federados irão, repetidamente, alterar os seus atributos e através da chamada ao método *updateAttributeValues* da classe *RTIAmbassador* [PRO98] vão comunicar aos demais federados sobre o ocorrência desta alteração. O número de alterações de atributos é parametrizável dentro da simulação. Em intervalos periódicos, também configuráveis, cada federado irá enviar uma interação através da chamada ao método *sendInteraction* da classe *RTIAmbassador* [PRO98] para os demais federados.

Ao final do número de interações definido para a execução, os federados se retiram da federação. Quando o último federado se retira, a execução da federação é encerrada.

O gerente de *rollback* somente entrará em operação quando o federado adotar a política de sincronismo de eventos otimista. Dentro da arquitetura HLA, esta situação ocorre quando o federado utiliza o método *flushQueueRequest* da classe *RTIAmbassador* [PRO98]. Neste exato momento, a chamada a este método será desviada para o meta objeto gerente conforme explicado na seção 4.4. A partir deste ponto, o gerente irá salvar o estado do federado em intervalos de checagem através do método *getState* (*st*). O gerente também irá proceder com o *rollback* caso seja necessário.

O *rollback* é necessário quando ocorre uma violação da causalidade da simulação, ou seja, o federado recebe uma mensagem no seu passado. Ao receber uma mensagem TSO, o gerente irá verificar a sua estampilha (T_m) e compará-la com o tempo lógico corrente do federado (T_c). Se $T_m < T_c$ então é detectada a violação da causalidade e o gerente irá automaticamente retornar o federado ao último estado seguro [*st*, T_s] (último instante de *checkpoint*), onde $T_s < T_m$. O gerente utiliza o método *setState* (*st*) para restaurar o estado do federado.

Para que mecanismo possa ser testado é preciso que um federado receba esta mensagem TSO com estampilhas inferior ao seu tempo corrente. Para forçar esta situação foi utilizado um terceiro federado também regulador (*time regulating*). A função deste federado, chamado *gatilho*, será enviar uma mensagem através da chamada ao método *updateAttributeValues* da classe *RTIAmbassador* com estampilha menor que o tempo lógico dos federados. O federado *gatilho* envia a mensagem, conforme mostrado na figura 30, para forçar a violação do causalidade do federado ativando assim o gerente de *rollback* .

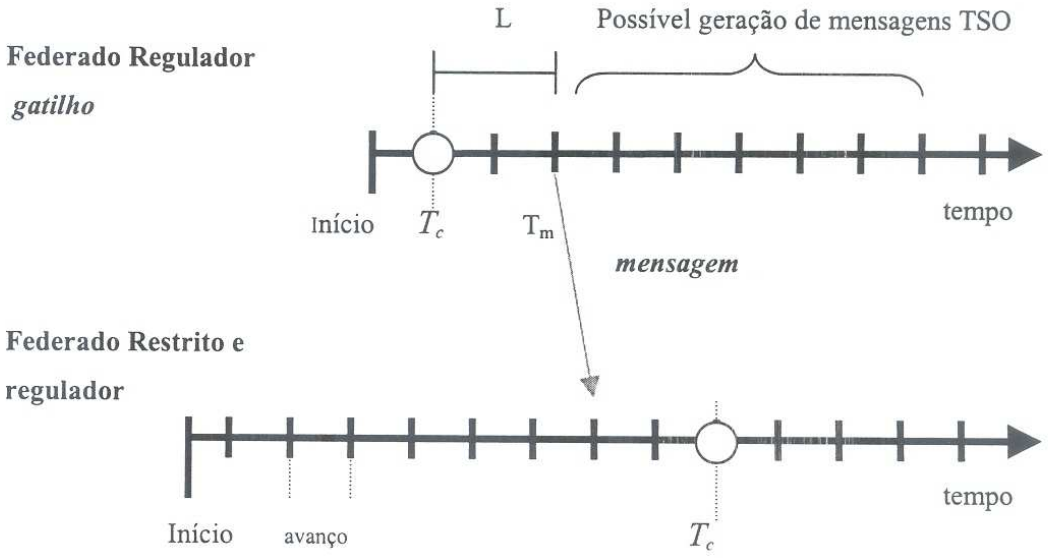

Figura 30 : Mensagem enviada no passado do Federado

Assim que o gerente de *rollback* interceptar esta mensagem enviada pelo federado gatilho, ele irá comparar a estampilha da mensagem (T_m) com o tempo lógico corrente (T_c).

Como $T_c > T_m$, o gerente irá detectar a violação da causalidade e o mecanismo descrito na seção 4.4 entrará em funcionamento.

5.3 Dificuldades Encontradas

Durante o desenvolvimento da simulação otimista, usando o gerente de *rollback*, algumas dificuldades foram encontradas. Existe todo um processo definido para o desenvolvimento de simulações baseadas na arquitetura HLA. O desenvolvimento de uma federação deve seguir seis etapas sugeridas pelo DMSO, que também fornece um conjunto de ferramentas para auxiliar os desenvolvedores [DMS99]. A simulação desenvolvida para validar o mecanismo proposto foi codificada através de uma simplificação de toda a metodologia indicada pelo DMSO.

Durante a fase de programação, uma dificuldade encontrada foi uma limitação técnica da RTI versão 1.3. O gerente de *rollback* precisa se comunicar com o federado e com a RTI para poder interceptar e tratar certas mensagens, conforme foi descrito na seção 4.2. Porém, nem todas as mensagens são desviadas para o gerente, logo o federado ainda precisa se comunicar com a RTI. A comunicação com a RTI é possível graças a criação de uma instância da classe *RTIAmbassador*. Desta forma, tanto o federado quanto o gerente de *rollback* precisam criar instâncias desta classe. A limitação técnica da versão 1.3 acontece quando as duas instâncias são criadas na mesma aplicação, ou seja, dentro de uma mesma federação. Dentro da RTI 1.3, cada federação é executada como uma única *thread*, logo a existência de duas instâncias da classe *RTIAmbassador* pode gerar problemas, especialmente quando os serviços de gerenciamento de tempo estão sendo utilizados. A classe *RTIAmbassador* utiliza uma série de variáveis estáticas. Quando esta classe é instanciada várias vezes dentro da mesma *thread* e espaço de endereçamento podem ocorrer inconsistências nos seus valores. Esta limitação é conhecida e está documentada nos comentários sobre a versão 1.3 (*release notes*)[1]. Uma alternativa é interceptar todas as chamadas a métodos das classes *RTIAmbassador* e *FederateAmbassador*, desta forma a comunicação entre o federado e a RTI não seria mais necessária. Assim somente uma instância da classe *RTIAmbassador* no gerente de *rollback* seria suficiente para a implementação do

[1] *"The known bugs/limitations in the RTI release notes states that multiple RTIAmbassadors cannot be created in multiple threads of the same application"*. [SUP99]

mecanismo proposto. Durante a fase de análise das alternativas, esta hipótese se revelou inviável pois existem mais de 180 métodos que precisariam ser desviados. Cada chamada interceptada acarreta em um acréscimo no tempo total de execução, mesmo que não seja tratada, ou seja, apenas repassada para o federado ou à RTI. Foram analisadas várias alternativas outras técnicas juntamente com a equipe de suporte do DMSO[2] para o contorno desta situação. A solução encontrada foi desabilitar o gerenciamento de tempo efetuado pela RTI e codificar estas funcionalidades dentro do gerente de *rollback*. Esta codificação adicional aumentou o tempo de desenvolvimento, mas agregou conhecimentos valiosos sobre os serviços de gerenciamento de tempo.

5.4 Resultados Obtidos

A simulação desenvolvida foi executada em uma estação Sun SparcStation 2 com 64 Mb de memória RAM, no mesmo ambiente que foi usado para o seu desenvolvimento. O objetivo principal do gerente de *rollback* proposto neste trabalho é a simplificação do processo de desenvolvimento e execução de federados otimistas. Todo o ônus envolvido na detecção da violação do princípio da causalidade e posterior processo de *rollback* do federado fica sob responsabilidade do gerente. Desta forma, a simulação com a inclusão do gerente de *rollback* apresentou tempos de execução maiores do que a simulação original. Este acréscimo já era esperado, pois o gerente precisa interceptar algumas chamadas a métodos que são feitas pelo próprio federado ou pela RTI, conforme já explicado na seção 4.2. Estas chamadas a métodos representam a troca de mensagens entre os federados e a RTI. As estampilhas das mensagens interceptadas pelo gerente precisam ser analisadas para verificar se houve ou não violação da causalidade. Este processo adiciona um processamento extra à simulação, conforme foi observado nas medições de desempenho efetuadas.

Para medir o acréscimo no tempo de execução da simulação foram feitos vários experimentos com a simulação desenvolvida. O primeiro experimento

---

[2] Contatos via telefone e correio eletrônico com Robert Head e Allen Skees, engenheiros de software da empresa Virtual Technology Corporation, EUA. (entre março e abril de 1999)

elaborado consistiu na variação do número de interações da simulação. Primeiramente foi executada a simulação original com diferentes valores para o parâmetro "número de interações". Este parâmetro variou de 500 a 2000 interações com o objetivo de medir o desempenho da simulação sem o mecanismo gerenciador de *rollback*. Cada simulação foi repetida diversas vezes para ampliar o tamanho da amostra de dados, pois a cada execução o tempo pode variar de acordo com a carga de processamento do computador naquele momento específico. Desta forma, foram feitas cinco simulações com cada número de interações e ao final das cinco execuções foi calculada a média aritmética dos tempos obtidos. A medição dos tempos foi feita através do comando *time* disponível no sistema operacional Solaris. Este comando não proporciona valores absolutamente confiáveis, porém servem perfeitamente para uma comparação dos tempos obtidos sem e com o mecanismo gerenciador de *rollback*. Os resultados obtidos estão apresentados na figura 31.

Simulações	**Número de Interações**			
	500	**1000**	**1500**	**2000**
1ª	2:21	4:25	6:42	8:30
2ª	2:42	4:24	6:44	8:16
3ª	2:29	4:35	6:12	8:18
4ª	2:33	4:31	6:37	8:22
5ª	2:41	4:22	6:23	8:27
média:	2:33	4:27	6:32	8:23

Figura 31 : Tempos de execução sem o Gerente de *Rollback*

Todos os tempos apresentados na figura 31 estão medidos em minutos e segundos no formato (min:seg) e representam o tempo (*user* + *system time*) total de execução da simulação.

Para efeitos de comparação também foram medidos os tempos de execução, com os mesmos números de interações, com a utilização do gerente de *rollback*. Os resultados obtidos comprovaram o acréscimo nos tempos totais de execução da simulação. Os valores obtidos são apresentados na figura 33 a seguir.

Simulações	Número de Interações			
	500	1000	1500	2000
1ª	3:34	6:21	9:16	12:13
2ª	3:36	6:14	9:34	12:16
3ª	3:44	6:22	9:22	12:23
4ª	3:22	6:31	9:45	12:12
5ª	3:52	6:33	9:39	12:17
média:	3:38	6:24	9:31	12:16

Figura 32 : Tempos de execução com o Gerente de *Rollback*

Cada interação da simulação gera a troca de duas mensagens (*updateAttributeValues* e *reflectAttributeValues*) entre os federados passando pela RTI. Estas mensagens precisam ser interceptadas e analisadas pelo gerente de *rollback*, o que acresce o tempo total de execução. Logo, quanto maior for o número de mensagens processadas maior será a influencia da existência do gerente no desempenho da simulação. A figura 33 mostra um gráfico relacionando os tempos totais de execução da simulação original e com o *rollback manager* proposto.

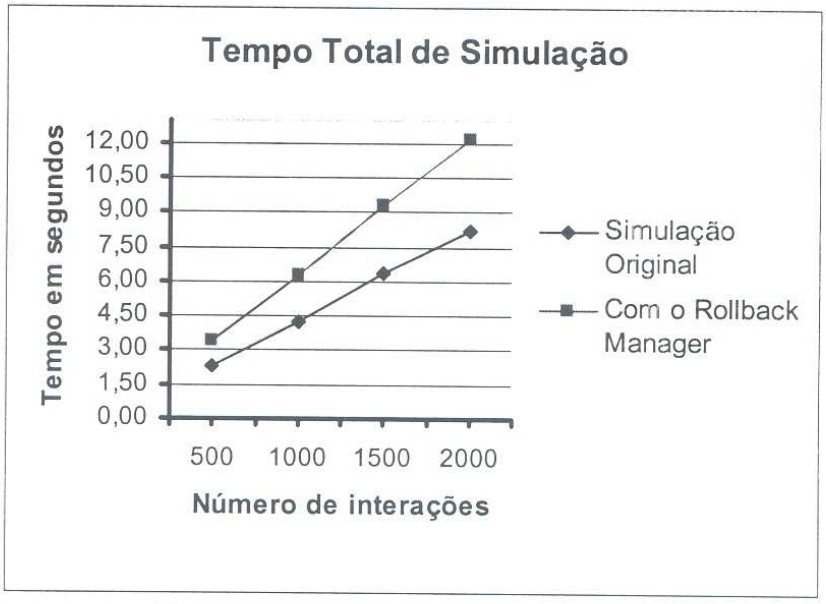

Figura 33 : Gráfico com os tempos de execução

Um outro indicador importante que deve ser analisado é a relação entre os tempos de execução obtidos com o uso do gerente e na simulação original. Segundo [CHI93], esta relação entre os tempos, denominada *fator*, deve ser medida para verificar o acréscimo no tempo de execução que é ocasionado pela reflexão computacional. Os fatores são calculados através da divisão dos tempos de execução com o gerente de *rollback* pelos tempos obtidos durante a execução da simulação original. Os valores obtidos estão apresentados na figura 34 a seguir.

	Tempos de Execução			
Número de Interações:	**500**	**1000**	**1500**	**2000**
Simulação Original	2:33	4:27	6:32	8:22
Rollback Manager (RM)	3:38	6:24	9:31	12:16
Fator	1:45	1:46	1:47	1:48

Figura 34 : Relações entre os tempos de execução

Estes números mostram que a execução da simulação com o gerente de *rollback* (RM) é 1,45 vezes mais lenta do que a simulação original quando são processadas 500 interações. Já no caso de 2000 interações, o fator indica que a simulação com o gerente é 1,48 vezes mais lenta. A evolução dos fatores pode ser visualizada na figura 35.

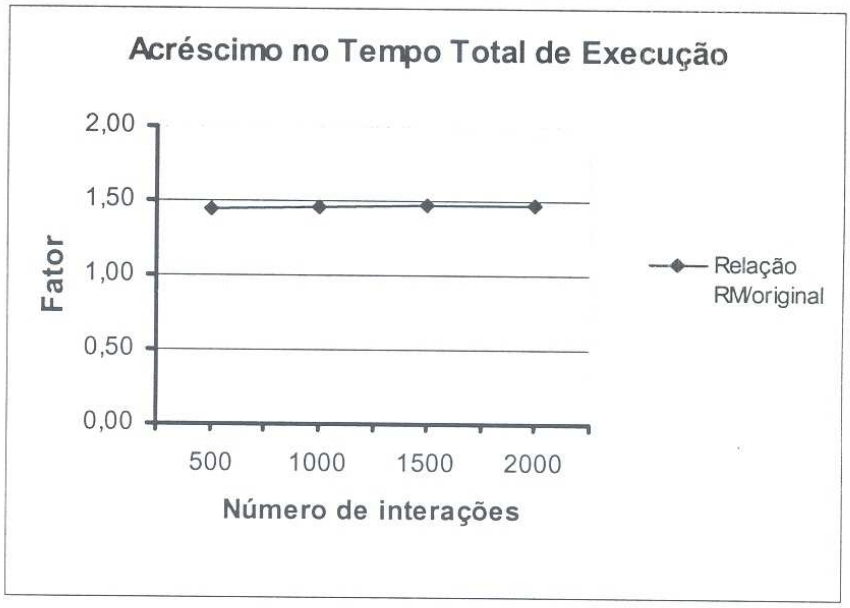

Figura 35 : Evolução dos fatores

Ainda segundo [CHI93], se o aumento no processamento causado pela técnica de reflexão, através da existência do nível meta, for limitado a um fator de até 10 vezes, então o uso da reflexão é justificado.

Nas medições o fator obtido ficou bem abaixo deste limite de 10, logo pode-se concluir que a utilização da reflexão computacional dentro do mecanismo proposto é justificável e aceitável.

Existem algumas considerações importantes que devem ser feitas em relação aos tempos de execução obtidos. Primeiramente, o desenvolvimento da classe que implementa as funcionalidades do gerente de *rollback* não teve com objetivo final a geração de um código otimizado, e sim mostrar a viabilidade do mecanismo proposto. Desta forma, o código do meta objeto gerente poderia ser otimizado de forma a melhorar o seu desempenho.

Outra consideração importante é o fato do gerente só entrar em operação quando o federado adota um comportamento otimista. Assim, se durante a execução da simulação nenhum federado invocar o método *flushQueueRequest* da classe *RTIAmbassador*, o desempenho da federação não será afetado de nenhuma forma.

O gerente de *rollback* foi implementado como uma classe adicional que só é instanciada no momento em que o federado assume um comportamento otimista, conforme já foi mencionado. No momento em que esta classe é instanciada existe um acréscimo também no consumo de memória. Este aumento na utilização de memória RAM também foi medido durante os experimentos realizados. Os valores obtidos através desta medição estão apresentados na figura 36.

Simulação	**Memória Alocada (Mb)**	
	Total	**Residente**
Original	15	10
Com o gerente	16	13
Acréscimo	6,67%	30%

Figura 36 : Alocação de memória

As medições efetuadas levaram em conta dois indicadores distintos: memória total e memória residente. A memória total representa o tamanho total da simulação (processo) englobando área de código, área de dados e pilha. Já o outro indicador representa a parte do processo que fica residente na memória durante a execução. A nível de comparação, foi elaborado o gráfico apresentado na figura 37.

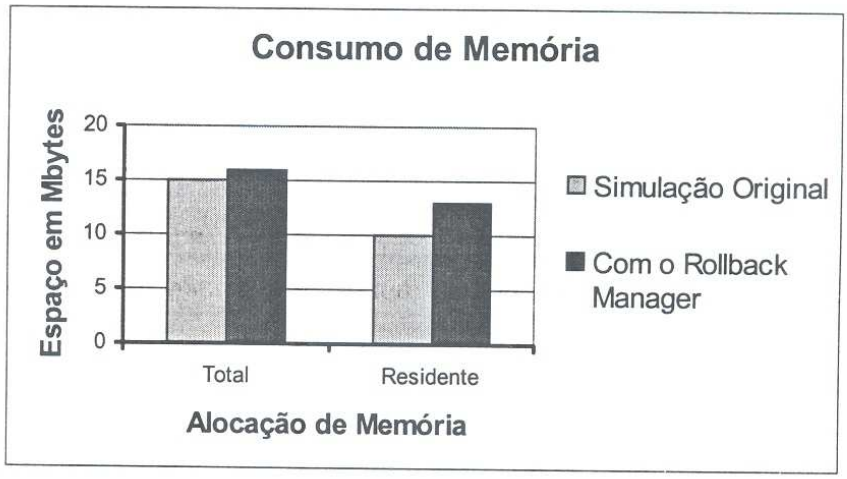

Figura 37 : Gráfico de alocação de memória

A utilização de gerente de *rollback* gerou um acréscimo de 6,67% na quantidade total de memória alocada em relação a simulação original. Já a área de memória residente aumentou 30% devido ao instanciamento da classe referente ao gerente de *rollback*. Estes valores de alocação de memória foram obtidos a partir de ferramentas nativas do sistema operacional Solaris e representam apenas uma aproximação dos valores reais, porém são suficientes para efeitos de comparação.

Além destes números obtidos existem outros indicadores que precisam ser levados em consideração. Na simulação original, o próprio federado é encarregado de gerenciar todo o processo de *rollback*. Neste caso, o código do federado é bem maior e mais complexo, o que eleva o tempo de desenvolvimento. Cada federado precisa implementar o seu próprio mecanismo de gerência de *rollback*. Com o uso do mecanismo proposto, o mesmo gerente de *rollback* pode ser utilizado por todos os federados otimistas já que o gerente independe da implementação do federado. Este reaproveitmento de código diminui significativamente o esforço e o tempo para o desenvolvimento de simulações otimistas.

6 Conclusão e Perspectivas

A utilização de técnicas de reflexão computacional no contexto apresentado mostra-se bastante útil, por permitir que o processamento de eventos indevidos seja desfeito de forma quase que transparente ao federado. Neste caso o gerente de *rollback* irá gerenciar os estados do federado e as mensagens enviadas/recebidas através de estruturas de controle internas. O gerente é capaz de identificar a necessidade de *rollback*, bem como tomar todas as ações necessárias para fazer com que o federado retorne a um estado seguro anterior à violação de causalidade. Além de aliviar o processamento interno do federado, o gerente toma para si a responsabilidade de cancelar mensagens indevidamente enviadas a outros federados, resultantes deste processamento incorreto.

O gerente de *rollback* irá realizar as tarefas que são comuns a todos os federados que adotam a estratégia otimista para sincronismo de eventos e conseqüente avanço de tempo. Como este gerente pode ser reutilizado para vários federados otimistas, o esforço empreendido no desenvolvimento desta solução pode resultar em um ganho de tempo no desenvolvimento dos federados. O código dos federados torna-se mais simples e claro pois todo o controle e gerenciamento do *rollback* fica sob responsabilidade do gerente. Além da simplificação do código, a utilização do gerente de *rollback* pode aumentar a confiabilidade do código, pois reduz a chance de erros de lógica e programação por parte dos desenvolvedores da simulação.

Os resultados obtidos mostraram que o acréscimo de tempo ocasionado pela interecptação e tratamento de algumas mensagens não chega a comprometer o desempenho da simulação. O fator calculado, cerca de 1,50 , representa um valor praticamente estável e bem abaixo do fator 10 sugerido por [CHI93] como valor máximo admitido em um sistema com reflexão computacional. O consumo de memória aumentou com a utilização do gerente de *rollback* devido a instanciação de uma nova classe, porém este acréscimo não depende do número de interações e sim da quantidade e tamanho dos atributos dos federados.

Apesar de agregar novas funcionalidades ao serviço de gerenciamento de tempo da arquitetura HLA, o mecanismo proposto neste trabalho não altera nenhuma interface definida na *I/F Spec*. Apenas foram incluídos os métodos *setState* e

getState que permitem ao gerente de *rollback* ter acesso ao estado dos federados. Desta forma, um modelo de simulação construído usando o gerente de *rollback* pode interagir com outros modelos que não utilizem este mecanismo ou mesmo com modelos desenvolvidos por terceiros.

Pesquisas Futuras

Um campo muito interessante para futuras pesquisas é o inter-relacionamento entre a arquitetura HLA e a orientação a objetos. O desenvolvimento de modelos para simulações HLA está baseado no paradigma da orientação a objetos. Através deste paradigma, as entidades do modelo de simulação são perfeitamente expressadas por objetos, que podem ser reusados tornando o desenvolvimento de simulações muito mais rápido.

Em uma simulação distribuída, os federados podem ser processados em diversos computadores, o que torna fundamental um mecanismo seguro de comunicação entre eles. A proposta inicial da arquitetura HLA apresenta apenas uma série de especificações de serviços que devem ser providos pela RTI. As implementações iniciais da RTI possuem facilidades de comunicação entre os federados que estão sendo executados em diferentes processadores conectados via rede. Cada implementação da RTI utiliza protocolos e mecanismos de comunicação proprietários que se encarregam de garantir a interoperabilidade entre os federados. Em última instância, trata-se de uma questão de comunicação entre objetos distribuídos [KUH94]. Nesta área de pesquisa existem várias propostas de sistemas operacionais distribuídos orientados a objetos, como *Amoeba* [MUL90] e *Clouds* [DAS91], todos proprietários e pouco aderentes a padrões de mercado. A solução é utilizar uma arquitetura padrão adotada por uma gama grande de empresas de software. A arquitetura CORBA [OMG98] proposta pela OMG e X/Open atende a todos os requisitos necessários para a simulação via HLA, proporcionando um ambiente padrão para desenvolvimento e execução de aplicação baseadas em objetos distribuídos. Um dos maiores benefícios desta arquitetura é tornar transparente para a aplicação os aspectos referentes a comunicação entre os objetos.

A extensão da arquitetura CORBA para atender aos requisitos de simulações distribuídas abre um campo novo e com futuro promissor para novas pesquisas e proposições de novos serviços que possam ser incorporados ao padrão CORBA. A

própria OMG percebeu a importância e utilização crescente das técnicas de simulação distribuídas e criou um grupo responsável pela sua padronização, o SimSIG (*Simulation Special Interest Group*). Este grupo tem como objetivo desenvolver uma arquitetura para simulação baseada na definição de modelos de dados e objetos, interfaces e padrões CORBA. O resultado deste esforço será padronizar este domínio de aplicações vertical, de forma a garantir a interoperabilidade entre os sistemas de simulação baseados em objetos, através da arquitetura CORBA. Contudo, este trabalho ainda está em fase inicial definido em uma RFI (*Request for Information*) [RFI98] onde a comunidade científica e empresas são convidadas a enviar sugestões para ajudar o SimSIG a tomar decisões úteis e coerentes na definição de padrões para esta área.

Os federados se comunicam com a RTI através de interfaces bem definidas. Já existe uma proposta para definição de interfaces expressas IDL CORBA. Estas interfaces podem ser usadas para que os federados e a RTI se comuniquem através de uma infra-estrutura CORBA padrão, o que garante a transparência em relação aos aspectos de comunicação entre os objetos distribuídos via rede.

Esta fase inicial, abre um campo novo e com futuro promissor para novas pesquisas e proposições de novos serviços que possam ser incorporados ao padrão CORBA. Neste caso poderia ser estudada uma extensão das definições IDL no que se refere ao gerenciamento de tempo, hoje muito dependente da implementação da RTI.

Outro campo de pesquisa é a utilização de linguagens de programação que suportem as técnicas de reflexão computacional. Existem várias linguagens disponíveis no mercado e no mundo acadêmico que permitem a criação de objetos reflexivos em tempo de compilação ou pré-compilação. Os testes realizados com o protótipo do gerente de *rollback* foram feitos através da substituição manual das chamadas aos métodos da RTI. Estas chamadas foram alteradas para os métodos do meta objeto de forma a validar o mecanismo proposto. Já com o uso de uma linguagem com suporte a reflexão esta tarefa poderia ser feita de forma transparente. Os objetos desejados poderiam ser especificados no código como reflexivos e durante a compilação toda a estrutura de controle seria montada. Durante a execução as chamadas aos métodos dos objetos reflexivos seriam automaticamente desviadas, no caso para o código do meta objeto gerente. Existem

diversas linguagens de programação que utilizam protocolos de meta objetos [KIC93] que poderiam ser pesquisadas para uso neste mecanismo de *rollback* proposto. Entre as linguagens mais utilizadas estão: *CLOS* [KIC93] , *OpenC++* e *OpenJava*. Como todo o código da RTI está disponível na linguagem C++, a utilização do *OpenC++* [CHI95] [CHI93] poderia ser estudada pois utiliza a mesma sintaxe, o que facilitaria a sua integração dentro do ambiente de desenvolvimento de simulações dentro da arquitetura HLA. No caso da simulação desenvolvida, uma alternativa seria a utilização da linguagem *OpenJava* [TAT99] que está sendo desenvolvida no Japão. Neste caso o código Java com os meta objetos e métodos definidos de acordo com o *OpenJava MOP* (*Metaobject Protocol*) é pré-processado. Durante este processo o código é dividido em pequenas partes, que são então codificadas em classes de meta objetos. As classes são unidas em um código fonte Java que é então processado através do JDK (*Java Development Kit*). Contudo, o *OpenJava* ainda está em fase de amadurecimento (versão 1.0) e não suporta muitas características essenciais para o desenvolvimento de simulações baseadas na arquitetura HLA. Assim com o *OpenJava*, existem outras propostas de arquiteturas reflexivas para o ambiente Java, como a *metaXa* [GOL98] que também poderiam ser estudas e aplicadas para a construção do gerente de *rollback* proposto neste trabalho.

Referências Bibliográficas

[BAU93] Bauer, H. e Spoorer, C., "**Reducing Rollback Overhead in Time Warp based Distributed Simulation with Optimized Incremental Sate Saving**", 26th Annual Simulation Symposium, 1993

[BEL90] Bellenot, S., "**Global Virtual Time Algorithms**", Proceedings of the 1990 Multiconference on Distributed Simulation, págs 122-127, 1990

[BEL92] Bellenot, S., "**State Skipping Performance with the Time Warp Operating System**", 6th Workshop on Parallel and Distributed Simulation, págs 53-61, 1992

[BIR93] Birman, Kenneth, "**The Process Group Approach to Reliable Distributed Computing**", Communications of the ACM, Dezembro 1993.

[BRA93] Braudes, R. e Zabele, S., "**Requirements for Multicast Protocols**", RFC 1458, Maio, 1993

[CAL96] Calvin, J. e Weatherly, R., "**An Introduction to the HLA RTI**", MIT e MITRE, 1996

[CAR94] Carothers, C.D., Fujimoto, R., Lin, Y.B. e England, P., "**Distributed Simulation of Large Scale PCS Networks**", Proceedings of the 2nd International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE Computer Society Press, Los Alamitos, CA, 1994

[CHA79] Chandy, K.M. e Misra, J., "**Distributed Simulation: A case study in design and verification of distributed programs**", IEEE Transactions on Software Engineering, vol.SE-5, n. 5, págs 440-452, Setembro, 1979

[CHA83] Chandy, K.M. , Misra, J. Haas, M., **"Distributed Deadlock Detection"**, ACM Transactions on Computer Systems, vol.1, n. 2, págs 144-156, Maio, 1983

[CHE94] Cheung e Loper, **"Synchronizing simulations in DIS"**, 1994 Winter Simulation Conference Proceedings, págs 1316-1323, 1994

[CHI93] Chiba, S. e Masuda, T., **"Designing an Extensible Distributed Language with Meta-level Architecture"**, Proceedings of ECOOP`93, págs 482-501, Germany, 1993

[CHI95] Chiba, S., **"A Metaobject Protocol for C++"**, Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), págs 285-299, Outubro, 1995

[CLE94] Cleary, J., Gomes, F., Unger, B., Zhonge, X. e Thudt, R., **"Cost of State Saving & Rollback"**, Proceedings of the 1994 Workshop on Parallel and Distributed Simulation, págs 94-101, 1994

[COM88] Comer, Douglas, **"Internetworking with TCP/IP – Principles, Protocols, and Architecture"**, Prentice-Hall, 1988

[COM96] DMSO, **"HLA Compliance Rules"**, DoD HLA for Simulations, http://www.dmso.mil/dmso/docslib/, 1996

[DAH98] Dahmann, J., **"Introduction to High Level Architecture"**, Proceedings of the 1998 Winter Simulation Conference, Washington, Dezembro, 1998

[DAS91] Dasgupta P., R.J. LeBlanc et al, **"The Clouds distributed operating System"**, IEEE Computer, 24, 34-44, 1991

[DAS94] Das, R.S., Fujimoto, R., **"An Adaptive Memory Management Protocol for Time Warp Parallel Simulation"**, Proceedings of the 1994 Conference on Measurement and Modeling of Computer Systems, págs 201-210, 1994

[DAS97] Das, R.S., Fujimoto, R., **"Adaptive Memory Management and Optimistic Control in Time Warp"**, ACM Transactions on Modeling and Computer Simulation, vol.7, n 2, págs 239-271, 1997

[DET88] Detlefs, D.L. e Herlihy, M.P., **"Inheritance of Synchronization and Recovery Properties in Avalon/C++"**, Computer, págs. 57-69, Dezembro, 1988

[DMS99] DMSO, **"Federation Development Tools"**, DoD HLA for Simulations, http://hla.dmso.mil/hla/fed_tools/, 1999

[FAB96] Fabre, J., Nicomette, V., Pérennou, T., Stroud R. e Wu, Z., **"Implementing Fault Tolerant Applications using Reflective Object Oriented Programming"**, Proceedings of the 25th IEEE International Symposium on Fault Tolerant Computing, Junho, 1996

[FER94] Ferscha, A. and Tripathi, S.K., **"Parallel and Distributed Simulation of Discrete Event Systems"**, University of Maryland, Agosto, 1994

[FUJ89] Fujimoto, R. M., **"Performance Measurements of Distributed Simulation Strategies"**, Transactions of the Society for Computer Simulation, 1989

[FUJ89a] Fujimoto, R. M., **"Time Warp on a Shared Memory Multiprocessor"**, Transactions of the Society for Computer Simulation, 6(3), págs 211-239, Julho, 1989

[FUJ90] Fujimoto, R. M., "**Parallel Discrete Event Simulation**", Communications of the ACM 33, págs 31-53, Outubro, 1990

[FUJ92] Fujimoto, R. M. e Nicol, D.M., "**State of the Art in Parallel Simulation**", 1992 Winter Simulation Conference Proceedings, págs 122-127, Dezembro, 1992

[FUJ95] Fujimoto, R. M., "**Parallel and Distributed Simulation**", Proceedings of the 1995 Winter Simulation Conference, Dezembro, 1995

[FUJ96] Fujimoto, R. M., "**Zero Lookahead and Repeatability in the High Level Architecture**", 1996

[FUJ98] Fujimoto, R. M., "**Time Management in the High Level Architecture**", SCS Simulation Magazine, Dezembro, 1998

[FUJ99] Fujimoto, R. M., "**Parallel and Distributed Simulation Systems**", Wiley Interscience, 1999

[FUL96] Fullford, D., "**Distributed Interactive Simulation: It's past, Present and Future**", Proceedings of the 1996 Winter Simulation Conference, 1996

[GAF88] Gafni, A., "**Rollback Mechanisms for Optimistic Distributed Simulation**", Proceedings of the 1988 SCS Multiconference on Distributed Simulation, págs 61-67, Fevereiro, 1988

[GHO94] Ghosh, P. e Fujimoto R., "**PORTS: A parallel, optimistic, real-time simulator**", 8th Workshop on Parallel and Distributed Simulation, Julho, 1994

[GLO96] DMSO, "**HLA Glossary**", DoD HLA for Simulations, http://hla.dmso.mil, 1996

[GOL94] Golner e Pollak, "**The application of network time protocol to implement DIS absolute timestamps**", 11th Workshop on Standards for the Interoperability of Distributed Simulations, vol.2, págs 431-440, 1994

[GOL98] Golm, M., "**metaXa and the Future of Reflection**", OOPSLA Workshop on Reflective Programming in C++ and Java, October, 1998, Vancouver, Canadá

[HLA97] Defense Modeling & Simulation Office, "**HLA Overview**", DMSO, 1997

[HON94] Honda, Y. e Tokoro, M., "**Reflection and Time Dependent Computing: Experiences with the R2 Architecture**", Technical Report, SONY Laboratory Inc., Japão, 1994

[IEE93] IEEE, "**IEEE Standard for Information Technology – Protocols for DIS Applications**", IEEE standard 1278, http://standards.ieee.org, Maio, 1993

[IEE95] IEEE, "**IEEE Standard for Distributed Interactive Simulation – Applications Protocols**", IEEE standard 1278.1, http://standards.ieee.org, 1995

[IEE99] IEEE, "**Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules** ", IEEE draft standard 1516, http://standards.ieee.org, Julho, 1999

[IST91] Military Standard, "**Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation**", Institute for Simulation and Training, Flórida, Outubro, 1991

[JEF85] Jefferson, D., "**Virtual Time**", ACM Transactions on Programming Languages and Systems, vol.7, n.3, págs 404-425, Julho, 1985

[JEF87] Jefferson, D., Beckman, B., Wieland, F., Blume, L., DiLorento, M., Hontalas, P., Laroche, P., Sturdevant, K,. Tupman, J., Warren, V., Wedel, J., Younger, H. e Belenot, S., "**Distributed Simulation and the Time Warp Operating System**", 10th Symposium on Operating System Principles, págs 77-93, Novembro, 1987

[JEF90] Jefferson, D., "**Virtual Time II**", ACM Symposium on Principles of Distributed Computation, págs 75-90, Agosto, 1990

[JHA96] Jha, V. e Bagrodia, R., "**Simultaneous Events and Lookahead In Simulation Protocols**", University of California, Los Angeles, Technical Report 960043, 1996

[KAN91] Kanarick, "**A Technical overview and history of the SIMNET project**", Advances in Parallel and Distributed Simulation, volume 23, SCS Series, Janeiro, 1991

[KIC93] Kiczales, G., Ashley, M., Rodriguez L., Vahdat A. e Bobrow, D., "**Metaobject Protocols: Why We Want Them and What Else They Can Do**", Object Oriented Programming: The CLOS Perspective, MIT Press, 1993

[KOI96] Koifman,A. e Zabele,S., "**RAMP: A Reliable Adaptive Multicast Protocol**", 5th Annual Joint Conference of the IEEE Computer and Communication Societies, California, Março, 1996

[KUH94] Kuhl, Frederick, et al, "**Object Request Broker: Foundation for Distributed Simulation**", Wiley, 1994

[LIN89] Lin, Yi-Bing e Lazowska, E.D., "**The Optimal Checkpoint Interval in Time Warp Parallel Simulation**", Technical Report 89-09-04, Department of Computer Science and Engineering, University of Washington, 1989

[LIN90] Lin, Yi-Bing, Preiss,B. e Lazowska, E.D., "**Optimally Considerations for Time Warp Parallel Simulation**", Proceedings of the 1990 SCS Multiconference on Distributed Simulation, págs 29-34, Janeiro, 1990

[LIN90a] Lin, Yi-Bing e Lazowska, E.D., "**Reducing the State Saving Overhead for Time Warp Parallel Simulation**", Technical Report 90-02-03, Department of Computer Science and Engineering, University of Washington, 1990

[LIN91] Lin, Yi-Bing e Preiss,B., "**Optimal Memory Management for Time Warp Parallel Simulation**", ACM Transactions on Modeling and Computer Simulation, 1(4), págs 283-307, Outubro, 1991

[LIN93] Lin, Yi-Bing, Preiss,B., Loucks, W.M. e Lazowska, E.D., "**Selecting the Checkpoint Interval in Time Warp Simulation**", Proceedings of the 1993 Workshop on Parallel and Distributed Simulation, 1993

[LIP90] Lipton, R.J., e Mizell, D.W, "**Time Warp vs. Chandy-Misra: A Worst Case Comparison**", Proceedings of the 1990 SCS Multiconference on Distributed Simulation, págs 137-143, Janeiro, 1990

[LOC95] Locke, John, "**An Introduction to the Internet Networking Environment and SIMNET/DIS**", Naval Postgraduate School, 1995

[MAE87] Maes, P., "**Concepts and Experiments in Computational Reflection**", Proceedings of OOPSLA'87, 1987

[MAS95] U.S. Department of Defense, "**DoD Modeling and Simulation Master Plan**", DMSO, http://www/dmso.mil, Outubro, 1995

[MAZ94] Maziero, C.A., "**Conception et réalisation d'un noyau de système réparti pour la simulation parallèle**", PhD thesis, Université de Rennes 1 – França, 1994

[MEH92] Mehl, H., "**A Deterministic Tie-Breaking Scheme for Sequential and Distributed Simulation**", Proceedings of the Workshop on Parallel and Distributed Simulation, Society for Computer Simulation, 1992

[MIL92] Mills, L., "**Network Time Protocol – specification, implementation and analysis**", 1992

[MIS86] Misra, J,. "**Distributed Discrete-Event Simulation**", ACM Computing Surveys, vol.18, n.4, págs 39-65, Março, 1986

[MUL90] Mullender, S.J., et al, "**Amoeba: a distributed operating system for the 1990's**", IEEE Computer, 23, 44-53, 1990

[NIC94] Nicol, D. e Fujimoto, R., "**Parallel Simulation Today**", Operations Research, 1994

[NIS96] Nishida, D., "**Estudo e Implementação do Modelo Reflexivo Tempo Real RTR sobre a Linguagem Java**", Dissertação de Mestrado, UFSC, 1996

[NPS] Naval Postgraduate School Research Group, "**The NPSNET Project**", http://www.cs.nps.navy.mil/research/npsnet

[NYG78] Nyggard, K. e Dahl, O.J., "**The development of the Simula language**", ACM Conference on the History of Programming Languages, Junho, 1978

[OMG98] OMG, "**Common Object Request Broker: Architecture and Specification (CORBA)**", revisão 2.2, Fevereiro 1998, http://www.omg.org/corba/corbaiiop.html

[OMT96] DMSO, "**Object Model Templates**", DoD HLA for Simulations, http://www.dmso.mil/dmso/docslib/OMT_v0.2.doc, 1996

[ORF98] Orfali e Harkey, "**Client/Server Programming with JAVA and CORBA**", Willey, 1998

[PAL93] Palaniswamy, A.C.. e Wilsey, P.A., "**An Analytical Comparison of Periodic Checkpointing and Incremental State Saving**", Proceedings of the 1993 Workshop on Parallel and Distributed Simulation, págs 127-134, 1993

[POP91] Pope, A. e Schaffer, R., "**The SIMNET Network and Protocols**", BBN Systems and Technologies Corporation report n.7627, Junho, 1991

[PRA91] Prakash, A., Subramanian, R., **"Optimistic Distributed Simulations**", Proceedings of the 24th Annual Simulation Symposium, págs 123-132, Abril, 1991

[PRE91] Preiss, B., Loucks, W. e MacIntyre, I., **"Null Message Cancellation in Conservative Distributed Simulation**", Simulation Councils, 1991

[PRE93] Preiss, B., Loucks, W. e Lazowska, E., **"Selecting the Checkpoint Interval in Time Warp Simulation**", Simulation Councils, 1993

[PRO98] DMSO, "**HLA RTI Programmer's Guide 1.3**", DoD, Setembro, 1998

[REI89] Reiher, P., Wieland, F. e Jefferson, D., "**Limitation of Optimistic in the Time Warp Operating System**", Winter Simulation Conference Proceedings, págs 765-770, Dezembro, 1989

[REI90] Reiher, P., Fujimoto, R., Bellenot, S. e Jefferson, D., "**Cancellation Strategies in Optimistic Execution Systems**", Proceedings of the 1990 SCS Multiconference on Distributed Simulation, págs 112-121, Janeiro, 1990

[RFI98] Object Management Group, "**Distributed Simulation: Request for Information**", OMG, Março, 1998

[RIG89] Righter, R. e Walrand, J.C., "**Distributed simulation of discrete event systems**", Proceedings of the IEEE 77, págs 99-113, Janeiro, 1989

[RTI96] DMSO, "**Interface Specification**", DoD HLA for Simulations, http://www.dmso.mil/dmso/docslib/IF_Spec_v0.35.doc, 1996

[RUB98] Rubira, C. e Buzato, L., "**Sistemas OOTF usando Reflexão Computacional**", UNICAMP, 1998

[SIE96] Siegel, J., "**CORBA: Fundamentals and Programming**", Wiley ,1996

[SMI96] Smith, W.Garth, et al, "**A Distributed Interactive Simulation Intranet using RAMP**", TASC, Inc., 1996

[SOW85] Sowizral, H. e Jefferson, D., "**Fast Concurrent Simulation using the Time Warp Mechanism**", Distributed Simulation 1985, págs 63-69, SCS, Simulation Councils, Inc., La Jolla, California, 1985

[SUP99] DMSO, "**RTI Bugs/Limitations Notes**", RTI Support Help Desk, http://vtcs20a.virtc.com/fom-serve/cache/1.html, 1999

[TAY96] Taylor, D.,"**DIS-Lite and Query Protocol: Message Structures**", Proceedings of the 14th DIS Workshop on Standards for Interoperability of Distributed Simulation, 1996

[UNG93] Unger, B.W., Cleary, J.G., Covington, A. e West, D., "**An External State Management System for Optimistic Parallel Simulation**", Winter Simulation Conference Proceedings, Dezembro, Los Angeles, CA, 1993

[TAT99] Tatsubori, M., "**An Extension Mechanism for the Java Language**", Dissertação de Mestrado, Tsukuba University, Japão, Fevereiro,1999

[VAR99] Vardânega, F. e Maziero, C.A., "**Simplifying Optimistic Distributed Simulations in HLA**", Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'99), Cambridge, MA (MIT), EUA, Novembro, 1999

[VAR99] Vardânega, F. e Maziero, C.A., "**Using Computational Reflection in Optimistic Distributed Simulations**", Proceedings of the IEEE International Conference of the Chilean Computer Science Society (SCCC´99), Talca, Chile, Novembro 1999

[VAR99] Vardânega, F. e Maziero, C.A., "**Reflective Management of Optimistic Federates in HLA**", Proceedings of the 11th European Simulation Symposium and Exhibition Simulation in Industry (ESS'99), Erlangen, Alemanha, Outubro 1999

[VIS93] DIS Vision Document, ftp.sc.ist.ucf.edu/SISO/dis/library/vision.doc, 1993

[WIE89] Wieland, F., et al., "**Distributed Combat Simulation and Time Warp: The Model and its performance**", Proceedings of the 1989 SCS Multiconference on Distributed Simulation, págs 14-20, Março, 1989

[WIL94] Wilson, L. e Weatherly, R., "**The Aggregate Level Simulation Protocol: An evolving system**", 1994 Winter Simulation Conference Proceedings, págs 781-787, Dezembro, 1994

[YON88] Yonezawa, A. e Watanabe, T., "**Reflection in an Object Oriented Concurrent Language**", Proceedings of OOPSLA'88, Setembro, 1988