UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ARQUITETURA PARA INTERCOMUNICAÇÃO DE AMBIENTES DE REALIDADE VIRTUAL DISTRIBUÍDOS BASEADOS EM AGLOMERADOS GRÁFICOS REMOTOS

DIEGO ROBERTO COLOMBO DIAS

ORIENTADOR: PROF. DR. LUIS CARLOS TREVELIN

São Carlos – SP

maio de 2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ARQUITETURA PARA INTERCOMUNICAÇÃO DE AMBIENTES DE REALIDADE VIRTUAL DISTRIBUÍDOS BASEADOS EM AGLOMERADOS GRÁFICOS REMOTOS

DIEGO ROBERTO COLOMBO DIAS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Sistemas Distribuídos e Redes de Computadores.
Orientador: Prof. Dr. Luis Carlos Trevelin

São Carlos – SP

maio de 2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

D541a

Dias, Diego Roberto Colombo
Uma arquitetura para intercomunicação de ambientes de realidade virtual distribuídos baseados em aglomerados gráficos remotos / Diego Roberto Colombo Dias. -- São Carlos : UFSCar, 2016.
150 p.

Tese (Doutorado) -- Universidade Federal de São Carlos, 2016.

1. Computação distribuída. 2. Aglomerado gráfico. 3. Realidade virtual. 4. Atividade colaborativa. I. Título.

UNIVERSIDADE FEDERAL DE SÀO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Tese de Doutorado do candidato Diego Roberto Colombo Dias, realizada em 23/05/2016:

Prof. Dr. Luis Carlos Trevelin
UFSCar

Prof. Dr. Hermes Senger
UFSCar

Prof. Dr. Cesar Augusto Cavalheiro Marcondes
UFSCar

Prof. Dr. Seiji Isotani
Unesp

Prof. Dr. Rodrigo Bonacin
FACAMP

Dedico esta tese aos meus pais e amigos.
E também a minha esposa Amanda, por sua paciência sem fim.

AGRADECIMENTOS

Gostaria de agradecer em especial ao meu orientador, Prof Luis Carlos Trevelin, por ter acreditado no meu potencial e por ter sido de grande valia para a condução da minha pesquisa, sempre presente e atencioso durante todo o meu período de doutorado.

Ao Prof Torsten Kuhlen, por ter me aceitado como membro do grupo de pesquisa em Realidade Virtual da RWTH Aachen University – Alemanha. Este período foi de extremamente importante no desenvolvimento deste trabalho, além do crescimento pessoal e cultural.

Eu não poderia deixar de agradecer ao Prof Marcelo Guimarães, mais que um prof, um amigo. Por suas incontáveis horas desprendidas as nossas conversas. Com certeza, esta tese não seria a mesma sem as suas contribuições.

Ao Prof José Remo, meu orientador de mestrado e amigo. Saiba que eu não estaria hoje aqui se você não tivesse acreditado em mim. Procede a informação, professor?

Ao meu amigo Bruno Gnecco. Esta pesquisa não seria a mesma se não fossem nossas discussões, parceiro durante todo o tempo.

A todos os meus amigos, Rafael, Patrícia, Mateus Viana, Mateus Ferrer, Mário e todos, mesmo não citados, um agradecimento especial.

Ao companheiro de pesquisa Alexandre, tendo colaborado muito no meu crescimento acadêmico. Companheiro de café, mas tinha que ser na Guanabara, do outro lado da cidade.

Aos colegas do DC–UFSCAR, em especial do LAVIIC, Glesio, Alfredo e Jéssica.

À Capes, pelo apoio financeiro durante todo o período do meu doutorado, assim como a bolsa de doutorado sanduíche no exterior (processo N 99999.002852/2014-04).

A todos os professores e funcionários do DC–UFSCar, por terem dado todo o suporte e contribuição à minha formação acadêmica.

Gostaria de agradecer especialmente minha esposa Amanda, que tem me acompanhado desde o início. Sua paciência e amor me mantiveram sempre firme no meu caminho.

Por vezes sentimos que aquilo que fazemos não é senão uma gota de água no mar.
Mas o mar seria menor se lhe faltasse uma gota

Madre Teresa de Calcuta

RESUMO

A Realidade Virtual é a tecnologia que combina a criação de ambientes virtuais sintéticos e tridimensionais (3D), caracterizados por proporcionar alto grau de imersão e interação. Paralelamente, a área de Trabalho Colaborativo permite que usuários realizem tarefas em conjunto estando em localidades remotas. Da combinação de aplicações de Realidade Virtual e de Trabalho Colaborativo, tem-se os ambientes virtuais colaborativos em 3D, que exigem soluções capazes de atender desde os requisitos de comunicação entre usuários em locais remotos até a geração de imagens em tempo real. Como consequência, as vantagens proporcionadas pelos aglomerados gráficos de computadores, como a capacidade de processamento, os indicam como possível solução a esta classe de aplicação. Por outro lado, torna-se um desafio a superação das desvantagens desses aglomerados, como a necessidade de comunicação inter–aglomerados para execução de tais ambientes, principalmente em redes baseadas em Internet. Esta tese teve como objetivo principal mitigar a influência das adversidades ocasionadas pelas redes de dados em ambientes virtuais colaborativos em 3D baseados em aglomerados gráficos remotos, de tal forma que fosse possível realizar adequações nas conexões entre estes em tempo de execução, visando manter a qualidade de serviço. Para isso, uma arquitetura foi definida e implementada, permitindo a comunicação inter–aglomerados de forma otimizada. Além disso, foi apresentado e implementado um modelo comportamental para análise de aplicações distribuídas baseado na arquitetura desenvolvida. Ao final foram apresentados estudos de caso com o intuito de avaliar a arquitetura.

Palavras-chave: Computação Distribuída, Aglomerado Gráfico, Realidade Virtual, Atividade Colaborativa

ABSTRACT

Virtual reality is a technology that combines the creation of synthetic and three-dimensional virtual environments (3D), providing a high degree of immersion and interaction. At the same time, the Collaborative Work area allows users to perform tasks together remotely. By combining Virtual Reality and Collaborative Work it is possible to create collaborative virtual 3D environments. However, collaborative virtual 3D environments pose communication challenges that essentially entail generating images in real time for remote users. Moreover, it is challenging to overcome the disadvantages of the involved clusters such as the need for interprocess communication, specially on Internet-based networks. In this thesis we set out to devise and implement an architecture that is able to mitigate issues caused by remote graphics clusters based on data networks, resulting in an optimized inter-cluster connection. Furthermore, a behavioral model for analyzing distributed applications based on our architecture was devised and implemented. It was also carried out case studies to evaluate the architecture.

Keywords: Distributed Computing, Graphic Cluster, Virtual Reality, Collaborative Activity

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE SIGLAS E ABREVIATURAS

3DCVE – *3D Collaborative Virtual Environment*

ACK – *Acknowledgement*

API – *Application Programming Interface*

CAVE – *Cave Automatic Virtual Environment*

CRT – *Cathodic Ray Tube*

CVE – *Collaborative Virtual Environment*

DLP – *Digital Light Processing*

DMP – *Distributed Multimedia Plays*

DVE – *Distributed Virtual Environment*

DWTP – *Distributed Worlds Transfer and Communication Protocol*

EVL – *Eletronic Visualization Laboratory*

FCL – *Fuzzy Control Language*

ICMP – *Internet Control Message Protocol*

ISTP – *InterStream Transit Protocol*

LAN – *Local Area Network*

LCD – *Liquid Crystal Display*

LIV – *Laboratório de Interfaces e Visualização*

LaVIIC – *Laboratório de Visualização Imersiva, Interativa e Colaborativa*

MMVE – *Massively Multi-user Virtual Environment*

MPI – *Message Passing Interface*

NACK – *Negative Acknowledgements*

NHVE – *Networked Haptic Virtual Environment*

NSF – *National Science Foundation*

NVE – *Networked Virtual Environment*

PDA – *Personal Digital Assistant*

PGM – *Pragmatic General Multicast*

PeX – *Projection Explorer*

QoS – *Quality-of-Service*

RA – *Realidade Aumentada*

RTEP – *Real Time Events Protocol*

RTP – *Real-time Transport Protocol*

RTSP – *Real Time Streaming Protocol*

RTT – *Round Trip Time*

RV – *Realidade Virtual*

SCTP – *Stream Control Transmission Protocol*

SDP – *Sockets Direct Protocol*

SGI – *Silicon Graphics*

SIP – *Session Initiation Protocol*

SVE – *Shared Virtual Environment*

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

UTP – *Unshielded Twisted Pair*

VI – *Visualização de Informação*

VRTP – *Virtual Reality Transfer Protocol*

WAN – *Wide Area Network*

XML – *Extensible Markup Language*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contexto

Atualmente, existe uma busca por novos meios para melhorar a interface e a interação dos usuários com os sistemas computacionais. Dentre as tentativas, existem as relacionadas com a Realidade Virtual (RV), que possibilitam que os usuários se envolvam e interajam com simulações. Para tanto, as aplicações dessa área, como o Second Life (SECONDLIFE, 2016), o Active Worlds (ACTIVEWORLDS, 2016), o IMVU (IMVU, 2016), o Eye gaze (EYEGAZE, 2016) e o VizGrid (OKUDA *et al.*, 2007) usufruem de diversas soluções, desde um computador pessoal até ambientes distribuídos.

Os ambientes distribuídos de RV são capazes de resolver problemas complexos, como o suporte a centenas de usuários e a comunicação entre eles. Entretanto, tradicionalmente, não foram projetados para aplicações que exijam alto poder computacional, como, por exemplo, as que exibem imagens em alta resolução. Além disso, estas aplicações são voltadas a um domínio específico de problema e não oferecem suporte a dispositivos avançados de entrada/saída 3D (tridimensional), como CAVEs (*Cave Automatic Virtual Environment*) (CRUZ-NEIRA *et al.*, 1992) e rastreadores de movimentos.

Para suprir limitações como essas, nos dias atuais utiliza-se aglomerados (*clusters*) gráficos de computadores (DIAS *et al.*, 2015; NETO *et al.*, 2015; DIAS *et al.*, 2014; VIEL *et al.*, 2012; DIAS, 2011; DIAS *et al.*, 2010b; DIAS *et al.*, 2010a; RAFFIN *et al.*, 2006; SOARES, 2005; GUIMARAES, 2004) e *softwares* específicos, como o FlowVR (ALLARD *et al.*, 2004), Syzygy (SCHAEFFER; GOUDESEUNE, 2003), Chromium (CHROMIUM, 2016), Net Juggler (ALLARD *et al.*, 2002), Vista VR Toolkit (VISTA, 2016) e a libGlass (LIBGLASS, 2016). Assim, é possível superar as barreiras para a adoção de dispositivos avançados de entrada/saída

3D, mas não as referentes à localidade, ou seja, a aplicação é executada somente em um local físico.

Efetuar tarefas independentemente da localidade e de forma colaborativa é algo corriqueiro nos dias atuais em diversas atividades, como, por exemplo, na escrita de textos. Assim, expandir as fronteiras das aplicações de RV com suporte a dispositivos avançados e adicionar a possibilidade de colaboração é uma oportunidade e, consequentemente, um desafio a ser superado. Esta tese apresenta o tratamento da comunicação inter–aglomerados necessária ao desenvolvimento de tais ambientes, aqui denominados ambientes virtuais colaborativos em 3D, do inglês *3D collaborative virtual environments* (3DCVE) (DIAS *et al.*, 2015; PICK *et al.*, 2014; PONTONNIER *et al.*, 2014). Dessa forma, espera-se permitir a construção de ambientes virtuais colaborativos em 3D, como, por exemplo, os que possibilitam a discussão de um projeto arquitetônico ou mecânico, o que pode beneficiar tanto a academia quanto a indústria. Para alcançar esse objetivo é necessário superar problemas de comunicação referentes a latência, ao atraso na entrega e a perda de pacotes e problemas comuns a redes de dados baseadas em Internet.

Nesta tese é apresentado o desenvolvimento de uma arquitetura de comunicação e sincronismo para 3DCVEs baseados em aglomerados gráficos remotos que, normalmente, possuem requisitos em relação à troca de mensagens síncronas e assíncronas, barreiras de sincronismo e definição de topologias de redes e protocolos de comunicação. Esta solução diferencia das presentes na literatura, geralmente baseadas em objetos distribuídos (SEMENTILLE, 1999) e chamadas remotas de procedimentos (RODELLO, 2003; CORRêA, 2010).

A Figura 1.1 apresenta o ambiente de teste utilizado para validar a arquitetura desenvolvida, sendo composto por sistemas de RV interligados via Internet colaborando na mesma simulação. Cada localidade possui um aglomerado gráfico, um sistema próprio de multiprojeção e dispositivos específicos de interação. Como estudo de caso, foram utilizados os sistemas de multiprojeção da Universidade Federal de São Carlos (UFSCar), da Universidade Estadual Paulista (Unesp)de Bauru e da RWTH Aachen University. Assim, uma das principais contribuições desta tese foi criar uma solução de comunicação que permita a interligação de ambientes de multiprojeção baseados em aglomerados gráficos remotos.

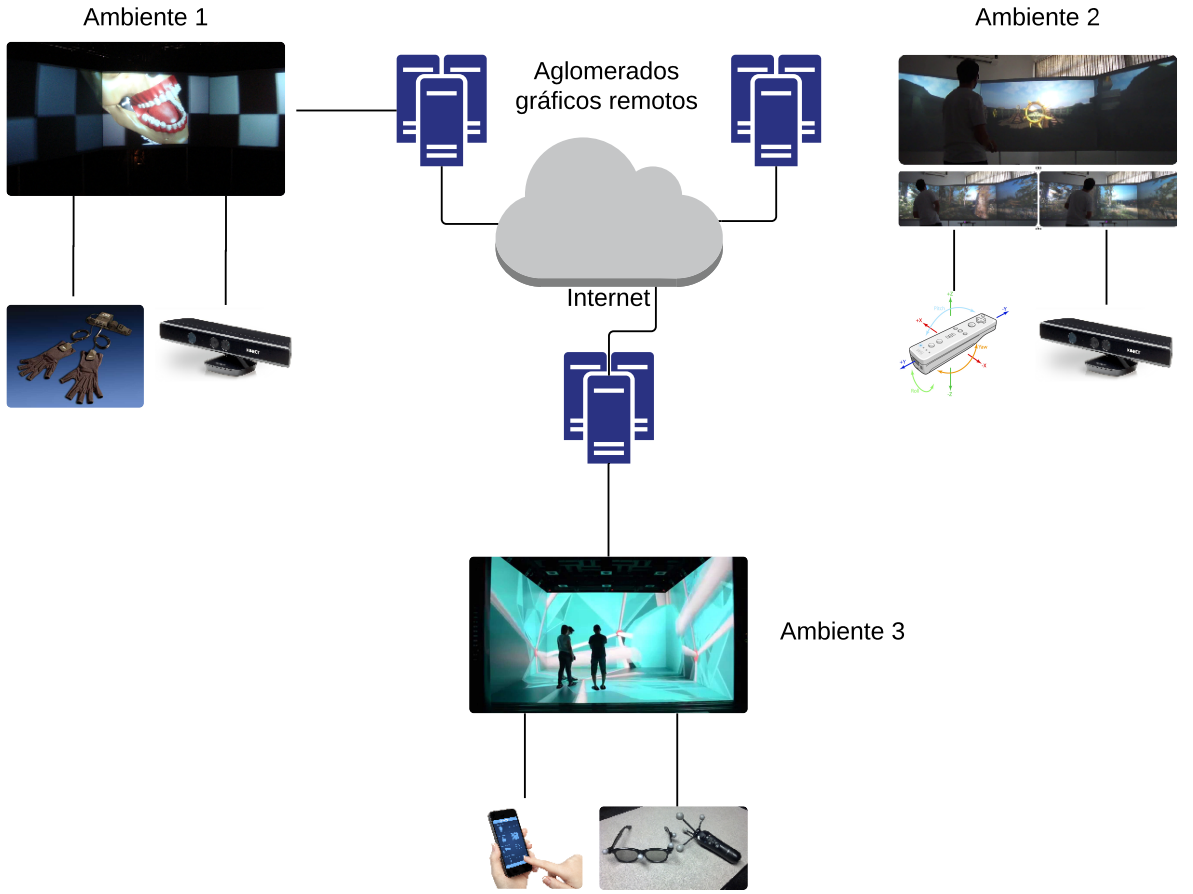


Figura 1.1: Aglomerados gráficos remotos.

Dentre as principais contribuições científicas contidas nesta tese, pode-se destacar o modelo de comunicação e o modelo de análise de funcionalidades realizados via troca de mensagens, como também um modelo de funcionalidades utilizado para verificar o funcionamento das aplicações. Esse modelo de funcionalidades foi adicionado na ferramenta GTracer (GUIMARAES, 2004), que originalmente apenas permitia a visualização de mensagens de nodos de um aglomerado (intra–aglomerado). O GTracer foi estendido, permitindo que mensagens trafegadas em arquiteturas inter–aglomerados pudessem ser verificadas, possibilitando aos desenvolvedores/usuários usufruir de uma interface gráfica para efetuar verificações em relação ao funcionamento de aplicações distribuídas de RV. A contribuição tecnológica foi alcançada por meio do desenvolvimento de *plugins* baseados na biblioteca libGlass (LIBGLASS, 2016).

1.2 Objetivos e motivação

Várias aplicações, como, por exemplo, o Google Docs, acabaram com as barreiras de tempo e disposição geográfica existentes nas interações entre membros de um grupo, possibilitando a

comunicação de maneira ubíqua. Esse fim de limite de tempo e de localização presentes nas aplicações colaborativas é um dos fatores iniciais que motivam esta tese. Porém, em se tratando de 3DCVEs, as soluções encontradas na literatura são limitadas, pois não suportam imagens de alta resolução ou o uso de qualquer dispositivos de multiprojeção, sendo limitadas a um domínio específico de problema. A carência de soluções que tratam esses problemas estimulam a investigação na área e abre diversos pontos de pesquisa.

Outro fato motivador é que a RV possibilita a interação e o envolvimento imersivo de usuários com simulações, por intermédio de dispositivos de visualização 3D e dispositivos de rastreamento corporal. Assim, a junção das atividades colaborativas com aplicações de RV é um caminho natural da evolução tecnológica. Por isso, espera-se que diversas áreas do conhecimento possam se beneficiar de 3DCVEs. A necessidade de tais ambientes é evidente quando se efetua a análise das aplicações provenientes das comunidades científicas e industriais. Na área da RV, vários investimentos têm sido realizados pela indústria na produção de *hardware*, *software* e dispositivos de entrada/saída, motivando seu crescimento acelerado. Na área de atividades colaborativas, as grandes corporações têm realizado vários investimentos para que funcionários resolvam os problemas em grupo, mesmo em situações onde seus membros estejam localizados em locais diferentes. Juntas, essas áreas indicam perspectivas bastante promissoras.

Um dos grandes desafios de implementação de 3DCVE, sejam estes baseados em aglomerados gráficos remotos ou não, está relacionado à qualidade de serviço, em inglês *Quality of Service* (QoS). Mesmo com a evolução das redes de dados, ambientes que fazem uso de redes de dados baseadas em Internet sofrem com problemas já conhecidos, tais como atraso e perda de pacotes e variação no tempo de entrega de pacotes.

Esta tese investigou soluções que mantivessem QoS entre as conexões de ambientes distribuídos baseados em aglomerados gráficos. Portanto, diferentes áreas do conhecimento foram envolvidas durante seu desenvolvimento, tais como protocolos de rede, paralelismo, sistemas distribuídos, computação gráfica, interfaces, entre outras. O estudo dessas áreas é necessário para resolver problemas relacionados ao gerenciamento de consistências na troca de mensagens entre processos distribuídos.

3DCVEs baseados em aglomerados gráficos remotos são compostos por um conjunto de tecnologias, que inclui a infraestrutura de comunicação e a capacidade de geração e apresentação de imagens de maneira sincronizada entre diversos processos, espalhados por diversos nodos. Assim, como possível solução ao desenvolvimento de 3DCVE baseados em aglomerados gráficos remotos, foi desenvolvido um modelo de arquitetura de conexão dinâmica e adaptável, capaz de realizar verificações e adequações sobre conexões de maneira autônoma.

Os objetivos abordados nesta tese foram:

- Investigar soluções de conexão para 3DCVEs, focando em topologias de rede e protocolos utilizados na transferência de dados, de modo a permitir QoS sobre as conexões;
- Experimentar diferentes técnicas e propostas para identificar soluções que pudessem viabilizar a colaboração dentro dos requisitos esperados, em termos de desempenho;
- Desenvolver uma arquitetura de gerenciamento de conexão dinâmico-adaptável, composta por diferentes topologias, protocolos e algoritmos em geral;
- Avaliar a arquitetura desenvolvida por meio de estudos de caso realizados sobre diferentes redes de dados, tais como *Local Area Network* (LAN) (gigabit) e *Wide Area Network* (WAN) (Internet);
- Prover QoS, independente do tipo de rede utilizada, por meio de adequações nas conexões entre servidores de aglomerados gráficos remotos; e
- Criar um modelo comportamental para 3DCVEs, a fim de facilitar o desenvolvimento de tais aplicações.

1.3 Estrutura da tese

Este documento está estruturado da seguinte maneira:

- Capítulo 1: apresentou uma breve introdução sobre 3DCVEs baseados em aglomerados gráficos já existentes e seus desafios de implementação; foi abordada também a importância dos 3DCVEs nos dias atuais; e, por último, os objetivos e motivações para o desenvolvimento desta tese;
- Capítulo 2: apresenta o estado da arte dos 3DCVEs, focando nos problemas relacionados à concepção destes; a revisão bibliográfica realizada foi conduzida de forma sistemática (KITCHENHAM *et al.*, 2009);
- Capítulo 3: apresenta detalhes sobre o projeto e implementação de 3DCVEs baseados em aglomerados gráficos; são apresentadas duas soluções de ambientes virtuais imersivos: o CAVE e o miniCAVE; também são apresentadas algumas das principais ferramentas de desenvolvimento de 3DCVEs; e, por último, são discutidos conceitos relacionados à colaboração e distribuição de dados em 3DCVEs;

- Capítulo 4: apresenta a definição geral da arquitetura voltada a 3DCVEs baseados em aglomerados gráficos remotos. A infraestrutura utilizada também é apresentada;

- Capítulo 5: apresenta a descrição e desenvolvimento da arquitetura, sendo especificada na forma de módulos (*plugins*) de uma biblioteca de desenvolvimento. Os *plugins* contemplam os comportamentos de comunicação e sincronização intra–aglomerado e inter–aglomerados;

- Capítulo 6: a verificação da troca de mensagens entre os processos é algo de extrema importância, sendo possível identificar comportamentos inesperados e *deadlocks*, por exemplo. Desta forma, este capítulo apresenta a implementação de um modelo de análise de funcionalidades de mensagens trafegadas entre processos, abordando tanto a comunicação intra–aglomerado quanto a inter–aglomerados;

- Capítulo 7: apresenta os estudos de casos desenvolvidos com o intuito de avaliar a arquitetura; e

- Capítulo 8: as discussões relacionadas à tese são abordadas neste capítulo, assim como as limitações e possíveis pesquisas futuras; também são elencadas as publicações realizadas pelo candidato durante o período do doutorado.

Capítulo 2

O Estado da arte de 3DCVE

O tipo de 3DCVE tratado no contexto desta tese é caracterizado como sendo ambientes de simulação em 3D, imersivos e interativos, que são compostos por aglomerados de computadores remotos. Eles permitem que múltiplos usuários interajam entre si em tempo real, mesmo estando localizados em lugares diferentes, também possuindo características de compartilhamento de espaço, presença e tempo, além de prover um meio para que os usuários possam se comunicar. Segundo Singhal e Zyda (1999), os 3DCVEs são compostos por quatro componentes básicos: motores gráficos e exibição, dispositivos de comunicação e controle, sistema de processamento e tráfego de dados.

O desenvolvimento de 3DCVEs requer conhecimento em áreas computacionais diversificadas, tais como redes, sistemas paralelos e distribuídos, computação gráfica, sistemas multi *threads* e interface de usuário. Vários problemas devem ser resolvidos com relação à sua concepção, como, por exemplo, o gerenciamento consistente de informação distribuída, a garantia de interação em tempo real e a adequação das aplicações a bandas de rede limitadas. Os 3DCVEs têm sido foco de pesquisa por mais de 20 anos, sendo destaque na década de 90 devido à evolução das redes.

Nos últimos anos, pesquisas têm sido realizadas com o intuito de minimizar as características adversas ocasionadas por diferentes redes de dados, principalmente via Internet. Este capítulo apresenta uma revisão sistemática – referente aos 12 últimos anos – sobre 3DCVEs e seus desafios de implementação. Os estudos utilizados nesta seção, sumarizam alguns problemas e possíveis soluções com relação a atraso na entrega de pacotes, variação de tempo na entrega de pacotes, perda de pacotes, entre outros.

Este capítulo apresenta apenas a discussão sobre os resultados obtidos durante a revisão. O Apêndice A apresenta, em detalhes, as etapas realizadas.

2.1 Tipos de 3DCVE

Nos estudos primários [1] revisados, foram encontradas diferentes definições para ambientes virtuais que utilizam redes de dados como meio de intercomunicação/colaboração entre locais distintos. Os termos encontrados foram: *Collaborative Virtual Environment* (CVE), *Networked Virtual Environment* (NVE), *Distributed Virtual Environment* (DVE), *Shared Virtual Environment* (SVE), *Networked Haptic Virtual Environment* (NHVE), *Massively Multi-user Virtual Environment* (MMVE), *Teleoperation*, *Virtual Conference* and *Teleimmersion*. A seguir, são apresentadas breves descrições de cada termo encontrado.

- CVE: são ambientes que incorporam múltiplos usuários, provendo meios de interação de objetos em colaboração. Algumas aplicações utilizam este tipo de ambiente para prover interação entre grupos de pessoas em diferentes locais (CORRÊA *et al.*, 2010; ISHIDA *et al.*, 2009; WANG *et al.*, 2011; STEINER; BIERSACK, 2006; SUNG *et al.*, 2006). A interação, normalmente, é efetuada por um usuário de cada vez, sendo que, quando outro usuário desejar interagir com o ambiente, este deve efetuar uma solicitação (CORRÊA *et al.*, 2010). Isto ocorre porquê os usuários compartilham um mesmo ponto de vista, isto é, a mesma câmera;

- NVE: são ambientes distribuídos por topologias ponto-a-ponto utilizados por vários tipos de aplicações, tais como jogos multijogadores, treinamento militar e simulações em geral. É caracterizado por possibilitar que múltiplos usuários interajam entre si em tempo real, mesmo estando localizados em lugares distintos (WANG *et al.*, 2011; BALADI *et al.*, 2008; YUAN; LU, 2004);

- DVE: são ambientes virtuais distribuídos sobre a WAN. Estes ambientes são compostos por vários servidores, cada um responsável pelo gerenciamento de múltiplos clientes que desejam participar de um ambiente virtual. Segundo Ali *et al.* (2009), existem dois tipos de arquiteturas, sendo elas compostas por um servidor ou múltiplos servidores;

- SVE: são ambientes virtuais que propiciam a múltiplos usuários a sensação de estarem presentes no mesmo ambiente e interagindo entre si (SCHROEDER, 2006);

- NHVE: You e Sung (2008) afirmam que este tipo de ambiente possui interação entre usuários sob a rede, mas com foco em dispositivos hápticos, que geram posições espaciais em 3 eixos (x,y,z). O *feedback* do usuário também é importante neste tipo de ambiente;

[1] Termo utilizado para se referir aos artigos revisados. Esta nomenclatura é utilizada por Kitchenham *et al.* (2009)

- MMVE: Hariri *et al.* (2008) definem este tipo de ambiente como um tipo de sistema que incorpora computação gráfica, som e meios hápticos em simulações de ambientes virtuais entre múltiplos usuários. Tais aplicações lidam com distribuições de atualizações entre os usuários. Utiliza topologia ponto-a-ponto;

- *Teleoperation*: Kadavasal e Oliver (2008) definem este tipo de ambiente como sendo uma combinação entre ambiente virtual, sensores e processamento de imagens em dispositivos operados remotamente. O uso de ambientes virtuais, ao invés de imagens reais (vídeos), se dá devido ao atraso na geração de imagens reais e o campo de visão limitado;

- *Virtual Conference*: são ambientes que conectam vários usuários em um ambiente virtual, de modo que os usuários possam se comunicar e interagir entre si. A interação, normalmente é feita por meio de avatares; o uso de áudio e vídeo também é aplicado (LORENZO *et al.*, 2012; RHEE *et al.*, 2007);

- *Teleimmersion*: integra soluções de RV e Realidade Aumentada (RA) distribuída com imagens geradas por câmeras. Normalmente, utilizam redes de alta velocidade, devido a alta banda de rede requerida para a realização de *streams* de vídeo (ISHIDA *et al.*, 2009).

Foi adotado o termo 3DCVE nesta tese, visto que seria inviável utilizar todos os termos elencados acima na redação desta. Assim sendo, o termo 3DCVE é aqui caracterizado como sendo um ambiente virtual colaborativo em 3D, que são compostos por aglomerados de computadores remotos e são providos da imersão e interação.

2.2 Relatando a revisão sistemática – problemas e soluções na implementação de 3DCVE

O foco desta seção é a apresentação do estado da arte em relação as pesquisas realizadas nos últimos 12 anos (2003 à 2015) referentes à concepção de 3DCVEs, em especial, abordando o uso de protocolos e topologias de rede. Além disso, as informações coletadas dos estudos primários foram utilizadas para responder as seguintes questões de pesquisa:

QP$_1$ Quais são os protocolos de comunicação utilizados na intercomunicação entre 3DCVEs?

QP$_2$ Quais são as topologias de rede utilizadas na intercomunicação entre 3DCVEs?

QP$_3$ Quais são os desafios de implementação de 3DCVEs?

QP$_4$ Quais técnicas de avaliação (estratégias empíricas) tem sido empregadas na avaliação dos estudos encontrados? Por exemplo, estudos de caso, experimentos entre outros.

A maioria dos estudos primários selecionados foram publicados no portal da IEEE (48 estudos). Os outros estudos primários foram selecionados nas bases ACM, Scopus, Springer e Science Direct, com quantidades de 34, 23, 19 e 8, respectivamente. Foram selecionados estudos primários publicados em conferências, livros, *workshops* e *journals*. A maioria dos estudos são de conferências (75), seguidos de journals (30), livros (15) e *workshops* (12).

Os resultados apresentados nos estudos primários podem ser classificados como experimentos, estudos de caso ou não especificado. Para que as questões de pesquisa (QP) fossem respondidas, os 132 estudos primários foram analisados individualmente.

Com relação aos problemas de implementação de 3DCVEs, estes estão relacionados à variação de tempo na entrega de pacotes, perda de pacotes e alta latência. Esses problemas foram encontrados em ambientes que utilizam a Internet, com 25 estudos apresentando soluções para tal infraestrutura. Em sua maioria, os estudos que abrangem problemas relacionados ao uso de Internet, também apresentam estudos relacionados a redes de dados locais, porém, as redes de dados locais não são afetadas pelos mesmos problemas encontrados nas que utilizam a Internet. Dentre as dificuldades encontradas no desenvolvimento de 3DCVEs, têm-se:

- Latência: é o tempo que um pacote leva para ser transmitido entre a origem e o destino, sendo caracterizada pela combinação dos atrasos de transmissão, propagação e processamento. Dentre os estudos primários, alguns autores (YOU *et al.*, 2007; CHEN, 2005; LI *et al.*, 2006; SUNG *et al.*, 2006; TUMANOV *et al.*, 2007) afirmam que a alta latência tem influência direta na interação de 3DCVEs, entretanto, apresentam diferentes afirmações com relação ao valor máximo aceitável. Altas latências podem ocasionar problemas que vão além da interferência da interação do usuário com 3DCVEs. Allison *et al.* (2004) afirmam que a alta latência pode ocasionar oscilopsia, causando a sensação de que o ambiente virtual esteja "boiando"no espaço. Com relação à atividade colaborativa, é mencionado em Tumanov *et al.* (2007) um experimento para avaliar as dificuldades que altas latências podem ocasionar em ambientes colaborativos. O experimento apresenta um ambiente onde um usuário controla um bastão virtual, enquanto outro usuário controla um anel, de modo que os dois usuários devem passar o anel de um lado para o outro pelo bastão, porém, com o mínimo de colisões entre os objetos. Neste estudo, foi concluído que os usuários se adequam aos atrasos do ambiente para que seus movimentos sejam sincronizados;

- Variação de tempo na entrega de pacotes: é uma variação estatística na diferença de tempo entre a entrega dos pacotes sucessivos em uma comunicação via rede. Alguns estudos (YOU *et al.*, 2007; CHEN, 2005; YOU *et al.*, 2007; YOU; SUNG, 2008; LING *et al.*, 2003; SUNG *et al.*, 2006) apresentam a variação no tempo de entrega como uma das principais causas de problemas na implementação de 3DCVEs sobre à Internet. De acordo com Ling *et al.* (2003), latências de até 200ms não são prejudiciais. Todavia, se for constatada a presença de variações, até mesmo latências de 20ms podem se tornar prejudiciais. Experimentos apresentados no estudo de Ling *et al.* (2003) comprovam que mensagens de atualização de estado são severamente afetadas pela variação de tempo na entrega de pacotes, de modo que, com uma latência de 10ms e a presença de variação, os resultados foram piores que em um ambiente com 200ms de latência sem a presença de variação.

- Perda de pacotes: é caracterizada por pacotes que são enviados pelo nodo origem, mas não são entregues ao nodo destino. Um dos motivos para que a perda de pacotes ocorra está relacionado à fila cheia, isto é, o pacote enviado chega ao destinatário, porém, encontra uma fila cheia, sendo assim descartado pelo roteador. Quanto maior o tráfego de rede, mais pacotes tendem a ser perdidos (KUROSE; ROSS, 2012). De acordo com alguns autores (STEINBACH *et al.*, 2011; YOU *et al.*, 2007; CHEN, 2005; YOU *et al.*, 2007; YOU; SUNG, 2008; LING *et al.*, 2003; YUAN; LU, 2004; WANG *et al.*, 2011), a perda de pacotes pode ser contornada com técnicas de predição de dados. Boukerche *et al.* (2006) apresentam um modelo de predição de pacotes, que efetua um cálculo sobre valores de atualização de estado do ambiente virtual. Por exemplo, se a interação de algum usuário em um 3DCVE gerar três atualizações de coordenadas (9900, 300 e 1200), (9800, 300 e 1220) e (9700, 300 e 1250), mas somente o primeiro e último estados forem recebidos, o módulo de predição deverá informar o valor aproximado do estado intermediário como sendo (9800, 300 e 1225). Chen (2005) afirma que a perda de pacotes não gera perda de qualidade em 3DCVEs que possuam, no mínimo, 40 atualizações de estado por segundo. Se a quantia de estados for de 15 a 40 por segundo, a perda de qualidade na atividade colaborativa, ou até mesmo na visualização do 3DCVE, é perceptiva. No entanto, a quantidade de estados enviados por segundo está diretamente relacionada ao poder de vazão da rede.

O uso de protocolos específicos pode amenizar esses problemas. Foram encontrados 9 protocolos de rede utilizados nos 132 estudos primários revisados:

- *Transmission Control Protocol* (TCP): é um protocolo de transporte confiável da camada

de transporte e orientado a conexão; possui características importantes com relação à transferência confiável de dados, tais como detecção de erro, retransmissões, reconhecimentos cumulativo, temporizadores e campos de cabeçalho para números de sequência e reconhecimento (KUROSE; ROSS, 2012);

- *User Datagram Protocol* (UDP): é um protocolo de transporte simplificado, se comparado ao TCP; não provê transporte confiável e nem é orientado a conexão. O UDP é indicado para aplicações de tempo real que possam tolerar uma certa quantia de perda de pacotes, mas evitando controle de congestionamento e os cabeçalhos complexos do protocolo TCP (KUROSE; ROSS, 2012);

- *Stream Control Transmission Protocol* (SCTP): é um protocolo de transporte confiável e orientado a conexão; realiza transferências de datagramas livre de erros e duplicações; utiliza mecanismo de *checksum* e números sequênciais para detectar corrupção, perda e duplicação de pacotes (SCTP, 2016);

- *Real-time Transport Protocol/Real Time Control Protocol* (RTP/RTCP): é um protocolo utilizado para a transmissão de pacotes de áudio e vídeo sobre redes IP. É aplicado, normalmente, a sistemas que utilizam *streams*, tais como telefonia IP e vídeo conferências. O RTP transmite os dados, já o RTCP transmite estatísticas de monitoramento e QoS (KUROSE; ROSS, 2012);

- *Real Time Events Protocol* (RTEP): é um protocolo que estende o RTP/RTCP, incorporando métodos para codificar e decodificar pacotes por meio dos mecanismos *SmartBeat, EvPack e MessageRouting*. A codificação consiste em serializar a informação relacionada a eventos e gerar os pacotes de dados (CORRÊA *et al.*, 2010);

- *Pragmatic General Multicast* (PGM): é um protocolo de transporte destinados a redes *multicast*. Ele provê a entrega de sequências de pacotes a múltiplos nodos simultaneamente, sendo indicado a aplicações de transferência de arquivos para múltiplos nodos. Em 3DCVEs é indicado a aplicações que fazem uso de aglomerados interconectados. Diferentemente do TCP, utiliza um padrão *Negative Acknowledgements (NACK)*, que envia uma mensagem de volta à origem quando a perda de pacotes é detectada (SPEAKMAN *et al.*, 2013); e

- *AppTraNet*: é um protocolo que combina três camadas: aplicação, transporte e rede. Utiliza um cabeçalho que pode ser processado em paralelo (RONNINGEN *et al.*, 2010).

Uma importante característica que deve ser definida no momento da implementação de um 3DCVE é a topologia de rede. Foram elencadas quatro topologias utilizadas nos estudos primários. As topologias físicas mais utilizadas foram: malha, estrela, ponto–a–ponto e híbridas. Com relação à topologias lógicas, foram encontradas intra–domínio, inter–domínio, *metadata overlay*, *overlay*. A topologia física mais utilizada foi a ponto–a–ponto, sendo mencionada em 9 estudos primários. As topologias são descritas da seguinte maneira:

- Ponto–a–ponto: é uma topologia onde cada nodo da rede pode realizar comunicação como sendo cliente e servidor, permitindo o compartilhamento de dados entre os nodos sem a necessidade de um servidor central. Este tipo de topologia é altamente difundido na distribuição de músicas, vídeos e imagens entre múltiplos usuários (COULOURIS *et al.*, 2011);

- Estrela: é uma topologia caracterizada por possuir um elemento central voltado ao gerenciamento do fluxo de dados da rede, ou seja, um servidor central ligado diretamente a cada nodo da rede. Para que as informações possam ser enviadas de um nodo a outro, elas devem ser enviadas a esse servidor central. Uma das vantagens em se utilizar esse tipo de topologia está relacionada a facilidade na administração, porém, se houver algum problema no servidor central, toda a estrutura é afetada (KUROSE; ROSS, 2012);

- Malha: neste tipo de topologia todos os nodos trocam dados entre si por meio de ligações diretas entre eles, isto é, sem a necessidade de um servidor central. Topologias em malha podem ser conectadas de duas formas: totalmente e parcialmente conectadas; e

- Híbrida: neste tipo de topologia são combinadas outros tipos de topologias. A topologia *Distributed Multimedia Plays* (DMP) (RONNINGEN *et al.*, 2010), por exemplo, combina características de duas topologias: estrela e malha.

A topologia ponto–a–ponto foi a mais utilizada nos últimos anos na concepção de 3DCVEs, visto que permite total distribuição, baixa latência e gerenciamento eficiente de mensagens, se implementada com alguns cuidados. Wang *et al.* (2011) utilizaram a topologia ponto-a-ponto na proposta de uma arquitetura voltada a 3DCVEs. Eles exploraram a localidade do usuário, de modo que o nodo do usuário só recebia informações dos nodos que estavam próximos a ele no ambiente. Steiner e Biersack (2006) utilizaram a topologia ponto–a–ponto. O algoritmo é dinâmico com relação a conexão de nodos em 3DCVEs. A ideia básica do algoritmo é classificar os *links* entre os nodos como sendo intra–aglomerado ou inter–aglomerados. A Figura 2.1 apresenta a frequência de uso de cada uma das topologias físicas encontradas, respondendo a QP2.

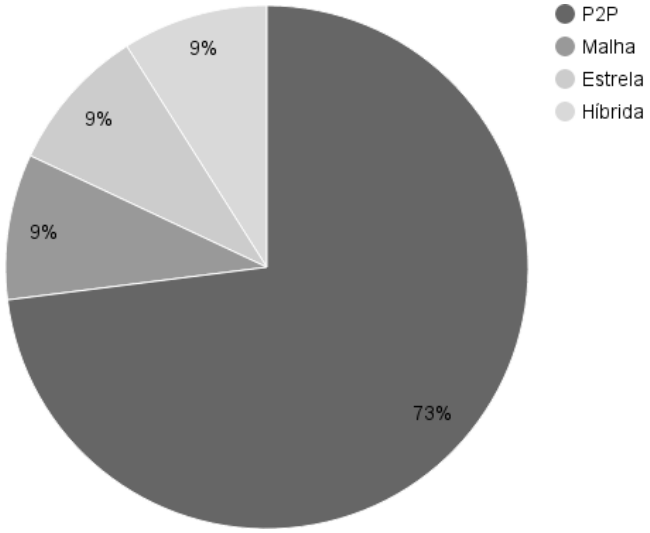


Figura 2.1: Topologias físicas utilizadas nos estudos primários.

Alguns dos estudos primários também apresentam detalhes sobre os tipos de topologias lógicas utilizadas. No entanto, da mesma forma que as topologias físicas, elas podem ser classificadas de acordo com a aplicação e experimentos realizados. As topologias lógicas classificadas foram:

- Intra–domínio: permite a troca de informações entre nodos que estejam em uma mesma subrede;
- Inter–domínio: permite a troca de informações entre nodos que estejam conectados por meio de diferentes subredes;
- *Overlay*: permite a troca de informações utilizando sobreposição, de modo que os nodos são caracterizados como sendo processos, e os enlaces representam os meios de comunicação possíveis (KUROSE; ROSS, 2012); e
- *Metadata Overlay Topology*: permite a troca de informações utilizando sobreposição, mas também considerando o tempo de ida e volta (*Round Trip Time* (RTT)) entre os nodos. Um grafo completo com os custos de cada enlace é calculado e difundindo para todos os nodos (NAHRSTEDT *et al.*, 2010).

Com relação as infraestruturas de rede utilizadas, foram encontrados estudos utilizando Internet e redes gigabit (locais e regionais). No geral, em se tratando de aplicações executadas em redes gigabit, não foram identificados os problemas mencionados na subseção 2.2, pois são redes de alta velocidade confiáveis. Neste tipo de rede, o protocolo utilizado com maior

frequência foi o TCP (BALADI *et al.*, 2008; YUAN; LU, 2004; DROLET *et al.*, 2009; TUMANOV *et al.*, 2007; STEINBACH *et al.*, 2011; LING *et al.*, 2003; SUNG *et al.*, 2006; YOU *et al.*, 2007; LI *et al.*, 2006).

Os estudos primários, em sua maioria, apresentam o uso de simuladores de rede para criar características adversas na realização dos experimentos, tal como o NetDisturb (CHEN, 2005) e o NISNet (YOU *et al.*, 2007).

Dentre os estudos primários, 17 apresentaram algum tipo de experimento relacionados a latência e variação de tempo na entrega de pacotes (MORILLO *et al.*, 2008; NAHRSTEDT *et al.*, 2010; BALADI *et al.*, 2008; TUMANOV *et al.*, 2007; CHEN, 2005; LI *et al.*, 2006; ROBERTS *et al.*, 2003), perda de pacotes (NAHRSTEDT *et al.*, 2010; WANG *et al.*, 2011; CHEN, 2005; YOU *et al.*, 2007; BOUKERCHE *et al.*, 2006) e comparações entre diferentes protocolos (YUAN; LU, 2004; SUNG *et al.*, 2006; CORRÊA *et al.*, 2010). Os estudos que realizam estudos de caso, apresentam exemplos de uso de 3DCVEs, tais como medicina (UBIK *et al.*, 2012; AI *et al.*, 2008; KENYON; LEIGH, 2004; WATANABE *et al.*, 2007; KADAVASAL; OLIVER, 2008), engenharias (OPIYO; HORVÁTH, 2010; DONG *et al.*, 2010; BRUNS *et al.*, 2007), educação (SHAHAB *et al.*, 2006; LIANG, 2012; GUO *et al.*, 2009) etc. Também foram encontrados estudos que não apresentam nenhum tipo de resultado real, os quais apresentam futuras tendências e conceitos de 3DCVEs (DEFANTI *et al.*, 2009; APOSTOLOPOULOS *et al.*, 2012; SCHROEDER, 2006). Este parágrafo responde a QP4.

A frequência de publicações que utilizam diferentes protocolos de comunicação é apresentada em um *bubble plot* na Figura 2.2. O *bubble plot* é composto por dois eixos(X, Y), que possui bolhas que representam categorias. O tamanho da bolha determina a quantidade de estudos primários que foram classificados como pertencente à categoria (protocolos) correspondente da coordenada da bolha. Este resumo visual fornece uma visão panorâmica que possibilita identificar quais categorias foram enfatizadas em pesquisas recentes, juntamente com lacunas e oportunidades para futuras pesquisas. Os dois eixos apresentados na Figura 2.2 representam o ano de publicação e os protocolos, evidenciando quais foram os protocolos mais utilizados na implementação de 3DCVEs nos últimos anos, o que responde a QP1. Os protocolos encontrados nos estudos primários foram: TCP, UDP, AppTraNet, SCTP, RTP/RTCP, RTEP, PGM, *InterStream Transit Protocol* (ISTP) e *Internet Control Message Protocol* (ICMP).

O protocolo mais utilizado foi o UDP, sendo encontrado em 8 estudos primários. Uma das principais razões para essa alta frequência, está relacionada a dois fatores: em redes de acesso confiáveis, o UDP pode ser utilizado sem que a comunicação seja prejudicada, pois esses ambientes são isentos ou possuem perda de pacotes não considerável; ou em redes de acesso

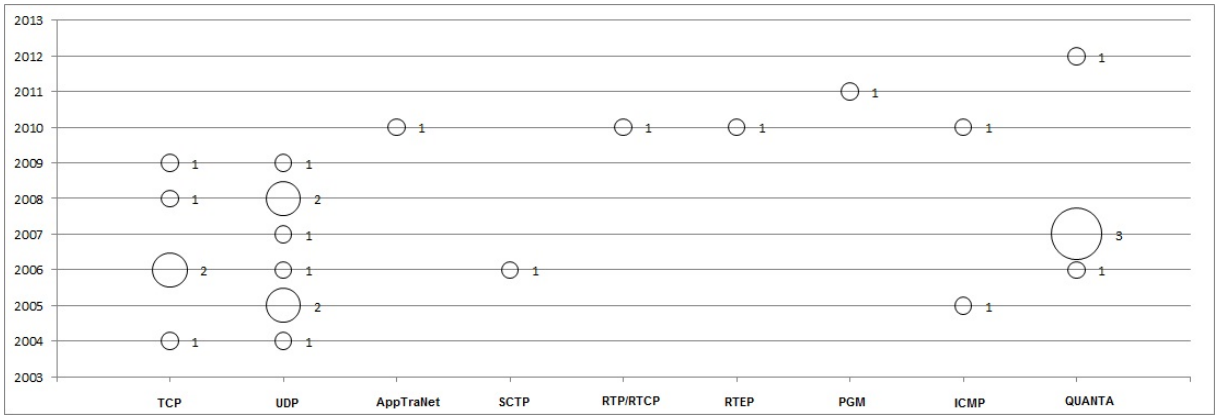


Figura 2.2: Frequência do uso dos protocolos por ano.

baseadas em Internet, por ser baseado em datagramas e não realizar verificação de entrega, aumentando a vazão de informação de 3DCVEs, deixando a cargo da aplicação a verificação de entrega de pacotes e também a ordem de chegada.

O segundo protocolo mais utilizado foi o TCP, sendo mencionado na implementação de 3DCVEs sobre redes de acesso de alta velocidade. O uso do protocolo TCP sobre redes de acesso baseadas em Internet pode acarretar em problemas de vazão, pois o tamanho do cabeçalho e o *Acknowledgement* (ACK) são responsáveis por diminuir sua capacidade (SUNG *et al.*, 2006). Para este tipo de ambiente, como mencionado, o protocolo UDP é mais indicado, por ser orientado a datagramas. Alguns estudos apresentam comparações entre os protocolos UDP e TCP (SUNG *et al.*, 2006; YUAN; LU, 2004), sendo o UDP o protocolo com melhores resultados (considerando-se apenas a vazão de dados). No entanto, outros protocolos também são indicados na comunicação de 3DCVEs: o SCTP e o RTP/RTCP; visto que estes possuem características combinadas do TCP e UDP, sendo otimizados para aplicações multimídia (*stream* e tempo real). De acordo com Boukerche *et al.* (2006), o protocolo SCTP provê sistema de entrega confiável para pacotes chave e entrega não confiável para pacotes de atualização; já o *Smoothed* SCTP, agrega estratégias para lidar com a variação de tempo na entrega de pacotes por meio da implementação de um pequeno *buffer* no lado do receptor.

Alguns protocolos foram encontrados em apenas um estudo primário. Estes podem ser considerados "desertos de evidência"(onde deve ser realizada uma nova ou aprofundada pesquisa). Os protocolos que obtiveram apenas uma ocorrência foram: AppTraNet, RTP/RTCP, RTEP, PGM e o ISTP. O RTP/RTCP foi utilizado como parâmetro para a criação do protocolo RTEP (CORRÊA *et al.*, 2010); foram realizados experimentos com o RTEP e o ICMP. O RTP/RTCP, apesar de ser utilizado em um estudo recente (CORRÊA *et al.*, 2010), possui apenas uma ocorrência, porém, é citado em livros textos e alguns estudos como tendência (KUROSE; ROSS, 2012; NAHRSTEDT *et al.*, 2010; RONNINGEN *et al.*, 2010). O PGM foi utilizado em con-

junto com o QUANTA *framework* (QUANTA, 2016) para prover o módulo de rede do iTILE (CHO *et al.*, 2012). O ISTP foi utilizado no encapsulamento de pacotes trocados entre os nodos de um 3DCVE, sendo que, se o tipo da mensagem fosse um estado chave, este era transportado de forma confiável (ACK e NACK) (LING *et al.*, 2003). O protocolo ICMP possui duas ocorrências, mas em ambos os estudos, foi utilizado como base de comparação para outros protocolos (CORRÊA *et al.*, 2010; ANTHES *et al.*, 2005).

O arcabouço QUANTA também aparece em alguns estudos (CHO *et al.*, 2012; STEINBACH *et al.*, 2011; SUNG *et al.*, 2006; YOU *et al.*, 2007; YOU *et al.*, 2007). O QUANTA é um *toolkit* de rede financiado pela *National Science Foundation* (NSF), que provê um modo fácil de transferência de dados entre sistemas distribuídos. A transferência é feita em alto nível de aplicação, sendo invísivel ao desenvolvedor. Nesta revisão, o QUANTA é apresentado no *bubble plot* por ser utilizado como opção ao uso de protocolos de rede, mas ele não deve ser tratado como um protocolo, e sim, um conjunto de soluções.

Foram mencionados outros protocolos que poderiam ter sido utilizados no desenvolvimento de 3DCVEs, entretanto, esta revisão abrange um período de 12 anos, de forma que tais protocolos não foram encontrados em nenhum dos estudos primários revisados. Dentre os protocolos que foram citados nos estudos primários, mas não foram utilizados, têm-se: SIP (RONNINGEN *et al.*, 2010; NAHRSTEDT *et al.*, 2010), SDP (RONNINGEN *et al.*, 2010; NAHRSTEDT *et al.*, 2010), RTSP (NAHRSTEDT *et al.*, 2010), VRTP (ANTHES *et al.*, 2005) e DWTP (ANTHES *et al.*, 2005).

Dentre os estudos primários revisados, grande parte, utilizaram redes de acesso com características de alta velocidade e confiabilidade, ou seja, redes locais ou regionais interligadas por meio de fibra óptica. O foco principal desta revisão foi encontrar possíveis soluções de desenvolvimento de 3DCVEs voltados a redes de acesso baseadas em Internet, de modo a apresentar os problemas e soluções abordados.

Os estudos primários avaliados respondem a QP3. Alguns estudos primários abordam problemas relacionados à perda de pacotes (STEINBACH *et al.*, 2011; YOU *et al.*, 2007; CHEN, 2005; YOU *et al.*, 2007; YOU; SUNG, 2008; LING *et al.*, 2003; YUAN; LU, 2004; WANG *et al.*, 2011). No entanto, nenhum deles apresenta experimentos em um ambiente real, apresentando resultados simulados; a perda de pacotes não é o problema mais prejudicial na implementação de 3DCVEs, podendo ser mitigada por meio do uso de técnicas não elaboradas de predição de dados (BOUKERCHE *et al.*, 2006).

Como já mencionado com relação à latência de rede, pesquisadores afirmam que mais de 200ms podem influenciar na experiência em 3DCVEs (CHEN, 2005). No entanto, pode-se

encontrar latências de até 600 ms em conexões intercontinentais sobre à Internet, assim, este é um dos principais desafios de implementação. Para tal, pode-se dispensar tempo de pesquisa no desenvolvimento de novos protocolos ou otimizar os já existentes. Porém, dentre os desafios, a variação de tempo na entrega de pacotes é o mais prejudicial, pois em 3DCVEs é importante que a taxa de entrega de pacotes seja mantida em uma velocidade constante. Alguns pesquisadores têm resolvido problemas relacionados a variação com técnicas de bufferização (YOU *et al.*, 2007; ANTHES *et al.*, 2005; BOUKERCHE *et al.*, 2006).

Os problemas identificados na Seção 2.2 são preocupações que devem ser tratadas quando redes de acesso baseadas em Internet são utilizadas. Em redes locais gigabit ou redes de alta velocidade, não existem tais problemas, inclusive, existem autores, tal como DeFanti *et al.* (2009), que afirmam que o gargalo atribuído as redes de dados nos dias de hoje não será problema em um futuro próximo, passando a ser ocasionado pelos barramentos, interfaces de rede e protocolos.

2.3 Considerações finais

Este capítulo apresentou os problemas e soluções encontrados durante os últimos 12 anos de pesquisa, buscando evidenciar os pontos relevantes a serem pesquisados. Os protocolos e topologias mais utilizados foram encontrados, assim como algoritmos de bufferização e predição de dados.

O protocolo mais utilizado foi o UDP devido ao tipo de rede utilizado na maioria dos estudos primários, redes locais de alta velocidade. Os protocolos TCP e SCTP também foram destacados. O SCTP, apesar de não ser mencionado, é apontado como tendência.

A topologia física mais utilizada foi a ponto–a–ponto, por permitir total distribuição, baixa latência e eficiente gerenciamento de mensagens de troca entre pares (*peers*).

Após a revisão, podê-se notar que os estudos avaliados apresentam, no geral, apenas avaliações e propostas isoladas dos problemas encontrados no desenvolvimento de 3DCVEs.

Capítulo 3

3DCVEs BASEADOS EM AGLOMERADOS GRÁFICOS

Este capítulo apresenta os componentes que compõem um 3DCVE. São apresentados sistemas de multiprojeção, que provêem características imersivas e interativas a ambientes virtuais. É apresentado também o aglomerado gráfico, a solução de renderização utilizada nesta tese. Ainda são apresentadas algumas das principais bibliotecas de desenvolvimento de aplicações de RV distribuídas, destacando a libGlass (LIBGLASS, 2016), biblioteca que foi utilizada como base para a implementação da arquitetura.

3.1 Ambientes de multiprojeção

Os sistemas de multiprojeção, geralmente compostos por múltiplas telas, proporcionam ao usuário diferentes pontos de vista de um mesmo ambiente, possibilitando uma experiência altamente imersiva. Os sistemas de multiprojeção têm sido pesquisados a mais de uma década como solução de renderização para ambientes virtuais complexos (CRUZ-NEIRA *et al.*, 1992; GNECCO *et al.*, 2003; DEFANTI *et al.*, 2009; DROLET *et al.*, 2009).

Entre os diversos sistemas de multiprojeção existentes, tem-se o CAVE, que se implementado utilizando aglomerados gráficos, pode ser implantado com baixo custo, dependendo dos dispositivos empregados. Os aglomerados gráficos são caracterizados por um conjunto de nodos interligados por uma rede de dados, dando ao usuário a impressão de que o processamento é efetuado por um único sistema (GNECCO *et al.*, 2003). O objetivo é oferecer uma visão múltipla de um mesmo conjunto de dados, sendo que cada nodo processa somente as tarefas associadas a ele, como a geração da imagem de um determinado ponto de vista e/ou recebimento das interações dos usuários.

O primeiro CAVE foi desenvolvido pelo *Eletronic Visualization Laboratory* (EVL) na Universidade de Illinois, Chicago, sendo apresentado no SIGGRAPH'92 (CRUZ-NEIRA *et al.*, 1992). Outros ambientes semelhantes foram desenvolvidos ao redor do mundo. No Brasil, o Laboratório de Sistemas Integráveis da Universidade de São Paulo, desenvolveu a Caverna Digital (SOARES, 2005), que segue o modelo de infraestrutura de multiprojeção proposto por Cruz-Neira *et al.* (1992).

Cruz-Neira *et al.* (1992) implementaram o primeiro CAVE utilizando projetores *Cathodic Ray Tube* (CRT) com resolução 1280x1024 e paredes de 3mx3m, capaz de gerar projeções estéreo ativo. A estrutura do sistema foi proposta com 3 paredes *rear-project* e uma projeção no chão por meio de *down-project*. Tal estrutura foi proposta para gerar um novo senso de imersão ao usuário de ambientes virtuais. As projeções eram geradas por estações *Silicon Graphics* (SGI) Crimson, porém, as aplicações atingiam apenas 8 quadros por segundo, não sendo suficiente para uma animação. Posteriormente, as estações SGI foram substituídas por sistemas SGI Onyx.

Na segunda geração do CAVE, proposta em 2001, o EVL focou suas pesquisas em melhorar a qualidade das projeções utilizando projetores Christie Mirage *Digital Light Processing* (DLP), que possuíam ganho em brilho, porém, sendo mais caros que os da geração anterior. A taxa de quadros também foi aprimorada, passando a gerar 25 quadros por segundo (DEFANTI *et al.*, 2009). A Figura 3.1 apresenta as duas versões do CAVE, a primeira com projetores CRT, e a segunda, com projetores Mirage.

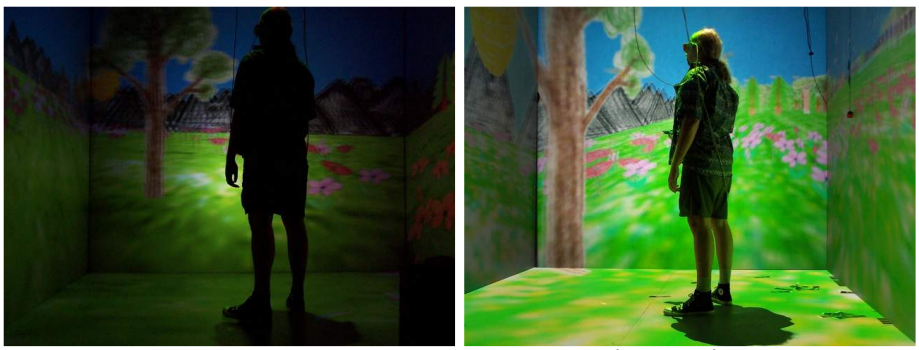


Figura 3.1: Projetores da primeira e segunda geração do CAVE™ - Figura adaptada de (PAPE, 2013).

Foi proposta ainda uma terceira geração do CAVE, que foi denominada StarCAVE (DEFANTI *et al.*, 2009). Esta geração utiliza projetores *Liquid Crystal Display* (LCD), que provêem alta resolução de imagem. As paredes quadradas, anteriormente propostas nas duas outras gerações, foram alteradas para uma sala em forma de pentágono. A Figura 3.2 apresenta a estrutura

do StarCAVE.

Figura 3.2: StarCAVE.
(DEFANTI *et al.*, 2009)

Com o intuito de prover um ambiente de multiprojeção de baixo custo, Dias *et al.* (2010b) propuseram um ambiente inspirado no CAVE, contudo, possuíndo uma angulação diferente, objetivando a imersão de um número maior de usuários, diferente do CAVE original, onde poucos usuários podem utilizá-lo ao mesmo tempo. Este ambiente foi denominado miniCAVE.

O ambiente é composto por um conjunto de 3 telas com projeção frontal. Para a geração de imagens são utilizados projetores convencionais. A polarização das imagens é feita por meio da utilização de lentes polarizadoras. A Figura 3.3 apresenta a infraestrutura do miniCAVE implantado no Laboratório de Visualização, Imersiva, Interativa e Colaborativa (LaVIIC) – UFSCar, composto por um aglomerado gráfico de quatro computadores pessoais: um mestre e três escravos.

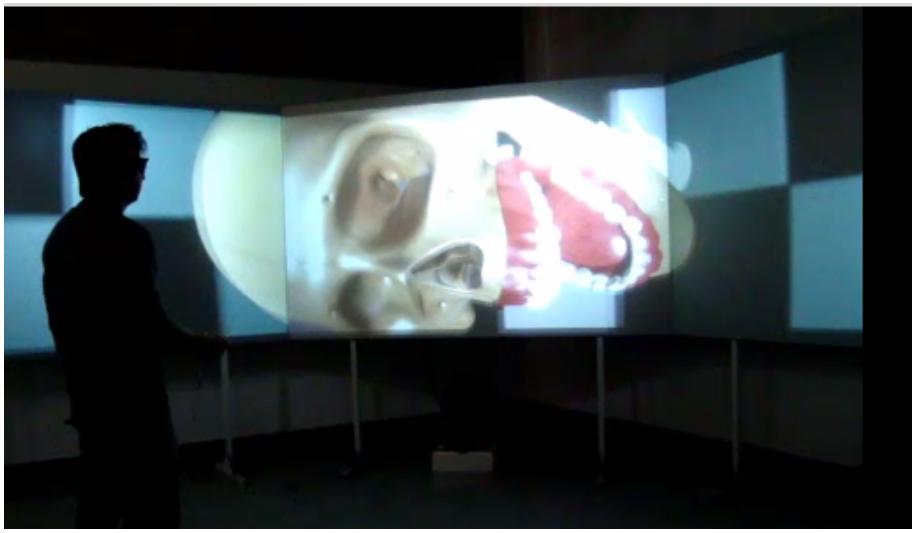


Figura 3.3: miniCAVE LaVIIC – UFSCar.
(DIAS, 2011)

A Figura 3.4 apresenta o miniCAVE implantado no Laboratório de Interfaces e Visualização (LIV) – UNESP que, apesar de possuir um aglomerado de seis computadores, possui arquitetura de *software* semelhante a utilizada no LaVIIC, diferenciando apenas na quantidade de instâncias por nodo. O aglomerado gráfico do LSTR permite alcançar uma maior taxa de quadros por segundo em relação ao aglomerado gráfico do LaVIIC, devido a possibilidade de manter apenas um processo referente a renderização por computador.

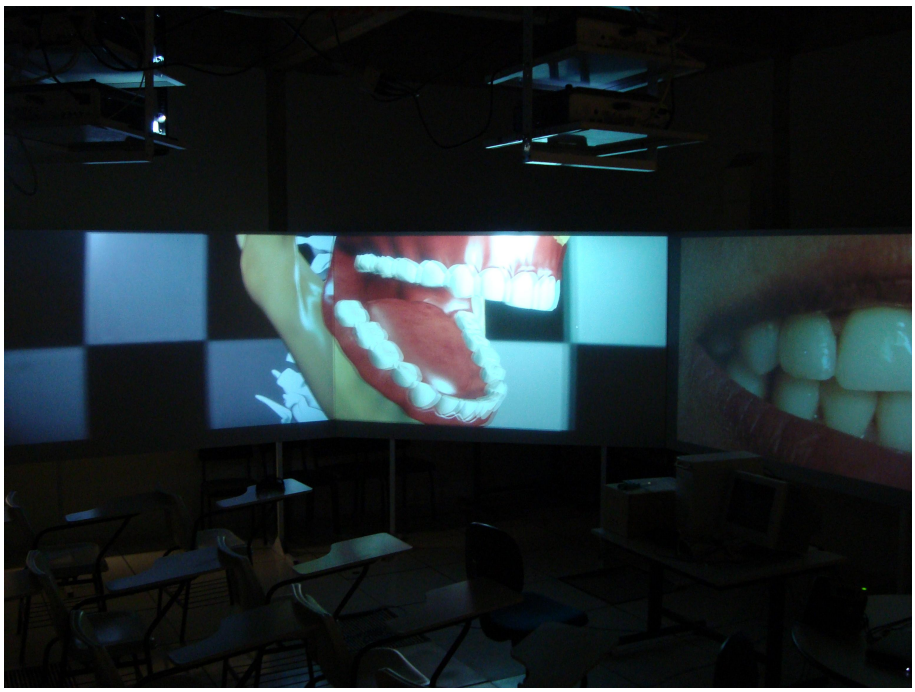


Figura 3.4: miniCAVE LSTR – UNESP.

3.2 Aglomerados gráficos

Os aglomerados gráficos, abordados nesta tese, são compostos pela combinação de *hardware* e *software*. Os componentes de *software* podem ser classificados em 4 componentes: a camada de aplicação, ferramentas de visualização, gerenciador de sistema do aglomerado e o sistema operacional. A Figura 3.5 apresenta as camadas da pilha de *software*.

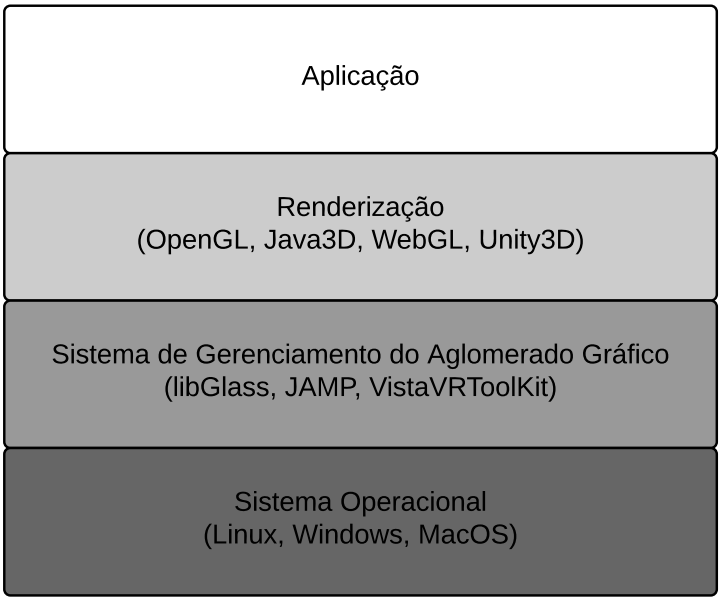


Figura 3.5: Pilha de *software*.

- A camada de aplicação é representada por ambientes virtuais utilizados de forma distribuída. Os ambientes podem ser desenvolvidos especificamente como ambientes distribuídos ou podem ser portados, dependendo da biblioteca de gerenciamento do aglomerado gráfico;

- A camada de renderização é composta por bibliotecas responsáveis pela geração das imagens do ambiente virtual. A renderização do ambiente virtual pode ser feita por bibliotecas nativas, tais como o OpenGL e o Java3D, ou por meio do uso de motores gráficos, tais como o Ogre3D e o Unity3D. Os motores gráficos tendem a facilitar o desenvolvimento de ambientes virtuais, visto que possuem primitivas específicas para interação, renderização e compatibilidade com vários formatos de ambiente. No entanto, não possuem suporte nativo a ambientes de multiprojeção com características de distribuição;

- A camada de gerenciamento do aglomerado gráfico é responsável pela comunicação entre os nodos do aglomerado. Esta camada é responsável por abstrair as formas de comuni-

cação entre os nodos, incluindo a intercomunicação entre aglomerados localizados em locais distintos. Esta camada foi o foco desta tese; e

- A camada do sistema operacional é responsável pelo ambiente no qual as camadas superiores são implantadas. Ela provê a comunicação entre as camadas superiores (*software*) e o *hardware*.

3.3 Ferramentas de desenvolvimento

Esta seção apresenta algumas ferramentas de desenvolvimento de ambientes virtuais distribuídos baseados em aglomerados gráficos. Porém, nem todas as bibliotecas apresentadas suportam a distribuição de dados em diferentes nodos de um aglomerado, sendo limitadas ao uso de memória compartilhada na tarefa de distribuição de dados entre processos.

Dentre as bibliotecas/arcabouços, pode-se citar o CaveLib (CAVELIB, 2016), o FlowVR (ALLARD *et al.*, 2004), o Syzygy (SCHAEFFER; GOUDESEUNE, 2003), o Chromium (BROWN; SEALES, 2001), o VR Juggler (VRJUGGLER, 2016), o FreeVR (SHERMAN *et al.*, 2013), o ViSTA VR Toolkit (VISTA, 2016) e a libGlass (LIBGLASS, 2016). No entanto, tais ferramentas limitam as aplicações a uma determinada localidade, não provendo suporte a aplicações colaborativas, ou quando não, permitem apenas a construção dessas para domínios específicos. Ao final desta seção é apresentada a comparação entre as ferramentas de desenvolvimento.

O primeiro arcabouço destinado ao desenvolvimento de aplicações de RV para ambientes de multiprojeção foi desenvolvido pelo EVL, sendo denominado CaveLib. O CaveLib possui uma *Application Programming Interface* (API) de baixo nível que abstrai o controle das tarefas ao desenvolvedor. Dentre as tarefas oferecidas pela API, têm-se a criação de pontos de vista, o sincronismo do aglomerado e compartilhamento de dados. A integração com o OpenGL é suportada. Atualmente é comercializado pela Mechdyne (CAVELIB, 2016).

O FlowVR (ALLARD *et al.*, 2004) é um *middleware* utilizado no desenvolvimento de aplicações de RV baseadas em aglomerados ou ambientes *grid*. Ele implementa uma grande gama de políticas para balancear a performance da aplicação com relação aos níveis de detalhes, latências e taxa de atualização de quadros (*framerate*).

A Syzygy (SCHAEFFER; GOUDESEUNE, 2003) é uma biblioteca desenvolvida pelo ISL, Universidade de Illinois. Ela provê um conjunto de ferramentas para a programação de ambientes de RV em aglomerados gráficos por meio de dois módulos principais: um arcabouço de grafo de cena distribuído, que permite que uma aplicação gráfica convencional seja renderizada

por múltiplos nodos; e um arcabouço baseado na arquitetura cliente–servidor para aplicações distribuídas síncronas.

O VR Juggler permite ao desenvolvedor portar aplicações de RV convencionais para aglomerados gráficos. A configuração do ambiente VR Juggler é feita por meio de arquivos *Extensible Markup Language* (XML) que definem o número de nodos, disponibilizando endereços de *hardware* e rede; dispositivos de entrada, tais como luvas, rastreadores etc; configurações de telas, *layout* físico e atribuição de nós. O VR Juggler é configurável em tempo de execução e possui uma interface gráfica de configuração, possibilitando que as configurações do aglomerado sejam alteradas durante a execução das aplicações. Suporta a maioria do sistemas operacionais e a renderização é feita utilizando o OpenGL (VRJUGGLER, 2016).

O Chromium é um arcabouço voltado ao desenvolvimento de aplicações que necessitem realizar renderização em tempo real com alta escalabilidade. Ele é derivado do projeto Stanford WireGL, que permite que aplicações nativas OpenGL possam ser renderizadas em aglomerados gráficos. O seu funcionamento se dá por meio de interceptações de chamadas OpenGL, dividindo-as entre os nodos de um aglomerado gráfico (CHROMIUM, 2016).

O Equalizer (EQUALIZER, 2016) é um *middleware* que permite que aplicações baseadas em OpenGL possam ser renderizadas em paralelo, possibilitando assim a execução utilizando várias placas de vídeo, processadores e computadores, visando aumentar o poder de processamento, qualidade e tamanho das imagens. Um dos benefícios do Equalizer é que a aplicação não precisa ser modificada para ser executada em um ambiente de multiprojeção. No entanto, funciona apenas em ambientes baseados em OpenGL. A proposta do Equalizer é semelhante a utilizada pelo Chromium.

A FreeVR (SHERMAN *et al.*, 2013) foi desenvolvida com o objetivo de ser o mais flexível possível. A biblioteca possui interfaces para a grande partes dos dispositivos. Segundo Sherman *et al.* (2013), a FreeVR atende aos requisitos requiridos de uma biblioteca de RV. Ela fornece multiprocessamento por meio de várias operações simultâneas.

A ViSTA VR *toolkit* (VISTA, 2016) é uma plataforma de *software* que permite a integração de tecnologias de RV e visualização 3D no desenvolvimento de aplicações voltadas à Visualização de Informação e Científica. Ela é utilizada como solução de desenvolvimento voltada ao aixCAVE na RWTH Aachen University.

3.4 A libGlass

A libGlass (LIBGLASS, 2016) é uma biblioteca de sincronização de ambientes gráficos distribuídos baseados em aglomerados gráficos. O objetivo principal desta biblioteca é facilitar o desenvolvimento de aplicações que necessitem de sincronização entre nodos de um aglomerado gráfico, possibilitando o uso de múltiplas projeções devido as possíveis configurações de diferentes pontos de vista e coerência nas imagens geradas por diferentes nodos (GUIMARAES, 2004). A libGlass foi utilizada como base no desenvolvimento da arquitetura apresentada nesta tese, sendo o candidato um dos desenvolvedores da biblioteca.

A libGlass provê um ambiente de fácil uso, tanto para o desenvolvimento de aplicações desde o início, como o porte de aplicações já existentes. Diferente de outras soluções disponíveis, a biblioteca não requer grande número de modificações no código fonte e na arquitetura da aplicação.

A biblioteca consiste em um conjunto escalável de componentes que podem ser utilizados para desenvolver aplicações baseadas em aglomerados gráficos. Ela também pode ser utilizada para portar aplicações de RV já utilizadas em sistemas fortemente acoplados. A Figura 3.6 apresenta uma visão geral dos seus componentes. Inicialmente, tem-se o Arcabouço, que é composto por três componentes:

- O componente Instanciação tem por objetivo inicializar as aplicações conforme a arquitetura interna da libGlass, o que permite criar aplicações servidor–cliente ou mestre–escravo;
- O componente Protocolo encapsula bibliotecas, escondendo as diferenças entre os protocolos de comunicação TCP, UDP, *Message Passing Interface* (MPI), entre outros; e
- O componente *Plugins* fornece serviços de transmissão de eventos, de compartilhamento de dados, de barreiras de sincronização e de associação de funções. Porém, vale ressaltar que a libGlass não está restrita apenas a esses *plugins*, podendo ser adicionados outros conforme a necessidade de cada aplicação, sem nenhuma modificação interna da biblioteca.

A biblioteca também possui um componente relacionado a criação ou portabilidade de novas aplicações. Este componente é denominado Aplicações, o qual possibilita o uso de diferentes protocolos, sem que a biblioteca seja modificada ou recompilada. Esse componente possui uma infraestrutura de empacotamento e desempacotamento de mensagens, que suporta todos os

tipos básicos (*integer*, *float*, *string*, entre outros). Este tipo de comportamento garante a interoperabilidade entre sistemas operacionais. Assim, pode-se, por exemplo, realizar execuções de alguns nodos da mesma aplicação no Linux, Windows ou Mac OS. Têm-se ainda as ferramentas de suporte, como: o GEditor (geração de aplicações para *palms*), o GTracer (análise de pacotes trafegados inter–processos) e o GVoicer (controle por áudio para ambientes imersivos) (GUIMARAES, 2004).

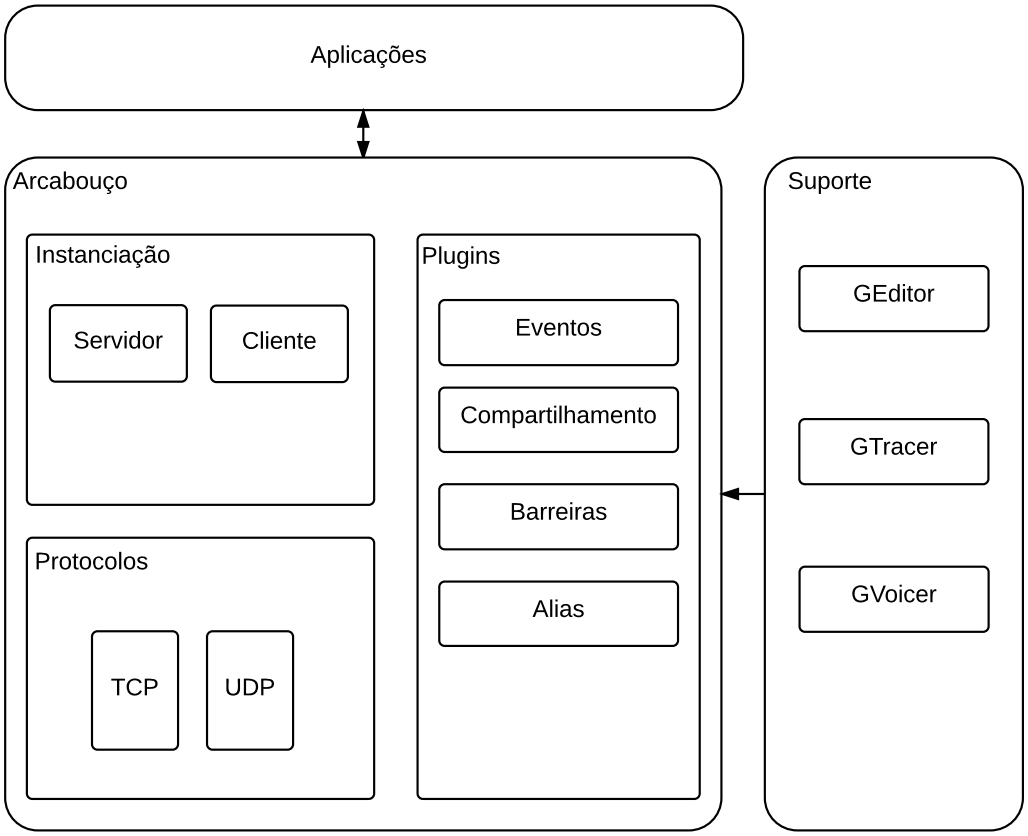


Figura 3.6: Arquitetura libGlass - baseada na tese de Guimaraes (2004).

A libGlass oferece abstração dos dados e independência de outras bibliotecas. Deste modo, os tipos comuns de dados podem ser criados e manipulados. Entretanto, pode-se escolher outras bibliotecas para trabalhar em conjunto com a libGlass, por exemplo, a biblioteca OpenGL ou Java 3D (DIAS *et al.*, 2010a).

3.5 Comunicação e sincronização de dados em 3DCVEs

No geral, todas as bibliotecas apresentadas na Seção 3.3 não abordam soluções com características de colaboração entre ambientes virtuais. As únicas bibliotecas que possuem implementações prévias relacionadas à comunicação inter–aglomerados são a FreeVR e a VRJuggler,

entretanto, não foram encontrados artigos científicos que abordem com profundidade esta característica.

Esta seção apresenta soluções de arquiteturas utilizadas na distribuição de ambientes colaborativos que podem ser extendidas a 3DCVEs. Essas arquiteturas abordam algumas características, tais como sincronismo, tolerância a falhas, transparência na distribuição, extensibilidade, escalabilidade, interoperabilidade e portabilidade.

3.5.1 Técnicas de comunicação

A comunicação entre nodos de uma rede pode ser feita de diferentes maneiras, tais como *unicast*, *broadcast* e *multicast*. A Figura 3.7 apresenta exemplos do uso dessas técnicas de envio/recebimento de dados.

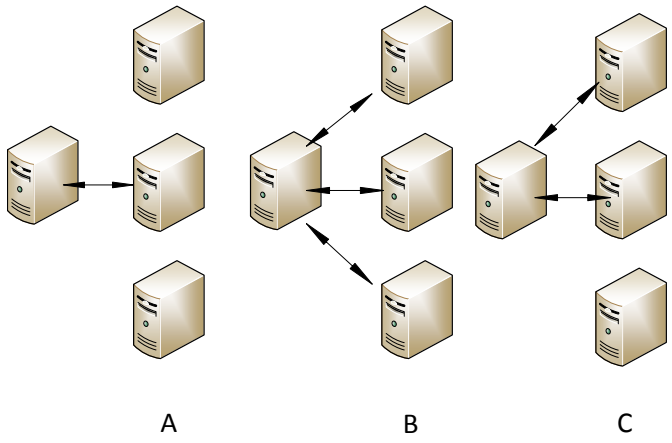


Figura 3.7: Tipos de comunicação A) Unicast - B) Broadcast - C) Multicast.

A técnica *unicast* realiza o envio de dados utilizando uma conexão ponto–a–ponto, que permite que uma mensagem seja enviada a apenas um nodo da rede. Dessa forma, se a mesma mensagem tiver de ser enviada a outros nodos, esta deverá ser transmitida novamente e individualmente, porém, por possuir essa característica, podem ocorrer problemas dependendo do tipo de tráfego exigido (TANENBAUM; VIEIRA, 2011).

Nas transmissões *broadcast*, um nodo envia dados para todos os outros nodos presentes na rede. No entanto, existe uma desvantagem, pois mesmo que um nodo não necessite de tal mensagem, ele receberá e deverá processá-la ou ignorá-la. Outra desvantagem, é que as mensagens enviadas por *broadcast*, geralmente, utilizam o protocolo UDP, o que, dependendo do tipo de rede de dados utilizada, pode ser um problema, como por exemplo, a Internet. Mensagens *broadcast* são interceptadas por roteadores da Internet por motivo de segurança (KUROSE; ROSS,

2012).

A técnica de *multicast* é a mais interessante para usos em 3DCVEs, pois implementa o conceito de canais. Quando um nodo envia dados utilizando *multicast*, somente os nodos conectados ao mesmo canal recebem esses dados. Todavia, esta técnica não permite o envio de dados diretamente sobre redes WAN, o que pode ser contornado com o uso de redes *overlay* (CHEN *et al.*, 2007; LIU; MA, 2010).

3.5.2 Arquiteturas de distribuição

A interação entre os multiusuários de um 3DCVE pode ser complexa, de modo que a especificação de uma arquitetura conceitual pode ser de grande ajuda. Uma arquitetura conceitual é definida como sendo uma organização apropriada do sistema, que permita o desenvolvimento eficiente, apresentando os componentes ou partes do sistema sob a forma de unidade de desenvolvimento de *software* (PIMENTEL; FUKS, 2011).

A arquitetura conceitual foca o desenvolvimento e manutenção do sistema. No entanto, a implementação do sistema depende da arquitetura de distribuição, que é construída com base na arquitetura conceitual do 3DCVE. Como apresentado no Capítulo 2, foram encontradas algumas arquiteturas que utilizam diferentes modelos de distribuição. Os três tipos principais de arquitetura são: centralizada, descentralizada e híbrida.

A arquitetura centralizada é baseada no modelo lógico de cliente–servidor. Assim, todo o processamento existente é realizado no nodo servidor. Os nodos clientes são responsáveis apenas pela apresentação das primitivas gráficas aos usuários. Este tipo de arquitetura, apesar de ser um modelo simples, possui alguns problemas, tais como:

- Escalabilidade: o fato de o servidor ser o ponto central de sincronismo e distribuição de dados, podem ocorrer problemas com relação à quantidade de nodos clientes suportados pelo ambiente. A medida que a quantidade de usuários aumenta, o servidor pode não conseguir suprir a necessidade de processamento; e
- Baixa tolerância a falhas: devido o servidor ser considerado um ponto único de falha, todo o sistema é interrompido no caso de uma falha deste. Este tipo de falha pode ser contornado com o uso de técnicas de replicação, porém, prejudicando a simplicidade inicial provida pela arquitetura.

Apesar das desvantagens mencionadas, a arquitetura centralizada também possui algumas vantagens. A implementação da arquitetura centralizada é simples, visto que só existe uma

instância compartilhada entre todos os nodos. Esta característica elimina problemas de consistência e sincronismo de dados. A gerência de segurança também é facilitada neste modelo de arquitetura, pois o servidor gerencia todos os acessos aos recursos. Outro ponto está relacionado à conexão de nodos posteriormente ao início de execução do ambiente. Um 3DCVE totalmente centralizado permite a conexão de nodos facilmente durante o tempo de execução, devido ao forte sincronismos provido.

A arquitetura descentralizada realiza o processamento em todos os nodos do ambiente, replicando todos os dados distribuídos. Cada nodo possui uma instância completa do ambiente colaborativo. Por possuir replicação, este modelo de arquitetura possui maior tolerância a falhas. A ausência de um servidor central também permite que a escalabilidade do ambiente seja maior, pois manter este ambiente síncrono e consistente não é uma tarefa trivial. Esta arquitetura é baseada no modelo mestre–escravo.

A escalabilidade alcançada em arquiteturas descentralizadas é alta, entretanto, a alta distribuição causa uma maior troca de mensagens entre os nodos. O uso de protocolos e formatos otimizados de troca de mensagens são utilizados. O uso de outros recursos de *hardware* também são maiores, tais como processadores e memórias, devido a replicação dos dados. A gerência de dados também é uma tarefa complexa, visto que ambientes com esta arquitetura são, comumente, propícios a inconsistências temporárias. A inconsistência temporária causa problemas de interação entre os usuários e torna mais difícil a conexão de usuários durante a execução do ambiente (PIMENTEL; FUKS, 2011).

As arquiteturas híbridas são caracterizadas por combinar as centralizadas e descentralizadas. Esta combinação permite que o ambiente se comporte de maneira distinta, dependendo da necessidade. Um modo de arquitetura híbrida bem difundido é o uso de um servidor para gerenciar o espaço de dados colaborativo, de modo que os nodos clientes gerenciam o espaço privado. Este modelo combina as vantagens dos dois modelos (centralizado e descentralizado), mantendo a simplicidade do modelo centralizado, mas ao mesmo tempo permitindo uma maior escalabilidade.

Existem ainda modelos baseados na topologia de rede ponto–a–ponto, que vêm sendo utilizados nos últimos anos em pesquisas relacionadas a 3DCVEs (vide seção 2). Na topologia ponto–a–ponto, todos os nodos são caracterizados como sendo clientes e servidores. Por possuir características de sincronismo fraco e alta escalabilidade, ele á altamente indicado para o desenvolvimento de 3DCVEs baseados em redes de dados sobre a Internet. Assim, este modelo é implementado sobre redes físicas existentes. No entanto, as redes não podem ser modificadas pelas aplicações que necessitam trocar mensagens sobre elas. As redes ponto–a–ponto se adap-

tam criando redes com uma topologia própria. As redes sobrepostas (*overlay*) são um tipo de rede criada sobre uma rede física existente. Na Figura 3.8 é apresentado um exemplo de rede ponto–a–ponto sobreposta. Os enlaces lógicos não correspondem, necessariamente, aos enlaces físicos, de modo que um enlace lógico pode ser, na verdade, um conjunto de enlaces físicos, ou até mesmo rotas entre vários enlaces físicos.

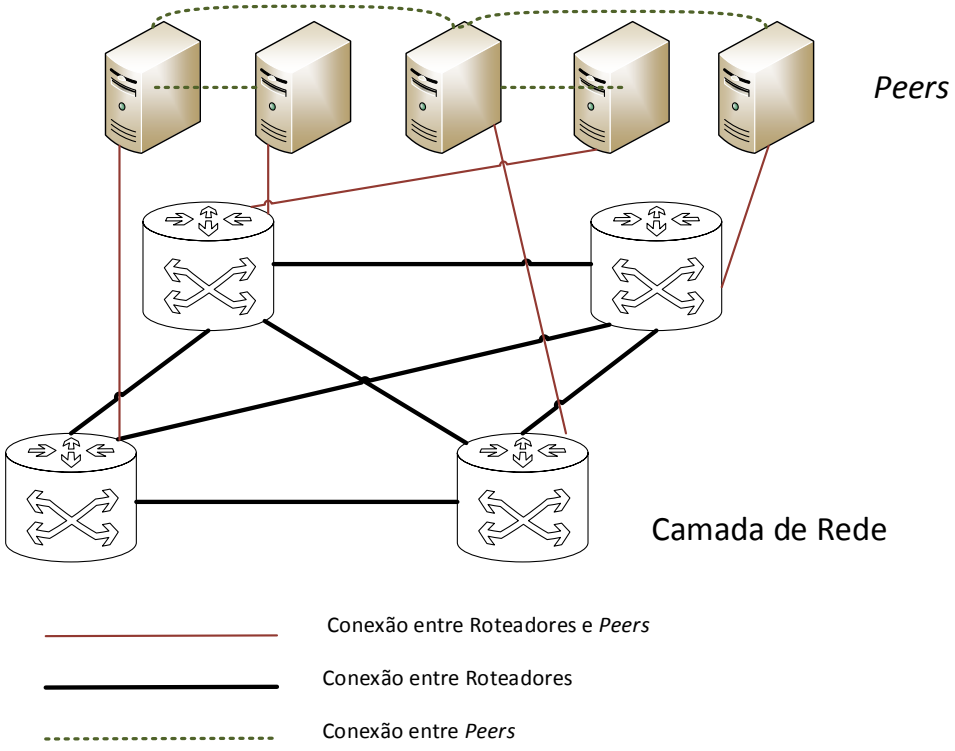


Figura 3.8: Rede ponto–a–ponto *Overlay*.

As redes ponto–a–ponto também são divididas em três diferentes modelos de arquitetura: centralizado, descentralizado e semicentralizado.

O modelo centralizado possui um servidor central que mantém um índice de busca. Quando um nodo realiza a busca de um recurso compartilhado na rede, este envia uma requisição ao servidor central, o qual retorna a lista dos nodos que compartilham este recurso. A troca de dados posterior é realizada diretamente entre os nodos. Devido a existência deste nodo centralizado, o modelo apresenta baixa tolerância a falhas e pouca estabilidade (PIMENTEL; FUKS, 2011).

O modelo descentralizado e não estruturado não utiliza um servidor central para manter o índice de busca. A busca neste modelo é feita por inundação, ou seja, é realizada de forma colaborativa entre os nodos, que consultam seus vizinhos repassando aos outros o índice.

O modelo semicentralizado utiliza um modo misto, que combina as vantagens dos modelos centralizado e descentralizado. Os nodos possuem papéis distintos. Alguns nodos atuam como supernodo, responsável pelas tarefas administrativas, enquanto que outros são associados a um supernodo.

3.5.3 Protocolos de comunicação: TCP, UDP e SCTP

A arquitetura apresentada nesta tese foi baseada em soluções focadas em topologias de redes e protocolos de comunicação. Assim, inicialmente, os protocolos de comunicação que foram utilizados para a comunicação foram: TCP, UDP e SCTP. Esses protocolos foram citados em alguns trabalhos relacionados (Capítulo 2).

O protocolo TCP utiliza uma grande quantidade de mensagens e pacotes em suas transmissões, pois além de realizar testes para envio de dados em fila, ainda realiza verificações com relação a entrega de pacotes, característica relacionada a garantia de entrega. Em ambientes baseados em aglomerados gráficos o seu uso pode ser realizado em diferentes tipos de aplicações, por exemplo, em comunicações realizadas inter–aglomerados, o uso do TCP/IP é indicado, pois garante a entrega e ordem de recebimento dos pacotes, o que, em alguns casos, é extremamente importante, como no caso de pacotes chave (*handshake* e liberação de barreiras). No entanto, em comunicações intra aglomerado, onde a comunicação é realizada sobre redes de dados Gigabit, o uso do TCP/IP pode prejudicar o desempenho das aplicações, visto que além das características já mencionadas, ele ainda não permite a realização de *broadcast* e *multicast*, sendo um grande problema de aglomerados com muitos nodos.

Ao contrário do TCP, o protocolo UDP realiza o envio de dados com custo menor de atraso, já que não existe enfileiramento de mensagens, também permitindo o uso de *broadcast* e *multicast*, contudo, este não garante a entrega e ordem de chegada dos pacotes. Outra característica prejudicial, é que as comunicações realizadas com o UDP não mantêm a conexão aberta, o que não é interessante quando a quantidade de dados a ser enviada é muito grande.

O protocolo SCTP tem sido utilizado no desenvolvimento de aplicações distribuídas sobre a Internet. Com ele é possível realizar a transmissão de dados de datagramas (UDP) de forma confiável. Os datagramas são enviados com garantias de serem livres de erros e de duplicações, características alcançadas por meio de ACKs, semelhantes aos utilizados pelo protocolo TCP. Sua principal característica é a possibilidade de envio de múltiplos *streams* em uma mesma associação, enquanto o TCP permite o envio de apenas um *stream* por sessão.

3.5.4 Modelo de base de dados de 3DCVEs

Uma das preocupações na concepção de 3DCVEs é forma como o conteúdo é compartilhado. Portanto, o modelo de base de dados utilizado possui total influência no desempenho, consistência e escalabilidade dos 3DCVEs (OLIVEIRA *et al.*, 2004; AQUINO; KIRNER, 2001). O modelo de distribuição é classificado em quatro tipos: centralizado, replicado, distribuído e híbrido.

Nas bases de dados centralizadas os dados são armazenados em um único nodo central da rede, que atende as requisições de dados de outros nodos. Uma das grandes desvantagens deste tipo de modelo, é que o nodo central pode se tornar um gargalo. Devido as inúmeras requisições de outros nodos, o nodo central pode não ser capaz de prover dados a todos os requerentes. Em se tratando do uso de redes de dados que não sejam de alta velocidade na interligação dos nodos, este tipo de base de dados também não é indicado.

Nas bases de dados replicadas, todos os nodos possuem os dados disponibilizados localmente, o que permite maior interatividade, mas menor consistência. A manutenção de consistência destas bases de dados não é uma tarefa trivial, pois quando um nodo realiza alguma alteração na base de dados, os outros nodos devem ser atualizados.

Já nas bases de dados distribuídas, vários nodos possuem os dados, de modo que cada um desses é responsável por prover dados a um conjunto de nodos requerentes. Somente os nodos conectados ao nodo central recebem as mensagens de atualização, assim, cada conjunto de nodos é atualizado por somente um dos nodos centrais, diminuindo o surgimento de possíveis gargalos.

As bases de dados híbridas combinam características dos modelos distribuído e replicado. Os dados são armazenados em nodos centrais. No entanto, no decorrer da execução da aplicação, quando os nodos clientes necessitam de dados, estes são fornecidos sob demanda. Os dados recebidos pelos nodos clientes são mantidos localmente, por meio de *cache*.

3.5.5 Sincronismo

O uso de técnicas de sincronismo em aplicações distribuídas não é uma tarefa trivial como em sistemas locais, sejam esses mono ou multiprocessados. Em sistemas centralizados, o tempo não possui ambiguidade, de modo que, quando um processo necessita saber o tempo corrente, basta enviar uma solicitação por meio de uma chamada de sistema ao *kernel* do sistema operacional. No caso de sistemas distribuídos, a tarefa de prover o sincronismo entre processos

independentes é uma tarefa complicada, pois cada computador possui um relógio local (TANENBAUM; STEEN, 2006).

Existem alguns algoritmos presentes na literatura capazes de realizar o sincronismo de processos distribuídos. Um dos principais algoritmos mencionados na literatura é o *Network Time Protocol* (NTP). Nele, os nodos de um aglomerados contatam um servidor para manter os seus relógios sincronizados. Mesmo que eles não estejam em um horário correto (UTC), eles ainda estarão sincronizados. Todavia, quando um nodo requer a configuração do relógio ao servidor, o atraso das mensagens de rede possuem influência no horário adquirido. Para resolver este tipo de problema, são utilizadas técnicas de estimação de atraso na conexão.

A Figura 3.9 apresenta um exemplo de solicitação de sincronismo de um nodo A a um nodo B. O nodo A envia uma solicitação ao nodo B, porém, a mensagem de A leva um certo tempo para ser entregue ao nodo B (T2). O nodo B retorna a configuração de relógio requisitada pelo nodo A (T3). Finalmente, o nodo A recebe a resposta (T4).

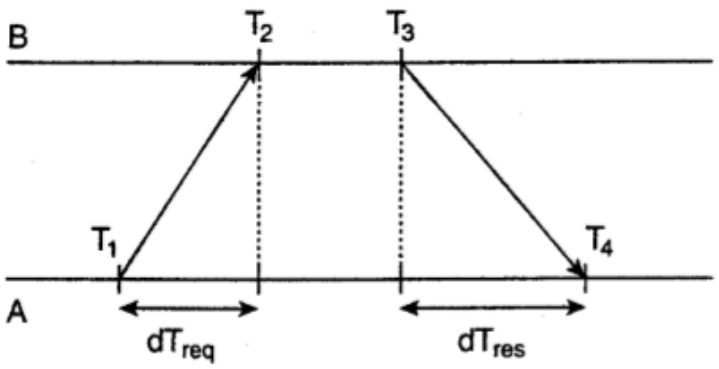


Figura 3.9: Requerendo o tempo corrente de um servidor de sincronismo.
(TANENBAUM; STEEN, 2006).

3.6 Considerações finais

Este capítulo apresentou o estado da arte e definições de conceitos e tecnologias sobre ambientes de multiprojeção. Foram apresentados ambientes específicos, tais como o CAVE e o miniCAVE, que utilizam aglomerados gráficos na geração de imagens.

As bibliotecas de desenvolvimento apresentadas são específicas a ambientes virtuais distribuídos que não provêem a possibilidade de colaboração entre diferentes aglomerados. Dentre as bibliotecas de desenvolvimento, a libGlass foi evidenciada, por se tratar da solução de *software* que foi utilizada para validar a arquitetura proposta nesta tese. A Tabela 3.1 apresenta as principais características de cada uma das bibliotecas/arcabouços/*middlewares* mencionados

nesta seção. São apresentadas algumas informações, tais como a compatibilidade com diferentes soluções gráficas, os sistemas operacionais compatíveis, a forma de distribuição (livre ou privada) e suas principais características.

Tabela 3.1: Principais características das soluções de desenvolvimento.

Biblioteca	**Caracteristicas**	**Bibliotecas gráfica**	**Plataforma**	**Distribuição**
CaveLib	Fácil de programar; o desenvolvedor não tem acesso ao código fonte	Possui soluções fechadas de renderização, como por exemplo, o getReal3D (Unity3D)	Windows e Linux	Privada, distribuída pela Mechdyne
FlowVR	Fácil de programar, com poucas modificações no código	Permite a integração com diversas soluções de renderização	Linux e MacOS	Código aberto
Syzygy	Distribui dados em aglomerados; difícil de programar	Permite integração com soluções baseadas em C/C++ e Python	Windows, Linux e MacOS	Código aberto
Chromium	Multiprojeções sem alteração no código fonte das aplicações; última versão lançada em 2006	Funciona apenas com o OpenGL	Windows e Linux	Código aberto
VRJuggler	Possui arquitetura cliente–servidor e ponto–a–ponto; muitas modificações no código para efetuar a distribuição	Permite integração com diversas soluções	Windows e Linux	Código aberto
Equalizer	Multiprojeções sem alteração no código fonte das aplicações	Funciona apenas com o OpenGL	Windows, Linux e MacOS	Código aberto
FreeVR	É compatível com vários dispositivos de entrada	Compatível com renderizações baseadas em OpenGL	Windows, Linux e MacOS	
ViSTA VR	Multiprojeções baseadas em aglomerado gráficos (mestre–escravo). Compatível com diversos dispositivos de entrada e rastreamento	Utiliza o OpenSG como solução de renderização	Windows e Linux	Código aberto
libGlass	Possui arquitetura cliente–servidor e mestre–escravo; são necessárias poucas modificações no código para realizar distribuição	Permite integração com diversas soluções de renderização, inclusive em diferentes linguagens, tais como C/C++, Java e Python	Windows e Linux	Código aberto

Capítulo 4

DEFINIÇÃO DA ARQUITETURA DE COMUNICAÇÃO INTRA–AGLOMERADO E INTER–AGLOMERADOS VOLTADA A 3DCVES

Segundo SHAW e GARLAN (1996) uma arquitetura de *software* é definida como sendo um sistema de componentes computacionais e seus relacionamentos. Outra definição semelhante é a de BASS e CLEMENTS (1998), onde eles definem uma arquitetura de *software* como sendo um conjunto de componentes, suas propriedades externas e seus relacionamentos, que constitui a abstração do sistema perante ao desenvolvedor.

Neste capítulo é descrita a arquitetura geral da pesquisa desenvolvida, que teve como objetivo principal investigar os problemas e soluções relacionados à influência de redes de dados na concepção de 3DCVEs. Um modelo de conexão denominado dinâmico-adaptável é apresentado neste capítulo, sendo descrito do seguinte modo:

- Verificação dinâmica: como apresentado no Capítulo 2, existem diversas adversidades a serem sanadas/mitigadas em relação ao desenvolvimento de 3DCVEs. Essas adversidades tendem a ser dinâmicas, devido as variações presentes nas redes de dados, principalmente nas redes baseadas na Internet. Assim, um modelo de verificação dinâmico de conexões inter–aglomerados é uma contribuição que não está presente em outras pesquisas relacionadas; e

- Tomada de decisão adaptável: diversas soluções de desenvolvimento de 3DCVEs são mencionadas na literatura (Capítulo 2). Porém, ainda não existe uma solução de desenvolvimento unificada para tal classe de aplicação. A arquitetura desenvolvida visa proporcionar diferentes configurações, utilizando diversas topologias, protocolos e algo-

ritmos na concepção de conexões intra–aglomerado e inter–aglomerados. Essas configurações devem ser realizadas em tempo de execução, permitindo que o 3DCVE se adapte às adversidades ocasionadas pelas redes de dados.

4.1 Aglomerados gráficos remotos

Nesta pesquisa foram realizadas investigações relacionadas à comunicação intra–aglomerado e inter–aglomerados. Assim, para este fim, foram utilizados aglomerados gráficos remotos geograficamente. A Figura 4.1 apresenta os aglomerados gráficos que compuseram a arquitetura de experimentação. São apresentados três lugares distintos.

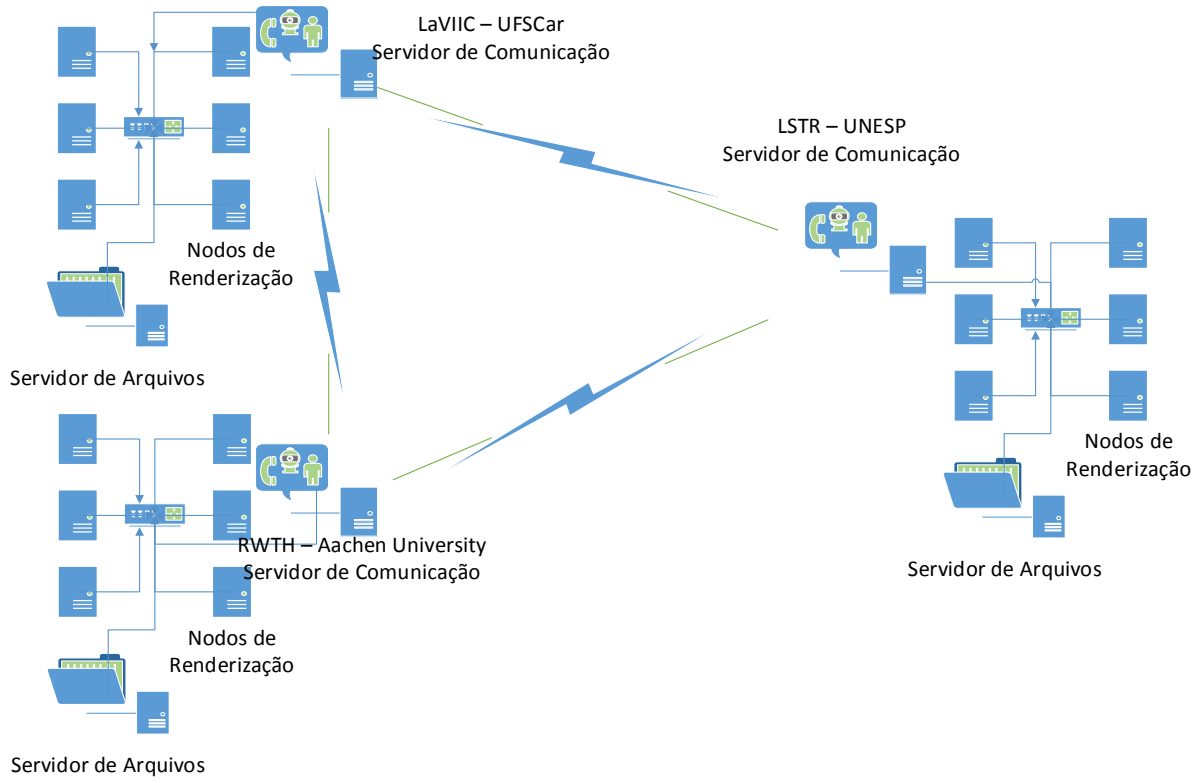


Figura 4.1: Aglomerados gráficos remotos.

Como apresentado na Figura 4.1, cada local possui um aglomerado gráfico reponsável pela renderização de ambientes virtuais, localmente. Os aglomerados gráficos são compostos por diferentes tipos de nodos que, nesta tese, são classificados como sendo:

- Servidor de Arquivos: armazena as aplicações e arquivos referentes aos modelos virtuais, que são replicados entre os aglomerados gráficos. Cada aglomerado possui o seu

servidor de arquivos. Uma das razões em se replicar os arquivos, está relacionada ao desempenho, sendo extremamente importante em sistemas distribuídos que necessitem de escalabilidade em diferentes áreas geográficas (TANENBAUM; VIEIRA, 2011);

- Servidor de Comunicação do aglomerado gráfico: responsável por sincronizar os nodos de renderização e realizar a comunicação com os outros servidores utilizando um modelo de sincronismo fraco para ambientes inter–aglomerados. Este nodo é o servidor/mestre do aglomerado gráfico local, visto que o modelo de comunicação intra–aglomerado utiliza um modelo cliente–servidor ou mestre–escravo. Os Servidores de Comunicação se comunicam uns com os outros por meio de uma topologia ponto–a–ponto;

- Servidor de Conexão Dinâmico-Adaptável: responsável por realizar as verificações em relação as adversidades da rede, capaz de modificar as configurações de comunicação intra–aglomerado e inter–aglomerados em tempo de execução, ou emitir avisos ao desenvolvedor, para que este possa tomar a melhor ação, a fim de manter a qualidade do ambiente. Esta tarefa é realizada em um nodo local centralizado do aglomerado gráfico, responsável por obter resultados referentes as conexões entre os Servidores de Comunicação dos aglomerados gráficos, visto que esses podem estar conectados por diferentes tipos de redes de dados (gigabit ou Internet). Assim, diferentes configurações de topologias e protocolos são utilizadas na intercomunicação entre os nodos. Por exemplo, dois servidores podem estar interligados por uma mesma subrede, entretanto, outros servidores podem estar interligados por meio da Internet; e

- Nodo de Renderização: responsável por realizar o desenho do ambiente virtual. Um conjunto de Nodos de Renderização são gerenciados pelo Servidor de Comunicação do aglomerado gráfico. Diferentes soluções podem ser usadas na geração de ambientes virtuais, tais como primitivas gráficas OpenGL ou até mesmo motores de jogo, como o Unity3D (NETO *et al.*, 2015). Na arquitetura apresentada, foi utilizada a abordagem de renderização com replicação (GUIMARAES, 2004), reduzindo assim a troca de informações entre os Nodos de Renderização e o Servidor de Comunicação, visando otimizar a comunicação entre Servidores de Comunicação sobre redes de dados baseadas em Internet.

4.2 *Plugins* para 3DCVEs baseados em aglomerados gráficos

Segundo Gnecco (2003) a libGlass possui otimizações que possibilitam a criação de aplicações distribuídas de alto desempenho, sendo limitadas pela banda das redes de dados, o que

é comprovado por meio de experimentos em sua pesquisa. Além disso, a libGlass foi escolhida devido outras características providas, tais como:

- Transparência ao usuário: o desenvolvimento de aplicações utilizando a libGlass é simples e intuitivo, o que torna o seu aprendizado rápido e fácil;
- Facilidade de uso e porte: tanto o desenvolvimento, quanto o porte de aplicações, é facilitado pela biblioteca, sendo possível utilizá-la mesmo em soluções de código legado;
- Desempenho e eficiência: por ser uma biblioteca utilizada no desenvolvimento de aplicações gráficas, é extremamente importante que a biblioteca tenha um ótimo desempenho;
- Independência de protocolos de rede: uma das características mais importantes para a sua escolha como solução de desenvolvimento para este trabalho, é que sua camada de abstração possibilita o uso de diferentes protocolos de comunicação de maneira simples e rápida;
- Portabilidade e interoperabilidade: a biblioteca é portável a vários dispositivos. Dispositivos móveis, tais como *smartphones*, podem ser utilizados como nodo, sendo normalmente utilizados como meio de interação. A interoperabilidade da biblioteca permite a sua utilização em aglomerados híbridos;
- Arquitetura de rede reconfigurável: permite utilizar diferentes tipos de topologias de rede. Originalmente, a biblioteca permitia o uso das topologias cliente–servidor e mestre–escravo; e
- Suporte a processos leves: a maioria das bibliotecas não possui suporte a *threads*.

Para se desenvolver 3DCVEs baseados em aglomerados gráficos utilizando a libGlass alguns requisitos tiveram de ser alcançados por meio do desenvolvimento de novos módulos. Os módulos foram estendidos ou desenvolvidos em forma de *plugins*. A Figura 4.2 apresenta a arquitetura atual dos novos *plugins* e protocolos voltados à arquitetura apresentada nesta tese. É importante destacar que a arquitetura poderia ser desenvolvida sobre outras bibliotecas/arcabouços, ou até mesmo a partir de um novo projeto, porém, demandaria maior esforço do ponto de vista de codificação.

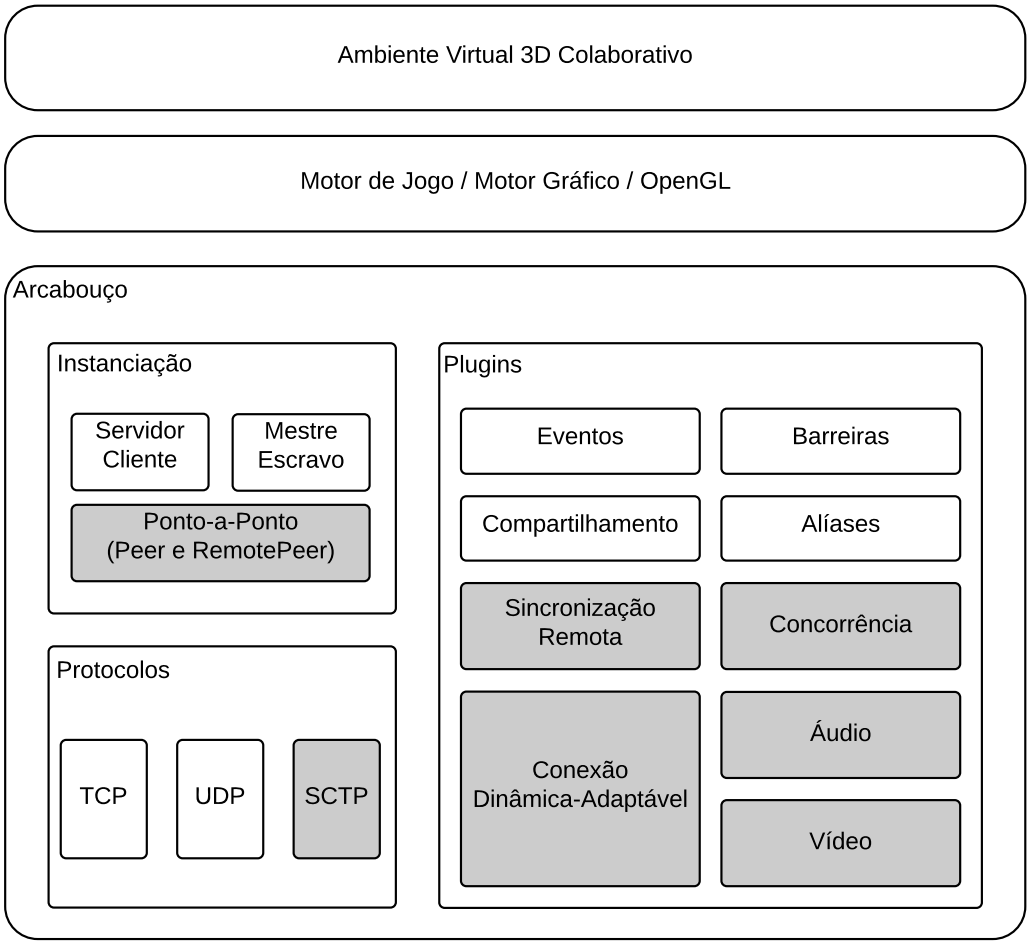


Figura 4.2: Arquitetura dinâmica e adaptável modularizada.

Os *plugins* têm como objetivo prover suporte a aplicações colaborativas síncronas/assíncronas, que permitam que usuários distribuídos geograficamente realizem atividades ou alcancem objetivos comuns, de tal forma que os requisitos necessários ao desenvolvimento de 3DCVEs sejam atendidos. Alguns dos *plugins* previamente disponíveis na libGlass foram reutilizados para a construção dos novos, que são:

- Sincronização remota: extensão do *plugin* Compartilhamento (*Shared*) da libGlass, que permite a cada Servidor de Comunicação manter-se conectado a outros servidores. O Servidor de Comunicação local obtém os dados dos Nodos de Renderização e os envia aos Servidores de Comunicação remotos;

- Concorrência: *plugin* que provê para cada localidade o acesso concorrente sobre objetos compartilhados. Para que a libGlass ofereça suporte aos diversos tipos de aplicações cooperativas, vários métodos foram disponibilizados. Inicialmente, foi implementado o método de bloqueio não-otimista, que assegura que todos os eventos sejam sempre

executados na ordem correta em todos os usuários, não permitindo que eventos sejam executados fora de ordem (TANENBAUM; VIEIRA, 2011);

- Vídeo: *plugin* que possibilita que cada localidade transmita ou receba vídeos, visto que é necessário um canal de comunicação entre os diferentes locais. Esse *plugin* é uma extensão do *plugin* Evento (Events);

- Áudio: *plugin* que permite que cada localidade transmita ou receba áudio. Assim, como o *plugin* Vídeo, este também é uma extensão do *plugin* Evento (Events); e

- Conexão Dinâmica e Adaptável: *plugin* que permite que as adversidades sejam verificadas em tempo de execução, permitindo que diferentes modos de conexão e configuração sejam utilizados. Este *plugin* permite que um Servidor de Conexão Dinâmico-Adaptável seja instanciado. A seção 5.8 apresenta detalhes de implementação deste *plugin*. A análise dos pacotes é realizada seguindo um modelo de *logs* gerados pela biblioteca.

Os componentes Ponto–a–ponto e Servidor Regional são importantes dependendo do tipo de rede de dados utilizado, visto que em redes em que a latência e o atraso na entrega de pacotes sejam altos não é aconselhável manter um modelo de sincronismo forte (aplicado em arquiteturas cliente–servidor ou mestre–escravo).

- *Peer*: este componente de instanciação é responsável por permitir que os Servidores de Comunicação dos aglomerados gráficos possam trocar informações entre si utilizando um modelo de compartilhamento de dados sem a necessidade de um servidor centralizado, visando reduzir ao máximo o atraso na comunicação. Todos os Servidores de Comunicação são tratados como servidores e clientes (*peers*);

- *Remote Peer*: este componente tem o intuito de prover a possibilidade de escalabilidade à arquitetura, de modo que apenas os Servidores de Comunicação que façam parte de uma dada região possam se comunicar com ele. Assim, dois ou mais Servidores Regionais podem ser utilizados. Este componente é útil no caso da existência de uma grande quantidade de aglomerados gráficos interconectados, visto que se seria enviável manter conexões entre um grande número de Servidores de Comunicação, devido á quantidade de conexões que um processo deveria manter paralelamente. Assim, um Servidor Regional é responsável pelo gerenciamento de um ou mais Servidores de Comunicação;

A geração de *log* da biblioteca também foi aprimorada. Informações referentes aos pacotes são geradas pela biblioteca. Assim, por meio de arquivos *log* é possível obter informações

relacionadas as conexões, tais como vazão e atraso na entrega de pacotes, além de verificar o comportamento esperado para ambientes inter–aglomerados. A geração de *logs* é discutida em detalhes na Seção 6.

A solução desenvolvida nesta tese utiliza um modelo de arcabouço, no que diz respeito as implementações. Um arcabouço permite a reutilização de código por meio de classes e objetos, sendo composto por classes abstratas que são utilizadas por subclasses. A Figura 4.3 apresenta o diagrama de classe referente à arquitetura geral do arcabouço estendido da libGlass. A classe principal desse diagrama é a classe abstrata *Glass*. As classes filhas, *Cliente*, *Servidor*, *Mestre*, *Escravo*, *Peer* e *RemotePeer*, definem os objetos a serem instanciados pelas aplicações. A classe *Protocolo* também é utilizada pela classe *Glass*, definindo as funções que devem ser implementadas na subclasse. Dessa forma, a mesma classe abstrata *Protocolo* permite que diversos protocolos de comunicação possam ser implementados seguindo o mesmo padrão. A classe *Gerenciamento de Plugin* é responsável pelo gerenciamento dos *plugins*, garantindo que os pacotes de dados sejam processados pelo *plugin* correto. A classe plugin define as funções que devem ser implementadas pelas subclasses, tais como barreiras de sincronismo, mensagens assíncronas (Eventos), mensagens síncronas (Compartilhamento), nomeação de processos distribuídos (Alíases) e gerenciamento de conexão.

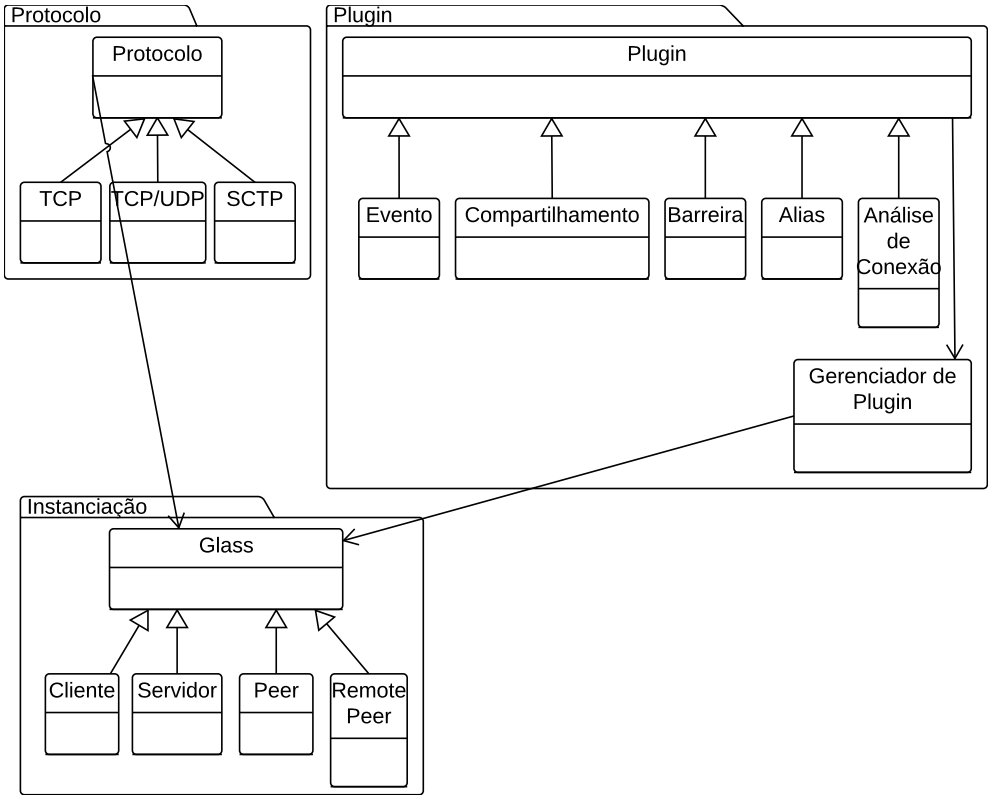


Figura 4.3: Diagrama de classes – libGlass.

4.2.1 Comunicação intra–aglomerado gráfico

A comunicação intra–aglomerado gráfico é realizada por meio do modelo cliente–servidor ou mestre–escravo, visto que os nodos de um aglomerado gráfico local são interconectados por meio de uma rede de dados local, sendo este o modelo de conexão ideal para manter todos os nodos com um alto grau de sincronismo, pois todas as mensagens trocadas entre os nodos clientes/escravos devem ser intermediadas pelo servidor/mestre. O compartilhamento de mensagens é realizado utilizando um modelo *multicast* ou *unicast*. A Figura 4.4 apresenta um diagrama de sequência que ilustra a troca de mensagens em um modelo intra–aglomerado, onde o Cliente_1 realiza o envio de um valor atualizado ao Servidor, executando logo em seguida uma chamada de sincronização. Os outros clientes, Cliente_2 e Cliente_N, requisitam o valor atualizado ao Servidor. Assim que o Servidor recebe todas as chamadas de sincronismo, este envia uma mensagem de liberação de barreira a todos os clientes.

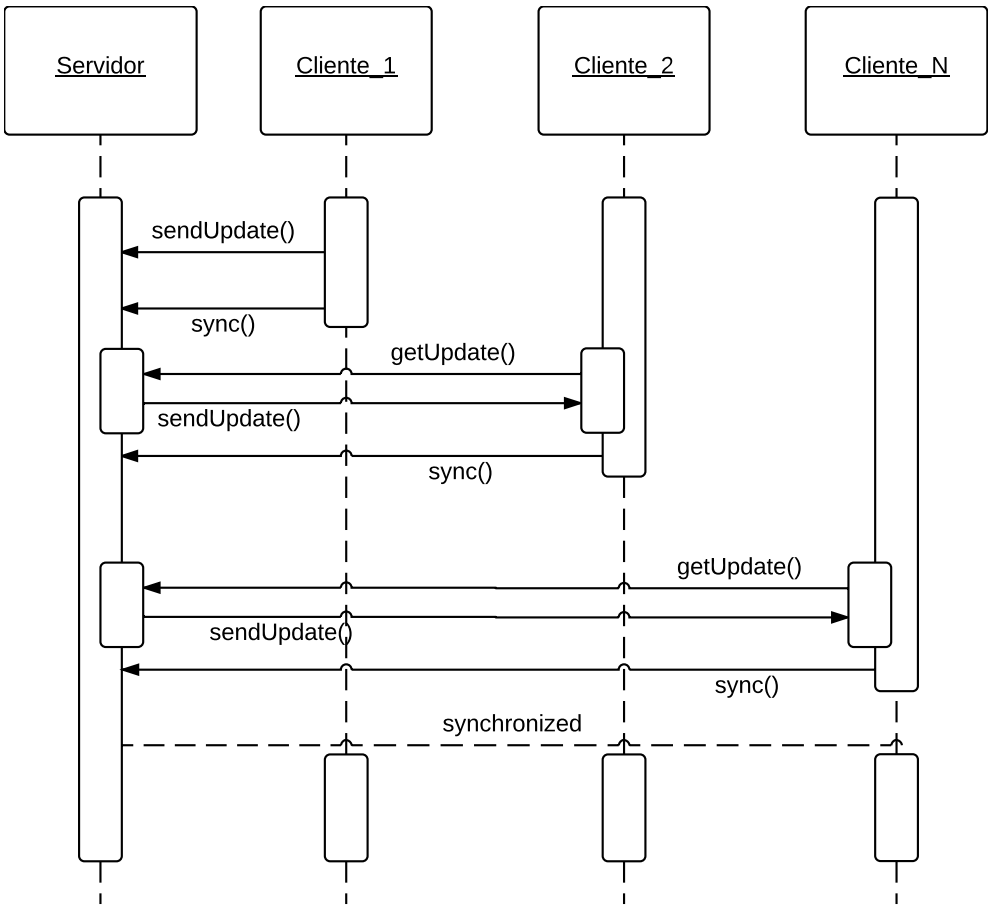


Figura 4.4: Diagrama de comunicação intra–aglomerado – sincronismo forte.

4.2.2 Comunicação inter–aglomerados gráficos

Como mencionado, o uso de barreiras de sincronismo forte não é uma solução viável no modelo de conexão inter–aglomerados, pelo menos não no caso de 3DCVEs. Por isso, os Servidores de Comunicação também funcionam como *peers*. Deste modo, os Servidores podem trocar informações entre si em um modelo de sincronismo fraco. A Figura 4.5 apresenta um diagrama de sequência que ilustra a troca de mensagens entre os Servidores de Comunicação em um modelo inter–aglomerados, onde o Peer_1 envia mensagens de atualização a os outros Peers. As mensagens de atualização enviadas pelo Peer_1 são geradas pelos processos clientes gerenciados por ele. Neste modelo, o tipo de mensagem trocada entre os *Peers* é síncrona, porém, sem o uso de barreiras. Desta forma, se por algum motivo, o Peer_2 atrasar o seu processamento, ele pode perder algum estado de atualização enviado pelo Peer_1, por exemplo. No entanto, dependendo do tipo de mensagem, este comportamento pode não ser prejudicial (vide Capítulo 5).

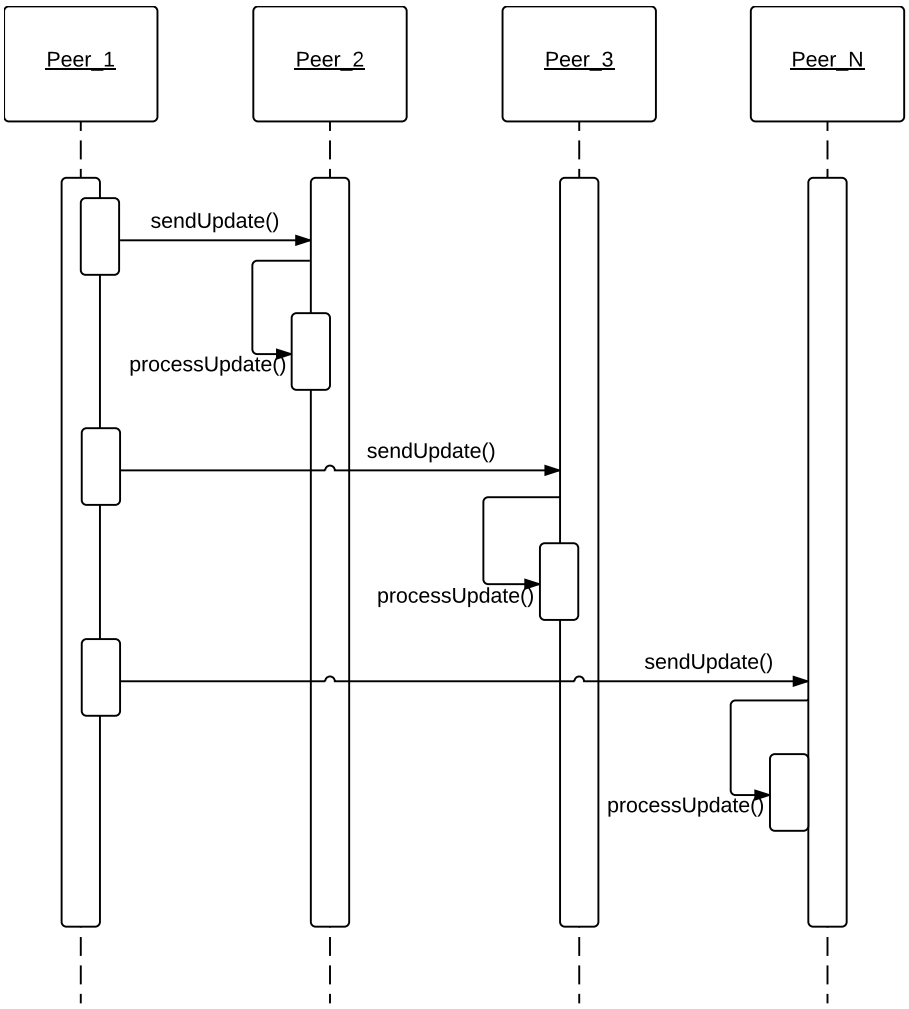


Figura 4.5: Diagrama de comunicação inter–aglomerados – sincronismo fraco.

4.3 Considerações finais

O ponto principal deste capítulo foi apresentar a arquitetura de forma geral, sem especificar em detalhes seu comportamento. Como apresentado, diversas soluções foram propostas para diferentes tipos de ambientes virtuais colaborativos, sendo essas baseadas em diferentes redes de dados, protocolos e algoritmos diversos, entretanto, não foi encontrada na literatura uma solução que integrasse tais soluções.

A libGlass foi utilizada como base no desenvolvimento da arquitetura por meio da criação de *plugins*, que tendem a satisfazer os requisitos necessários ao desenvolvimento de 3DCVEs em relação as adversidades ocasionadas pelas redes de dados. O modelo de conexão dinâmico–adaptável foi apresentado como um *plugin*. Este modelo realiza avaliações sobre as conexões inter–aglomerados A descrição referente ao desenvolvimento dos *plugins* é apresentada no Capítulo 5.

Capítulo 5

IMPLEMENTAÇÃO COMPORTAMENTAL DA ARQUITETURA DE COMUNICAÇÃO INTRA–AGLOMERADO E INTER–AGLOMERADOS VOLTADA A 3DCVES

Este capítulo apresenta a definição do conjunto de comportamentos desejáveis aos ambientes baseados em aglomerados gráficos, sejam eles remotos ou locais. Os componentes Instanciação, Protocolos, Compartilhamento, Eventos, Barreiras, Alíases, Áudio e vídeo, e Conexão dinâmica–adaptável, descritos no Capítulo 4, são apresentados em detalhes quanto as suas implementações.

A principal contribuição deste capítulo é a descrição comportamental do modelo de comunicação proposto para ambientes baseados em arquitetura inter–aglomerados sobre redes de dados baseadas em Internet.

A comunicação entre processos distribuídos, de forma geral, é feita por meio de interfaces de rede, portanto é necessário o uso de protocolos de comunicação e topologias de rede. Diversos são os protocolos e topologias presentes na literatura.

5.1 Instanciação

Existem diversos tipos de topologias de redes mencionadas na literatura, sejam elas físicas ou lógicas, sempre são discutidas em pesquisas relacionadas a redes de computadores. Assim, esta seção descreve o modelo Instanciação, que utiliza três diferentes protocolos de rede: TCP, TCP/UDP e SCTP. Com relação as topologias de redes, duas foram utilizadas: mestre–escravo

e ponto–a–ponto. Nas Subseções 5.1.1 e 5.1.2 são apresentados as duas topologias.

5.1.1 Mestre–escravo

Segundo Kurose (KUROSE; ROSS, 2012) a arquitetura mestre–escravo possui um nodo sempre presente, o mestre, responsável por gerenciaroutros nodos, os escravos. A Figura 5.1 apresenta o comportamento do módulo de instanciação mestre–escravo. A arquitetura mestre–escravo é utilizada para o modelo de comunicação intra–aglomerado.

Na Figura 5.1 o Mestre_1 é o primeiro processo a ser instanciado. O Mestre_1 cria uma lista vazia de escravos. Quando o Escravo_1 é instanciado, este conecta-se ao Mestre_1, que o adiciona a sua lista de escravos. O Escravo_2 é instanciado, sendo adicionado à lista de escravos do Mestre_1, da mesma forma que o Escravo_1.

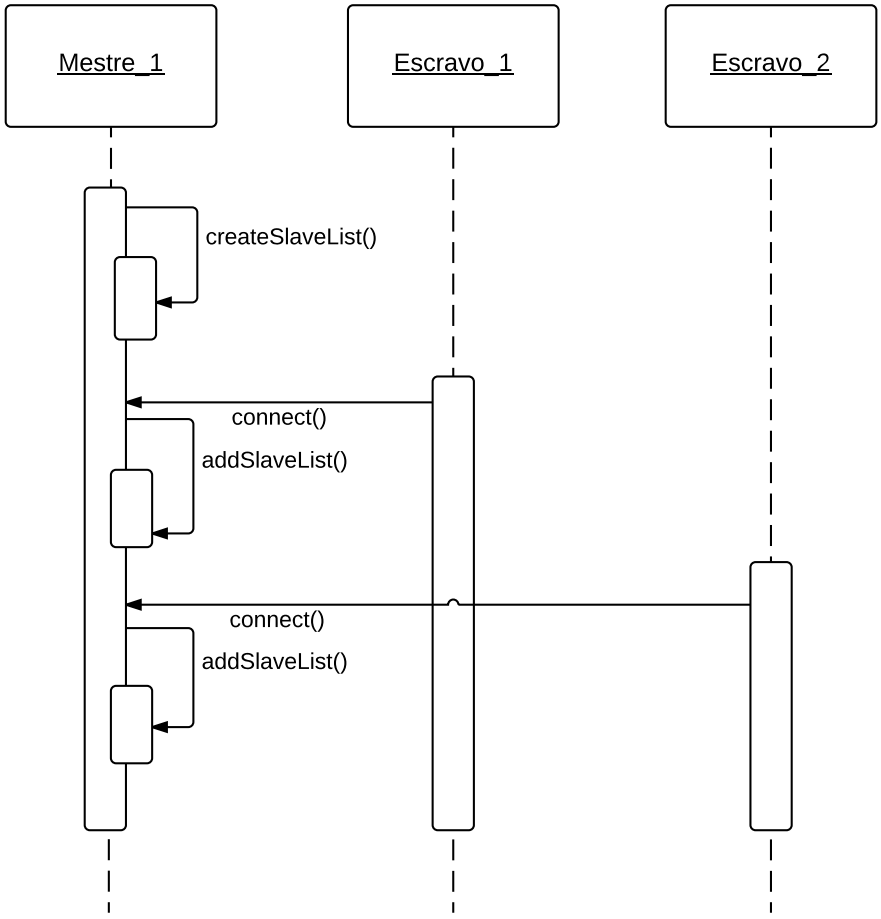


Figura 5.1: Modelo de instanciação para arquitetura mestre–escravo.

Quando um processo escravo é finalizado, este deve ser removido da lista de escravos do mestre. A Figura 5.2 apresenta este comportamento para os processos Mestre_1 e Escravo_1.

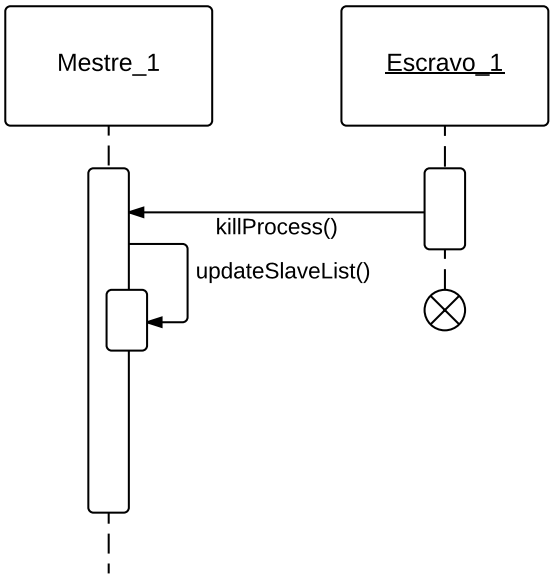


Figura 5.2: Modelo de instanciação para arquitetura mestre–escravo – finalização de escravos.

5.1.2 Ponto–a–ponto

A topologia ponto–a–ponto tem sido utilizada em larga escala em ambientes distribuídos, inclusive no desenvolvimento de 3DCVEs, sejam estes baseados em aglomerados gráficos ou não. Assim, esta subseção apresenta uma solução de topologia baseada na topologia ponto–a–ponto. Dois diferentes modelos de instanciação ponto–a–ponto foram desenvolvidos: *Peer* e *Remote Peer*.

O modelo *Peer* permite que processos mestres, que são responsáveis por um conjunto de processos escravos, possam ser tratados também como servidores e clientes, do ponto de vista de outros *peers*. Assim, mensagens antes difundidas apenas para processos escravos, passam a ser também difundidas a outros processos *peers*. A Figura 5.3 apresenta o comportamento esperado para o módulo de instanciação para a topologia ponto–a–ponto.

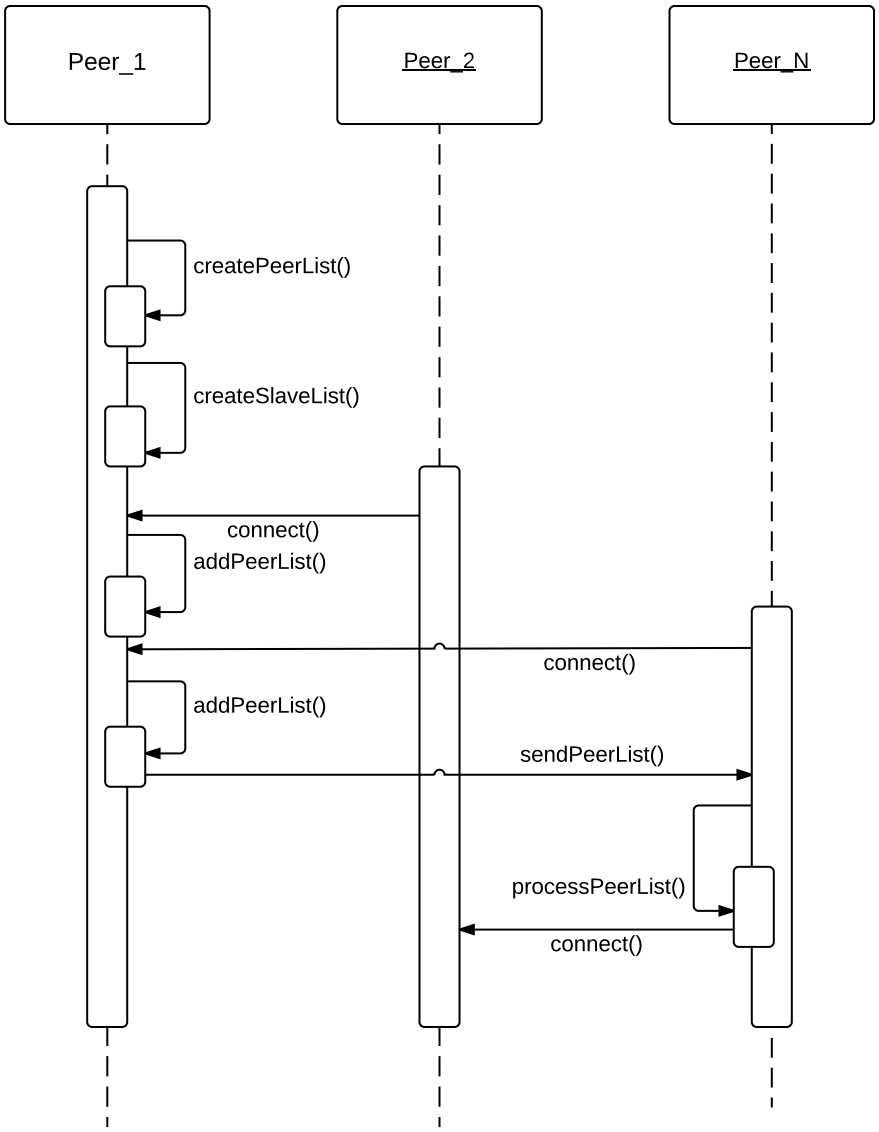


Figura 5.3: Modelo de instanciação para arquitetura ponto–a–ponto.

Na Figura 5.3 o Peer_1 é o primeiro processo a ser instanciado. Assim, ele cria uma lista vazia de *peers*. Quando o Peer_2 conecta-se ao Peer_1, este é adicionado à lista. O Peer_1 envia a lista que, neste momento, possui apenas o Peer_2. O Peer_2, assim que recebe a lista, a verifica, comparando os *peers* presentes na lista do Peer_1 com a sua lista (criada no momento de sua instanciação), que neste momento está vazia. Por último, o Peer_3 conecta-se ao Peer_2. O Peer_2 adiciona o Peer_3 a sua lista e a envia ao Peer_3. O Peer_3 efetua a comparação da lista recebida de Peer_2 com a sua própria lista, que está vazia neste momento. O Peer_3 irá tentar se conectar com todos os *peers* presentes na lista de Peer_2, excluindo apenas o Peer_2, que enviou a lista atualizada a ele. Assim, o Peer_3 conecta-se ao Peer_1, que envia a sua lista ao Peer_2. O Peer_2 efetua a comparação entre a sua lista e a lista enviada pelo Peer_1. Como o Peer_2 já possui conexão com o Peer_3, este apenas ignora a lista recebida do Peer_1. Dessa

forma, cada peer é responsável por manter sua própria lista atualizada.

No caso da finalização de um processo *peer*, este deverá ser excluído da lista de todos os outros processos. Isso é feito localmente por cada processo, sem a necessidade de difundir novamente a lista de *peers* entre todos os outros processos. Na Figura 5.4 é apresentado este comportamento. O Peer_3 é finalizado, assim o Peer_1 e Peer_2 excluem o Peer_3 de suas listas.

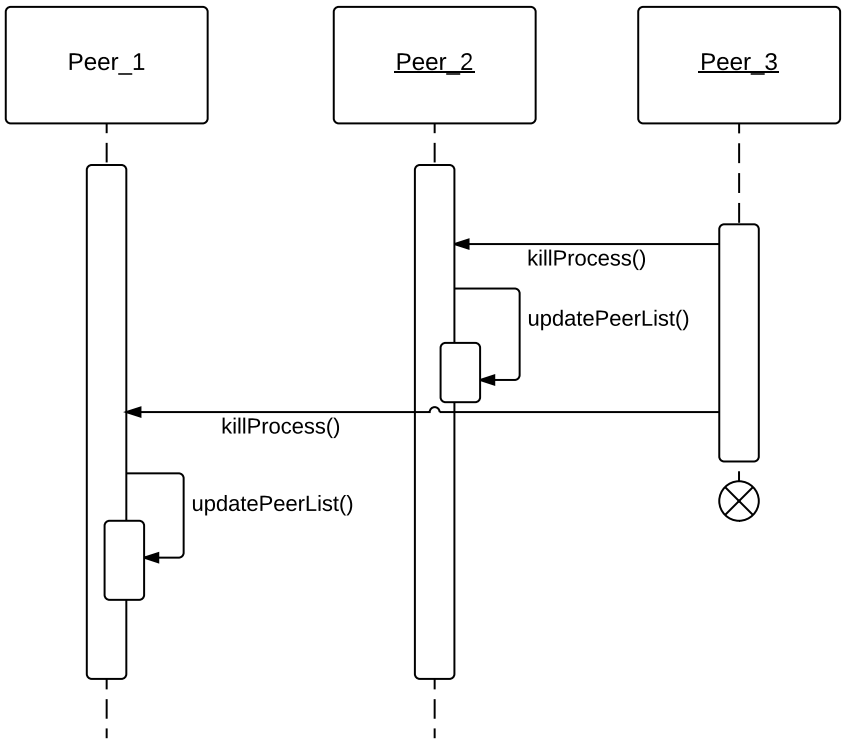


Figura 5.4: Modelo de instanciação de arquitetura ponto–a–ponto – finalização de *peers*.

Como mencionado, os processos *peers* possuem características tanto de servidor quanto de cliente. A Figura 5.5 apresenta o comportamento de um processo *peer* em relação a conexões entre outros *peers* e seus escravos. O processo Peer_1 é instanciado, gerando duas listas vazias: escravos e *peers*. Subseqüentemente, dois processos escravos são instanciados (Escravo_1 e Escravo_2), que conectam-se ao proceso Peer_1, sendo assim adicionados à lista de escravos. Finalmente, o processo Peer_2 é instanciado, conectando-se ao processo Peer_1. O Peer_2 é adicionado à lista de *peers* do Peer_1. O Peer_2 recebe a lista atualizada de *peers* do Peer_1, assim, como já apresentado (Figura 5.5), ele pode conectar-se a todos os outros *peers* que estejam presentes na lista do Peer_1. No entanto, em arquiteturas onde existam muitos processos *peers*, o modelo *Remote Peer* pode ser utilizado.

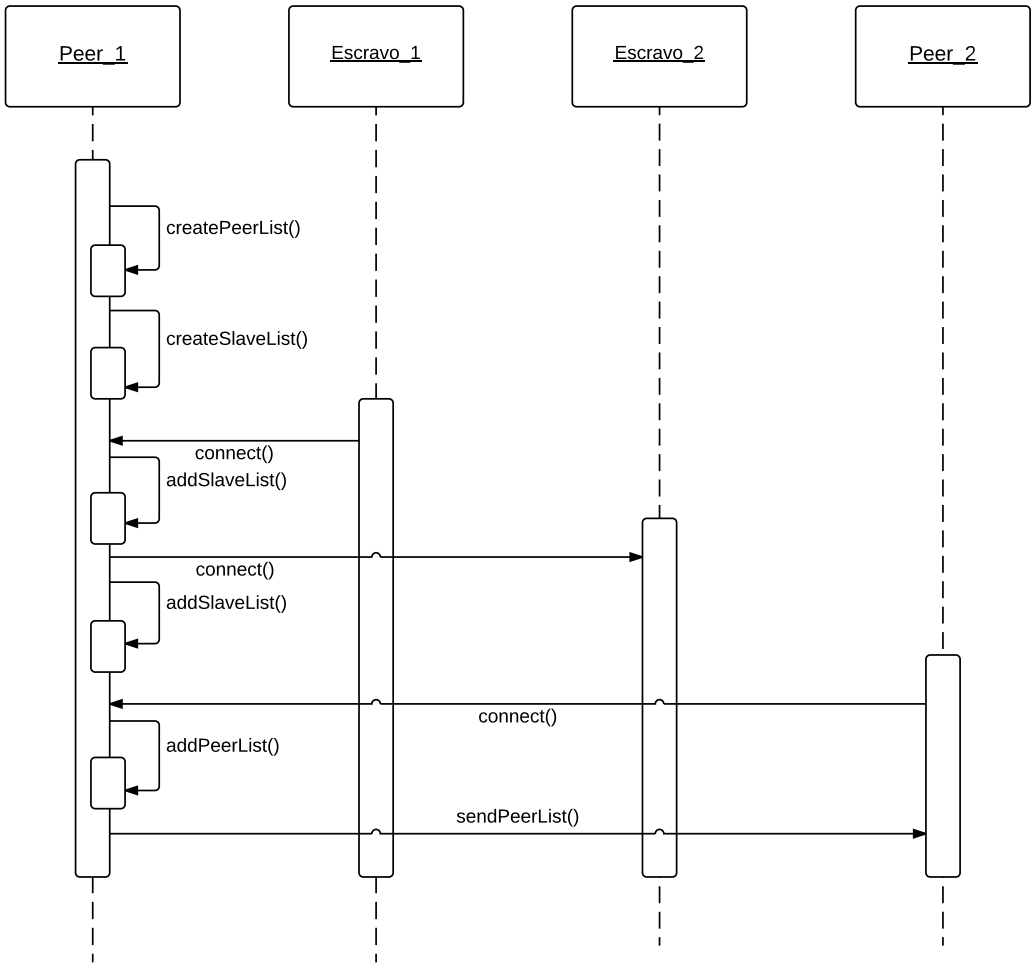


Figura 5.5: Modelo de instanciação de arquitetura ponto–a–ponto e escravos.

Em ambientes onde alta escalabilidade seja um requisito, a separação dos *peers* em menores grupos é uma possível abordagem. O *Remote Peer* é um modelo semelhante ao *Peer*, com a diferença de que este modelo gerencia conexões apenas entre *peers*, não difundindo mensagens para processos escravos. A Figura 5.6 apresenta o modelo *Remote Peer*. Da mesma forma que o *Peer*, o *Remote Peer* é utilizado no modelo de comunicação inter–aglomerados.

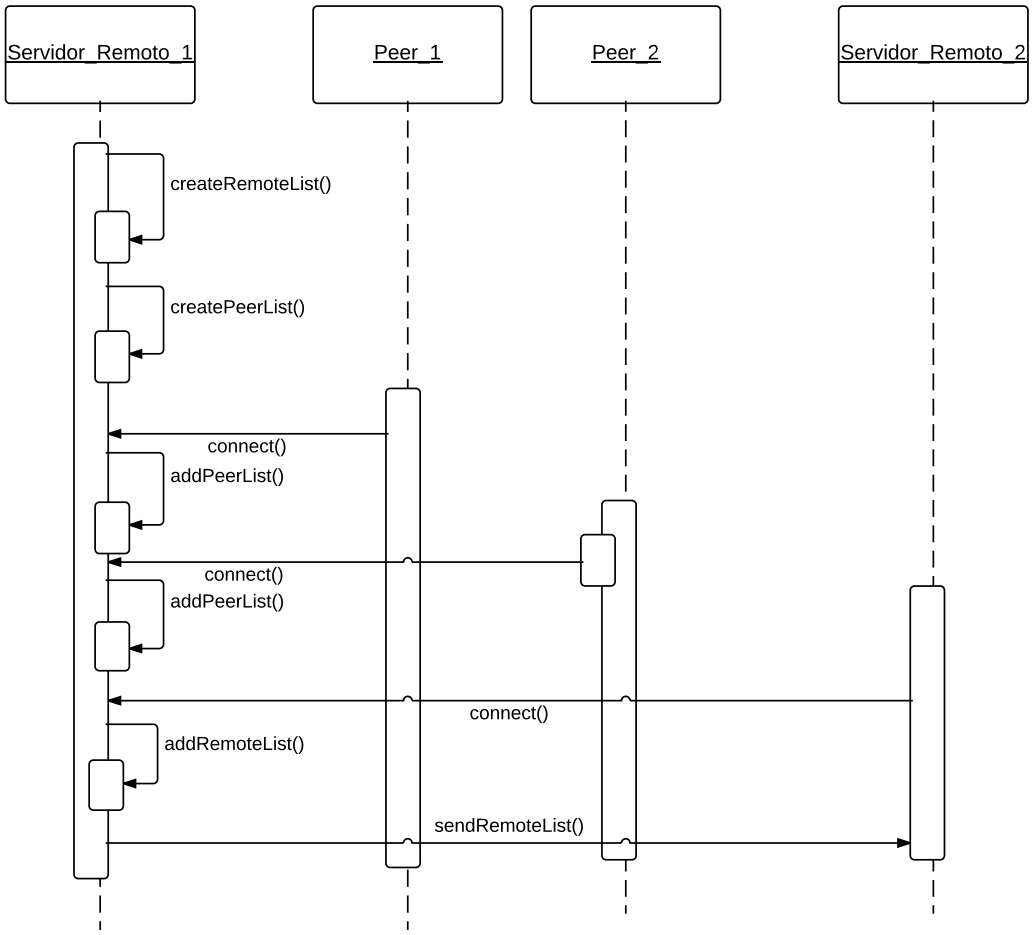


Figura 5.6: Modelo de instanciação para arquitetura ponto–a–ponto – *Remote Peer*.

Na Figura 5.6 é apresentado o comportamento esperado para o modelo *Remote Peer*. O processo ServerRemote_1 é instanciado, criando duas listas vazias: *remote peer list* e *peer list*. O processo ServerPeer_1 é instanciado, conectando-se ao ServerRemote_1. O ServerRemote_1 o adiciona a sua lista de *peers*. Da mesma forma, o processo *peer* ServerPeer_2 é instanciado e conecta-se ao ServerRemotePeer_1. O ServerRemotePeer_1 o adiciona a sua lista de *peers*. Por último, o ServerRemotePeer_2 é instanciado e conecta-se ao ServerRemotePeer_1. O ServerRemotePeer_1 o adiciona a sua lista de *peers* remotos e a envia, atualizada, para o ServerRemotePeer_2. Assim, se existirem outros processos *Remote Peers*, o ServerRemotePeer_2 pode conectar-se a eles, comportamento semelhante ao apresentado no modelo *Peer*.

O processo de finalização de um processo *Remote Peer* é tratado da mesma forma que o processo *Peer*. Por exemplo, se um processo *Remote Peer* desejar ser finalizado, ele deverá informar a todos os outros *Remote Peers* sobre a sua finalização.

5.2 Descrição dos protocolos de comunicação

Esta seção apresenta os protocolos de comunicação empregados no desenvolvimento do arcabouços de *software* utilizado como prova de conceito da arquitetura proposta nesta tese.

Os protocolos utilizados foram: TCP, UDP e SCTP. Estes foram escolhidos por serem altamente difundidos no meio acadêmico e comercial. Três diferentes soluções foram desenvolvidas. A primeira solução utiliza apenas o protocolo TCP na comunicação entre os processos distribuídos. Desta forma, toda troca de mensagens entre os processos é realizada utilizando um modelo de comunicação confiável, visto que essa é uma das principais características providas pelo protocolo TCP. A Figura 5.7 apresenta o modelo de troca mensagens desenvolvido utilizando o protocolo TCP, que segue o modelo padrão do protocolo, o *Three-way Handshake*.

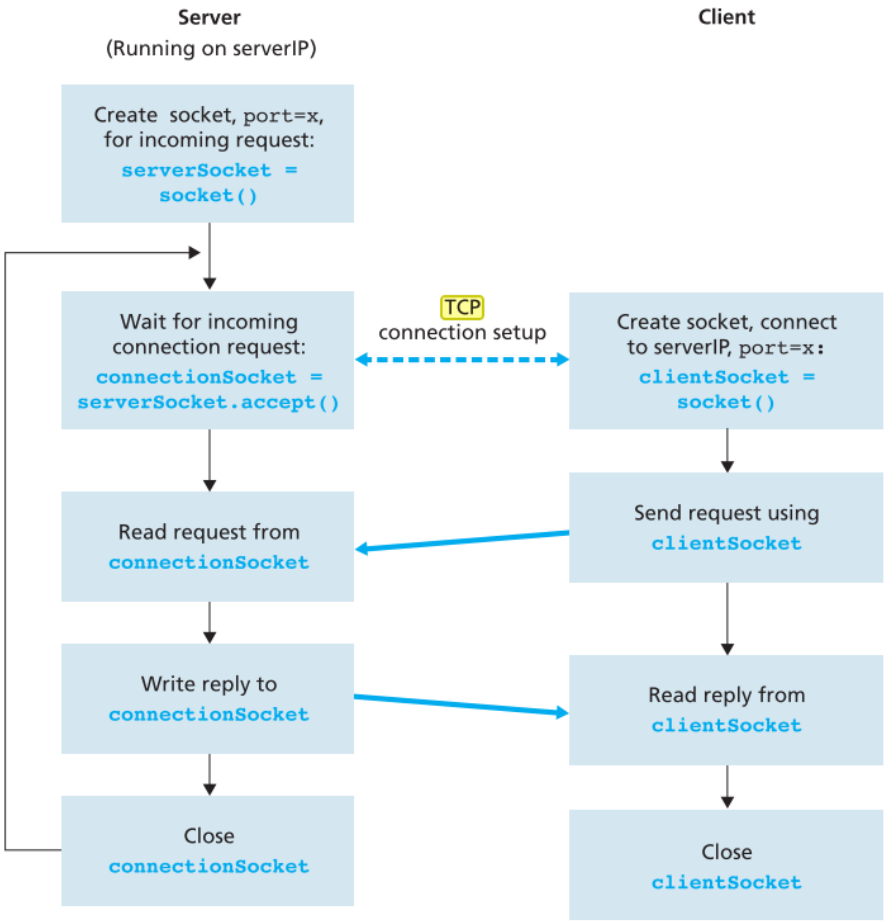


Figura 5.7: Aplicação cliente–servidor – protocolo TCP.
(KUROSE; ROSS, 2012)

Uma solução combinando os protocolos TCP e UDP também foi desenvolvida, que permite que mensagens sejam trocadas de duas formas distintas: confiável e não-confiável. Desta

forma, mensagens que necessitem serem enviadas de modo confiável utilizam o modelo de comunicação provido pelo protocolo TCP. Por outro lado, as mensagens que não necessitem serem enviadas em um modelo de comunicação confiável, podem fazer uso do protocolo UDP. Neste modelo, duas conexões são mantidas entre os pares de processos: uma conexão utilizando o protocolo TCP (Figura 5.7) e uma conexão utilizando o protocolo UDP (Figura 5.8).

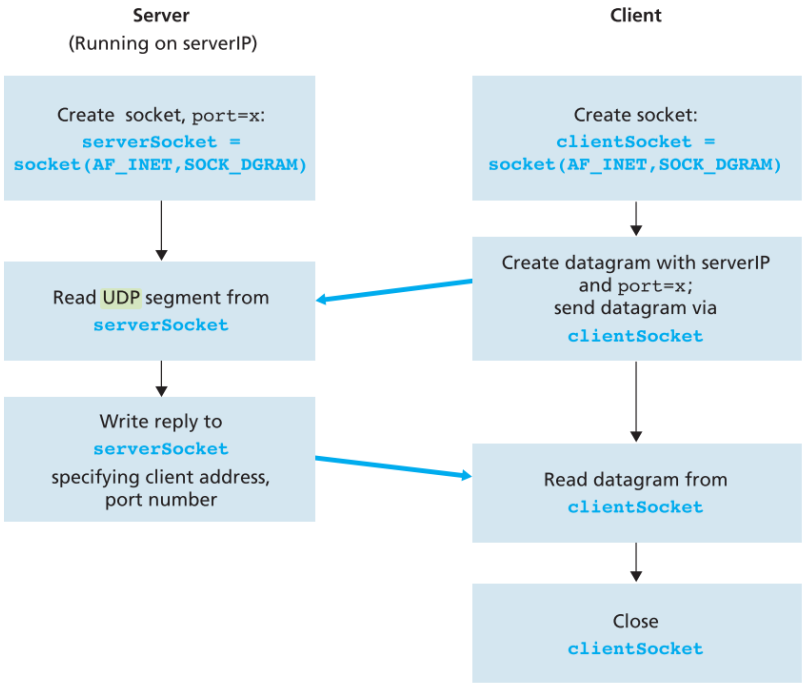


Figura 5.8: Aplicação cliente–servidor – protocolo UDP.
(KUROSE; ROSS, 2012)

Uma terceira solução foi abordada por meio do uso do protocolo de comunicação SCTP. Como já mencionado, o SCTP provê uma combinação das melhores características existentes nos protocolos TCP e UDP. A Figura 5.9 apresenta o modelo de *four–way handshake* utilizado pelo SCTP.

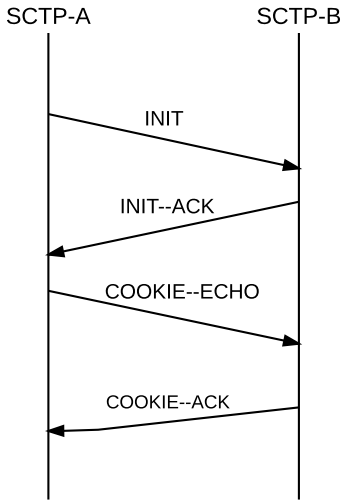


Figura 5.9: Aplicação *host–to–host* exemplificando o *four–way handshake* – protocolo SCTP.

Cada protocolo possui algumas distinções em relação aos seus cabeçalho, entretanto, um cabeçalho personalizado foi utilizado, contendo: tamanho do cabeçalho (2 bytes), tamanho da carga útil (4 bytes), ID do emissor (4 bytes), ID do *plugin* (2 bytes) e os IDs dos receptores (targetIds.size()+1)*sizeof(nodeId)). A Figura 5.10 apresenta o modelo de cabeçalho personalizado implementado pela libGlass.

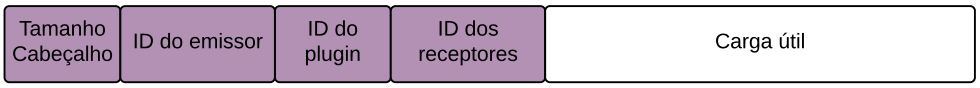


Figura 5.10: Cabeçalho padrão libGlass.

5.3 Barreiras

O componente Barreira é extremamente importante em aplicações de tempo real que necessitem de alto grau de sincronismo. Sua principal funcionalidade é prover sincronização de estados aos processos distribuídos em aglomerados gráficos. Seu funcionamento pode ser descrito em duas etapas: na primeira etapa, os processos devem ficar em um determinado estado de espera até que todos os processos alcancem o mesmo ponto, para que o processamento possa ser continuado. Assim, um processo servidor/mestre deve ser responsável por receber as mensagens de sincronização dos nodos clientes/escravos; na segunda etapa, o processo servidor/mestre deve enviar mensagens de sincronização a seus processos clientes/escravos, para que estes possam continuar suas execuções.

A Figura 5.11 apresenta o modelo de sincronização para arquiteturas intra–aglomerado.

Neste modelo, o processo Escravo_1 envia uma mensagem de sincronização ao Mestre_1 e fica esperando uma mensagem de volta. O mesmo acontece com o Escravo_2. O Escravo_N envia uma mensagem de sincronização ao Mestre_1, que difunde uma mensagem de sincronização a todos os nodos escravos, liberando-os para continuar com os seus fluxos de execução normalmente, até que outro ponto de sincronismo seja alcançado.

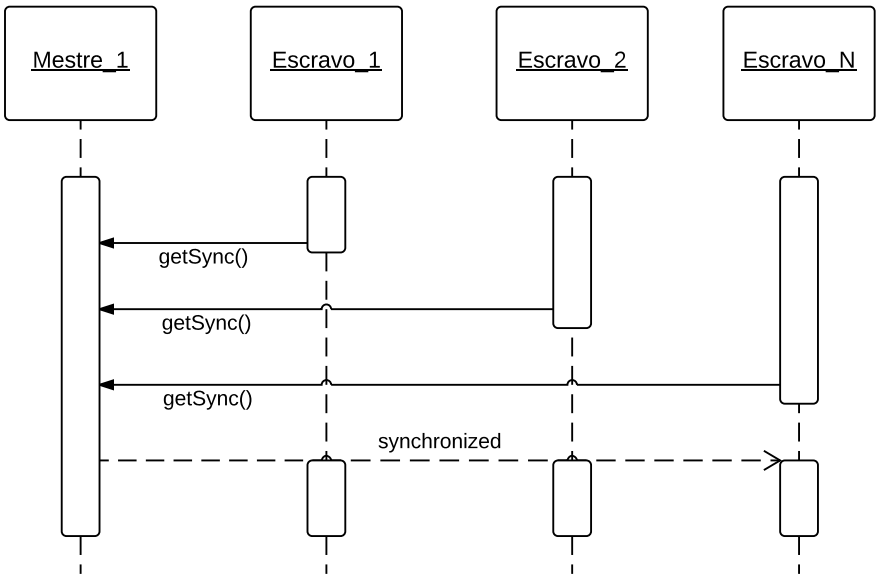


Figura 5.11: Modelo de comunicação do componente Barreira intra–aglomerado.

A Figura 5.12 descreve o comportamento do modelo de sincronização para arquiteturas inter–aglomerados. Os processos escravos, Escravo_1 e Escravo_2, enviam mensagens de sincronização ao seu mestre, neste caso, o Peer_1, que é tratado como mestre em relação aos seus escravos, mas também é tratado como sendo um *peer* em relação a outros *peers*. O Peer_1 envia uma mensagem de sincronização a todos os outros *peers*, avisando-os que os seus escravos já estão prontos para continuar seus fluxos de execução. O Escravo_N envia uma mensagem de sincronização ao Peer_N, que envia uma mensagem de sincronização ao Peer_1, avisando que todos os seus escravos também estão prontos. Neste momento, quando todos os *peers* já trocaram mensagens de sincronização entre si, todos os escravos são liberados para continuarem com seus fluxos de execução normalmente.

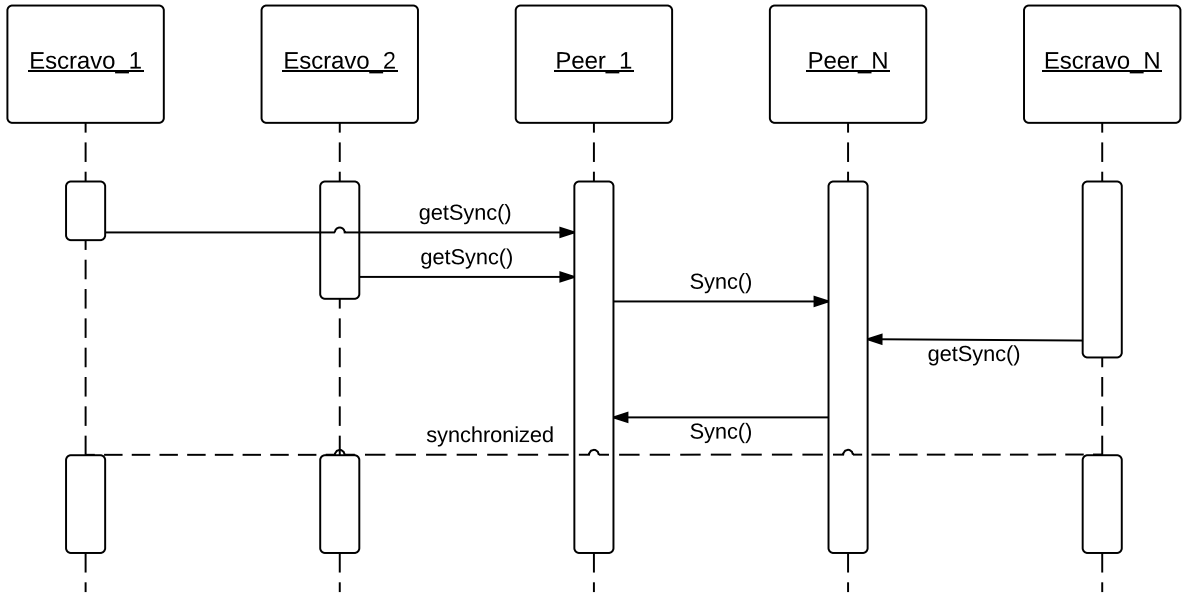


Figura 5.12: Modelo de comunicação do componente Barreira inter–aglomerados.

Em arquiteturas compostas por *peers* remotos, o funcionamento é semelhante ao apresentado na Figura 5.12, contudo, uma etapa extra é utilizada. O Peer_1 envia uma mensagem de sincronização ao Remote_Peer_1, que envia uma mensagem de sincronização ao Remote_Peer_2 e Remote_Peer_N, avisando que seus *peers* estão esperando uma mensagem de sincronização. O Peer_2 envia uma mensagem de sincronização ao Remote_Peer_2, que envia uma mensagem de sincronização ao Remote_Peer_1 e Remote_Peer_N, avisando que seus *peers* estão esperando uma mensagem de sincronização para que eles possam continuar seus fluxos de execução. O mesmo acontece com o Remote_Peer_N, que após receber as mensagens de sincronização dos seus *peers*, envia mensagens de sincronização aos outros *peers* remotos. Neste momento, os *peers* remotos enviam mensagens de sincronização a todos os seus *peers*, que as encaminham aos seus escravos. A Figura 5.13 apresenta o comportamento descrito.

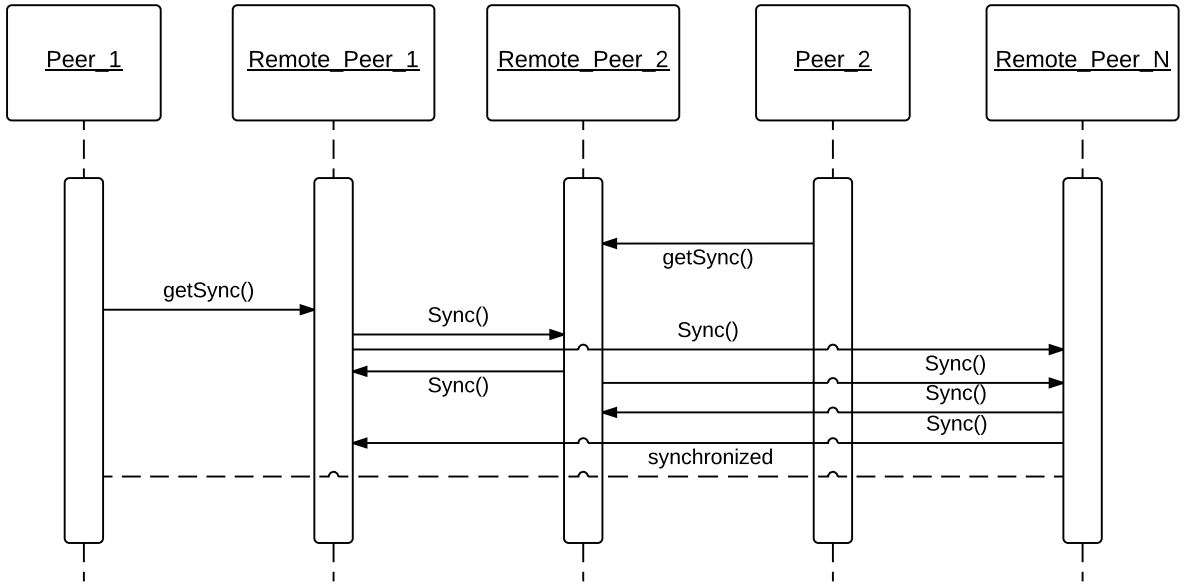


Figura 5.13: Modelo de comunicação do componente Barreira inter–aglomerados – *Remote Peer*.

Na Figura 5.14 é apresentado um outro modelo de barreira voltada à arquitetura *Remote Peer*, adaptável a altas latências. Este tipo de barreira compensa a latência média de comunicação entre os processos *Remote Peers* e seus *peers*, utilizando como base a maior latência presente nas conexões entre todos os *Remote Peers* presentes. Para ambientes em que a sincronização seja de extrema importância, não importando o possível atraso gerado, pode-se utilizar um modelo de barreira no qual a mensagem de desbloqueio emitida pelo *Remote Peer* (Remote_Peer_1) também informe o maior tempo de latência em relação a outros *Remote Peers*, neste caso o processo Remote_Peer_N. Isto é, um processo *peer*, que mesmo depois de receber a mensagem de sincronismo do processo *Remote Peer*, ainda espere a quantidade de tempo definida, para só então dar sequência ao seu fluxo de execução. Tem-se então a função *tf = tm - te*. O tempo final (tf) é definido pelo tempo máximo (tm) menos o tempo de envio (te).

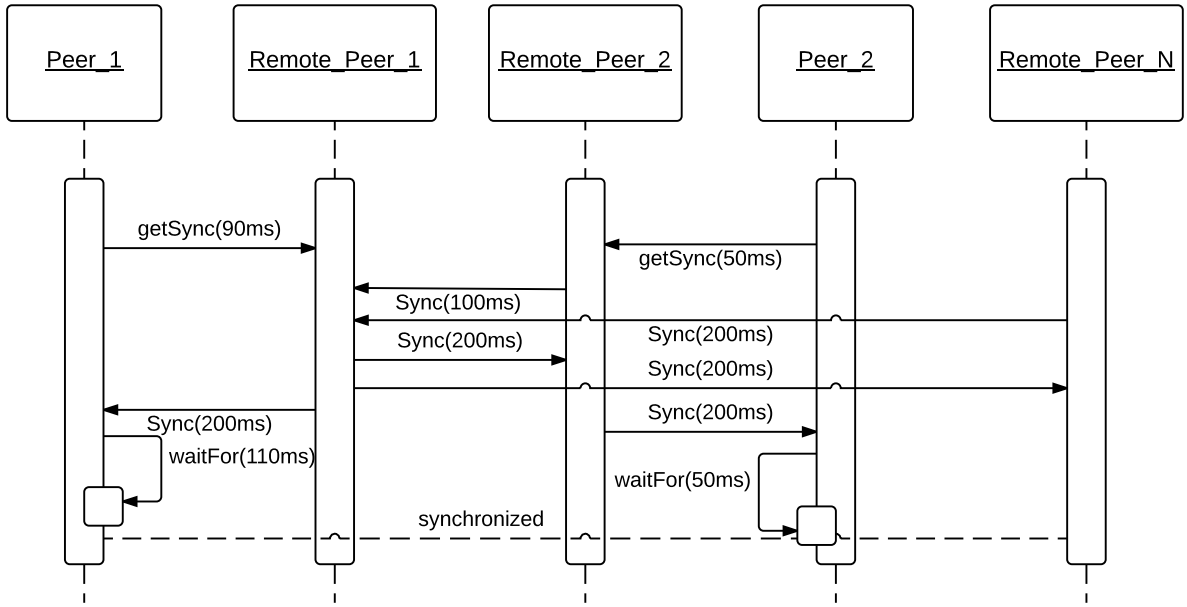


Figura 5.14: Modelo de comunicação do componente Barreira inter–aglomerados com compensação de latência média – *Remote Peer*.

5.4 Variáveis síncronas

O componente Compartilhamento (*Shared*) possui características distintas quanto a forma de propagação de mensagens. As variáveis Compartilhamento podem ser definidas como sendo locais ou globais, restringindo assim o seu escopo (intra–aglomerado ou inter–aglomerados). A Figura 5.15 apresenta a definição comportamental para o componente Compartilhamento em um ambiente intra–aglomerado.

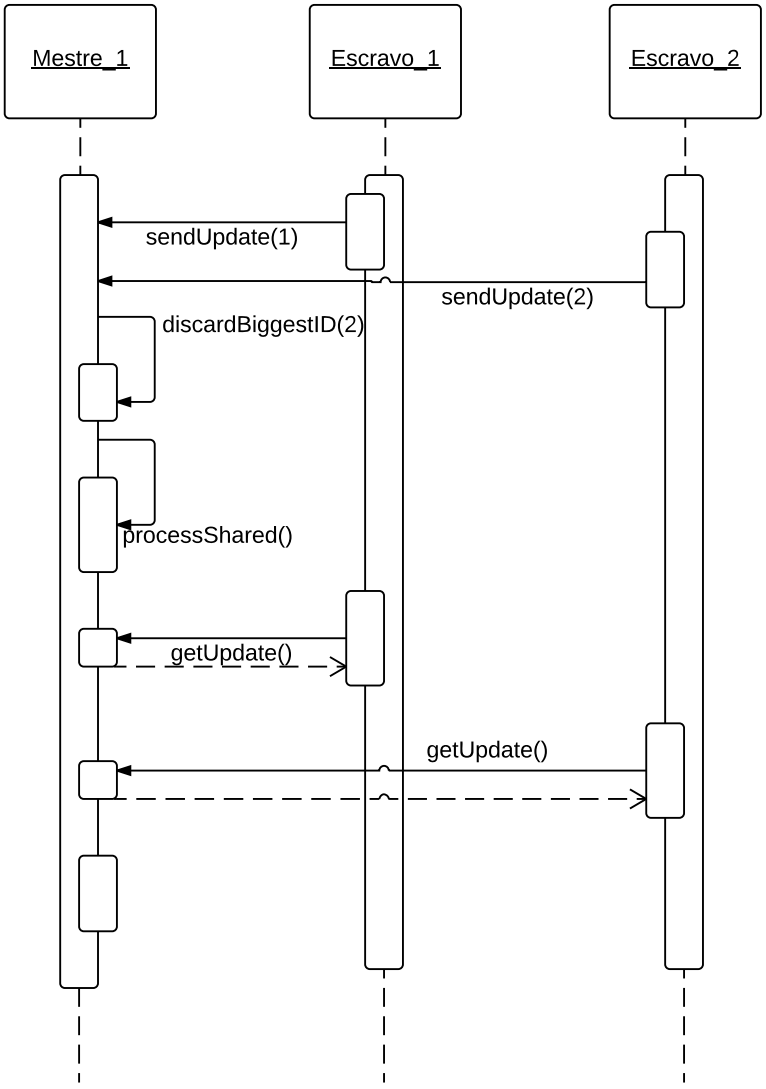


Figura 5.15: Modelo de comunicação do componente Compartilhamento intra–aglomerado.

De acordo com a Figura 5.15, sempre que o Escravo_1 requisita o valor atualizado de uma variável (SH1), este realiza uma chamada *getUpdate*. Dessa forma, o Mestre_1 só envia o valor atualizado ao Escravo_1 quando este o solicita.

O componente Compartilhamento, em arquiteturas intra–aglomerado, só envia dados atualizados aos seus processos quando estes realizam alguma solicitação (*getUpdate*). No caso de arquiteturas inter–aglomerados, os processos *peers* recebem o valor atualizado assim que a tabela interna de um processo *peer* é atualizada. Na Figura 5.16, por exemplo, quando a varíavel SH1 é atualizada pelo Escravo_1, o Peer_1, que é responsável por pelo Escravo_1, propaga a mensagem para todos os outros processos *peers* que tenham interesse na variável SH1. Assim, quando o Escravo_2 solicita o valor da variável SH1 ao Peer_2, este envia o valor da SH1 atualizado. Essa abordagem de envio imediato aos processos *peers* foi desenvolvida com o intuito

de diminuir o atraso no processo de atualização de variáveis Compartilhamento entre os processos do tipo *Peer* e *Remote Peer*, pois se a atualização fosse realizada apenas quando algum processo escravo realizasse a solicitação, o atraso seria prejudicial, visto que o processo *peer* responsável pelo escravo requerente necessitaria requisitar o valor ao processo *peer* que possua o valor atualizado.

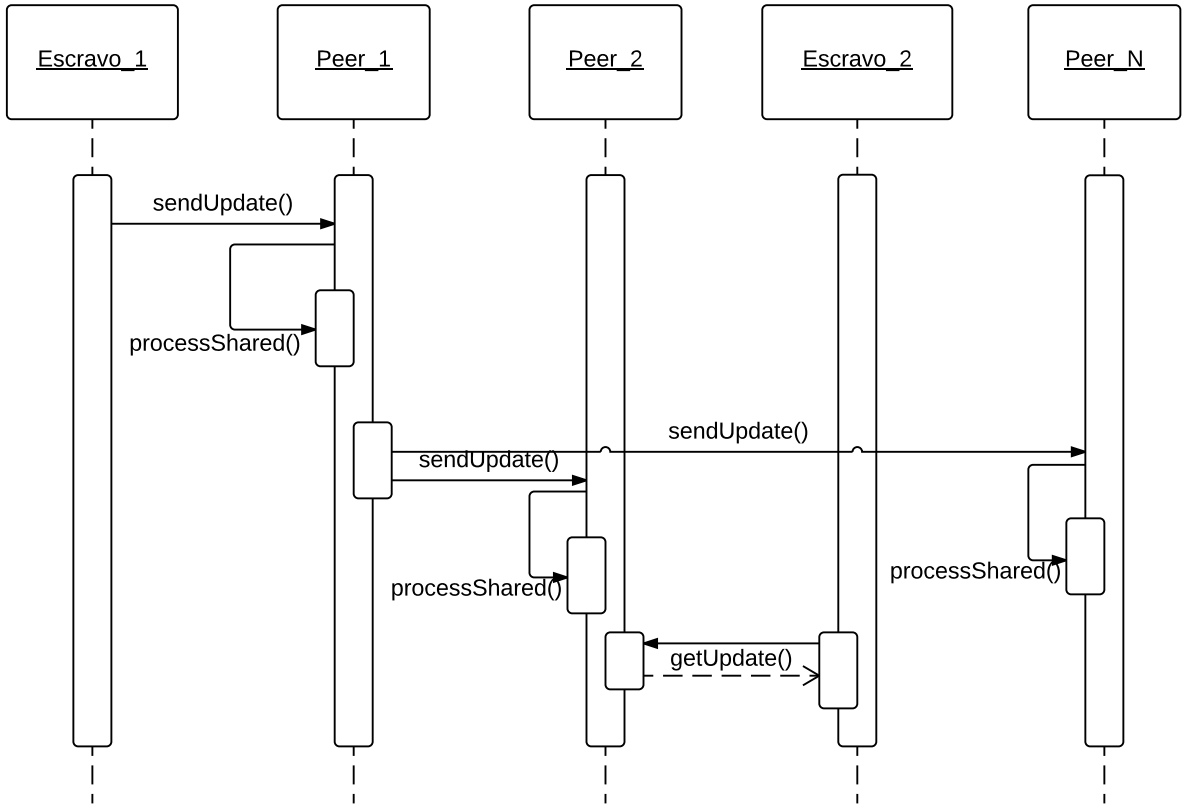


Figura 5.16: Modelo de comunicação do componente Compartilhamento inter–aglomerados.

5.5 Eventos assíncronos

O componente Evento (*Event*) permite troca de mensagens entre processos de forma assíncrona. Da mesma forma que o componente Compartilhamento, os eventos podem ser definidos como sendo locais ou globais. A Figura 5.17 apresenta a definição comportamental do componente Evento para a arquitetura intra–aglomerado.

De acordo com a Figura 5.17, o processo Mestre_1 recebe um evento do Escravo_1 e o coloca no final de sua fila. O processo Escravo_2 também envia evento ao Mestre_1, que o coloca no final de sua fila. A fila de mensagens é processada pelo Escravo_1, e então difundida aos outros processos (Escravo_2, Escravo_N), que recebem e processam os eventos. O modo de envio é realizado de modo assíncrono, desta forma, toda mensagem recebida pelos escravos

é consumida apenas quando estes desejarem. Os eventos ficam enfileirados em suas respectivas filas, quando estes não são consumidos assim que recebidos.

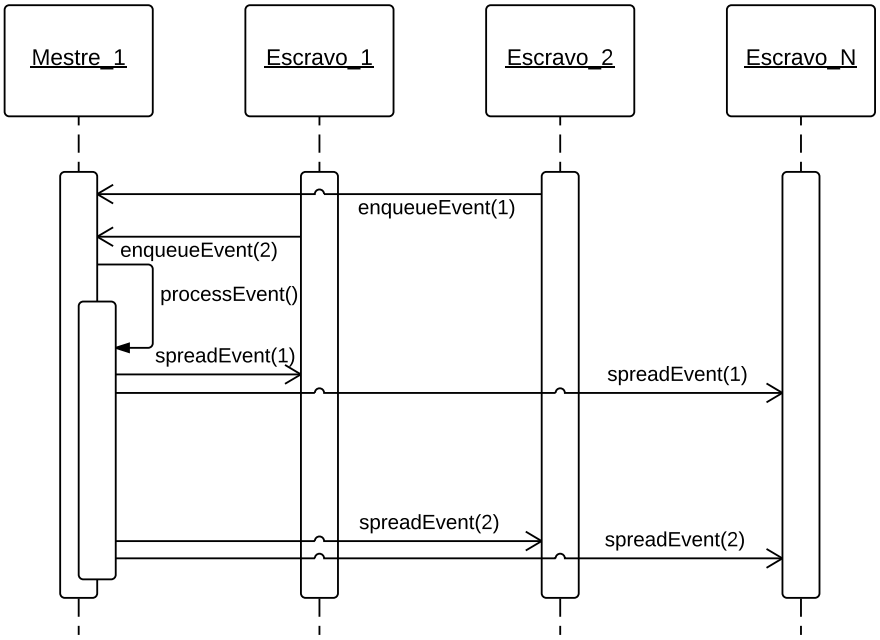


Figura 5.17: Modelo de comunicação do componente Evento intra–aglomerado.

O comportamento do componente Evento para ambientes inter–aglomerados possui algumas distinções em relação ao modelo intra–aglomerado. Por exemplo, em um ambiente composto por processos *peers* e escravos, o processo Escravo_1 envia um evento ao seu processo mestre (Peer_1), que também é um *peer* em relação a outros processos *peers*. O Peer_1 recebe e processa o evento enviado pelo Escravo_1, encaminhando este evento a todos os outros processos escravos e *peers*. Primeiramente, o evento deve ser enviado a todos os processos escravos que são gerenciados pelo Peer_1. Posteriorente, o processo Peer_1 encaminha o evento aos outros processos *peers* (Peer_2 e Peer_N). Os processos *peers*, Peer_2 e Peer_N, encaminham o evento recebido aos seus processos escravos. Por se tratar de evento assíncrono, este é processado e enfileirado, sendo consumido de acordo com a demanda da aplicação executada nos escravos. A Figura 5.18 apresenta o exemplo descrito.

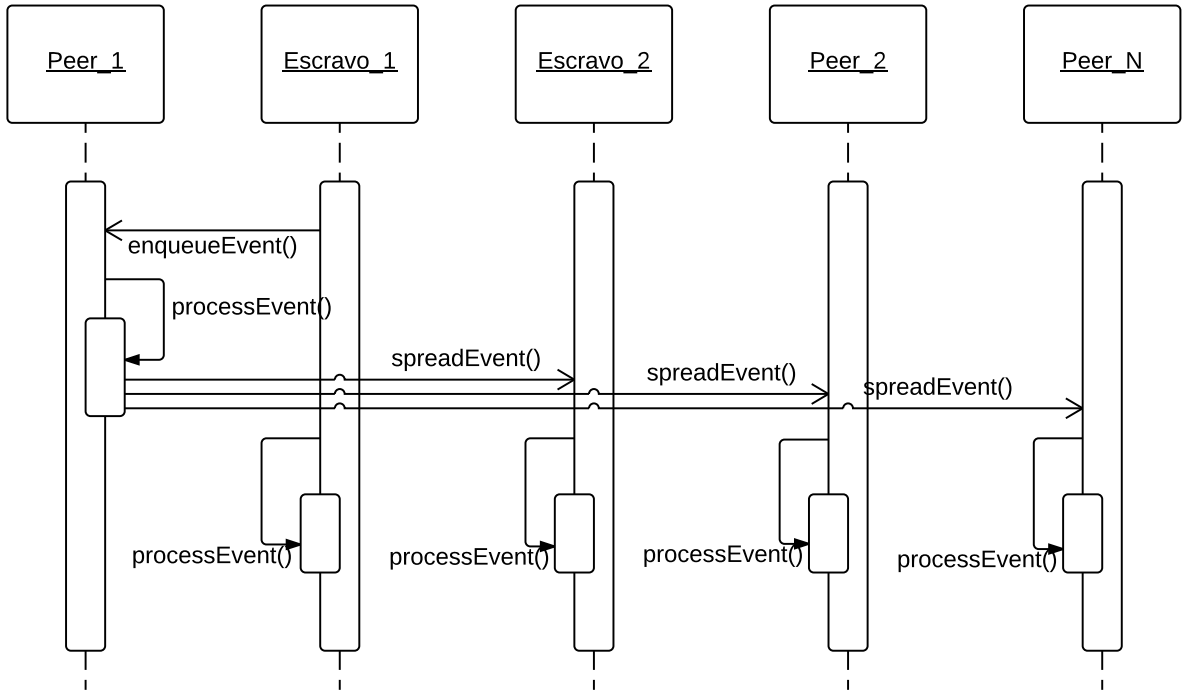


Figura 5.18: Modelo de comunicação do componente Evento inter–aglomerados.

Em um ambiente inter–aglomerados, composto por processos *peers* remotos, *peers* e escravos, a execução é semelhante ao apresentado na Figura 5.18. No entanto, possui uma fase extra na propagação dos eventos entre os processos *peers*. Em um modelo de arquitetura que possua *peers* remotos, os *peers* não trocam mensagens entre si. Por exemplo, o processo Peer_1 recebe um evento de um dos seus escravos e o envia ao seu processo Peer_Remoto_1, que o propaga aos *peers* remotos e *peers* a que ele esteja conectado (Peer_Remoto_N), que propagam a seus *peers* (Peer_N). Portanto, sempre que um *peer* remoto recebe um evento, este deve enviá-lo, primeiro, aos outros *peers* remotos. A Figura 5.19 apresenta o comportamento descrito.

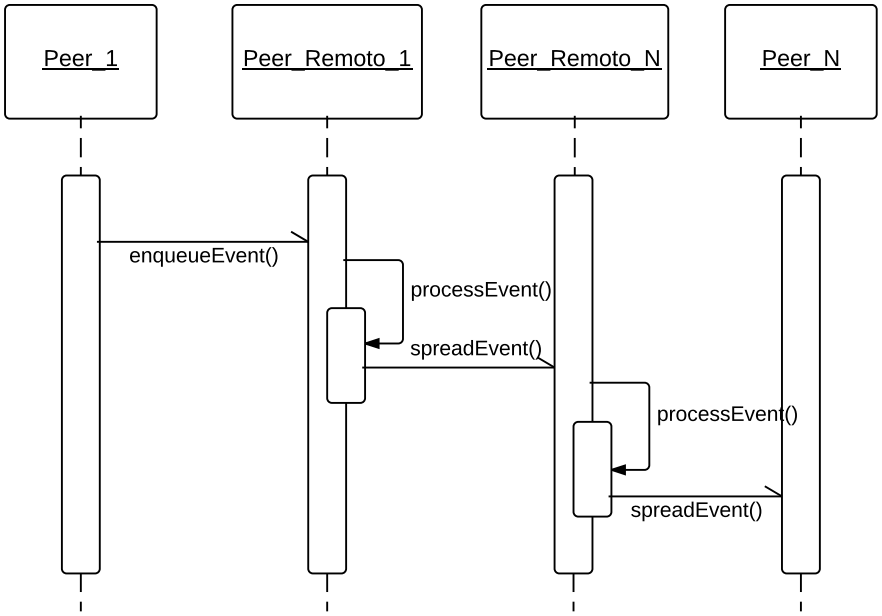


Figura 5.19: Modelo de comunicação do componente Evento inter–aglomerados – *Remote Peer*.

Visando evitar problemas de gargalo relacionados à vazão de pacotes enviados por um determinado processo, um modelo de envio que permita o agrupamento de múltiplos eventos utilizando um mesmo pacote de dados foi desenvolvido. Por se tratar de um modelo de envio assíncrono, este tipo de comportamento não é prejudicial à execução de aplicações, pelo contrário, pois permite que a quantidade de pacotes de dados trafegados entre os processos seja reduzida, transmitindo a mesma quantidade de dados (quando as mensagens forem menor que 1500 bytes). Este agrupamento é feito internamente pela libGlass.

5.6 Alíases

Esta seção descreve o componente Alíases, responsável por nomear os processos, de modo a facilitar sua identificação. Todo processo possui um número de identificação (ID) único, que é definido de forma dinâmica sempre que o processo é instanciado (este ID não é o referente ao processo criado pelo sistema operacional, mas sim o criado pela libGlass). Os IDs são compostos por uma longa cadeia de números, sendo, portanto, difíceis de serem lembrados. Por isso, um sistema de nomeação pode auxiliar na resolução deste problema. Um modelo de lista de alíases foi desenvolvido. A lista foi implementada por meio de um *map* (ID, alíases).

O componente Alíases permite que apelidos sejam atribuídos a cada um dos processos, facilitando assim a identificação pelos desenvolvedores. Por exemplo, um processo mestre é instanciado, de modo que ele mesmo atribui um ID para si; quando um processo escravo é

instanciado, conectando-se ao processo mestre, este recebe um ID criado e atribuído a ele pelo o seu processo mestre. A Figura 5.20 apresenta um exemplo de atribuição de apelidos aos processos.

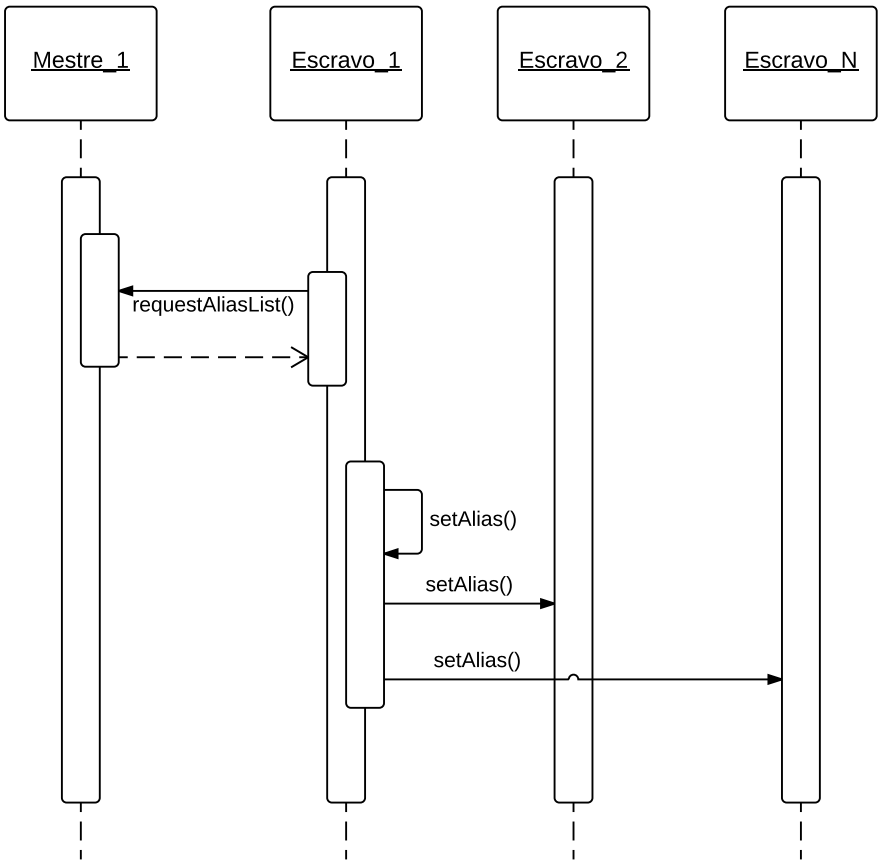


Figura 5.20: Modelo de comunicação do componente Alíases intra–aglomerado.

No modelo de comunicação inter–aglomerados o funcionamento para a atribuição de alíases é semelhante ao desenvolvido no modelo intra–aglomerado, diferindo apenas em uma etapa extra em sua execução. O primeiro processo *peer* (Peer_1) instanciado é responsável por gerenciar a lista de alíases. Assim, sempre que um novo *peer* for instanciado, este deve perguntar a este *peer* seu alíases. O comportamento descrito é apresentado na Figura 5.21. O mesmo se aplica aos *Remote Peers*. No entanto, cada *Remote Peer* é responsável também pela lista de alíases dos seus *peers*.

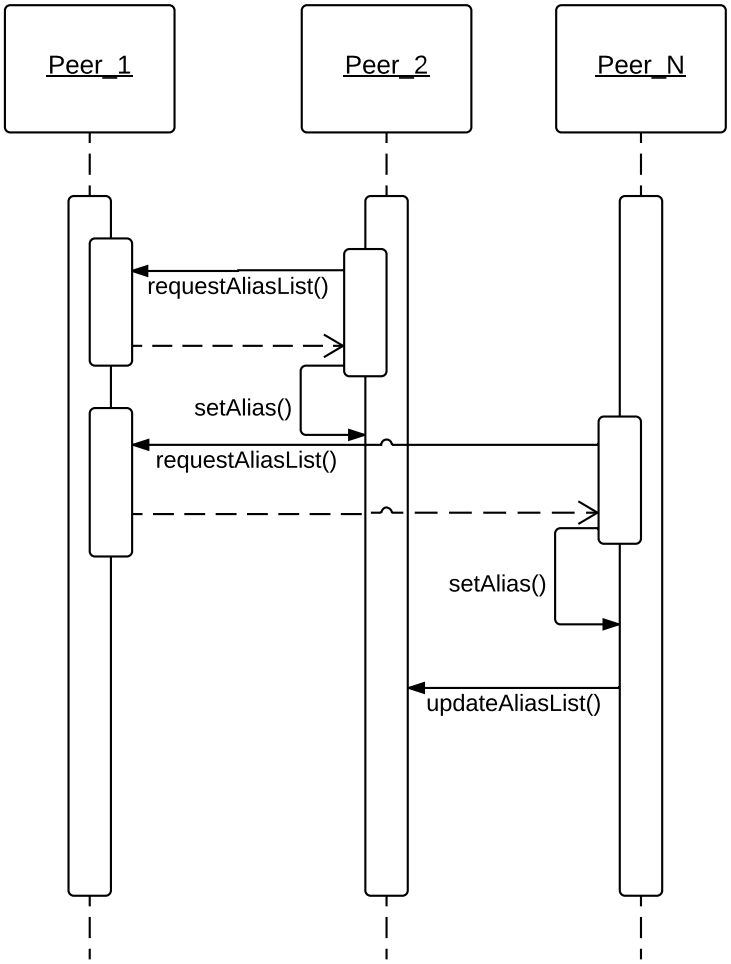


Figura 5.21: Modelo de comunicação do componente Alíases inter–aglomerados.

Sempre que um processo receber um novo alíases, todos os outros processos *peers* devem ser informados. Desta forma, o processo Peer_N envia uma mensagem ao Peer_2, informando o seu alíases. O alíases referente ao ID do processo Peer_N já é conhecido pelo Peer_1, portanto, não é necessário a atualização da sua lista de alíases.

5.7 Áudio e vídeo

O módulo de áudio e vídeo foi estendido a partir do modelo eventos assíncronos. Assim, os eventos são enfileirados no emissor e enviados ao receptor assim que possível. Os eventos recebidos pelo receptor são enfileirados, sendo consumidos assim que necessário. Por meio do protocolo SCTP, foi possível utilizar um modelo de comunicação multicanais, transmitindo assim, o vídeo e o áudio utilizando canais distintos. O principal benefício neste tipo de comunicação é a possibilidade de, por meio de apenas uma associação, realizar o envio de mais de um canal de dados, visto que se o TCP fosse utilizado, por exemplo, seriam necessárias duas

sessões para cada par de processos.

5.8 Conexão dinâmica–adaptável

Em todos os momentos o ser humano está tomando decisões. Normalmente, os modelos matemáticos que atingem a realidade são construídos para ajudar a tomar decisões. A fim de melhorar os modelos e, em seguida, o nível de segurança na tomada de decisões, é comum a utilização de ferramentas com potencial de modelagem que incluem incertezas envolvidas no processo modelado, como a Teoria dos Conjuntos Difusos.

O principal objetivo da Teoria dos Conjuntos Difusos é fornecer uma ferramenta matemática para o tratamento de informações de imprecisão ou de caráter impreciso. Geralmente classificados como um exemplo especial, da assim chamada "modelagem qualitativa", a modelagem difusa tem sido considerada como um dos problemas básicos na pesquisa de sistemas difusos (SUGENO; YASUKAWA, 1993).

Com o intuito de realizar tomadas de decisões autônomas em relação as conexões entre *peers*, um método de inferência difuso foi proposto. Nesta seção são apresentadas as variáveis linguísticas utilizadas para avaliar a qualidade de conexão entre os Servidores de Comunicação (processos *peers*) dos aglomerados gráficos remotos. As variáveis foram divididas em duas categorias: entrada e saída.

As variáveis de entrada (Tabela 5.1) são utilizadas para mensurar os principais problemas encontrados em 3DCVEs em relação as redes de dados. Estes problemas não devem ser classificados simplesmente como presentes ou não, visto que valores intermediários podem ser levados em conta. A existência de um ou mais problemas também pode ser avaliada de em conjuntos, permitindo assim que diferentes ações sejam tomadas. Os valores utilizados como parâmetros das variáveis de entrada foram elencados durante a etapa de revisão. Estes foram citados por vários autores (vide Capítulo 2). As variáveis de entrada identificadas e definidas em termos linguísticos foram:

- Atraso na entrega de pacotes: essa variável considera o atraso na entrega dos pacotes entre os Servidores de Comunicação, podendo ser classificada como prejudicial, não prejudicial e pouco prejudicial;
- Variação de tempo na entrega de pacotes: essa variável considera o valor médio do tempo de entrega dos pacotes entre os Servidores de Comunicação. É classificada como sendo pouco prejudicial, prejudicial e altamente prejudicial; e

- Perda de pacotes: essa variável considera os pacotes que são perdidos nas conexões entre os Servidores de Comunicação. É classificada como baixa, alta e sem perda.

As variáveis de saída (5.2) são utilizadas com o intuito de aplicar ações sobre o ambiente de acordo com os valores classificados pelo modelo de inferência difuso. As variáveis de saída identificadas foram:

- Alterar quantidade de mensagens em um mesmo pacote de dados: esta variável determina se a configuração do pacote está coerente com o estado da rede em um dado momento. Os seguintes termos linguísticos foram classificados:
  - Não: o pacote não é alterado, mantendo o tamanho padrão usado na inicialização (cabeçalho + carga útil). Apenas uma mensagem (tamanho médio de 30 bytes) de atualização é enviada por pacote;
  - Médio: o pacote é alterado de acordo com a combinação de valores classificados, podendo ser utilizado para o envio de múltiplas mensagens de atualização; e
  - Grande: o pacote é alterado de acordo com o necessário. Seu tamanho é variável, assim como a quantidade de mensagens a serem enviadas por ele. Não é especificado o tamanho do pacote, sendo este limitado pelo tamanho máximo dos datagramas IP e pela perda de desempenho devido à fragmentação.
- Utilizar outro protocolo de comunicação: esta variável determina qual o protocolo de rede deve ser utilizado na intercomunicação dos Servidores de Comunicação. Os termos linguísticos classificados foram:
  - TCP: o protocolo padrão é mantido;
  - TCP/UDP: de acordo com a combinação de valores classificados por meio das variáveis de entrada, diferentes protocolos podem ser utilizados. O protocolo TCP/UDP combina o TCP e UDP, de modo que pacotes chave são transmitidos utilizando o TCP e pacotes de atualização de estado utilizando o UDP;
  - SCTP: similar ao TCP/UDP, pois utiliza características presentes nos dois protocolos, mas provê outras soluções, tal como um pequeno buffer do lado do receptor.
- Bufferizar: esta variável determina se o uso de bufferização é necessário. Os termos linguísticos classificados foram:
  - Não bufferizar: o envio e recebimento de mensagens é realizado sem o uso de nenhuma técnica de bufferização;

- Nível 1: pacotes de atualização de estado do ambiente são bufferizados no emissor com intuito de manter a taxa de quadros do 3DCVE frequente, com o intuito de não prejudicar a experiência do usuário. Abordagem utilizada em conjunto com a variável Alterar quantidade de mensagens em um mesmo pacote de dados;
- Nível 2: aplica-se o Nível 1 em conjunto com um pequeno buffer utilizado no lado do receptor, para quando houver fragmentação de pacotes. O protocolo SCTP foi utilizado.

• Predizer pacotes: esta variável é responsável por predizer pacotes perdidos durante a conexão entre os Servidores de Comunicação. Os termos linguísticos classificados foram:

- Não predizer: o modo de conexão padrão é mantido;
- Nível 1: um modelo de predição linear é utilizado, onde o pacote perdido é previsto a partir dos pacotes recebidos;
- Nível 2: é utilizado um modelo de predição baseado em compensação de latência, o algoritmo de *dead reckoning*, por exemplo. Isto deve solucionar dois problemas, simultaneamente: reduzir o consumo de banda e a mitigação dos efeitos ocorridos nos atrasos e perda de entrega de pacotes.

A função de pertinência utilizada foi a Trapezoidal (Equação 1). Ela representa os graus de pertinência apresentados nas Tabelas 5.1 e 5.2 – onde a, b, c e d são os pontos que definem as arestas do trapézio, de modo que os valores devem ser representados da seguinte maneira: **a < b <= c < d**, definindo assim um trapézio. A variável **x** define o intervalo do universo de discurso, que deve ser um número real ou um vetor não vazio de valores estritamente crescentes. O grau de pertinência é um valor real compreendido no intervalo [0,1], que significa o quanto é possível que o valor de **x** pertença ao conjunto difuso. Essa função de pertinência é fundamental para que seja possível utilizar o modelo de inferência difuso (ZADEH, 1965).

$$trapmf = (x;a,b,c,d) = max\left(min \quad (\tfrac{x-a}{b-a}, 1, \tfrac{d-x}{d-c}), 0\right)$$

1

A Figura 5.22 apresenta os valores de pertinência das variáveis de entrada do modelo de inferência difuso. As informações são as mesmas da Tabela 5.1.

A Figura 5.23 apresenta os valores de pertinência das variáveis de saída do modelo de inferência difuso. As informações são as mesmas da Tabela 5.2.

Tabela 5.1: Variáveis de entrada.

Nome da Variável	Termos Linguísticos	Grau de Pertinência (intervalo)
Atraso na entrega de pacotes (*delay*)	Não prejudicial	[0, 60]
	Pouco prejudicial	[55, 200]
	Prejudicial	[190, 600]
Variação de tempo na entrega de pacotes (*time_variation*)	Pouco prejudicial	[0, 5]
	Prejudicial	[4, 10]
	Altamente prejudicial	[9, 100]
Perda de pacotes (*loss*)	Sem perda	[0,3]
	Baixa	[2,6]
	Alta perda	[5,20]

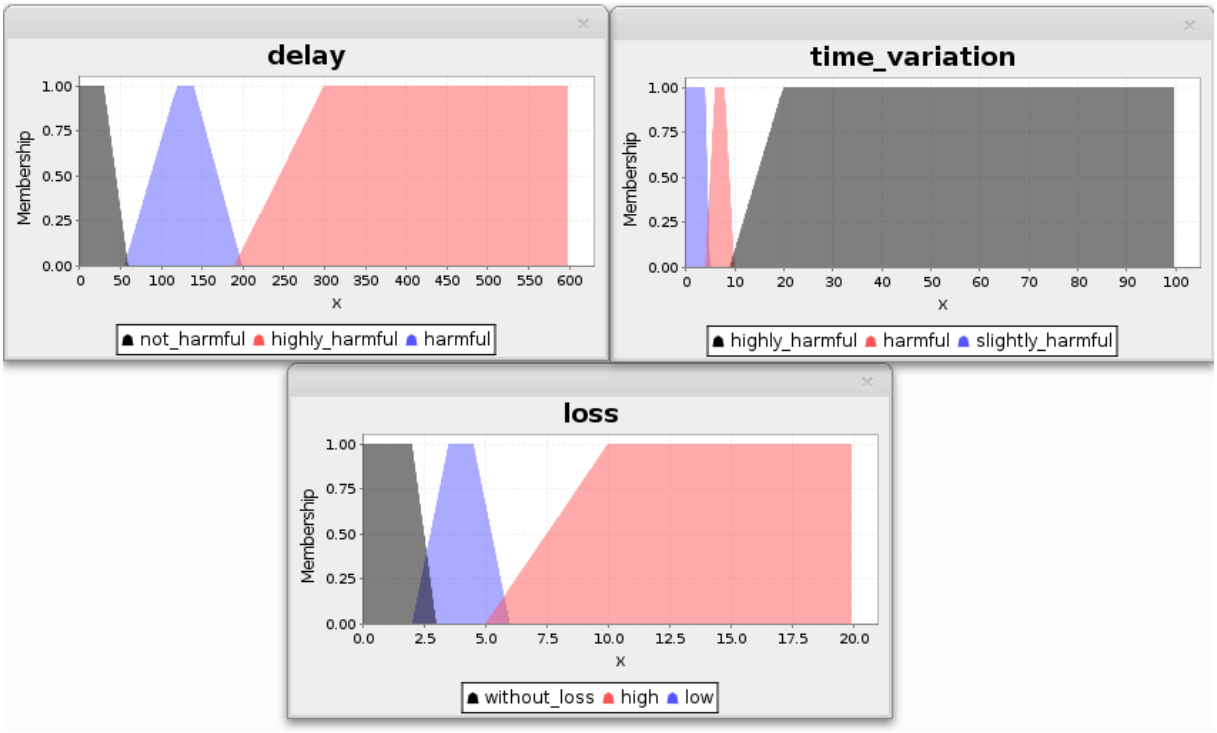


Figura 5.22: Valores de pertinência das variáveis de entrada.

Tabela 5.2: Variáveis de saída.

Nome da Variável	Termos Linguísticos	Grau de Pertinência (intervalo)
Alterar quantidade de mensagens (*data_packages*)	Não	[0, 0.3]
	Médio	[0.25, 0.6]
	Grande	[0.55, 1.0]
Utilizar outro protocolo de comunicaçao (*protocol*)	TCP	[0, 0.3]
	TCP/UDP	[0.25, 0.6]
	SCTP	[0.55, 1.0]
Buferizar (*buffer*)	Não buferizar	[0, 0.3]
	Nível 1	[0.28, 0.7]
	Nível 2	[0.68, 1.0]
Predizer pacotes (*predict*)	Não predizer	[0, 0.3]
	Nível 1	[0.25, 0.6]
	Nível 2	[0.55, 1.0]

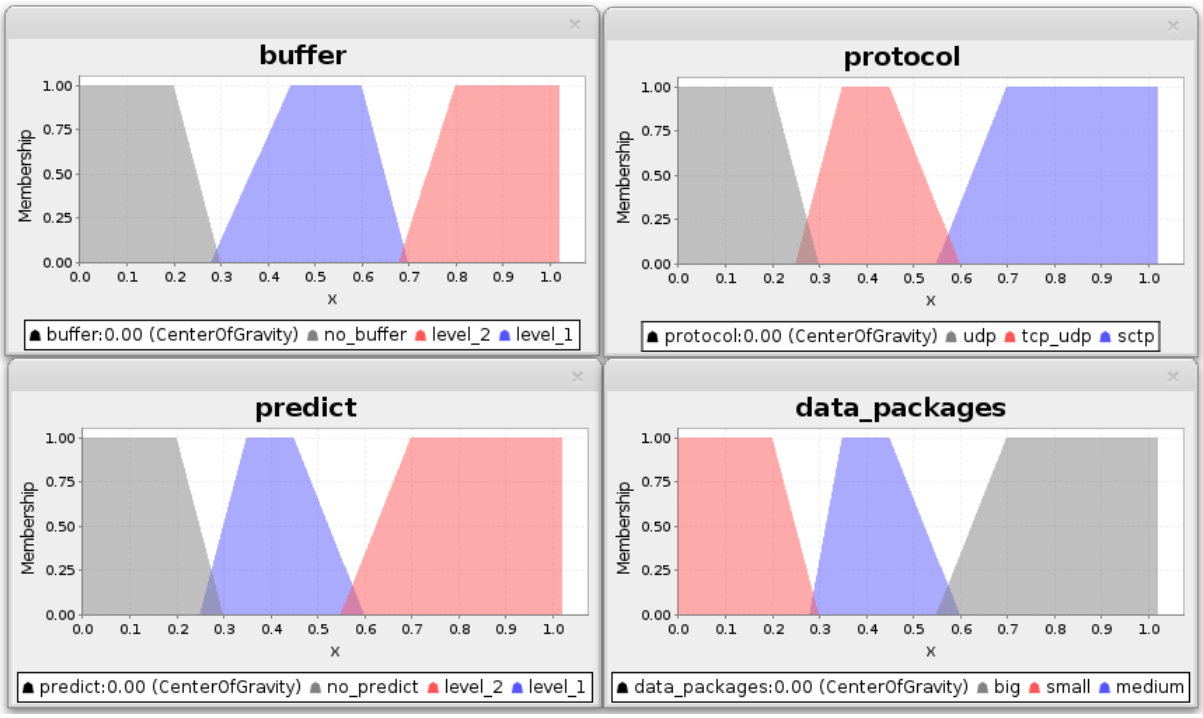


Figura 5.23: Valores de pertinências das variáveis de saída.

O cruzamento de diferentes valores das variáveis de entrada geram diferentes saídas. A Tabela 5.3 apresenta algumas das regras difusas criadas para o sistema de inferências. A definição completa das regras difusas são descritas no Apêndice B.

Tabela 5.3: Regras difusas.

SE delay É harmful E time_variation É slightly_harmful E loss É low ENTÃO buffer É level_1.
SE delay É harmful E time_variation É slightly_harmful E loss É low ENTÃO predict É level_1.
SE delay É harmful E time_variation É slightly_harmful E loss É low ENTÃO protocol É tcp_udp.
SE delay É harmful E time_variation É slightly_harmful E loss É low ENTÃO data_packages É medium.
SE delay É highly_harmful E time_variation É highly_harmful E loss É high ENTÃO buffer É level_2.
SE delay É highly_harmful E time_variation É highly_harmful E loss É high ENTÃO predict É no_predict.
SE delay É highly_harmful E time_variation É highly_harmful E loss É high ENTÃO protocol É sctp.
SE delay É highly_harmful E time_variation É highly_harmful E loss É high ENTÃO data_packages É big.
SE delay É not_harmful E time_variation É slightly_harmful E loss É without_loss ENTÃO buffer É no_buffer.
SE delay É not_harmful E time_variation É slightly_harmful E loss É without_loss ENTÃO predict É no_predict.
SE delay É not_harmful E time_variation É slightly_harmful E loss É without_loss ENTÃO protocol É udp.
SE delay É not_harmful E time_variation É slightly_harmful E loss É without_loss ENTÃO data_packages É small.
.....

5.9 Considerações finais

Este capítulo apresentou a definição comportamental dos componentes da arquitetura voltada a aglomerados gráficos remotos. Foram definidos os comportamentos para os ambientes intra–aglomerado e inter–aglomerados.

Os comportamentos dos componentes foram exemplificados por meio do uso de diagramas de sequência. Essa abordagem foi escolhida com o intuito de facilitar a forma de exemplificação, visando deixar clara a sequência exata de execução dos eventos.

Além dos componentes considerados básicos no desenvolvimento de 3DCVEs, também foi apresentado o componente de verificação dinâmica e adaptável de conexões. Este componente é uma das soluções originais apresentadas nesta tese.

As definições comportamentais para ambientes intra–aglomerado foram baseadas na tese de Guimaraes (2004), porém, novas características foram adicionadas.

Capítulo 6

DEFINIÇÃO DE UM MODELO DE ANÁLISE DE MENSAGENS

O processo de análise de mensagens é uma tarefa árdua se efetuada sem o apoio de ferramentas. Em ambientes distribúidos existem ainda mais complicações. Essa classe de aplicação está mais suscetível a erros de *software* (*bugs*) (KRANZLMÜLLER, 2000). A depuração e análise de aplicações distribuídas são problemas complexos. Diversas pesquisas têm sido efetuadas na área, porém, a maioria delas ainda realizam análises por meio de *prints* no código, ou seja, utilizando mensagens de saída criadas no próprio código fonte das aplicações, que são checadas ao final da execução, requerendo certo esforço do programador (REYNOLDS, 2006).

Este capítulo apresenta o modelo de análise de funcionalidades realizada por meio da ferramenta de apoio GTracer, utilizada na identificação de erros e falhas no desenvolvimento de aplicações baseadas na arquitetura apresentada nesta tese. A ferramenta foi estendida com o intuito de possibilitar a verificação inter–aglomerados.

6.1 Análise e depuração de mensagens de 3DCVEs

3DCVEs baseados em aglomerados gráficos são propensos a erros devido à sua arquitetura complexa. Estes incluem não apenas a rede e o multiprocessamento, mas também dependências de tempo, como por exemplo o *framelock* (garantia de que os novos quadros de vídeo são trocados em sincronia) e o *datalock* (dados da cena coerentes em todos os processos) (RAFFIN *et al.*, 2006). Deve-se também garantir que o desempenho seja aceitável para uma experiência em tempo real.

Abordagens de baixo nível, tais como *prints* ou depuradores convencioais, são inadequadas

para analisar 3DCVEs baseados em aglomerados gráficos, visto que existe vários nodos interligados por uma rede de computadores, executando vários processos simultaneamente. Isto conduz a várias questões: os pontos definidos para depuração podem gerar inconsistências e alterações no fluxo de execução; os *prints* geram grandes quantidades de dados que não podem ser lidos em tempo real.

Portanto, foi especificado um conjunto de funcionalidades de alto nível (modelo de sincronização e compartilhamento de mensagens assíncronas e síncronas) e um modelo padrão para o fluxo de execução esperado. O conjunto de funcionalidades foi projetado para atender de forma eficiente as necessidades previstas dessas aplicações. Neste sentido, este modelo pode ser vistos como sendo o núcleo de uma solução genérica para análise de dados trafegados em 3DCVEs. Ao invés de permitir a visualização do fluxo de execução do aglomerado como um *log* de texto apenas, o modelo permite visualizar o cruzamento entre as mensagens e o comportamento esperado.

O modelo permite que a checagem da troca de mensagens entre os processos possa ser visualizada de forma gráfica, de acordo com o comportamento esperado. É demonstrado o conjunto de funcionalidades que permite a análise dos processos. Em termos gerais, o GTracer possibilita checagem de violações em aplicações desenvolvidas por meio da solução apresentada nesta tese.

Segundo Guimaraes (2004), as contribuições geradas pelo GTracer são:

- Modelo funcional comportamental para aplicações de RV distribuídas em geral;
- Uma ferramenta de análise de comportamento funcional esperado para aplicações de RV distribuídas;
- Permite aos desenvolvedores maior entendimento do comportamento de aplicações por meio de uma visualização de alto nível (gráfica);
- Facilidade na identificação de comportamentos não esperados;
- Verificação em tempo de execução; e
- Auxílio no ensino de conteúdo relacionado à computação distribuída.

6.2 Ferramentas de depuração

O processo de análise de aplicações distribuídas é mais difícil do que o efetuado em programas sequênciais, porque possuem complexidade elevada e geram grande quantidade de dados, além de outros problemas, tais como condições de corrida e bloqueios (KRANZLMÜLLER, 2000). O processo de análise não deve interferir na codificação das aplicações.

Algumas ferramentas não requerem nenhum tipo de mudança do código fonte, colentando dados usando recursos nativos do sistema, como os dados fornecidos pelo sistema operacional ou monitoramento de dados tráfegados na rede por meio de *sniffers*. Apesar de ser uma solução satisfatória em alguns casos, não é suficientemente robusta para capturar detalhes sobre a execução de aplicações. É altamente desejável que o desempenho das aplicações não seja afetado de forma considerável, o que nem sempre é possível, sendo este um dos principais motivos para a não inclusão de sistemas de depuração na geração de *builds* por padrão. Tem havido um grande volume de pesquisas relacionadas à verificação e análise de aplicações distribuídas pós–execução, incluíndo análises estatísticas sobre o uso de recursos (AGUILERA *et al.*, 2003), correlação de eventos (BARHAM *et al.*, 2004), dependências de componentes (FONSECA *et al.*, 2007) e verificação de caminhos causais (REYNOLDS, 2006).

A Figura 6.1 apresenta o tamanho de um sistema *versus* o esforço realizado pelo desenvolvedor para utilizar a ferramenta. Sistemas podem variar de um simples nodo a uma aplicação de Internet de grande escala, enquanto o esforço do desenvolvedor varia entre o modelo de depuração manual e automático, feito por uma ferramenta ou por meio de comandos, como o *printf* ou os depuradores, tais como o *gbd* (REYNOLDS, 2006) e o *gprof* (GRAHAM *et al.*, 1982). A MaceMC (KILLIAN *et al.*, 2007) e a VeriSoft (GODEFROID, 2005) são ferramentas para a verificação de modelos que checam propriedades específicas, mas requerem implementação de código extra. A Pinpoint (CHEN *et al.*, 2002) tem como objetivo identificar componentes problemáticos, utilizando conceitos de probabilidade para detectar anomalias em um *log* de eventos problemáticos, ao invés de analizar o caminho completo. A Magpie (BARHAM *et al.*, 2003) é um exemplo de ferramenta que necessita de nenhum tipo de alteração no código da aplicação analisada, mas requer o desenvolvimento de um programa específico, escrito por um especialista para identificar informações por meio do caminho causal. O Project 5, WAP5 e Pip também utilizam a técnica de caminho causal (REYNOLDS, 2006). Há também as técnicas baseadas em testes automáticos, porém, apesar de estes apresentarem os erros, não são a solução ideal quando se precisa identificar a causa do erro, e não apenas sua existência.

O GTracer não almeja substituir depuradores tradicionais, mas sim auxiliar o desenvolvi-

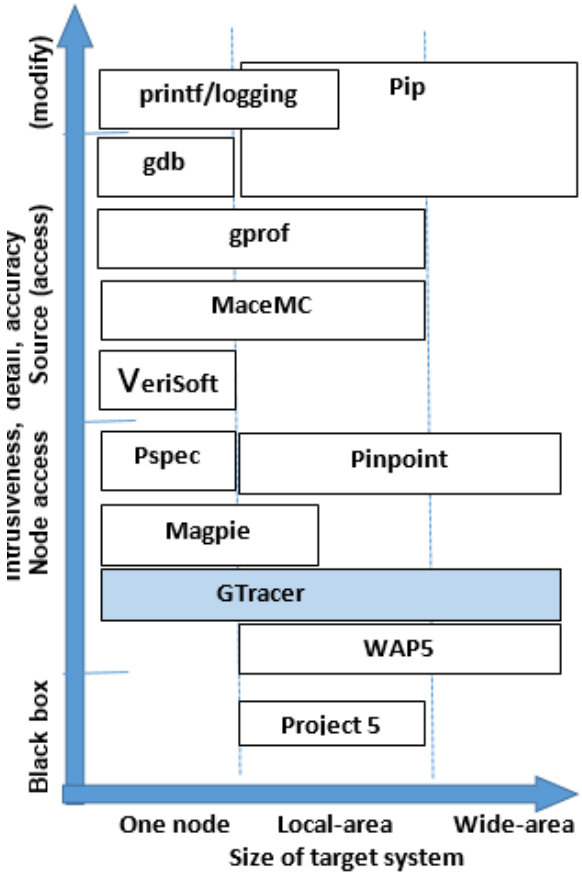


Figura 6.1: Tamanho do sistema *versus* o esforço do desenvolvedor para usar a ferramenta – adaptada de (REYNOLDS, 2006).

mento de aplicações distribuídas, sem a necessidade de nenhum tipo de alteração de código, visto que a análise é realizada por meio do modo de depuração da libGlass. O modo depuração da biblioteca pode ser habilitado no momento da compilação.

6.3 Requisitos para análise de mensagens

3DCVEs baseados em aglomerados gráficos visam renderizar múltiplas imagens de um mesmo conjunto de dados visuais em tempo real. Cada processo tem acesso a todo o conjunto de dados (modelos 3D e texturas), determinando, independetemente, quanto do conjunto de dados é renderizado por cada processo. Estas aplicações tem a tendência de ser implementadas no modelo mestre–escravo.

Com o intuito de manter o sincronismo e a coerência sobre os dados, os *locks datalock* e *framelock* são utilizados no desenvolvimento de 3DCVEs. De acordo com Raffin *et al.* (2006), *locks* do tipo *datalock* possibilitam a coerência dos dados entre os processos distribuídos em um sistema distribuído. Os *locks* do tipo *framelock*, também chamados de *swaplock*, garantem

a coerência da cena. Assim, cada processo executa a função de *render* ao mesmo tempo que os outros processos.

A abordagem clássica de desenvolvimento de 3DCVEs é suportada pelo uso de bibliotecas genéricas de computação distribuída, e, então, as funcionalidades de alto nível são implementadas sobre as primitivas de baixo nível. As soluções de baixo nível mais utilizadas são os BSD *sockets* e o MPI. É comum utilizar bibliotecas de alto nível, como a libGlass, visando esconder do desenvolvedor toda a complexidade da comunicação e gerenciamento dos dados das aplicações distribuídas.

6.4 Comportamento do GTracer – intra–aglomerado e inter–aglomerados

A libGlass pode gerar um arquivo de *log* de todos os pacotes trafegados entre os processos. Cada mensagem é vinculada à uma primitiva, que é armazenada em um arquivo. O arquivo *log* é composto pelos parâmetros: identificação do processo, identificação da primitiva, processo origem, projeto destino, dados modificados e o *timestamp*. O GTracer permite que erros de implementação sejam encontrados, sejam eles relacionados à aplicação ou à biblioteca em si. A Figura 6.2 apresenta a interface principal do GTracer, que possui botões para a leitura, análise e visualização dos arquivos *log*, e o filtro de mensagens. A visualização é dividida em guias, apresentando as mensagens transmitidas por cada processo ou por todos os processos. Uma que vez que todas as mensagens possuem um identificador único e um *timestamp*, é possível organizá-las cronologicamente, mesmo sem um relógio global de sincronismo. O cruzamento das informações armazenadas nos *logs* permite a identificação da ordem de execução das mensagens.

Uma importante característica do GTracer é que os desenvolvedores não precisam escrever nenhum código adicional, sendo apenas necessário habilitar a geração de *logs* durante a compilação da biblioteca. A libGlass é responsável por gerar toda a codificação de baixo nível referente as mensagens trocadas entre os processos. Isto facilita o uso da ferramenta. A ideia geral do GTracer é checar se a troca de mensagens funciona como o especificado e esperado. Por exemplo, o GTracer pode verificar o comportamento esperado de uma mensagem de sincronização baseando-se na troca de mensagens entre os processos origem e destino. Todas as mensagens são ordenadas sequenciamente, o que ajuda o desenvolvedor no entendimento do comportamento da aplicação. Essas mensagens ordenadas tendem à auxiliar o desenvolvedor no processo de localização de erros e falhas, como por exemplo, identificar qual processo pode

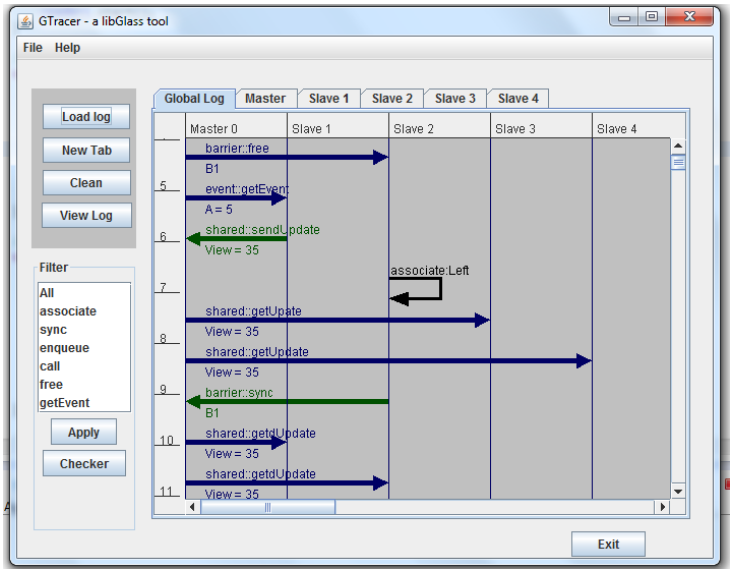


Figura 6.2: Interface do GTracer.

ser classificado como gargalo e em qual nodo ele está sendo executado.

Um modelo funcional de execução é apresentado nesta seção, que pode ser utilizado por uma ferramenta de análise comportamental. Além do mais, este modelo pode ser utilizado como base para a prototipação e teste de novas bibliotecas de desenvolvimento de aplicações de distribuídas, com o intuito de auxiliar o processo cognitivo do desenvolvedor em relação ao desenvolvimento. Com este propósito, foi especificado o comportamento de cada funcionalidade, primitivas, entre outras funções, visando mitigar potenciais problemas relacionados à troca de mensagens.

6.4.1 Análise de barreiras de sincronização

Uma barreira é um ponto onde a aplicação efetua uma pausa até que todos os processos que possuam essa barreira atinjam este mesmo ponto. Barreiras utilizam algoritmos lineares (ARENSTORF; JORDAN, 1989) e são compostas por duas fases, chegada e liberação. Na fase de chegada, um processo entra em uma barreira e fica congelado até que todos os processos atinjam a mesma barreira. Então, o processo passa para a fase de liberação. Uma aplicação pode possuir diversos pontos de sincronismo e barreiras em seu código. Nesta tese foram especificados comportamentos de barreiras de sincronismo, utilizadas no desenvolvimento de 3DCVEs na implementação de *locks* (*datalock* e *framelock*).

O GTracer permite ao desenvolvedor observar se as barreiras estão sincronizadas como esperado. Mais que isso, barreiras de sincronismo podem causar problemas relacionados à *deadlock*, que pode acontecer se um processo chegar à uma barreira e ficar esperando pela

chegada de outro processo na mesma barreira. Isto é algo que pode não acontecer, desse modo, ambas não chegarão à fase de liberação. A libGlass provê proteções contra *deadlocks* e o GTracer permite que o desenvolvedor identifique esses *deadlocks* de forma visual.

6.4.1.1 Barreira intra–aglomerado

O GTracer analisa o arquivo *log*, verifica se as barreiras possuem o comportamento esperado (definido pela libGlass) e emite um aviso se elas não o possuem. Informação extra, como a lista de barreiras declaradas em cada processo, também é disponibilizada. Na Figura 6.3 é apresentado um exemplo de comportamento inesperado: neste caso, o processo P1 registra apenas a barreia B1, entretanto, o processo mestre envia uma primitiva para liberar a barreira B2, que deveria ser identificada como um erro, segundo o modelo implementado pela libGlass.

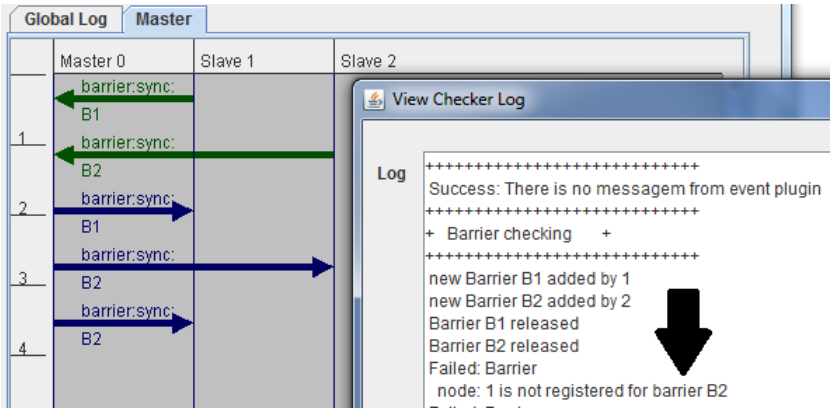


Figura 6.3: GTracer – processos escravos recebem o sinal de desbloqueio, mas não possuem a barreira instanciada.

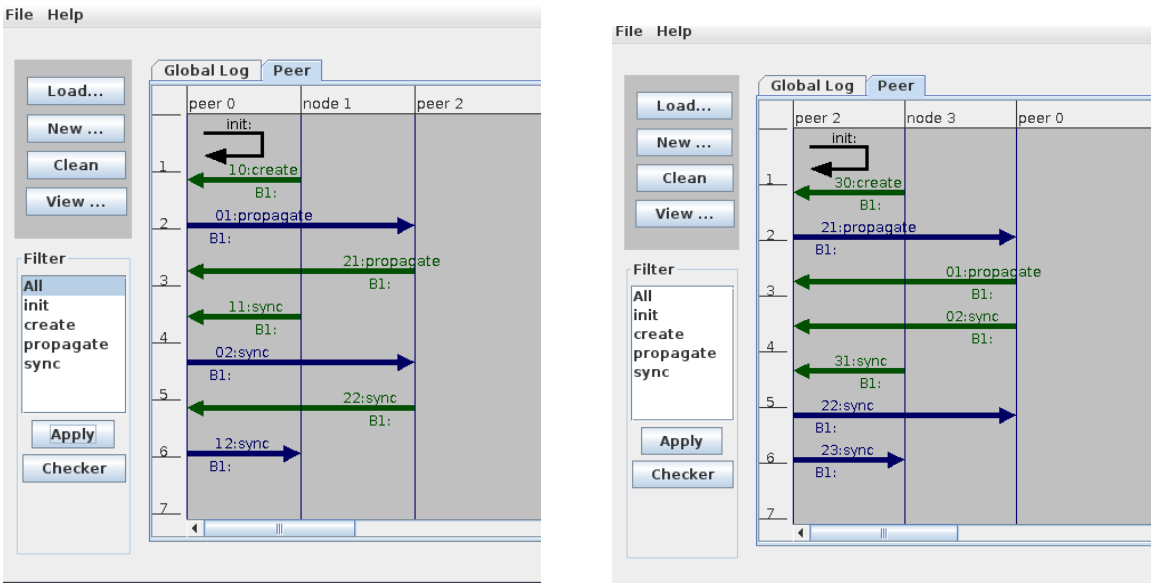
6.4.1.2 Barreira inter–aglomerados

O funcionamento de barreiras de sincronismo para ambientes inter–aglomerados é diferente do aplicado no modelo intra–aglomerado (que possui um servidor/mestre centralizado). No caso da comunicação inter–aglomerados, onde a troca de mensagens acontece entre *peers*, não existe um servidor para receber as solicitações de sincronismo e difundir a todos os processos. Neste caso, todos os *peers* trocam mensagem entre si. Cada *peer* possui uma lista contendo todos os *peers* que tem interesse em uma determinada barreira, isto é, que possua, pelo menos, a barreira instanciada em um de seus processos escravos.

O funcionamento para ambientes *Remote Peer*, no caso da relação com seu *peers*, é semelhante ao intra–aglomerado, uma vez que o papel do *Remote Peer* é centralizar um conjunto de *peers* sobre o seu controle. No entanto, na comunicação entre *Remote Peers*, o modelo aplicado

é o inter–aglomerados.

A Figura 6.4 apresenta o comportamento de barreiras de sincronismo em ambientes inter–aglomerados. Neste caso, o processo *Peer_0* difunde a mensagem de sincronismo ao *Peer_2*. As setas possuem cores diferentes devido a origem do *log* analisado. As setas verdes são referentes as mensagens trafegadas a partir do processo *Peer_2*; as setas azuis são referentes as mensagens trafegadas a partir do processo *Peer_0*. Neste exemplo, inicialmente, o *Node_1* possui a barreira B1 instanciada. Já o processo *Peer_2* possui o *Node_3*, que também possui a barreira B1 instanciada. Portanto, ambos os *peers* têm interesse nas mensagens relacionadas à barreira B1.



(a) Peer 0. (b) Peer 2.

Figura 6.4: Comportamento de barreiras de sincronismo – inter–aglomerados gráficos.

6.4.2 Análise de variáveis síncronas

3DCVEs compartilham variáveis que devem ser sincronizadas. O posicionamento de câmera é um exemplo comum. Essas variáveis são um tipo de variável síncrona compartilhada. O desenvolvedor especifica quando elas devem ser atualizadas globalmente e quando as variáveis locais devem ser atualizadas com os valores globais. Processos podem ter conjuntos de variáveis compartilhadas distintos. A mesma variável compartilhada pode ser utilizada em mais de um processo. A chamada de sincronismo (usualmente combinada com uma barreira) bloqueia a aplicação temporariamente e tende a gerar perda de desempenho, mas é uma tarefa necessária quando se precisa manter o alto nível de sincronismo. As variáveis síncronas compartilhadas devem ser definidas pelo desenvolvedor.

6.4.2.1 Variáveis síncronas intra–aglomerado

Em um modelo mestre–escravo, por exemplo, um processo escravo envia um novo valor de variável (*sendUpdate*) a um processo mestre (fase de envio de atualização), e outros processos recebem o valor atualizado (*getUpdate*) (fase de recebimento de atualização). A Figura 6.5 apresenta uma análise realizada aplicando um filtro, que apresenta somente as mensagens referentes a variáveis Compartilhamento (síncronas). Durante a checagem das mensagens, o GTracer detecta que um valor corrente é 15. O valor correto é enviado ao Escravo_1, mas uma valor errado (17) é enviado ao Escravo_2. Neste tipo de situação, a biblioteca, por algum motivo, alterou o valor da variável erroneamente, e, portanto, isto é considerado um *bug*. Além da checagem do comportamento esperado, o GTracer auxilia o processo de identificação do momento e ordem que os eventos acontecem. O desenvolvedor pode ver quais variáveis foram declaradas por cada processo.

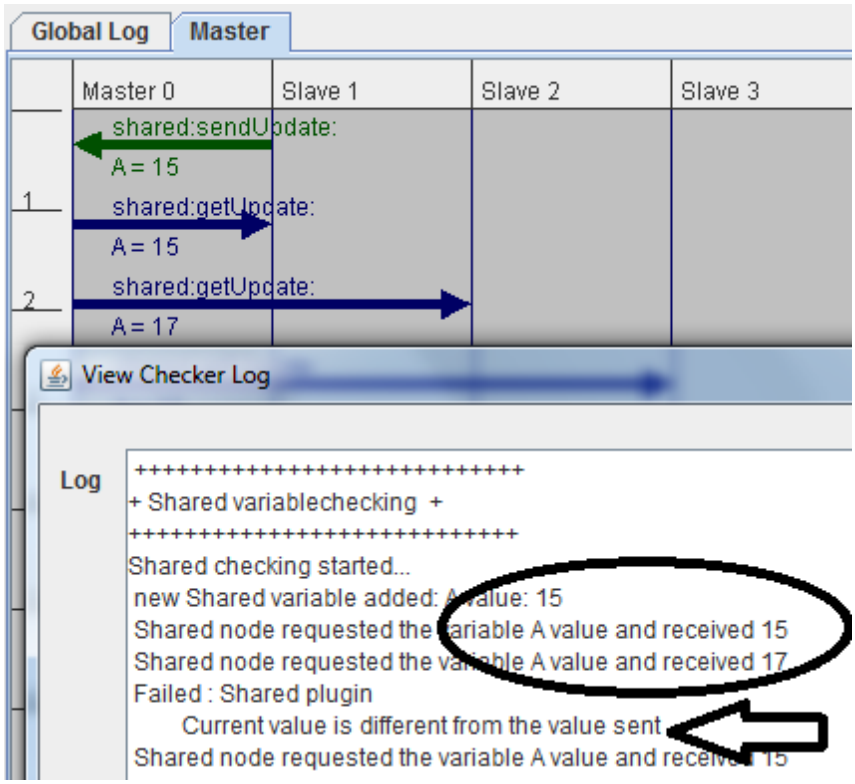


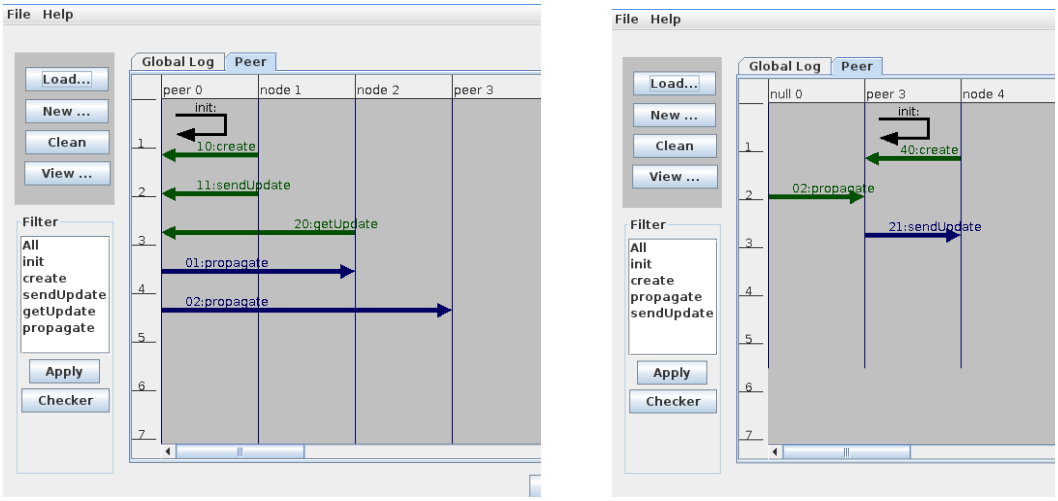
Figura 6.5: GTracer – valor incosistente enviado.

6.4.2.2 Variáveis síncronas inter–aglomerados

O funcionamento das variáveis síncronas para ambientes inter–aglomerados é diferente do aplicado no modelo intra–aglomerado, que fornece mensagens atualizadas aos processos escravos somente quando estes as solicitam. No caso da comunicação inter–aglomerados, em que a

troca de mensagens acontece entre *peers*, não existe um mestre para receber as chamadas (*getUpdate()*) de outros *peers*. Assim, sempre que um processo *peer* recebe um valor atualizado, este o propaga o mais rápido possível, desde que o processo escravo de outro *peer*, presente em sua lista, tenha a variável instanciada.

A Figura 6.6 apresenta o comportamento de variáveis síncronas em ambientes inter–aglomerados. Neste exemplo, o processo *Peer_0* recebe um valor atualizado (variável SH1) do seu processo escravo *Node_1*, que deve então, ser propagado aos outros processos escravos (*Node_2*) e *peers* (*Peer_3*). Já o *Peer_3*, recebe a atualização enviada pelo *Peer_0* e a difunde ao seu cliente *Node_4*, visto que este possui a variável SH1 instanciada.



(a) Peer 0. (b) Peer 2.

Figura 6.6: Comportamento de variáveis síncronas – inter–aglomerados gráficos.

6.4.3 Análise de eventos assíncronos

Os eventos assíncronos, como apresentados na Seção 5.5, são caracterizados, basicamente, por uma fila (FIFO (*firts–in, first–out*). Eles são úteis à propagação de alterações ou eventos assíncronos em ambientes distribuídos. Existem vários tipos de eventos, como por exemplo, os eventos de entrada. O GTracer auxilia na verificação de como estes eventos são propagados e tratados. Usualmente, em uma arquitetura mestre–escravo, um processo envia um evento a um processo mestre, que o propaga a outros processos, assim que possível. Um processo pode possuir diversas variáveis de evento assíncrono ao mesmo tempo.

6.4.3.1 Eventos assíncronos intra–aglomerado

A Figura 6.7 apresenta a análise de propagação de eventos assíncronos feita pelo GTracer, que tem o intuito de verificar inconsistências entre um valor recebido pelo mestre (evento com valor 5) e um valor propagado (valor 7). A ferramenta também apresenta os eventos registrados por cada processo e quando eles foram enviados e recebidos.

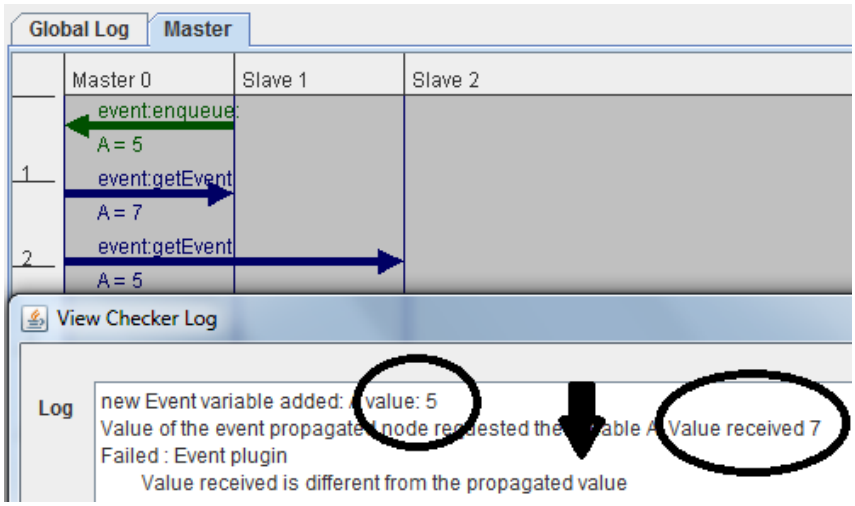
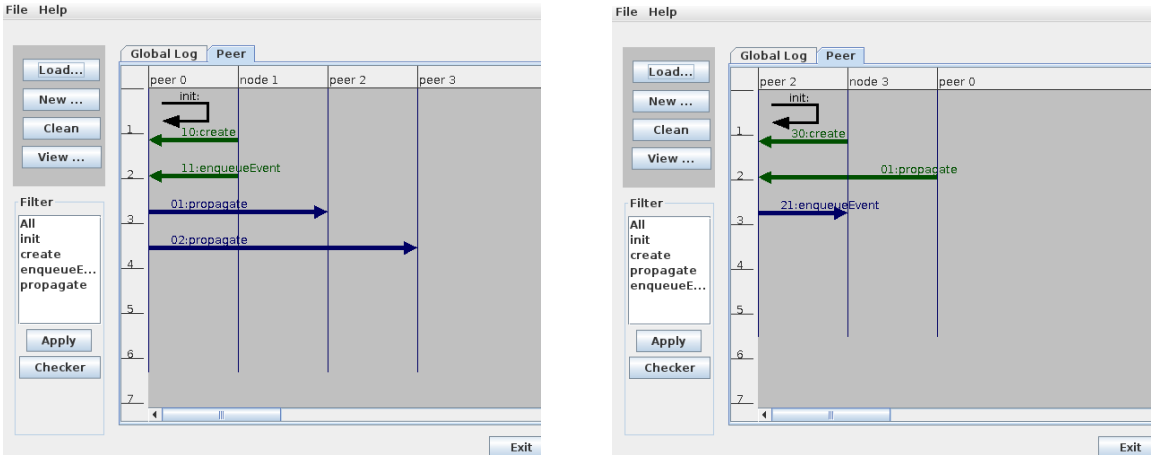


Figura 6.7: GTracer – análise de propagação de evento assíncrono.

6.4.3.2 Eventos assíncronos inter–aglomerados

O comportamento dos eventos assíncronos para ambientes inter–aglomerados é semelhante ao realizado no modelo de comunicação intra–aglomerado. Para cada mensagem enfileirada por um processo escravo, o processo *peer* deve encaminhá-la, o mais rápido possível, aos outros processos escravos e *peers*.

Na Figura 6.8 o *Peer_0* recebe um novo evento assíncrono (EV1). Este evento é enviado, assim que possível, aos processos escravos (*Node_1*) e *peers* (*Peer_2* e *Peer_3*). Os eventos assíncronos, diferente das variáveis síncronas, não dependem da chamada *getUpdate()* para serem propagados a outros processos. Assim, o processo *peer* que recebe o evento deve escolher para quais processos o evento deve ser enviado primeiro. No modelo desenvolvido, os eventos são sempre enviados seguindo um modelo hierárquico: mais alto para o mais baixo. Deste modo, os eventos devem ser enviados, primeiramente, aos processos *peers*. Esta regra foi tomada com o intuito de diminuir, ao máximo, a latência ocasionada pela rede, não acrescendo cargas extras de atraso do processamento interno da biblioteca.



(a) Peer 0. (b) Peer 2.

Figura 6.8: Comportamento de eventos assíncronos – inter–aglomerados gráficos.

6.4.4 Análise de alíases

Um dos problemas em lidar com ambientes distribuídos é o gerenciamento de IDs dos processos. Por exemplo, em um sistema de multiprojeção com três processos, é necessário renderizar as telas da frente, esquerda e direita. Os alíases, ou apelidos, possuem valores distintos para cada processo, associando-os com seus diferentes pontos de vista. O mesmo código é executado em todas as instâncias, efetuando a chamada de função *set_view()*, que vincula o ID do processo a um alíases. Esta função é utilizada para configurar a câmera. Assim, depois do alíases vinculado, o processo pode ser identificado pelo seu alíases, além do seu ID. A configuração é realizada por meio do processo mestre, o processo que executa a primitiva, associando um processo, que possui um ID, a uma apelido (fase de associação).

6.4.4.1 Alíases intra–aglomerado

A Figura 6.9 apresenta a análise realizada pelo GTracer, que detecta o comportamento da função "esquerda", que é executada ao invés da "frente"pelo Escravo_1. Os apelidos são apresentados, portanto, o desenvolvedor pode visualizar o seus valores em tempo–real.

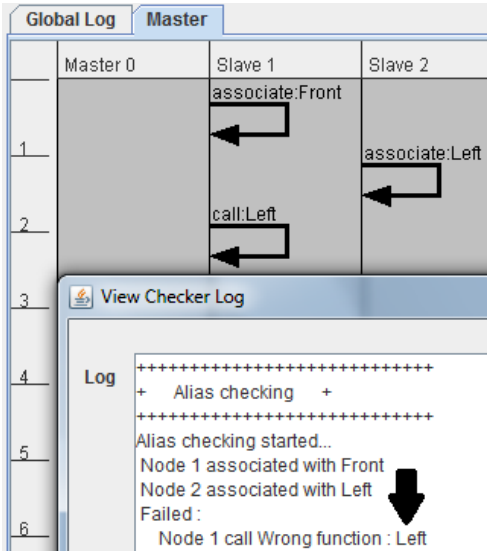
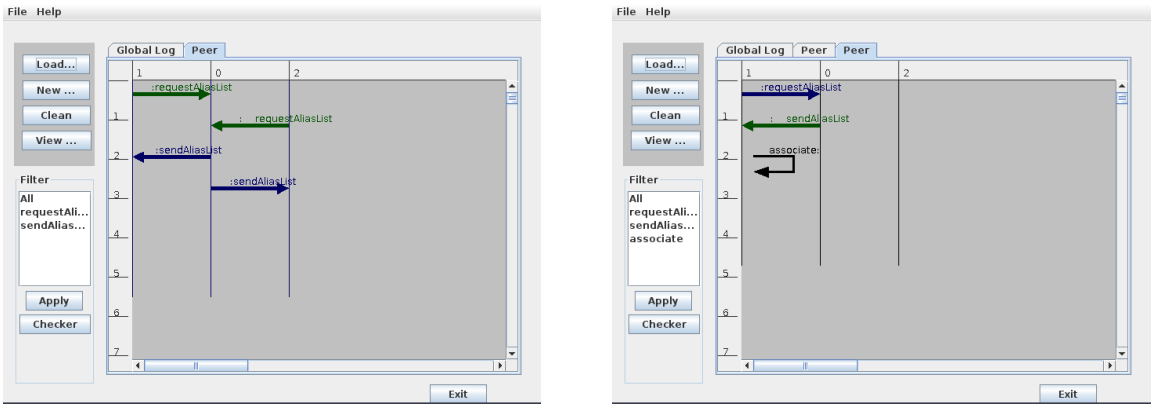


Figura 6.9: GTracer – análise de mensagem de associação.

6.4.4.2 Alíases inter–aglomerados

A atribuição de alíases em ambientes inter–aglomerados é realizada pelo primeiro *peer* instanciado. Este possui uma lista dos alíases (ID, alíases) disponíveis aos outros processos *peers*. Desta forma, sempre que um novo *peer* é instanciado, este deve requisitar a lista de alíases para saber quais alíases estão disponíveis. Por exemplo, o Peer_0 é instanciado e cria uma lista de alíases. Assim, quando o Peer_1 conecta-se ao Peer_0, este requisita a lista de alíases ao Peer_0, que recebe a solicitação e vincula o ID do Peer_1 a um alíases. Então, o Peer_0 envia a lista atualizada ao Peer_1 e aos outros *peers* a que ele esteja conectado. Todo *peer* conseguinte requisita a lista de alíases ao *peer* que ele se conectar primeiro, visto que todos mantêm a lista atualizada. Este comportamento é apresentado na Figura 6.10.



(a) Peer 0. (b) Peer 1.

Figura 6.10: Comportamento de atribuição de alíases – inter–aglomerados gráficos.

6.5 Considerações finais

O processo de análise de aplicações distribuídas pode ser uma tarefa complexa e custosa. Algumas ferramentas requerem maior envolvimento e esforço do desenvolvedor, visto que modificações de código podem ser requeridas, tais como inserção de pontos de controle, verificação e adição de chamadas específicas, entre outras. Por outro lado, algumas ferramentas realizam análise sem modificações de código.

As aplicações distribuídas são, normalmente, mais complexas, pois são caracterizadas por múltiplos processos executados em máquinas distintas. Enquanto bibliotecas auxiliam o processo de desenvolvimento, as ferramentas de depuração nem sempre são fáceis de se utilizar. Ainda existe muito a ser resolvido com relação à área de depuração de 3DCVEs, tal como a grande quantidade de informação transparente ao usuário.

As linhas de tempo gráficas são um modo prático para exemplicar o tráfego e estados de uma aplicação. A análise dos *logs* auxilia a detecção de comportamentos não esperados. Este tipo de depuração auxília o desenvolvimento da arquitetura definida nesta tese e as aplicações que a utilizam. Também é extremamente útil para propósitos pedagógicos, apresentando aos estudantes todo o tráfego de mensagens, relacionando o que acontece com a aplicação.

Capítulo 7

ESTUDOS DE CASO

Este capítulo apresenta a definição dos protocolos e execução dos estudos de caso realizados com o intuito de avaliar a arquitetura apresentada nesta tese. O comportamento definido no Capítulo 5 também é analisado neste capítulo, assim como os requisitos de troca de mensagens em 3DCVEs baseados em aglomerados gráficos remotos.

7.1 Definição dos estudos de caso

De acordo com Runeson *et al.* (2012) um estudo de caso é realizado com o intuito de investigar uma simples entidade ou fenômeno em seu contexto real em um tempo–espaço específico. Durante a realização de um estudo de caso, diversos procedimentos de coleta de dados e análise de diferentes perspectivas devem ser avaliados. Geralmente, estudos de caso são bastante aplicados em métodos de Engenharia de *Software* industriais e ferramentas. De acordo com Yin (2013) a principal diferença entre o estudo de caso e o experimento, é que o experimento baseia–se nas variáveis que são manipuladas, enquanto que o estudo de caso é baseado nos dados das variáveis que apresentam a situação típica. O termo estudo de caso também é utilizado em paralelo com termos como estudo de campo e estudo observacional, definição que se aplica ao apresentado nesta tese.

7.1.1 Objetivos dos estudos de caso

Os objetivos de um estudo de caso devem ser bem definidos, sendo a principal atividade na sua concepção. Nesta tese foram definidos quatro estudos de caso para avaliar a arquitetura apresentada. Os objetivos dos estudos de caso foram:

- Estudo de Caso 1: verificar se as tomadas de decisões sugeridas pelo modelo de comunicação dinâmico–adaptável são realmente as esperadas;
- Estudo de Caso 2: realizar análises referentes à influência das redes de dados na execução de 3DCVEs, e verificar se o ambiente é capaz de manter a qualidade de experiência do usuário na presença de adversidades de rede. A integração entre o miniCAVE e o aixCAVE foi realizada neste estudo;
- Estudo de Caso 3: definir o modelo de comunicação colaborativo do RehabGesture (BRANDAO *et al.*, 2016); e
- Estudo de Caso 4: realizar os testes de integração automáticos.

7.1.2 Casos e unidades de análise

Na Engenharia de *Software* um projeto de *software* pode ser classificado com sendo um caso. Projetos de *software* podem ser tratados como objetos de um estudo de caso, porém, com a possibilidade de serem analisados de forma modular, de modo que o pesquisador possa focar em algum aspecto específico, as unidades de análise (YIN, 2013). As unidades de análise definidas foram:

- Estudo de Caso 1: neste estudo foram analisadas as tomadas de decisões efetuadas pelo módulo difuso. De acordo com diversas configurações de rede, algumas tomadas de decisões eram esperadas pelo ambiente. As saídas esperadas foram comparadas com as obtidas, a fim de verificar se os resultados eram realmente os esperados;
- Estudo de Caso 2: neste estudo diferentes configurações de rede foram utilizadas com o intuito de realizar análises referentes à influência das redes de dados na execução de 3DCVEs. Por meio de aplicações exemplo, a influência das redes de dados foram analisadas de modo empírico, visando verificar se o ambiente era capaz de manter a qualidade de experiência do usuário na presença de adversidades da rede. Para este estudo foram utilizados ambientes reais de execução, o miniCAVE e o aixCAVE (RWTH);
- Estudo de Caso 3: foram realizadas avaliações de comunicação entre dois ambientes (1 e 2) executando instâncias do RehabGesture; e
- Estudo de Caso 4: Os módulos apresentados no Capítulo 5 foram testados por meio de testes automáticos de integração.

7.2 Adversidades de rede

As adversidades ocasionadas pelas redes de dados são diversas, dentre elas, as que possuem maior destaque são: latência, variação no tempo de entrega de pacotes e perda de pacotes. Estes problemas tendem a ser encontrados em redes de dados baseadas em Internet.

Com o intuito de simular essas adversidades a ferramenta NetEM (2016) foi utilizada. A ferramenta provê funcionalidades para efetuar testes em protocolos, emulando as principais propriedades de uma rede baseada em Internet. A versão utilizada durante a concepção desta tese permitiu emular latência variável, perda de pacotes, duplicação de pacotes, reordenação de pacotes e largura de banda.

Neste experimento foram executados ambientes virtuais em diversas configurações de rede, inclusive efetuando combinações entre elas. As adversidades emuladas são apresentadas na Tabela 7.1.

Tabela 7.1: Adversidades de rede de dados.

Adversidade	**Valores**
Latência	0ms, 10ms, 100ms, 200ms, 300ms, 400ms, 500ms, 600ms, 800ms e 1000ms
Variação no tempo de entrega	0ms, 5ms, 10ms, 30ms, 50ms, 100ms e 200ms
Perda de pacotes	0,1%, 0,5%, 1%, 2%, 5%, 10% e 25%
Pacotes corrompidos	0,1%, 0,5%, 1%, 2%, 5%, 10% e 25%
Reordenação de pacotes	0,1%, 0,5%, 1%, 2%, 5%, 10% e 25%
Largura de banda	256Kb, 1024Kb, 5120Kb, 10240Kb e 102400Kb

- Latência: uma taxa fixa foi atribuída aos pacotes enviados por uma interface de rede específica (eth0/lo);
- Variação no tempo de entrega de pacotes: a latência em uma rede de dados não é algo uniforme. Portanto, simulações de variação no tempo de entrega de pacotes foram levadas em consideração;
- Perda de pacotes: a perda de pacotes randômica é algo extremamente prejudicial a sistemas de tempo real, sendo eles reenviados ou não pelo emissor, sempre ocasiona algum tipo de inconsistências no ambiente;
- Pacotes duplicados: é especificado da mesma forma que os pacotes perdidos, sendo necessário o reenvio, descarte ou predição de dados;
- Pacotes corrompidos: inserir apenas um *bit* defeituoso no pacote de dados;

- Reordenação de pacotes: a existência de variação no tempo de entrega de pacotes pode acarretar na entrega fora de ordem, que, dependendo do protocolo utilizado, pode ser um problema; e
- Largura de banda: diferentes larguras de bandas foram experimentadas, visando avaliar sua influência na vazão de dados.

7.3 Estudo de caso 1 – modelo de inferência difuso

A fim de avaliar as definições do modelo de inferência difuso, foram realizadas algumas simulações. Alguns valores de entrada foram simulados, e as saídas obtidas foram verificadas com os resultados esperados.

Na primeira simulação foram utilizados os valores de 12ms para atraso na entrega de pacotes, 3ms para variação de tempo na entrega de pacotes e 0 para perda de pacotes, valores normalmente encontrados em comunicações entre conexões de qualidade sobre à Internet. Estes valores foram encontrados durante experimentos de conexão entre o LaVIIC (UFSCar) e o LIV (Bauru) utilizando a Internet. A Figura 7.1 apresenta os resultados obtidos.

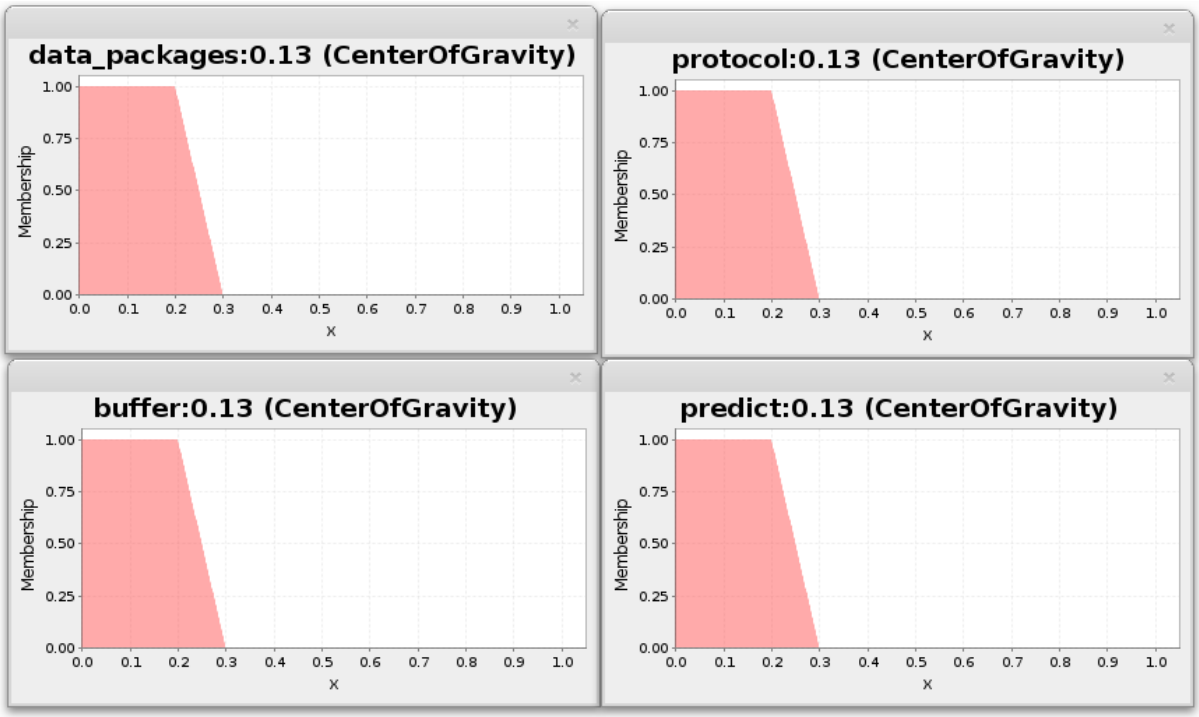


Figura 7.1: Saídas – Simulação 1.

Os resultados das variáveis de saída obtidas apresentam valores que devem ser usados como parâmetros nas conexões entre *peers*. Os resultados dos valores apresentados na Figura 7.1

definem o Alterar quantidade de mensagens em um mesmo pacote de dados como Não, isto é, o tamanho padrão do pacote é mantido, o que depende do protocolo utilizado, de modo que cada pacote contenha apenas uma mensagem, mesmo que essa não atinja o tamanho máximo do pacote (por exemplo, 1500 bytes); o protocolo de comunicação é definido como sendo o UDP; e, por ser uma conexão caracterizada como sendo de boa qualidade, não existe a necessidade de bufferização ou predição de dados. Os valores referentes as variáveis de saída são apresentados na Tabela 7.2:

Tabela 7.2: Simulação 1

Variável de Saída	**Valor Fuzzificado**
Alterar quantidade de mensagens em um mesmo pacote de dados	Não
Protocolo de comunicação	TCP/UDP
Bufferizar	Não bufferizar
Predizer	Não predizer

Na segunda simulação foram utilizados os valores de 100ms para atraso na entrega de pacotes, 3ms para variação de entrega de pacotes e 3 para perda de pacotes. Esses valores foram encontrados entre o LaVIIC (UFSCar) e o Laboratório de RV – UESB (Jequié), utilizando a Internet. A Figura 7.2 apresenta os resultados obtidos.

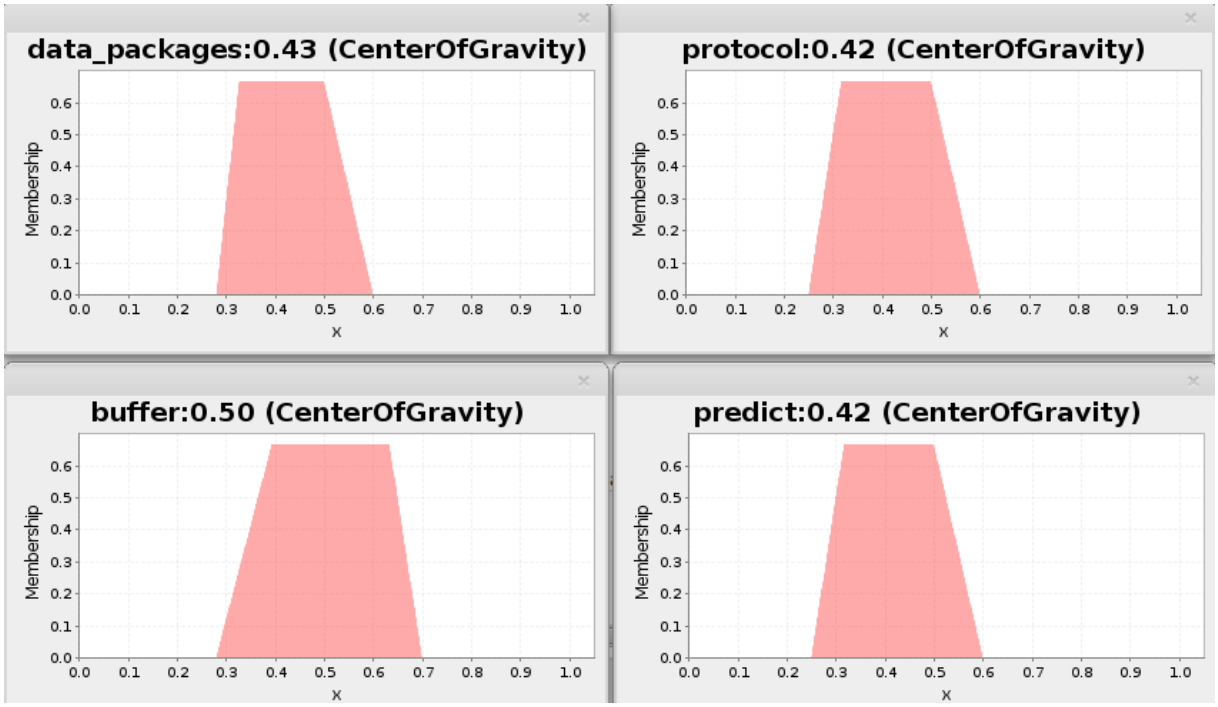


Figura 7.2: Saídas – Simulação 2.

Os resultados dos valores apresentados na Figura 7.2 definem o Alterar quantidade de mensagens em um mesmo pacote de dados como Médio, no caso do TCP, mantendo o tamanho máximo de 1500 bytes, mas agrupando mensagens para serem enviadas por um mesmo pacote; o protocolo de comunicação ficou definido como TCP/UDP, de modo que mensagens de atualização de estados são enviadas utilizando o protocolo UDP, e as mensagens chave são enviadas utilizando o TCP; o valor de Bufferizar como sendo Nível 1, sendo implementado um buffer na origem do pacote, com o intuito de agrupar as mensagens enviadas; e predição de dados como Nível 1, no caso de algumas mensagens UDP não serem entregues ao receptor. Os valores referentes as variáveis de saída são apresentados na Tabela 7.3:

Tabela 7.3: Simulação 2

Variável de Saída	**Valor Fuzzificado**
Alterar quantidade de mensagens em um mesmo pacote de dados	Médio
Protocolo de comunicação	TCP/UDP
Bufferizar	Nível 1
Predizer	Nível 1

A terceira simulação consistiu em avaliar características de conexão com relação à alta latência e variação na entrega de pacotes, com valores de 300ms para atraso na entrega de pacotes, 15ms para variação de entrega de pacotes e 5 para perda de pacotes. A Figura 7.3 apresenta os resultados obtidos.

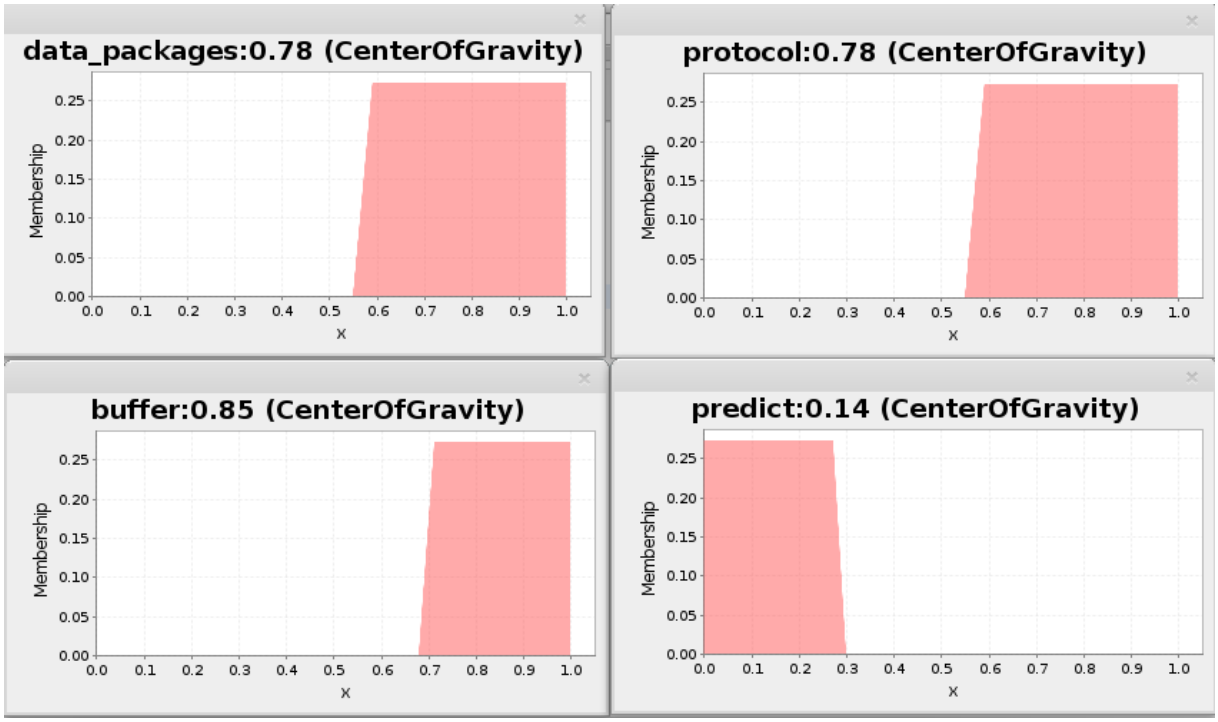


Figura 7.3: Saídas – Simulação 3.

Os resultados dos valores apresentados na Figura 7.3 definem o Alterar quantidade de mensagens em um mesmo pacote de dados como sendo Grande, nesse caso, o tamanho dos pacotes de dados é variável e a quantidade de mensagens enviadas pode ultrapassar, ou não, o tamanho padrão do pacote (1500 bytes); o protocolo de comunicação escolhido foi o SCTP, por prover características de garantia de entrega de pacotes; o Bufferizar como sendo Nível 2, isto é, *buffers* dos dois lados, tanto no envio quanto no recebimento das mensagens; quanto ao Predizer, este foi definido como Não predizer, devido a garantia de entrega do SCTP. Os valores referentes as variáveis de saída são apresentados na Tabela 7.4:

Tabela 7.4: Simulação 3

Variável de Saída	**Valor Fuzzificado**
Alterar quantidade de mensagens em um mesmo pacote de dados	Grande
Protocolo de comunicação	SCTP
Bufferizar	Nível 2
Predizer	Não

O protótipo foi desenvolvido utilizando a biblioteca Fuzzylite (FUZZYLITE, 2016). A linguagem utilizada na modelagem das variáveis de entrada, saída e regras foi a *Fuzzy Control Language* (FCL). A FCL permite o desenvolvimento de aplicações difusas de forma padroni-

zada, reduzindo o período de adaptação requeridos por diversas outras ferramentas, que possuem linguagem própria para as definições de conjuntos de variáveis e regras. O Apêndice B apresenta o arquivo FLC com as definições das variáveis e regras utilizadas nas simulações.

7.4 Estudo de caso 2 – aixCAVE e miniCAVE

Este estudo de caso foi realizado durante o período de doutorado sanduíche que o candidato conduziu no Grupo de Realidade Virtual da RWTH Aachen University durante o peŕiodo de agosto de 2014 à julho de 2015.

Durante o período de doutorado o candidato solicitou uma bolsa de doutorado sanduíche junto ao seu programa de pós–graduação, visto que ele foi aceito como membro do Grupo de Realidade Virtual no Instituto de Computação de Alta Performance da RWTH Aachen University, uma das universidades de excelência em pesquisa da Alemanha. Os interesses de pesquisa comuns e a possibilidade de maior produtividade das atividades ao longo do projeto por meio desta parceria motivaram a solicitação da bolsa para a realização do estágio.

O objetivo principal deste estudo foi realizar estudos relacionados à conexão entre o sistema de multiprojeção implantado no LaVIIC (UFSCar), a miniCAVE (DIAS, 2011), e a aixCAVE, do Grupo de Realidade Virtual da RWTH Aachen University. O propósito geral foi desenvolver um protótipo de um 3DCVE codificado utilizando duas soluções de *softwares* distintas: a libGlass (LIBGLASS, 2016; DIAS *et al.*, 2015; GNECCO *et al.*, 2003; GUIMARAES, 2004) e o Vista VR Toolkit (VISTA, 2016; SCHIRSKI *et al.*, 2003).

7.4.1 Materiais e métodos

Este estudo de caso foi realizado por meio de uma parceria entre o LaVIIC (Laboratório de Visualização Imersiva, Interativa e Colaborativa) - UFSCar, o LIV (Laboratório de Sistemas de Tempo Real) – UNESP e o Grupo de Realidade Virtual da RWTH Aachen University.

Duas diferentes soluções de desenvolvimento foram integradas, a libGlass e o Vista VR Toolkit. O Vista VR Toolkit permite a integração entre tecnologias relacionadas à RV, visualização 3D e científica. A biblioteca tem sido desenvolvida pelo Grupo de Realidade Virtual da RWTH Aachen University nos últimos doze anos, em cooperação com a DLR Braunschweig. O projeto está ativo, sendo utilizado como solução de desenvolvimento as aplicações do aixCAVE.

O principal objetivo foi prover uma solução de *software* que permitisse a interconexão entre o miniCAVE (LaVIIC) e o aixCAVE (RWTH) utilizando a arquitetura apresentada nesta tese.

A Figura 7.4 apresenta o comportamento esperado na comunicação inter–aglomerados gráficos. Em termos gerais, o processamento interno, como troca de mensagens, sincronismo, definição de pontos de vista e tratamento de interação, ficaram a cargo da VistaVRToolkit. No entanto, sempre que a mensagem tivesse de ser enviada a outro aglomerado gráfico remoto, esta tinha de ser enviada utilizando a libGlass. O protocolo de comunicação desenvolvido entre os diferentes locais definiu que toda interação realizada por um usuário em um dos ambientes deveria ser replicada aos outros locais. Assim, se um usuário gerasse um estímulo no miniCAVE (LaVIIC), este tinha de ser aplicado na aixCAVE (RWTH), porém, sempre visando o menor atraso nesta tarefa.

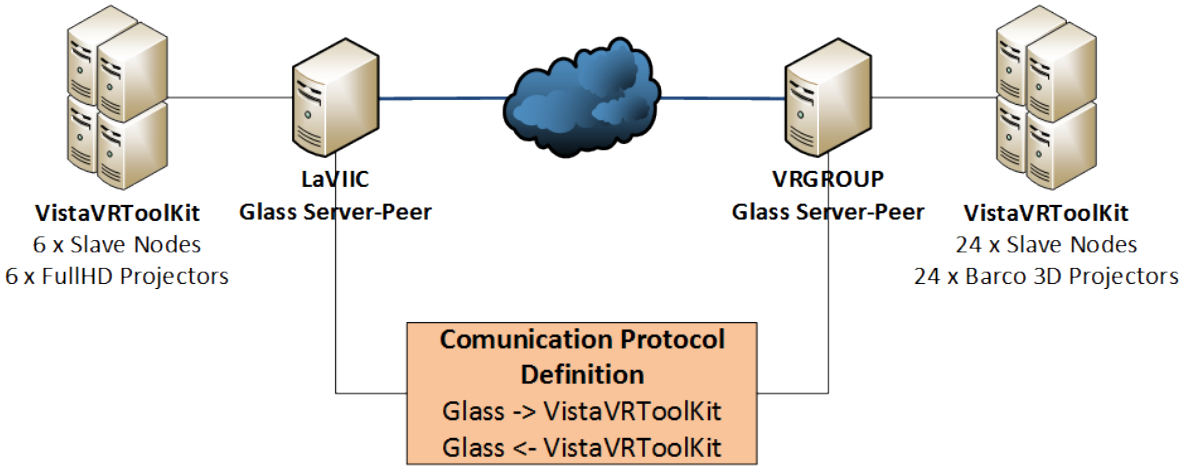


Figura 7.4: Modelo de comunicação – aixCAVE e miniCAVE.

Para que as mensagens pudessem ser convertidas entre as diferentes soluções de desenvolvimento um protocolo de comunicação foi definido. Desta forma, sempre que uma mensagem é enviada entre diferentes aglomerados gráficos, esta deve ser empacotada em um objeto da libGlass. Da mesma forma, quando a mensagem é entregue ao aglomerado gráfico destino, esta é desempacotada em um objeto da Vista VR Toolkit. Os objetos libGlass são definidos como sendo do tipo a ser empacotado, neste caso VistaVector3D e VistaQuaternion ❶. O Código 7.1 descreve este comportamento.

Código 7.1: Definição dos objetos libGlass

```
1
2 ❶ Shared<VistaVector3D, rawPack, rawUnpack> *glassTrans;
3 Shared<VistaQuaternion, rawPack, rawUnpack> *glassRotate;
```

```
4
5 void Handler::InitGlass() {
6  g = new GlassClient(new TCPUDP(tcpport, udpport), host);
7
8  glassTrans = new Shared<VistaVector3D, rawPack,
       rawUnpack>(glassTrans, vistaVec, sizeof(VistaVector3D), Global);
9  glassRotate = new Shared<VistaQuaternion, rawPack,
       rawUnpack>(glassRotate, vistaQuat, sizeof(VistaQuaternion),
       Global);
10 }
11
12 void Handler::HandleEvent( VistaEvent *pEvent ) {
13
14  if (isVistaVRMaster == true) {
15   if (isGlassMaster == true) {
16    if (vistaVec != GetTranslation()) {
17     vistaVec = GetTranslation();
18     *glassTrans = vistaVec;
19     glassTrans→sendUpdate();
20    }
21    if (vistaQuat != GetRotation()) {
22     vistaQuat = GetRotation();
23     *glassRotate = vistaQuat;
24     glassRotate→sendUpdate();
25    }
26   }
27   if (isGlassMaster == false) {
28    glassTrans→getUpdate();
29    glassRotate→getUpdate();
30    vistaVec = glassTrans→getData();
31    vistaQuat = glassRotate→getData();
32    SetTranslation(vistaVec);
33    SetRotation(vistaQuat);
34   }
35  }
36 }
```

O protocolo de comunicação utilizado neste estudo de caso permite que sejam realizadas transformações referentes à translação e rotação de câmera. Assim, dois métodos foram criados para cada uma das transformações (*get* e *set*). O Código 7.2 apresenta os métodos *get* e *set* relacionados à translação.

Código 7.2: Método de transformação – Translação

```
1 VistaVector3D Handler::GetTranslation() {
2  return m_pVirtualPlatform→GetTranslation();
3 }
4
5 void Handler::SetTranslation(VistaVector3D v3Trans) {
6  m_pVirtualPlatform→SetTranslation(v3Trans);
7 }
```

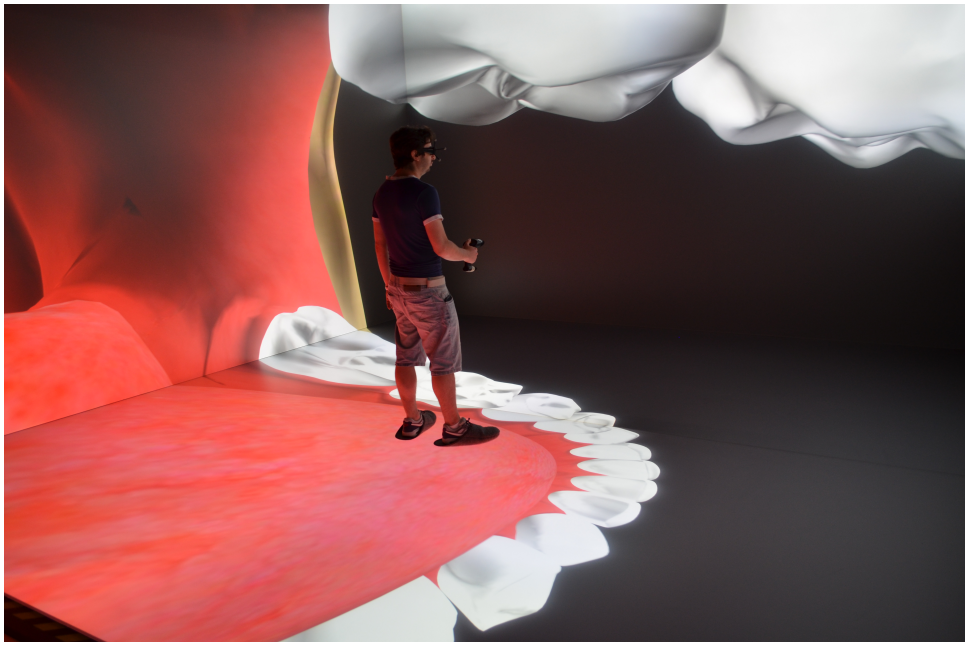
As transformações relacionadas à rotação são tratadas da mesma forma que as translações, diferenciando apenas no tipo de objeto utilizado. O Código 7.3 apresenta os métodos *get* e *set* para as transformações de rotação.

Código 7.3: Método de transformação – Rotação

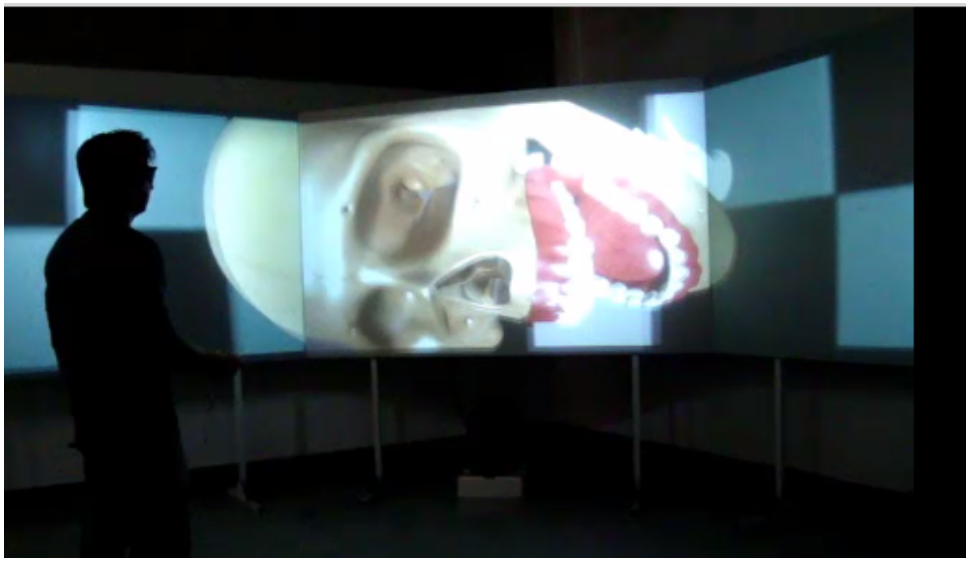
```
1 VistaQuaternion Handler::GetRotation() {
2  VistaQuaternion qRot = m_pVirtualPlatform→GetRotation();
3  return qRot;
4 }
5
6 void Handler::SetRotation(VistaQuaternion q4Rot) {
7  m_pVirtualPlatform→SetRotation(q4Rot);
8 }
```

A diferença básica entre as duas transformações é a forma como elas foram representadas. No caso das translações, foi utilizado um vetor 3D; já para as rotações, uma matriz 4D.

Os ambientes virtuais utilizados diferenciam-se entre si tanto em *layout* físico quanto no poder de processamento. Assim, algumas adequações tiveram de ser realizadas com relação ao posicionamento de câmeras. Toda a configuração do miniCAVE foi feita seguindo como modelo os *scripts* de configuração já utilizados no aixCAVE, contudo, adequações tiveram que ser realizadas com relação à quantidade de telas, computadores (nodos do aglomerado gráfico) e projetores. A Figura 7.5 apresenta os diferentes *layout*.



(a) aixCAVE



(b) miniCAVE

Figura 7.5: Pontos de vista – aixCAVE e miniCAVE.

7.4.2 Protocolo de comunicação

Para que as mensagens pudessem ser convertidas entre as diferentes soluções de desenvolvimento um protocolo de comunicação foi definido. Deste modo, sempre que uma mensagem tem de ser enviada entre os aglomerados gráficos, esta é empacotada em um objeto da libGlass. Da mesma forma, quando a mensagem é entregue ao aglomerado, esta é desempacotada em um objeto da Vista VR Toolkit. Os objetos libGlass foram definidos como sendo do tipo da variável

que deve ser empacotada, neste caso, duas variáveis foram utilizadas: VistaVector3D ❶ e VistaQuaternion ❷. Em termos gerais, sempre que um objeto da Vista VR Toolkit for criado, este tem de ser instanciado também como uma variável Compartilhamento (*Shared*), definida no modo de difusão Global. O termo Global ❸, utilizado no momento da instanciação do objeto, possibilita a troca de mensagens entre processos *peers*, não restrigindo somente à arquitetura local mestre–escravo. O Código 7.4 descreve este comportamento em um protótipo simples.

Código 7.4: Compartilhamento de Câmera – Nodo Mestre

```
1
2 ❶ Shared<VistaVector3D, rawPack, rawUnpack> *glassTrans;
3 ❷ Shared<VistaQuaternion, rawPack, rawUnpack> *glassRotate;
4
5 void Handler::InitGlass() {
6  g = new GlassClient(new TCPUDP(tcpport, udpport), host);
7
8 ❸ glassTrans = new Shared<VistaVector3D, rawPack, rawUnpack>
9 (glassTrans, vistaVec, sizeof(VistaVector3D), Global);
10  glassRotate = new Shared<VistaQuaternion, rawPack, rawUnpack>
11  (glassRotate, vistaQuat, sizeof(VistaQuaternion), Global);
12 }
13
14 void Handler::HandleEvent( VistaEvent *pEvent ) {
15
16  if (isVistaVRMaster == true) {
17   if (isGlassMaster == true) {
18    if (vistaVec != GetTranslation()) {
19     vistaVec = GetTranslation();
20     *glassTrans = vistaVec;
21     glassTrans→sendUpdate();
22    }
23    if (vistaQuat != GetRotation()) {
24     vistaQuat = GetRotation();
25     *glassRotate = vistaQuat;
26     glassRotate→sendUpdate();
27    }
28   }
29   if (isGlassMaster == false) {
30    glassTrans→getUpdate();
31    glassRotate→getUpdate();
32    vistaVec = glassTrans→getData();
33    vistaQuat = glassRotate→getData();
```

```
34    SetTranslation(vistaVec);
35    SetRotation(vistaQuat);
36   }
37  }
38 }
```

Algo importante que teve de ser definido foi a hierarquia dos processo mestres, escravos, clientes and *serverpeers*. Um processo *masterVista* (Vista VR Toolkit) pode não ser, obrigatoriamente, um nodo *masterclientGlass* (libGlass), visto que todo ambiente inter–aglomerados possui um processo *masterVista*, responsável pela troca de mensagens entre os processos de renderização (*slavesVista*); no caso do processo *masterclientGlass*, só deve existir um, visto que este realiza o envio de mensagens com dados atualizados a outros aglomerados gráficos remotos. Já os processos *clientGlass* (libGlass) solicitam as mensagens com os valores atualizados aos seus *serverpeers*, posteriormente, difundindo-as aos processos *slaveVista* por meio do processo *masterVista*. Os processos *serverpeers* são responsáveis por efetuar a comunicação entre os ambientes. A Figura 7.6 apresenta um diagrama de sequência descrevendo este comportamento.

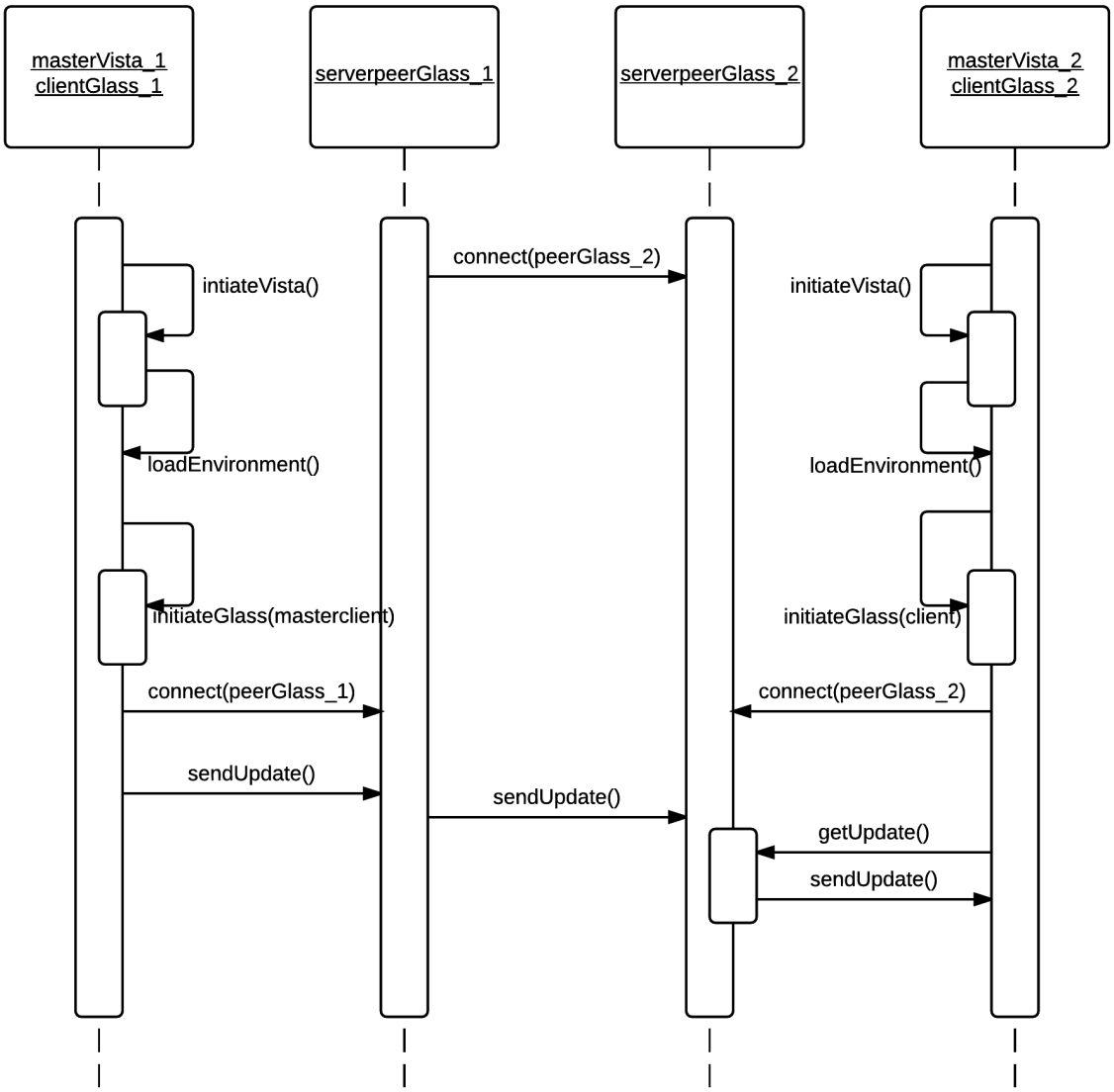


Figura 7.6: Modelo comportamental – VistaVRToolKit e libGlass.

Neste modelo, como a mesma configuração de câmera é compartilhada, apenas um usuário pode interagir por vez. Assim, apenas um dos processos mestre pode realizar transformações. Na Figura 7.6 fica claro este tipo de comportamento no modo como os objetos *clientGlass* são instanciados: o *clientGlass_1*, como um *masterclient* e o *clienteGlass_2*, como um *client*. Quando um usuário desejar interagir com o ambiente, este deve efetuar uma solicitação, que é feita por meio de um evento assíncrono do tipo Evento (*Event*), enviada a todos os processos *clientGlass*. A Figura 7.7 apresenta um diagrama de sequência que exemplifica a requisição de um processo *clientGlass* para se tornar um *masterclient*.

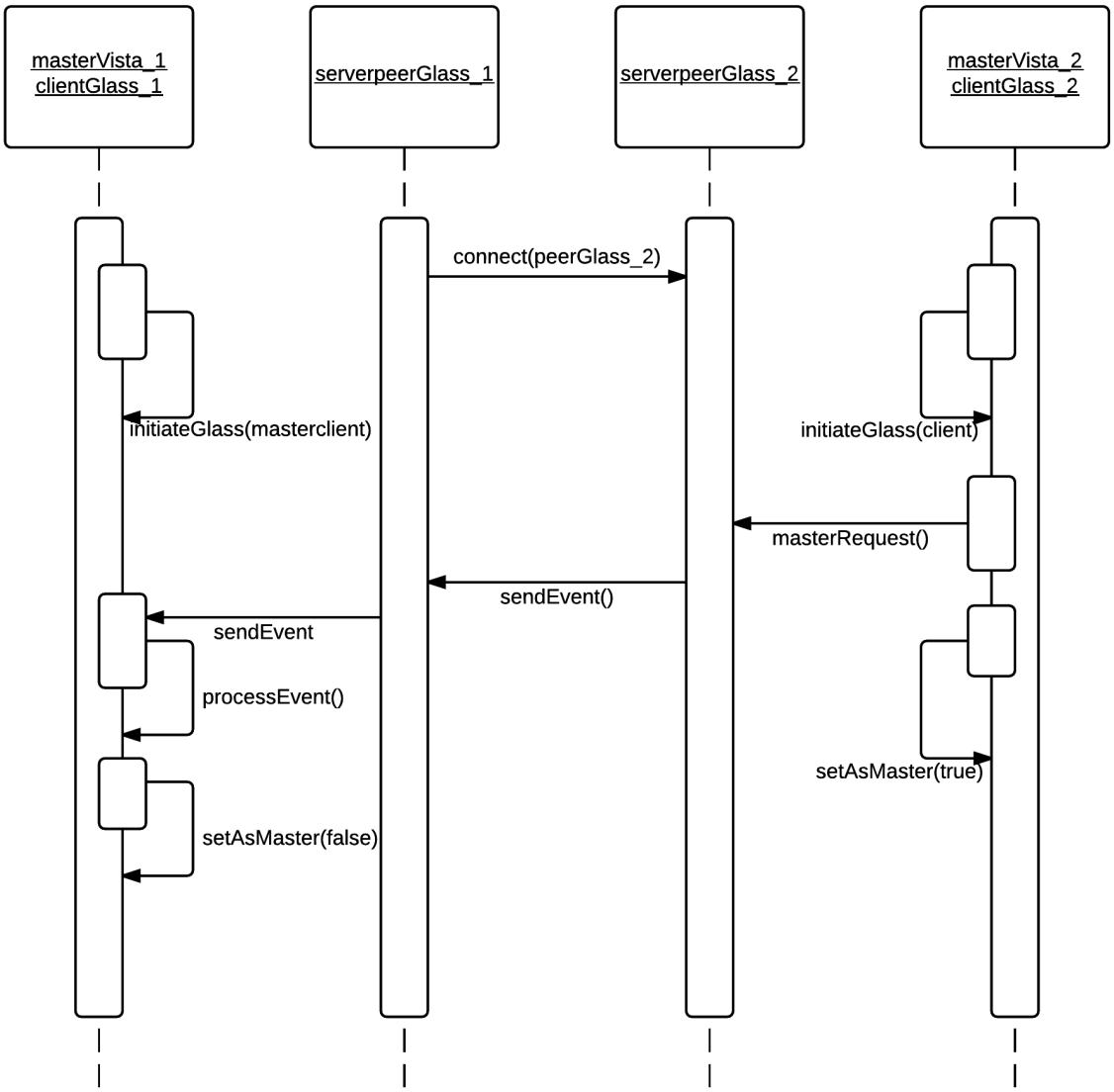


Figura 7.7: Requisição master – libGlass.

Para ser ter o controle sobre as trocas de mensagens entre os processos do aglomerado gráfico local foram utilizados os modelos de *callblacks* providos pela Vista VR Toolkit. Para o nodo *masterClientGlass*, o tipo de evento utilizado foi o VistaSystemEvent::VSE_ POSTAPPLICATIONLOOP. Desta forma, a todo quadro os dados atualizados são enviados a outros aglomerados gráficos, utilizando as variáveis libGlass (glassTrans e glassRotate). Já para o nodo *clientGlass*, o tipo de evento utilizado foi o VistaSystemEvent::VSE_PREAPPLICATIONLOOP. A diferença básica entre as duas transformações são os tipos de dados. No caso das translações, foi utilizado um vetor 3D; já para as rotações, uma matriz 4D.

7.4.3 Latência

Os estudos de caso apresentados nesta seção relacionam taxa de quadros por segundo e as diferentes configurações de rede utilizadas. A taxa de quadros mínima esperada foi de 60 quadros por segundo, a fim de manter qualidade à experiência do usuário.

Neste estudo de caso foi possível identificar a influência de diferentes valores de latência em um 3DCVE provido de barreiras de sincronismo. Foram utilizados alguns valores de latência a fim de emular tanto redes locais quanto redes regionais com infraestrutura baseadas sobre à Internet. A topologia de rede ponto–a–ponto foi utilizada. Com relação aos protocolos de comunicação, três diferentes soluções foram utilizadas: TCP, TCP/UDP e SCTP.

Os valores de configuração da rede de dados utilizados foram os apresentados na Tabela 7.1.

O uso de barreiras de sincronismo é indicado em ambientes distribuídos que necessitem de um alto grau de sincronismo. No caso dos 3DCVEs o sincronismo é de suma importância, principalmente em sistemas de multiprojeção, onde as projeções devem ser geradas ao mesmo tempo, para que não sejam notados atrasos na geração de imagens. Contudo, em se tratando de comunicações inter–aglomerados, o uso de barreiras pode ser ignorado, pelo menos na troca de informações entre os *peers*, visto que a existência de latência pode acarretar em problemas à execução do ambiente. Na Figura 7.8 são apresentados os resultados em relação ao uso de barreiras em um ambiente inter–aglomerados. Duas barreiras foram utilizadas: *datalock* (antes da atualização dos dados) e *framelock* (antes da execução do próximo quadro). A troca de mensagens apresentada neste estudo de caso foi efetuada entre dois *peers*, utilizando os protocolos SCTP, TCP e TCP/UDP. Com este estudo foi possível identificar os requisitos mínimos em relação à influência da latência em ambientes sincronizados.

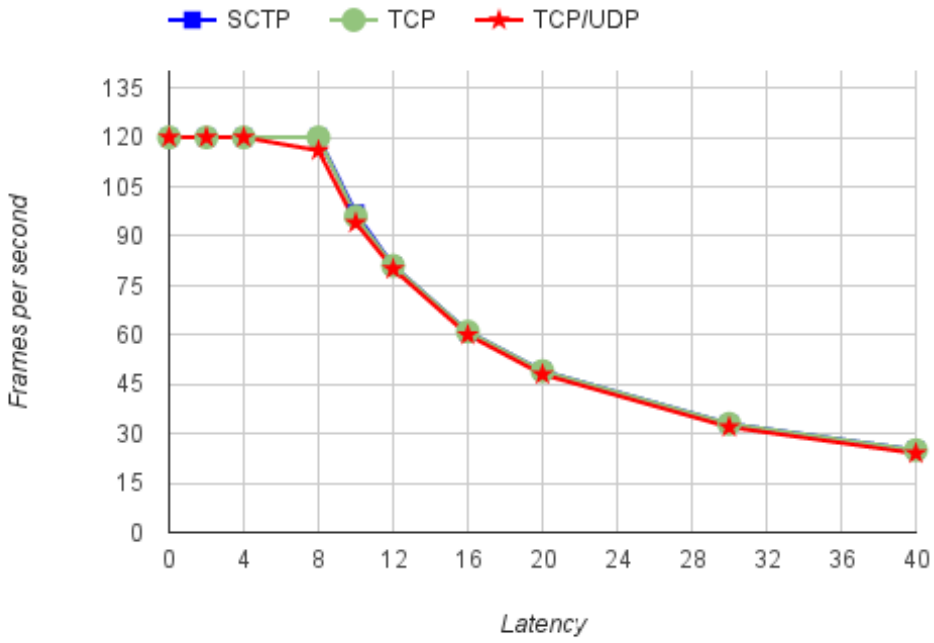


Figura 7.8: Aplicação VistaVRToolKit – latência vs quadros por segundo (com barreiras).

De acordo com a Figura 7.8 a execução do ambiente está dentro dos requisitos esperados com valor de latência de até 16ms, independente do protocolo de comunicação utilizado. Valores acima de 16ms tiveram alto impacto na experiência do usuário, visto que a troca de quadros tornou-se perceptível.

Em ambientes inter–aglomerados, em que não são utilizadas barreiras, a taxa de quadros não é afetada, sendo acrescido apenas o atraso presente na rede. As três soluções de protocolo de comunicação obtiveram resultados semelhantes em um ambiente com variações apenas de latência.

Um estudo de caso foi realizado a fim de avaliar apenas a latência. As execuções foram realizadas variando apenas os valores de latência apresentados na Tabela 7.1. A Figura 7.9 apresenta os valores obtidos em uma arquitetura ponto–a–ponto para os três protocolos de comunicação. No entanto, neste estudo, não foi aplicado o uso de barreiras à comunicação inter–aglomerados, somente no modelo intra–aglomerado, visando manter o sincronismo na geração de imagens.

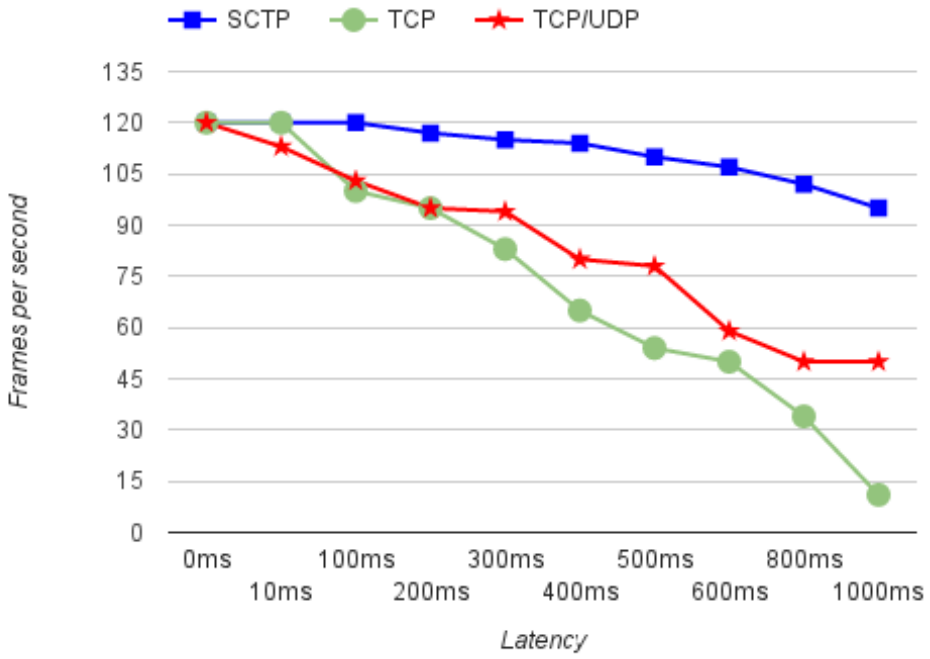


Figura 7.9: Aplicação VistaVRToolKit – latência vs quadros por segundo (sem barreiras).

Em relação a Figura 7.9 pode-se afirmar que o SCTP obteve melhor comportamento na presença de altos valores de latência, principalmente em relação ao protocolo TCP. O protocolo TCP/UDP apresenta melhor performance perante o protocolo TCP, contudo, com valores de latência acima de 600ms. Apesar de manter a taxa de quadros acima dos valores obtidos pelo TCP, o ambiente se tornou inconsistente, praticamente impossível de ser utilizado.

7.4.4 Tempo de variação na entrega de pacotes

Neste estudo de caso, apenas a variação de tempo de entrega de pacotes foi levada em consideração. A Figura 7.10 apresenta os valores de quadros por segundo utilizando as diferentes variações.

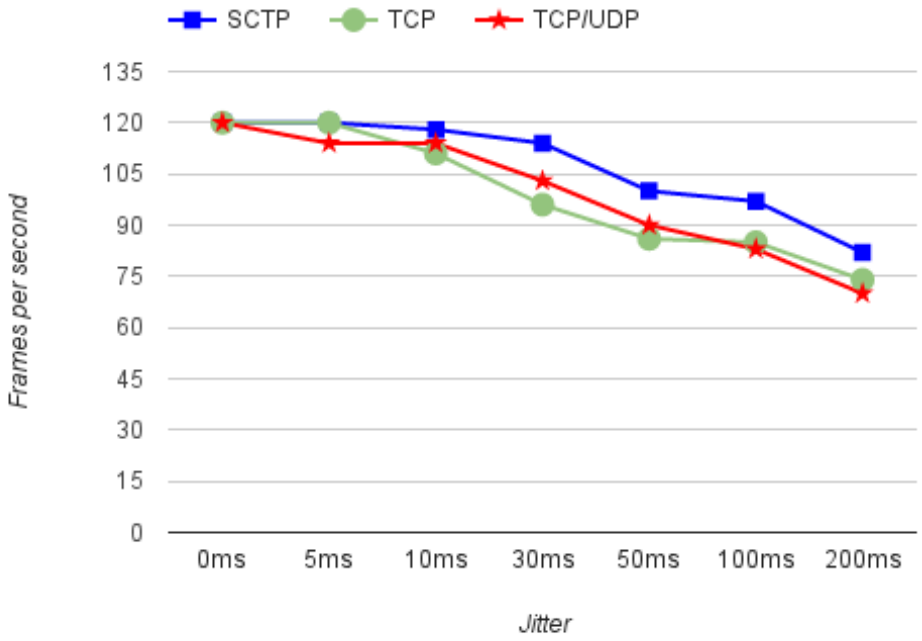


Figura 7.10: Aplicação VistaVRToolKit – variação de tempo na entrega de pacotes vs quadros por segundo.

Novamente, como no estudo referente à latência, o protocolo de comunicação SCTP obteve melhor resultado. Entretanto, vale ressaltar que a variação de tempo na entrega de pacotes foi prejudicial aos três protocolos de comunicação. Em ambientes com altas variações o protocolo TCP/UDP tende a ser o mais prejudicado, visto que os pacotes UDP passam a ser entregues fora de ordem, o que pode ocasionar inconsistências ao ambiente, principalmente em pacotes relacionados à atualização de estados.

7.4.5 Perda de pacotes

Este estudo de caso consistiu em verificar a quantidade de pacotes perdidos durante um período de conexão pré–definido. Foi avaliada a perda de pacotes entre os processos peers, visto que a perda e atraso de pacotes não é um problema encontrado em ambientes intra–aglomerado.

De modo a verificar o comportamento dos ambientes virtuais na ocorrência de perda de pacotes, alguns pacotes foram descartados, exemplificando assim um modelo de perda de pacotes simples.

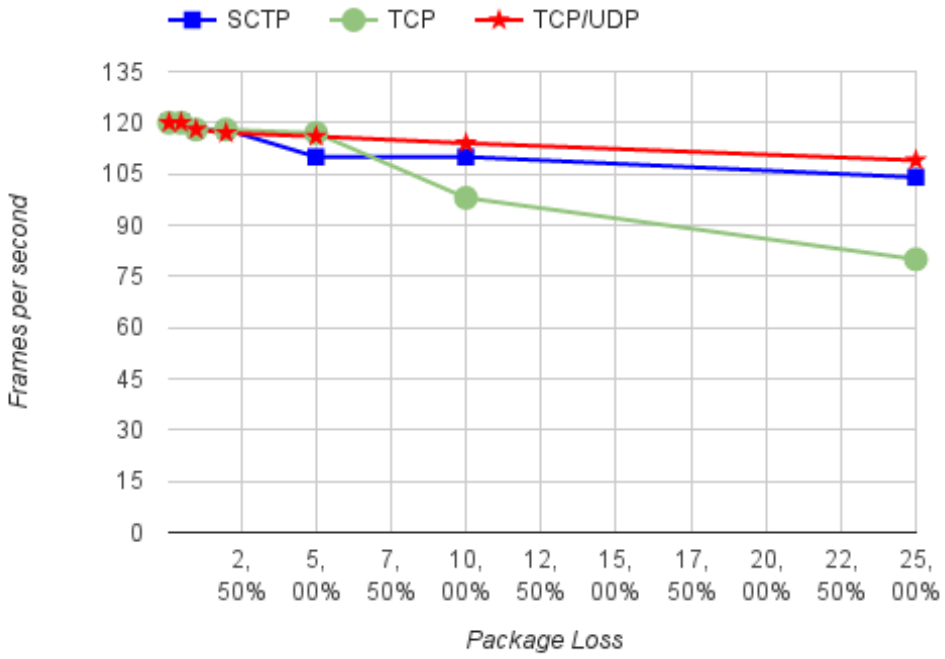


Figura 7.11: Aplicação VistaVRToolKit – perda de pacotes vs quadros por segundo.

De acordo com a Figura 7.11 o protocolo de comunicação TCP/UDP foi o que obteve melhor resultado, impactando menos na taxa de quadros do ambiente. Contudo, isso acontece devido a uma das principais características do UDP: o não reenvio de pacotes perdidos. Já o protocolo de comunicação TCP obteve um resultado não satisfatório quando utilizado em um ambiente com perda acima de 10%. O protocolo de comunicação SCTP obteve melhor desempenho se comparado ao TCP, mas pior do que o TCP/UDP devido ao *buffer* implementado.

7.4.6 Latência, variação no tempo de entrega de pacotes e perda de pacotes

As três principais adversidades relacionadas a redes de dados dificilmente aparecem isoladamente em ambientes reais. Assim sendo, nesta subseção foram definidos alguns valores combinando as adversidades. Apenas valores encontrados em redes de dados baseadas em Internet foram avaliados, visto que ambientes intra–aglomerado não são propícios a estes tipos de adversidades. A Tabela 7.5 apresenta as configurações.

Tabela 7.5: Adversidades

Configuração	Latência	Variação	Perda
Configuração 1	100ms	10ms	1%
Configuração 2	200ms	15ms	3%
Configuração 3	400ms	10ms	0,1%
Configuração 4	500ms	20ms	1%

Na Figura 7.12 pode-se notar que o protocolo de comunicação SCTP obteve resultados melhores do que os outros protocolos. Novamente, o protocolo TCP/UDP obteve os piores resultados, devido a ordem de entrega dos pacotes, problemas provenientes do uso do UDP; o TCP puro obteve um bons resultados, mas ainda sim piores que os do SCTP.

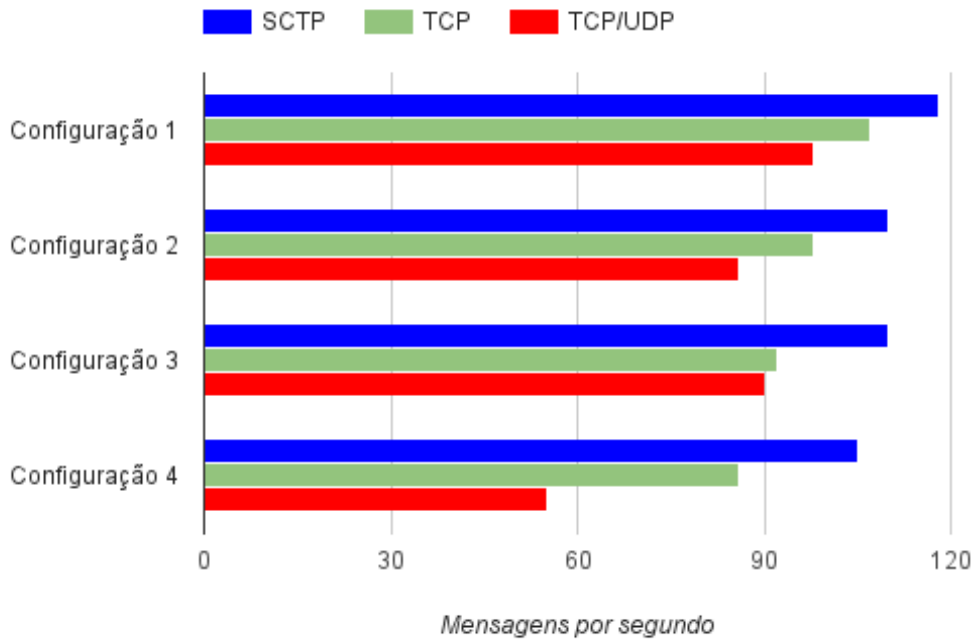


Figura 7.12: Adversidades vs mensagens por segundo.

7.5 Estudo de Caso 3 – RehabGesture: uma ferramenta alternativa para a mensuração do movimento humano

Esta subseção apresenta o desenvolvimento do módulo de inter–comunicação do aplicativo RehabGesture, a fim de torná–lo uma solução colaborativa sobre à Internet. O RehabGesture foi desenvolvido por meio de um estudo multidisciplinar no LaVIIC, no departamento de Ciência da Computação. O estudo foi aprovado pelo conselho de ética de pesquisas em humanos, com

o número de processo 11319712.4.0000.5504, vínculado ao Centro de Ciência e Tecnologia – UFSCar. O desenvolvimento do RehabGesture foi possível por meio da solução desenvolvida nesta tese. Os componentes utilizados foram: Evento, Compartilhamento e Barreiras.

O estudo dos movimentos angulares dos segmentos do corpo demanda parâmetros cinemáticos. A coordenação de um indivíduo permite o movimento preciso, principalmente por meio de contração dinâmica (isotónica), em que a força desenvolvida pelo músculo é maior ou menor do que a resistência. Esta força permite o controle do movimento contra a gravidade (contração concêntrica) ou com a gravidade (ação muscular excêntrica). Ele retorna medidas da aplitude de movimento das articulações dos ombros e cotovelos do plano coronal.

O processo de reabilitação realizado em um hospital é a atividade mais desejada para uma pessoa com um impedimento motor, entretanto, isso pode não ser sempre alcançado. Nestes casos, a telereabilitação pode ser utilizada. Um dos principais objetivos da telereabilitação é proporcionar uma continuação do processo de reabilitação do paciente, independentemente do lugar onde este esteja, por exemplo, em sua casa. Hoje em dia, as tecnologias de comunicação são capazes de realizar diversos serviços, tais como transferência de dados de áudio/vídeo e gráficos em tempo real sem grandes problemas (RIZZO *et al.*, 2011).

A fim de permitir que usuários possam realizar atividades colaborativas sobre à aplicação RehabGesture, a arquitetura colaborativa foi utilizada para desenvolver um módulo de sistema de telepresença, baseando-se em um modelo cliente–servidor.

A Figura 7.13 apresenta a interface do RehabGesture, incluindo os valores de medida de amplitude de movimento em graus, mostrados dentro dos círculos nas articulações do cotovelo e ombro. O logotipo RehabGesture é representado por uma imagem figurativa (um ângulo de 90°) no canto superior esquerdo da figura. O RehabGesture está registrado no INPI (Instituto Nacional de Propriedade Industrial / Brasil), sob o processo número 909054541/2015 e registro de *software*, também no INPI, sob processo número BR 51 2015 000130 2.

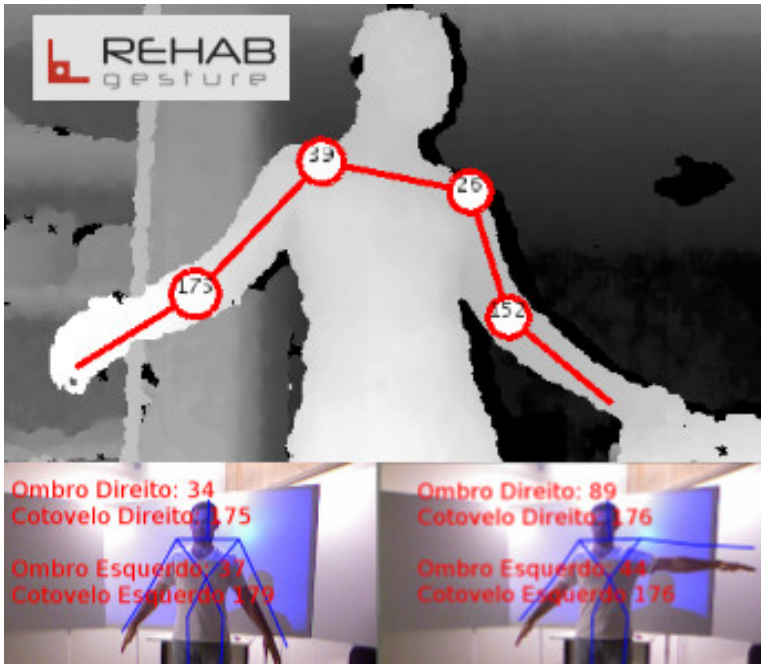


Figura 7.13: Interface do RehabGesture.

O *software* permite a medição de amplitude de movimento (no plano coronal) desde 0° de extensão a 145° de flexão da articulação do cotovelo, e de 0° de adução a 180° de abdução do glenoumeral (ombro) conjunta, deixando a posição de pé. O movimento de abdução passa por três etapas: 1ª etapa - 0° a 60° é realizada exclusivamente pela articulação do ombro ; 2ª etapa - 60° a 120° é realizada pela articulação do ombro com a participação da joint escapulotorácica; e a 3ª etapa - 120° a 180° combina inclinação do tronco com as demais articulações mencionadas acima.

A Figura 7.14 apresenta o modelo de conexão que fornece a colaboração entre os dois locais diferentes. O Ambiente 1 executa uma instância do RehabGesture, definida como um servidor (paciente), que fornece valores atualizados gerados pelo RehabGesture a outros ambientes. O Ambiente 2 também executa uma instância de RehabGesture, que é definido como um cliente (especialista em saúde). O ambiente apresentado na Figura 7.14 ilustra o modelo em que toda atividade realizada pelo paciente durante a sua telereabilitação (Ambiente 1) é monitorada por um especialista utilizando Ambiente 2. Mensagens síncronas foram utilizadas no desenvolvimento do RehabGesture, por isso, cada vez que o Ambiente 1 envia uma mensagem para o Ambiente 2, este a requisita e consome no mesmo quadro. Barreiras de sincronismo garantem este comportamento.

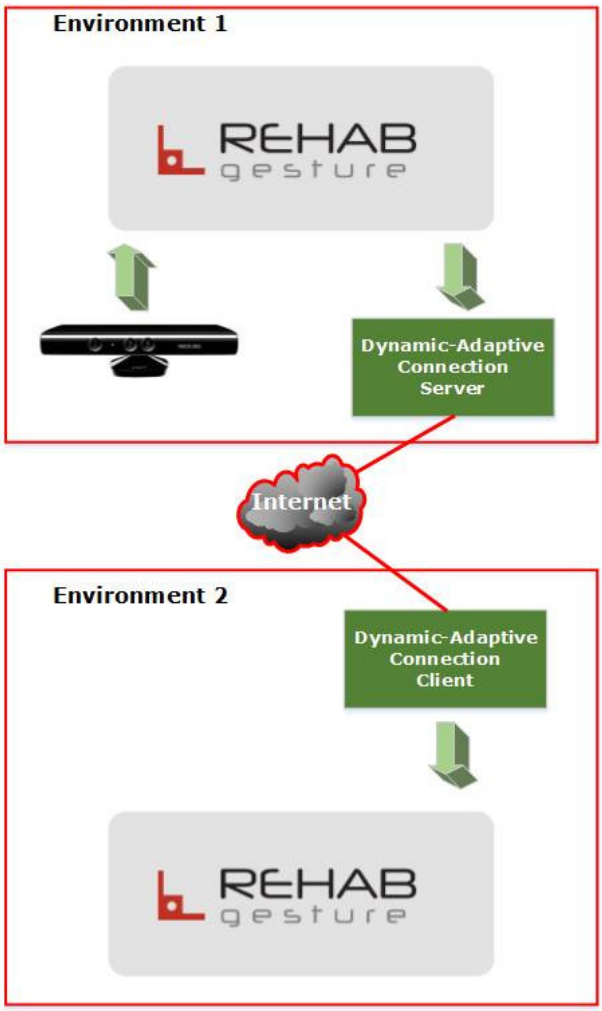


Figura 7.14: Modelo de conexão desenvolvido no RehabGesture.

A sequência de inicialização necessária para ambos os ambientes é apresentada na Figura 7.15. De acordo com o diagrama de sequência, o paciente inicia o Ambiente 1 (servidor), de modo que o RehabGesture inicializa o módulo de rastreamento e o módulo de conexão. A partir deste momento, todas as interações gerada pelo paciente são enviadas a os outros ambientes, neste caso o Ambiente 2, que é inicializado pelo especialista (cliente), podendo monitorar a atividade do paciente.

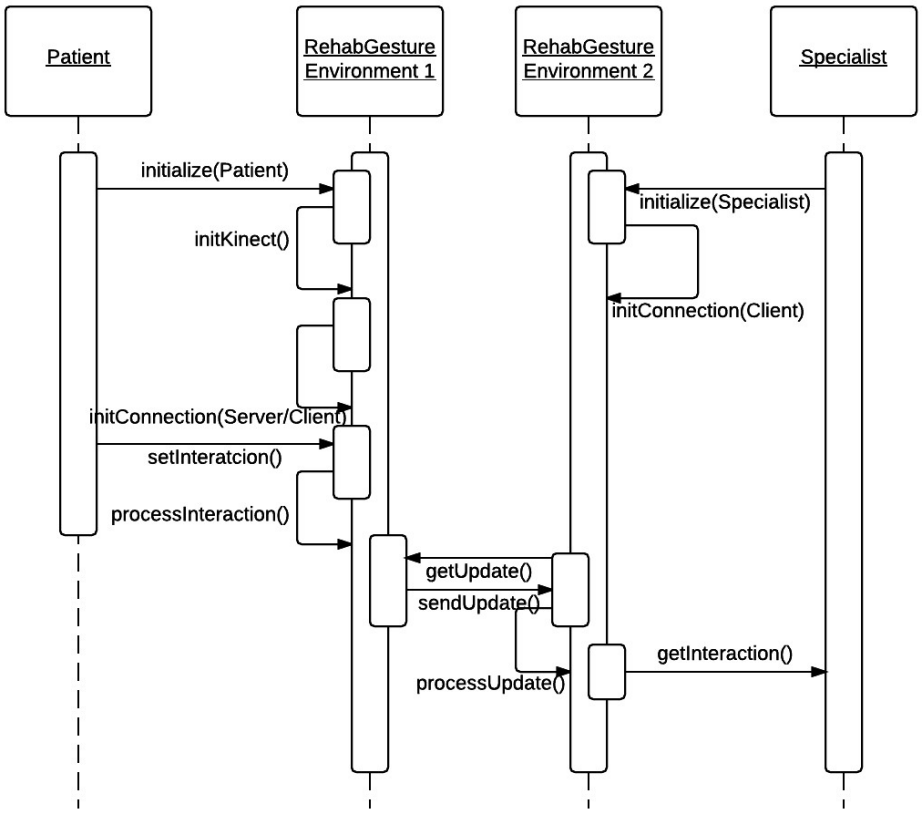


Figura 7.15: Diagrama de sequência para o modelo de telereabilitação do RehabGesture.

O Ambiente 1 envia mensagens atualizadas a cada quadro, mantendo os dados da sessão, se o especialista assim desejar (não é padrão). Da mesma forma , em cada quadro, o Ambiente 2 solicita dados atualizados ao Ambiente 1. O RehabGesture renderiza o ambiente virtual em uma taxa de 30 quadros por segundo.

7.6 Estudo de Caso 4 – testes de integração automáticos

Esta seção apresenta a definição e execução de testes de integração automáticos desenvolvidos a fim de validar e depurar os módulos e funcionalidades desenvolvidas nesta tese. Teste de integração é toda a aplicação de teste nas assinaturas de entradas e saídas de um sistema combinando seus módulos, que consiste em análisar dados válidos e inválidos via entrada/saída sendo aplicado por desenvolvedores ou analistas de teste.

A biblioteca *Boost Test* foi utilizada no desenvolvimento dos testes. Ela prove um conjunto de componentes para o desenvolvimento de casos de teste, que podem ser controlados em tempo de execução.

A execução dos testes é feita por meio da chamada BOOST_TEST_LOG_LEVEL=all [test-name]. Desse modo, os testes são executados de um modo verboso, apresentando todas as informações referentes à execução. Ao final do teste, se tudo ocorrer como esperado, nenhuma mensagem de erro será informada no log de execução.

O Código 7.5 apresenta as *includes* básicos necessários aos testes automáticos. A linha 6 é responsável por fazer o *include* do *unit_test*.

O ❶ (#define NUM_LOOPS 50) define a quantidade de vezes que o teste é executado. Este tipo de repetição pode auxiliar na identificação de erros e falhas na execução, visto que se a aplicação for executada apenas uma vez, o erro/falha pode não ocorrer. Por exemplo, falhas relacionadas à condição de corrida. O ❷ (#dfine TOTAL_CLIENTS 100) define a quantidade de processos clientes que serão instanciados. Cada cliente será instanciado em uma *thread*. Dessa forma, é possivel realizar testes com uma grande quantidade de processos clientes por *host*. O ❸ apresenta as definições de duas barreiras. Elas são utilizadas para que seja possível esperar o tempo de instancição de cada processo cliente.

Código 7.5: Teste de integração automático – mensagem síncrona

```
1 #include <iostream>
2 #include <unistd.h>
3 #define BOOST_TEST_DYN_LINK
4 #define BOOST_TEST_MODULE shared
5 #include <boost/test/unit_test.hpp>
6 #include
7
8 ❶ #define NUM_LOOPS 50
9 ❷ #define TOTAL_CLIENTS 100
10
11 using namespace libglass;
12
13 ❸ boost::barrier b1(TOTAL_CLIENTS), b2(TOTAL_CLIENTS);
14 unsigned int peerport = 1234;
```

No Código 7.6 o processo servidor é instanciado ❶, definindo a classe de protocolo (PROTOCOLCLASS) utilizada (TCP, TCP/UDP ou SCTP), que é passada como parâmetro na execução do teste.

Código 7.6: Teste de integração automático – *Thread* servidor

```
1 void first_server_thread() {
2       GlassServerPeer  *g;
```

```
3
4       BOOST_TEST_MESSAGE(                 );
5
6       try {
7             ❶ g = new GlassServerPeer(new PROTOCOLCLASS(1234));
8             BOOST_CHECK(g != NULL);
9       }
10      catch (Exception &e) {
11            BOOST_FAIL(                               + e.getMessage());
12            return;
13      }
14      delete g;
15      BOOST_TEST_MESSAGE(                 );
16
17      return;
18 }
```

No Código 7.7, um processo *peer* é instanciado ❶. Diferente do processo servidor, além da porta padrão para a criação do *socket*, também são passados como parâmetros o ip e a porta do *peer* ❷ que este deve-se conectar no momento da instanciação.

Código 7.7: Teste de integração automático – *Thread peer*

```
1 void peer_thread() {
2       peerport++;
3       ❶ GlassServerPeer  *g;
4
5       BOOST_TEST_MESSAGE(               );
6
7       try {
8             ❷ g = new GlassServerPeer(new PROTOCOLCLASS(peerport),
                              , 1234, true, true);
9             BOOST_CHECK(g != NULL);
10      }
11      catch (Exception &e) {
12            BOOST_FAIL(                             + e.getMessage());
13            return;
14      }
15      delete g;
16      BOOST_TEST_MESSAGE(               );
17
18      return;
19 }
```

No Código 7.8 são instanciadas todas as *threads* ❶. As chamadas BOOST_TEST_MESSAGE ❷ são utilizadas para gerar o *log* de execução do teste.

Código 7.8: Teste de integração automático do – Caso de teste Boost

```
1 BOOST_AUTO_TEST_CASE(Case) {
2       unsigned int clientnum = 5; // amount of clients
3
4       // open server
5       ❶ boost::thread server(&first_server_thread);
6       sleep(4);
7       ❶ boost::thread peer1(&peer_thread);
8       sleep(2);
9
10      boost::thread_group clients;
11      unsigned int i;
12
13      sleep(2);
14      unsigned int port = 1234;
15      for (i = 0; i < clientnum; i++) {
16            ❶ clients.create_thread(boost::bind(client_thread, i,
                  port, false));
17      }
18
19      port = 1235;
20      for (i = 0; i < clientnum; i++) {
21            ❶ clients.create_thread(boost::bind(client_thread, i+5,
                  port, true));
22      }
23
24      sleep(2);
25      ❷ BOOST_TEST_MESSAGE(            << clientnum);
26      clients.join_all();
27      ❷ BOOST_TEST_MESSAGE(        );
28 }
```

As *threads* clientes são criadas de acordo com o tipo de *plugin* que será testado. Portanto, a função é definida por **void client_thread(unsigned int id, unsigned int port, bool isMaster)** ❶ para todos os testes. No Código 7.9 a função *client_thread* ❶ recebe como parâmetro o id processo, a porta que este deverá utilizar para criar o seu *socket* (porta atribuída pelo processo

servidor), e um objeto booleano para definir se o processo cliente é um *master* ou não. O processo *master* é o responsável por gerar estímulos no ambiente distribuído. No ❷ o objeto *GlassClient* é declarado. Uma barreira é definida no ❸.

Código 7.9: Teste de integração automático – *Thread* cliente

```
1 ❶ void client_thread(unsigned int id, unsigned int port, bool
    isMaster) {
2       ❷ GlassClient *g;
3       ❸ Barrier *b;
4
5       BOOST_TEST_MESSAGE(                                          );
6
7       try {
8             std::ostringstream oss;
9             oss << id;
10
11            nodename =        + oss.str();
12            g = new GlassClient(new PROTOCOLCLASS(port),            ,
                  nodename);
13            BOOST_REQUIRE(g != NULL);
14      }
15      catch (Exception &e) {
16            BOOST_FAIL(                                + e.getMessage());
17            return;
18      }
19      BOOST_TEST_MESSAGE(          << boost::this_thread::get_id() <<
                              << (void *)g);
20
21      try {
22            b = new Barrier(0, 0, true);
23      } catch (Exception &e) {
24            BOOST_FAIL(                                  +
                  e.getMessage());
25            return;
26      }
27
28      .
29      .
30      .
31
32      delete g;
33      BOOST_TEST_MESSAGE(                 );
```

```
34
35        return;
36 }
```

7.6.1 Variáveis síncronas

No Código 7.10, como esse teste está relacionado as variáveis síncronas, um objeto do tipo Compartilhamento é declarado ❶. O tipo de dados compartilhado pelo objeto é *int*.

Depois de instanciar os objetos ❷ um novo valor é atribuído ao objeto sharedint ❸. O processo *master* envia os dados atualizados ao processo servidor ❹. Se o processo não for um *master*, ele recebe os dados atualizados ❺.

Código 7.10: Teste automático do *plugin* Compartilhamento – *Thread* cliente

```
1  void client_thread(unsigned int id, unsigned int port, bool isMaster) {
2        .
3        .
4        .
5
6        ❶ Shared<int, rawPack, rawUnpack> *sharedint;
7        std::string nodename;
8
9        try {
10             ❷ sharedint = new Shared<int, rawPack, rawUnpack>
                   (           , 0, 1, Global);
11       }
12       catch (Exception &e) {
13             BOOST_FAIL(                              + e.getMessage());
14             return;
15       }
16
17       ❸ *sharedint = 0;
18       BOOST_REQUIRE(sharedint→getData() == 0);
19
20       if (g→isMaster() && isMaster) {
21             BOOST_TEST_MESSAGE(nodename <<            );
22       }
23
24       b1.wait();
25
26       for (int i = 0; i < NUM_LOOPS; i++) {
```

```
27              if (g→isMaster() && isMaster) {
28                    BOOST_TEST_MESSAGE(nodename <<
                              );
29                    *sharedint = i;
30                    ❹ sharedint→sendUpdate();
31                    sleep(2);
32              }
33
34              b1.wait();
35              b→sync();
36
37              ❺ sharedint→getUpdate();
38
39              sleep(2);
40              b1.wait();
41
42              BOOST_CHECK_MESSAGE(sharedint→getData() == i,
                                          << sharedint→getData() <<
43                                      << i <<                 <<
                              nodename);
44
45              b2.wait();
46        }
47
48        delete b;
49        delete sharedint;
50        delete g;
51        BOOST_TEST_MESSAGE(                 );
52
53        return;
54  }
```

7.6.2 Eventos assíncronos

A execução dos testes para os outros componentes, tal como o Evento, é feita de forma semelhante, apenas utiilizando o envio de mensagens assíncronas, ao invés de mensagens síncronas.

No Código 7.11 um objeto do tipo Evento é declarado ❶. O tipo de dados compartilhado pelo objeto é *int*. Depois de instanciar os objetos ❷ um novo valor é adicionado à fila do objeto eventint ❸, que é enviado ao servidor/*peer*. Se o processo não for um *master*, este analisa se o

objeto não está vazio. O processo cliente recebe o valor enviado pelo processo *master* ❹.

Código 7.11: Teste automático do *plugin* Evento – *Thread* cliente

```
1  void client_thread(unsigned int id, unsigned int port, bool isMaster) {
2        .
3        .
4        .
5
6     ❶ Event<int, rawPack, rawUnpack> *eventint;
7        std::string nodename;
8
9        try {
10             ❷ eventint = new Event<int, rawPack, rawUnpack>
                     (           , NULL, Global);
11       }
12       catch (Exception &e) {
13             BOOST_FAIL(                              + e.getMessage());
14             return;
15       }
16
17       if (g→isMaster() && isMaster) {
18             BOOST_TEST_MESSAGE(nodename <<             );
19       }
20
21       sleep(1);
22
23       BOOST_MESSAGE(          );
24       BOOST_REQUIRE_MESSAGE(totalclients+1 == g→getTotalNodes(),
                        << totalclients+1 <<          <<
            g→getTotalNodes());
25
26       BOOST_REQUIRE(eventint→isEmpty() == true);
27
28       b1.wait();
29
30       for (int i = 0; i < NUM_LOOPS; i++) {
31             if (g→isMaster() && isMaster) {
32                   BOOST_TEST_MESSAGE(              << i);
33                   ❸ bool b = eventint→enqueueEvent(i, everybodyId, 1,
                        true);
34                   sleep (2);      //o bastante para o evento ser enviado
35                   BOOST_REQUIRE(b == true);
36             }
```

```
37
38              b1.wait();
39              b→sync();
40
41              BOOST_CHECK_MESSAGE(eventint→isEmpty() == false,
                                << nodename);
42              BOOST_CHECK_MESSAGE(eventint→queueSize() == 1,
                                << nodename);
43
44              try {
45                    ❹ int e = eventint→getEvent();
46                    sleep (2);      //o bastante para o evento ser recebido
47                    BOOST_CHECK_MESSAGE(e == i,
                           << e <<
48                                                  << i <<                 <<
                                    nodename);
49              }
50              catch (Exception e) {
51                    BOOST_FAIL(                          <<
                         e.getMessage());
52              }
53
54        }
55
56        b2.wait();
57
58        delete b;
59        delete eventint;
60        delete g;
61        BOOST_TEST_MESSAGE(                 );
62
63        return;
64  }
```

7.6.3 Barreiras de sincronismo

No Código 7.12, como esse teste está relacionado as barreiras de sincronismo, um objeto *Barrier* é declarado ❶. No ❷ o objeto b1 é instanciado como sendo uma barreira global, isto é, uma barreira que não apenas sincroniza os clientes e servidor, mas também os *peers*. No ❸ é efetuada a chamada *b1->sync()*, responsável pelo sincronismo. Somente quando todos

os processos que tenham esta barreira instanciada chegarem a este ponto é que a execução continuará.

Código 7.12: Teste automático do *plugin* Barrier – *Thread* barreira

```
1  void barrier_thread(unsigned int id, unsigned int port, int clientnum)
       {
2          GlassClient *g;
3          ❶ Barrier *b1;
4
5          unsigned int i;
6
7          BOOST_TEST_MESSAGE(          << id <<
                              );
8
9          try {
10               std::ostringstream oss;
11               oss << id;
12
13               g = new GlassClient(new PROTOCOLCLASS(port),            );
14
15               BOOST_REQUIRE(g != NULL);
16         }
17         catch (Exception &e) {
18               BOOST_FAIL(                              + e.getMessage());
19               return;
20         }
21
22         BOOST_TEST_MESSAGE(                  );
23
24               try {
25                     boost::mutex::scoped_lock l(m);
26                     g→setCurrent();
27                     BOOST_TEST_MESSAGE(                    );
28                     ❷ b1 = new Barrier(1, 0, false, Global);
29                     BOOST_CHECK(b1 != NULL);
30               }
31               catch (Exception e) {
32                     BOOST_FAIL(           << i <<
                                 + e.getMessage());
33               }
34
35
```

```
36              BOOST_TEST_MESSAGE(          << id);
37              sleep(2);
38              ❸ b1→sync();
39              BOOST_TEST_MESSAGE(           << id);
40              sleep(2);
41
42      delete b1;
43          delete g;
44          BOOST_TEST_MESSAGE(                );
45          return;
46  }
```

7.7 Considerações finais

Este capítulo apresentou resultados experimentais quanto à influência de redes de dados na concepção de 3DCVEs baseados em aglomerados gráficos. Foram apresentados resultados experimentais relacionados ao modelo de conexão dinâmico–adaptável proposto e a vazão de dados em ambientes distribuídos relacionada à taxa de quadros por segundo.

Como já esperado, o protocolo de comunicação SCTP apresentou melhorias à comunicação inter–aglomerados, independente das adversidades simuladas. No entanto, estudos de caso empíricos devem ainda ser realizados, com o intuito de avaliar a experiência do usuário nos 3DCVEs entre diferentes locais.

Os estudos de casos relacionados à banda de dados não tiveram valores significativos, visto que a quantidade de mensagens trocada entre os aglomerados é otimizada, isto é, apenas valores atualizados são enviados e não cópias de objetos, por exemplo. Este é um dos motivos da solução permitir que os ambientes mantenham a característica de baixa latência mesmo em redes de dados com largura de banda limitada.

Capítulo 8

DISCUSSÕES FINAIS

Os benefícios pertinentes ao suporte de atividades colaborativas para as aplicações de RV são essenciais em uma sociedade globalizada, onde cada vez mais a competitividade e a produtividade são fatores determinantes para o sucesso das empresas. Essa tese visou buscar soluções para essa necessidade atual.

Uma revisão sistemática foi realizada com o intuito de elencar os principais problemas e soluções relacionados à influência das redes de dados na concepção de 3DCVEs baseados em aglomerados gráficos remotos. Por meio desta, foi possível descobrir quais as principais topologias e protocolos de redes utilizados no desenvolvimento de 3DCVEs. A partir dos resultados encontrados durante a etapa de revisão foi possível propor um modelo de conexão para o desenvolvimento de 3DCVEs. Outro benefício da condução da revisão sistemática foi a sua publicação em um evento de alto impacto. Desta forma, outros autores poderão se beneficiar deste estudo.

Uma arquitetura dinâmica e adaptável de conexão foi proposta, desenvolvida e avaliada por meio de estudos de caso. O termo utilizado não existe na literatura, porém, é esperado da arquitetura um comportamento dinâmico, no que tange as modificações durante o tempo de execução das aplicações; e a característica de adaptabilidade, permitindo que as conexões sejam modificadas com o intuito de possibilitar a adaptação das aplicações em relação as adversidades ocasionadas pelas redes de dados. Os intervalos definidos na Seção 5.8 foram encontrados na literatura. Porém, experimentos foram realizados com o intuito de corroborar com os resultados já publicados por outros pesquisadores.

A avaliação da arquitetura foi realizada por meio de implementações sobre a biblioteca de desenvolvimento de aplicações distribuídas libGlass, a qual o candidato é um dos desenvolvedores. É esperado que desenvolvedores e pesquisadores possam utilizar as novas funcionalidades

da biblioteca, de modo que o tempo de desenvolvimento e manutenção de aplicações sejam reduzidos e os requisitos de tempo real sejam alcançados. Assim, esta tese teve o intuito de gerar resultados científicos relevantes e tecnologias que possam ser exploradas comercialmente.

8.1 Contribuições da tese

A principal contribuição desta tese foi o desenvolvimento de uma arquitetura de comunicação de baixa latência para ambientes baseados em aglomerados gráficos remotos. Os 3DCVEs têm sido fruto de pesquisas há vários anos, porém, as soluções encontradas na literatura não satisfazem os resquisitos de QoS (baixa latência e variação no tempo de entrega de pacotes). A solução apresentada nesta tese conseguiu mitigar problemas encontrados em redes de dados baseadas em Internet, visto que a troca de mensagem/eventos foi desenvolvida de forma otimizada, visando diminuir a quantidade de dados trafegados pela rede.

Os estudos de caso desenvolvidos foram todos baseados na solução apresentada na tese. Todos os estudos foram voltados a aplicações gráficas, entretanto, a solução pode ser aplicada em diversos problemas relacionado à computação em nuvem e ambientes distribuídos em geral, visto que o modelo de comunicação (variáveis síncronas, eventos assíncronos e barreiras de sincronismo) são altamente difundidos para essas áreas.

Além do modelo de comunicação inter–aglomerados desenvolvido, um modelo de análise de conexão dinâmico e adaptável baseado em lógica difusa foi apresentado. Por meio deste foi possível realizar verificações e adequações nas conexões em tempo de execução, permitindo que diferentes configurações de redes pudessem ser aplicadas de acordo com as adversidades de rede encontradas.

Indo além, na linha de depuração de mensagens distribuídas, a ferramenta de análise de troca de mensagens GTracer foi estendida, permitindo a análise de mensagens trocadas entre processos distribuídos em aglomerados gráficos remotos. Outra contribuição foi a forma de ordenação de mensagens trocadas entre os processos distribuídos. Por meio de um sistema de cruzamento de *logs*, foi possível identificar quando e qual processo gerou um evento específico, não necessitando utilizar nenhum tipo de relógio lógico.

8.2 Limitações e lições aprendidas

Dentre as limitações, o que pode ser mencionado até o presente estágio da pesquisa apresentado nesta tese, é a limitação em relação ao uso de dispositivos móveis. Visto que a RV

e RA já estão bem difundidas, soluções de desenvolvimento de ambientes colaborativos para esta classe de dispositivos é uma evolução pertinente. Problemas relacionados as bibliotecas de dependência são os mais corriqueiros, como a compilação da biblioteca *Boost*, que não é uma tarefa trivial de ser realizada em ambientes móveis. Uma das formas de se tentar resolver essa limitação seria utilizando um *web service*, implementando assim uma interface entre os dispositivos móveis e a solução apresentada na tese, que seria utilizada como um seviço *web*.

As funcionalidades apresentadas nesta tese podem não ser suficientes para atender a todos os requisitos de 3DCVEs baseados em aglomerados gráficos remotos, por isso, com o surgimento de novos serviços e requisitos durante pesquisas conseguintes, novas funcionalidades poderão ser propostas e implementadas.

O uso de soluções de baixo nível para efeutar comunicação inter–aglomerados se mostrou adequado quanto a desempenho. No entanto, algumas da funcionalidades poderiam ter sido tratadas com o uso de soluções de terceiros, tal como o ZooKeeper (APACHE, 2016), serviço provido pela Apache voltado a nomeação, gerenciamento de configuração, sincronização, e diversos outros serviços.

8.3 Sugestões de trabalhos futuros

Em relação a trabalhos futuros, pretende–se realizar experimentos utilizando a computação em nuvem como infraestrutura aos *peers* remotos, como também realizar o uso de dispostivos móveis para processos escravo (interação ou/e visualização) da arquitetura.

Outra proposta de aplicação futura é utilizar a solução para ambientes de cálculo baseados em aglomerados. Assim, a solução seria utilizada no processo de balanceamento de carga, não se limitando ao contexto de 3DCVEs. É esperado evidenciar a importância de sua pesquisa e as possíveis contribuições em outros domínios de aplicação.

8.4 Publicações no período do doutorado

Esta seção apresenta as publicações produzidas durante o período do doutorado. Nem todas foram diretamente relacionadas à proposta apresentada nesta tese, mas todas estiveram relacionadas aos interesses de pesquisa do candidato.

8.4.1 Journals e revistas

BRANDAO, A. F. ; DIAS, D. R. C. ; CASTELLANO, G. ; PARIZOTTO, N. A. ; TREVELIN, L. C. . RehabGesture: an alternative tool for measuring human movement. Telemedicine Journal and e-Health, 2015. (Qualis B1)

POPOLIN NETO, M. ; AGOSTINHO, I. A. ; DIAS, D. R. C. ; RODELLO, I. A. ; BREGA, J. R. F. . A Realidade Virtual e o Motor de Jogo Unity. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 5, p. 9-23, 2015.

FRANCISCO, L. A. V. ; BRANDAO, A. F. ; DIAS, D. R. C. ; PESSOA, J. D. C. ; TREVELIN, L. C. . Desenvolvimento de projetos para impressão de biosistemas em STL utilizando VTK. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 5, p. 121-138, 2015.

VIEL, Caio C. ; MELO Erick L. ; GODOY, A. P. ; DIAS, D. R. C. ; TREVELIN, Luis Carlos ; TEIXEIRA, C. A. C. . Multimedia Presentation Integrating Media with Virtual 3D Realistic Environment Produced in Real Time with High Performance Processing. SBC Journal on 3D Interactive Systems, v. 5, p. 34-43, 2014.

BRANDAO, A. F. ; BRASIL, G. J. C. ; DIAS, D. R. C. ; ALMEIDA, S. R. M. ; CASTELLANO, G. ; TREVELIN, Luis Carlos . Realidade Virtual e Reconhecimento de Gestos Aplicados às Àreas de Saúde. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 4, p. 33-48, 2014.

DIAS, D. R. C.; FRANCISCHETTI-CORRÊA, Moacyr ; BREGA, J. R. F. ; TREVELIN, L. C. ; GUIMARAES, M. P. . Teacher-learning process based on na immersive and interactive environment. IEEE Multidisciplinary Engineering Education Magazine, v. 8, p. 1-6, 2013.

DIAS, D. R. C.; BRASIL, G. J. C. ; GNECCO, B. B. ; POPOLIN NETO, M. ; AGOSTINHO, I. A. ; VALDIVIA, R. ; TREVELIN, L. C. . Desenvolvimento de Aplicações com Interface Natural de Usuário e Dispositivos PrimeSense como Meio de Interação para Ambientes Virtuais. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 3, p. 89-103, 2013.

GNECCO, B. B. ; DOMINGUES, R. G. ; BRASIL, G. J. C. ; DIAS, D. R. C. ; TREVELIN, L. C. . Estratégias Mistas de Mecanismos para Imersão em Modelos de Interação. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 3, p. 104-120, 2013.

GNECCO, B. B. ; DIAS, D. R. C. ; BRASIL, G. J. C. ; GUIMARAES, M. P. . Desenvolvimento de Interfaces Naturais de Interação usando o Hardware Kinect. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 2, p. 37-62, 2012.

GUIMARAES, M. P. ; GNECCO, B. B. ; DIAS, D. R. C. . Realidade Aumentada para aplicações Web e Mobile. Tendências e Técnicas em Realidade Virtual e Aumentada, v. 2, p. 159-176, 2012.

GNECCO, B. B. ; DIAS, D. R. C. ; GUIMARAES, M. P. . Traditional paintings and the digital medium. SBC Journal on 3D Interactive Systems, v. 3, p. 3, 2012.

8.4.2 Eventos

DIAS, D. R. C.; GUIMARAES, M. P. ; KUHLEN, T. W. ; TREVELIN, L. C. . A Dynamic-Adaptive Architecture For 3D Collaborative Virtual Environments Based On Graphic Clusters. In: The 30th ACM/SIGAPP Symposium On Applied Computing, 2015, Salamanca. The 30th ACM/SIGAPP Symposium On Applied Computing Proceedings, 2015. v. 1. p. 1.

GUIMARAES, M. P. ; GNECCO, B. B. ; DIAS, D. R. C. ; BREGA, J. R. F. ; TREVELIN, L. C. . Graphical High Level Analysis of Communication in Distributed Virtual Reality Applications. In: nternational Conference On Computational Science, ICCS 2015 ? Computational Science at the Gates of Nature, 2015, Reykjavík. Procedia Computer Science, 2015. v. 51. p. 1373-1382.

GUIMARAES, M. P. ; DIAS, D. R. C. ; MARTINS, V. F. ; BREGA, J. R. F. ; TREVELIN, L. C. . Immersive and Interactive Simulator to Support Educational Teaching. In: The 15th International Conference on Computational Science and Its Applications (ICCSA 2015), 2015, Banff. Computational Science and Its Applications – ICCSA 2015, 2015. p. 250-260.

POPOLIN NETO, M. ; DIAS, D. R. C. ; TREVELIN, L. C. ; GUIMARAES, M. P. ; BREGA, J. R. F. . Unity Cluster Package: Dragging and Dropping Components for Multi-projection Virtual Reality Applications Based on PC Clusters. In: The 15th International Conference on Computational Science and Its Applications (ICCSA 2015), 2015, Banff. Computational Science and Its Applications – ICCSA 2015, 2015. p. 261-272.

SANTOS, R. ; LUZ, B. N. ; MARTINS, V. F. ; DIAS, D. R. C. ; GUIMARAES, M. P. . Teaching-Learning Environment Tool to Promote Individualized Student Assistance. In: The 15th International Conference on Computational Science and Its Applications (ICCSA 2015), 2015, Banff. Computational Science and Its Applications – ICCSA 2015, 2015. p. 143-155.

DIAS, D. R. C.; GUIMARAES, M. P. ; TREVELIN, L. C. . Integração de Ambientes Virtuais 3D Colaborativos baseados em Aglomerados Gráficos. In: WRVA 2015, 2015, Presidente Prudente. Anais do WRVA, 2015. p. 119-124.

DIAS, D. R. C.; DURELLI, R. S. ; BREGA, J. R. F. ; GNECCO, B. B. ; TREVELIN, L. C. ; GUIMARAES, M. P. . Data Network in Development of 3D Collaborative Virtual Environments: A Systematic Review. In: The 14th International Conference on Computational Science and Its Applications (ICCSA 2014), 2014, Guimarães. Computational Science and Its Applications ICCSA 2014, 2014. p. 769-785.

DIAS, D. R. C.; BREGA, J. R. F. ; TREVELIN, L. C. ; GNECCO, B. B. ; PAPA, J. P. ; GUIMARAES, M. P. . 3D Network Traffic Monitoring Based on an Automatic Attack Classifier. In: The 14th International Conference on Computational Science and Its Applications (ICCSA 2014), 2014, Guimarães. Computational Science and Its Applications - ICCSA 2014, 2014.

BRASIL, G. J. C. ; BRANDAO, A. F. ; DIAS, D. R. C. ; PARIZOTTO, N. A. ; TREVELIN, L. C. . Natural User Interface Applied for SpatialDisorientation and Movement Disorder: GestureMaps. In: WRVA 2014, 2014, Marília. Anais do WRVA, 2014. v. 1. p. 1.

BREGA,J.R.F ; MARTINS, V. F. ; DIAS, D. R. C. ; RODELLO, I. A. ; GUIMARAES, M. P. . A virtual reality environment to support chat rooms for hearing impaired and to teach Brazilian sign language (LIBRAS). In: The 11th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'2014), 2014, Doha. AICCSA Proceedings, 2014. v. 1. p. 1.

BRANDAO, A. F. ; SOARES, M. C. ; BRASIL, G. J. C. ; DIAS, D. R. C. ; FABBRO, A. C. ; DUARTE, A. C. G. O. ; TREVELIN, L. C. . Prevenção de atrofia muscular da articulação glenoumeral por meio de atividade física adaptada à realidade virtual e reconhecimento de gestos. In: Simpósio SESC de Atividades Físicas Adaptadas, 2013, São Carlos. Anais do evento, 2013.

GUIMARAES, M. P. ; MARTINS, V. F. ; DIAS, D. R. C. ; CONTRI, L. F. ; GNECCO, B. B. . Desenvolvimento e Depuração de Aplicações de Realidade Virtual Distribuídas. In: CISTI'2013 - 8ª Conferência Ibérica de Sistemas e Tecnologias de Informação, 2013, Lisboa. Anais do CISTI'2013. Lisboa: CISTI'2013, 2013. v. 1. p. 1.

DIAS, D. R. C.; BRANDAO, A. F. ; BRASIL, G. J. C. ; GUIMARAES, M. P. ; BREGA, J. R. F. ; TREVELIN, L. C. . Gesture Chess - Interface Natural de Usuário na Atividade Motora e Cognitiva. In: WRVA 2013, 2013, Jataí. Anais do WRVA, 2013.

GNECCO, B. B. ; BREGA, J. R. F. ; DIAS, D. R. C. ; TREVELIN, L. C. ; GUIMARAES, M. P. . Sistema Imersivo e Interativo Baseado em Projeção Semi-cilíndrica. In: WRVA 2013, 2013, Jataí. Anais do WRVA, 2013.

POPOLIN NETO, M. ; AGOSTINHO, I. A. ; MORAES, A. ; DIAS, D. R. C. ; Weber, S. ; BREGA, J. R. F. . Sistema de Descrição Semântica para Visualização de Modelos 3D da Anatomia Humana. In: WRVA 2013, 2013, Jataí. Anais do WRVA, 2013.

MARTINS, V. F. ; DIAS, D. R. C. ; GUIMARAES, M. P. . Uso de Lógica Fuzzy no Auxílio ao Acompanhamento Automático de Alunos utilizando um Ambiente de Aprendizagem. In: SBIE 2013, 2013, Campinas. Anais do XXIV Simpósio Brasileiro de Informática na Educação, 2013. p. 707-716.

BREGA, J. R. F. ; MARCA, A. F. L. ; POPOLIN NETO, M. ; DIAS, D. R. C. ; TREVELIN, L. C. . Sistema Gerador de Apoio a um Dicionário Temático Visual-Gestual Baseado em Realidade Virtual. In: WAVE/CBIE 2013, 2013, Campinas. Anais do CBIE, 2013.

DIAS, D. R. C.; BREGA, J. R. F. ; TREVELIN, L. C. ; POPOLIN NETO, M. ; GNECCO, B. B. ; GUIMARAES, M. P. . Design and Evaluation of an Advanced Virtual Reality System for Visualization of Dentistry Structures. In: 18th International Conference on Virtual Systems and Multimedia, 2012, Milan. VSMM 2012, 2012. v. 1. p. 1-1.

VIEL, CAIO CÉSAR ; MELO, ERICK LAZARO ; GODOY, ARTHUR PEDRO ; DIAS, DIEGO ROBERTO COLOMBO ; TREVELIN, LUIS CARLOS ; TEIXEIRA, CESAR AUGUSTO CAMILLO . Multimedia presentation integrating interactive media produced in real time with high performance processing. In: the 18th Brazilian symposium, 2012, São Paulo/SP. Proceedings of the 18th Brazilian symposium on Multimedia and the web - WebMedia '12. New York: ACM Press, 2012.

MARCA, A. F. L. ; DIAS, D. R. C. ; BREGA, J. R. F. . Sistema de Multiprojeção para a Comunicação em Libras. In: WRVA 2012, 2012, Paranavaí. Anais do WRVA 2012, 2012. v. 1. p. 1-1.

REFERÊNCIAS

ACTIVEWORLDS. *ActiveWorlds: Home of the 3D Internet since 1995*. 2016. `https://www.activeworlds.com/web/index.php`. Último acesso em: março de 2016.

AGUILERA, M. K.; MOGUL, J. C.; WIENER, J. L.; REYNOLDS, P.; MUTHITACHAROEN, A. Performance debugging for distributed systems of black boxes. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2003. (SOSP '03), p. 74–89. ISBN 1-58113-757-5. Disponível em: <http://doi.acm.org/10-.1145/945445.945454>.

AI, Z.; EVENHOUSE, R.; LEIGH, J.; CHARBEL, F.; RASMUSSEN, M. Biomedical modeling in tele-immersion. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 4650 LNAI, p. 47–70, 2008. Cited By (since 1996)0.

ALI, A.; EL-DESOKY, A.; SALAH, M. An allocation management algorithm for dve system. In: *Computer Engineering Systems, 2009. ICCES 2009. International Conference on*. [S.l.: s.n.], 2009. p. 489–494.

ALLARD, J.; GOURANTON, V.; LECOINTRE, L.; MELIN, E.; RAFFIN, B. Net juggler: running vr juggler with multiple displays on a commodity component cluster. In: *Virtual Reality, 2002. Proceedings. IEEE*. [S.l.: s.n.], 2002. p. 273–274. ISSN 1087-8270.

ALLARD, J.; GOURANTON, V.; LECOINTRE, L.; LIMET, S.; MELIN, E.; RAFFIN, B.; ROBERT, S. Flowvr: A middleware for large scale virtual reality applications. In: DANELUTTO, M.; VANNESCHI, M.; LAFORENZA, D. (Ed.). *Euro-Par 2004 Parallel Processing*. [S.l.]: Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 3149). p. 497–505. ISBN 978-3-540-22924-7.

ALLISON, R. S.; ZACHER, J. E.; WANG, D.; SHU, J. Effects of network delay on a collaborative motor task with telehaptic and televisual feedback. In: *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*. New York, NY, USA: ACM, 2004. (VRCAI '04), p. 375–381. ISBN 1-58113-884-9.

ANTHES, C.; HAFFEGEE, A.; HEINZLREITER, P.; VOLKERT, J. A scalable network architecture for closely coupled collaboration. *Computing and Informatics*, ACM Press, v. 24, n. 1, p. 31–51, 2005. Disponível em: <http://www.cai.sk/ojs/index.php/cai/article/viewArticle-/365>.

APACHE. *Apache ZooKeeper*. 2016. `https://zookeeper.apache.org`. Último acesso em: março de 2016.

APOSTOLOPOULOS, J.; CHOU, P.; CULBERTSON, B.; KALKER, T.; TROTT, M.; WEE, S. The road to immersive communication. *Proceedings of the IEEE*, v. 100, n. 4, p. 974–990, 2012. ISSN 0018-9219.

AQUINO, P. T.; KIRNER, T. G. Desenvolvimento de um sistema de gerenciamento de dados e arquivos para um ambiente virtual colaborativo. In: *SBC Symposium on Virtual Reality – SVR 2001*. [S.l.: s.n.], 2001.

ARENSTORF, N. S.; JORDAN, H. F. Comparing barrier algorithms. *Parallel Computing*, v. 12, n. 2, p. 157 – 170, 1989. ISSN 0167-8191. Disponível em: <http://www.sciencedirect-.com/science/article/pii/0167819189900501>.

BALADI, M.; VITALI, H.; FADEL, G.; SUMMERS, J.; DUCHOWSKI, A. A taxonomy for the design and evaluation of networked virtual environments: its application to collaborative design. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, Springer-Verlag, v. 2, n. 1, p. 17–32, 2008. ISSN 1955-2513.

BARHAM, P.; DONNELLY, A.; ISAACS, R.; MORTIER, R. Using magpie for request extraction and workload modelling. In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA: USENIX Association, 2004. (OSDI'04), p. 18–18. Disponível em: <http://dl.acm.org/citation-.cfm?id=1251254.1251272>.

BARHAM, P.; ISAACS, R.; MORTIER, R.; NARAYANAN, D. Magpie: Online modelling and performance-aware systems. In: *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*. Berkeley, CA, USA: USENIX Association, 2003. (HOTOS'03), p. 15–15. Disponível em: <http://dl.acm.org/citation.cfm?id=1251054.1251069>.

BASS, L.; CLEMENTS, P. *Software Architecture in Pratice*. [S.l.]: Addison Wesley, 1998.

BOUKERCHE, A.; SHIRMOHAMMADI, S.; HOSSAIN, A. Moderating simulation lag in haptic virtual environments. In: *Simulation Symposium, 2006. 39th Annual*. [S.l.: s.n.], 2006. p. 269–277. ISSN 1080-241X.

BRANDAO, A. F.; DIAS, D. R. C.; CASTELLANO, G.; PARIZOTTO, N. A.; TREVELIN, L. C. Rehabgesture: an alternative tool for measuring human movement. *Telemedicine Journal and e-Health*, 2016. ISSN 1530-5627.

BROWN, M. S.; SEALES, W. B. *Low-Cost and Easily Constructed Large Format Display System*. 2001.

BRUNS, F.; ERBE, H.-H.; MÜLLER, D.; SCHAF, F.; PEREIRA, C.; REICHERT, C.; CAMPANA, F.; KRAKHECHE, I. Collaborative learning and engineering workspaces. In: . [S.l.: s.n.], 2007. v. 1, n. PART 1, p. 112–117. Cited By (since 1996)0.

CAVELIB. *Software from Mechdyne | Worldwide Technology Solutions*. 2016. `http://www.mechdyne.com/cavelib.aspx`. Último acesso em: março de 2016.

CHEN, L. Effects of network characteristics on task performance in a desktop cve system. In: *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*. [S.l.: s.n.], 2005. v. 1, p. 821–826vol.1. ISSN 1550-445X.

CHEN, M. Y.; KICIMAN, E.; FRATKIN, E.; FOX, A.; BREWER, E. Pinpoint: Problem determination in large, dynamic internet services. In: *Proceedings of the 2002 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2002. (DSN '02), p. 595–604. ISBN 0-7695-1597-5. Disponível em: <http://dl.acm.org/citation.cfm?id=647883.738238>.

CHEN, S.; SHI, B.; CHEN, S. Acom: Any-source capacity-constrained overlay multicast in non-dht p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, USA, v. 18, n. 9, p. 1188–1201, 2007. ISSN 1045-9219.

CHO, Y.; KIM, M.; PARK, K. Lotus: composing a multi-user interactive tiled display virtual environment. *The Visual Computer*, Springer-Verlag, v. 28, n. 1, p. 99–109, 2012. ISSN 0178-2789.

CHROMIUM. *Chromium Homepage*. 2016. `http://chromium.sourceforge.net/`. Último acesso em: março de 2016.

CORRÊA, M.; SCHPECTOR, J.; TREVELIN, L.; PAIVA GUIMARÃES, M. de. Immersive environment for molecular visualization to interaction between research groups geographically dispersed. In: *Applied Sciences in Biomedical and Communication Technologies (ISABEL), 2010 3rd International Symposium on*. [S.l.: s.n.], 2010. p. 1–5.

CORRêA, M. F. *Arquitetura de alto desempenho para integração de ambientes interativos e intensivos remotos para visualização molecular*. Tese (Doutorado) — Universidade Federal de São Carlos, 2010.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. *Distributed Systems: Concepts and Design*. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011, 9780132143011.

CRUZ-NEIRA, C.; SANDIN, D. J.; DEFANTI, T. A.; KENYON, R. V.; HART, J. C. The cave: audio visual experience automatic virtual environment. *Commun. ACM*, ACM, New York, NY, USA, v. 35, p. 64–72, June 1992. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/129888.129892>.

DEFANTI, T. A.; LEIGH, J.; RENAMBOT, L.; JEONG, B.; VERLO, A.; LONG, L.; BROWN, M.; SANDIN, D. J.; VISHWANATH, V.; LIU, Q.; KATZ, M. J.; PAPADOPOULOS, P.; KEEFE, J. P.; HIDLEY, G. R.; DAWE, G. L.; KAUFMAN, I.; GLOGOWSKI, B.; DOERR, K.-U.; SINGH, R.; GIRADO, J.; SCHULZE, J. P.; KUESTER, F.; SMARR, L. The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Generation Computer Systems*, v. 25, n. 2, p. 114–123, 2009. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X08001040>.

DIAS, D. C.; LA MARCA, A.; POPOLIN NETO, M.; ; BREGA, J.; PAIVA GUIMARAES, M. de; LAURIS, J. Java 3d para sistemas de multiprojeção utilizando aglomerados gráficos. In: *Workshop de Realidade Virtual e Aumentada (WRVA 2010)*. São Paulo: SBC, 2010. (WRVA 2010).

DIAS, D. C.; LA MARCA, A.; MOIA VIEIRA, A.; NETO, M.; BREGA, J.; PAIVA GUIMARAES, M. de; LAURIS, J. Dental arches multi-projection system with semantic descriptions. In: *Virtual Systems and Multimedia (VSMM), 2010 16th International Conference*

on Virtual Systems and Multimedia. Seoul: IEEE Xplore, 2010. (VSMM 2010, ISBN 978-1-4244-9027-1), p. 314 –317.

DIAS, D. R. C. *Sistema Avançado de Realidade Virtual para Visualização de Estruturas Odontológicas*. Dissertação (Mestrado) — IBILCE/UNESP, São José do Rio Preto, 2011.

DIAS, D. R. C.; GUIMARãES, M. P.; KUHLEN, T. W.; TREVELIN, L. C. A dynamic-adaptive architecture for 3d collaborative virtual environments based on graphic clusters. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2015. (SAC '15), p. 480–487. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695762>.

DIAS, D. R. C.; DURELLI, R. S.; BREGA, J. R. F.; GNECCO, B. B.; TREVELIN, L. C.; PAIVA GUIMARÃES, M. Computational science and its applications – iccsa 2014: 14th international conference, guimarães, portugal, june 30 – july 3, 2014, proceedings, part i. In: ____. Cham: Springer International Publishing, 2014. cap. Data Network in Development of 3D Collaborative Virtual Environments: A Systematic Review, p. 769–785. ISBN 978-3-319-09144-0. Disponível em: <http://dx.doi.org/10.1007/978-3-319-09144-0_53>.

DONG, K.; NAN, K.; TILAK, S.; ZHENG, C.; XU, D.; SCHULZE, J.; ARZBERGER, P.; LI, W. Real time biomedical data streaming platform (rimes): A data-intensive virtual environment. In: . [S.l.: s.n.], 2010. v. 2, p. 342–346. Cited By (since 1996)1.

DROLET, F.; MOKHTARI, M.; BERNIER, F.; LAURENDEAU, D. A software architecture for sharing distributed virtual worlds. In: *Virtual Reality Conference, 2009. VR 2009. IEEE*. [S.l.: s.n.], 2009. p. 271–272. ISSN 1087-8270.

EQUALIZER. *Equalizer: Parallel Renderings*. 2016. `http://www.equalizergraphics.com/`. Último acesso em: março de 2016.

EYEGAZE. *Eye Tracking | Advanced Eye – Tracking Technology by LC Technologies*. 2016. `http://www.eyegaze.com/`. Último acesso em: março de 2016.

FONSECA, R.; PORTER, G.; KATZ, R. H.; SHENKER, S.; STOICA, I. X-trace: A pervasive network tracing framework. In: *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2007. (NSDI'07), p. 20–20. Disponível em: <http://dl.acm.org/citation.cfm?id=1973430.1973450>.

FUZZYLITE. *C++ | fuzzylite*. 2016. `http://www.fuzzylite.com/cpp/`. Último acesso em: março de 2016.

GNECCO, B. B. *Uma Plataforma para Geração Interativa de Imagens por Traçados de Raios*. Dissertação (Mestrado) — Poli/USP, São Paulo, 2003.

GNECCO, B. B.; PAIVA GUIMARAES, M. de; ZUFFO, M. K. Um framework para computação distribuída. In: *Simpósio Brasileiro de Realidade Virtual*. Ribeirão Preto: SBC, 2003.

GODEFROID, P. Software model checking: The verisoft approach. *Form. Methods Syst. Des.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 26, n. 2, p. 77–101, mar. 2005. ISSN 0925-9856. Disponível em: <http://dx.doi.org/10.1007/s10703-005-1489-x>.

GRAHAM, S. L.; KESSLER, P. B.; MCKUSICK, M. K. Gprof: A call graph execution profiler. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 17, n. 6, p. 120–126, jun. 1982. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/872726.806987>.

GUIMARAES, M. P. *Um ambiente para o desenvolvimento de aplicações de realidade virtual baseadas em aglomerados gráficos*. Tese (Doutorado) — Universidade de São Paulo, 2004.

GUO, T.-T.; GUO, L.; WANG, Z.; LIN, S.; PAN, J.-H. A networked virtual experiment system based on virtual campus. In: . [S.l.: s.n.], 2009. v. 3, p. 884–888. Cited By (since 1996)1.

HARIRI, B.; RATTI, S.; SHIRMOHAMMADI, S.; PAKRAVAN, M. A distributed latency-aware architecture for massively multi-user virtual environments. In: *Haptic Audio visual Environments and Games, 2008. HAVE 2008. IEEE International Workshop on*. [S.l.: s.n.], 2008. p. 53–58.

IMVU. *IMVU*. 2016. `http://pt.imvu.com/`. Último acesso em: março de 2016.

ISHIDA, T.; YATSU, K.; SHIBATA, Y. Tele-immersion environment for video avatar based cve. In: *Network-Based Information Systems, 2009. NBIS '09. International Conference on*. [S.l.: s.n.], 2009. p. 608–611.

KADAVASAL, M.; OLIVER, J. Towards sensor enhanced virtual reality teleoperation in a dynamic environment. In: . [S.l.: s.n.], 2008. v. 2 PART B, p. 1057–1065. Cited By (since 1996)0.

KENYON, R.; LEIGH, J. Networked virtual environments and rehabilitation. In: . [S.l.: s.n.], 2004. v. 26 VII, p. 4832–4835. Cited By (since 1996)1.

KILLIAN, C. E.; ANDERSON, J. W.; BRAUD, R.; JHALA, R.; VAHDAT, A. M. Mace: Language support for building distributed systems. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 42, n. 6, p. 179–188, jun. 2007. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/1273442.1250755>.

KITCHENHAM, B.; PEARL BRERETON, O.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINKMAN, S. Systematic literature reviews in software engineering - a systematic literature review. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 51, n. 1, p. 7–15, jan. 2009. ISSN 0950-5849.

KRANZLMÜLLER, D. *Event Graph Analysis for Debugging Massively Parallel Programs*. Tese (Doutorado) — Institut für Technische Informatik und Telematik, 2000.

KUROSE, J. F.; ROSS, K. W. *Computer Networking: A Top-Down Approach (6th Edition)*. 6th. ed. [S.l.]: Pearson, 2012. ISBN 0132856204, 9780132856201.

LI, L.; LI, F.; LAU, R. A trajectory-preserving synchronization method for collaborative visualization. *Visualization and Computer Graphics, IEEE Transactions on*, v. 12, n. 5, p. 989–996, set. 2006. ISSN 1077-2626.

LIANG, J. Modeling an immersive vr driving learning platform in a web-based collaborative design environment. *Computer Applications in Engineering Education*, v. 20, n. 3, p. 553–567, 2012. Cited By (since 1996)0.

LIBGLASS. *Glass Library download | SourceForge.net*. 2016. `https://sourceforge.net/projects/vistavrtoolkit/`. Último acesso em: março de 2016.

LING, C.; XIAO-LEI, X.; GEN-CAI, C.; CHUEN, C. An effective communication architecture for collaborative virtual systems. In: *Communication Technology Proceedings, 2003. ICCT 2003. International Conference on*. [S.l.: s.n.], 2003. v. 2, p. 1598–1602vol.2.

LIU, J.; MA, Y. Implementing overlay multicast with sctp. In: *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*. [S.l.: s.n.], 2010. p. 1–4.

LORENZO, C.-M.; SICILIA, M. . A.; A¡NCHEZ, S. S. Studying the effectiveness of multi-user immersive environments for collaborative evaluation tasks. *Computers & Education*, v. 59, n. 4, p. 1361–1376, 2012. ISSN 0360-1315. Disponível em: <http://www.sciencedirect-.com/science/article/pii/S0360131512001443>.

MALHEIROS, V.; HOHN, E.; PINHO, R.; MENDONCA, M.; MALDONADO, J. C. A visual text mining approach for systematic reviews. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2007. (ESEM '07), p. 245–254. ISBN 0-7695-2886-4.

MORILLO, P.; BIERBAUM, A.; HARTLING, P.; FERNáNDEZ, M.; CRUZ-NEIRA, C. Analyzing the performance of a cluster-based architecture for immersive visualization systems. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 68, n. 2, p. 221–234, fev. 2008. ISSN 0743-7315. Disponível em: <http://dx.doi.org/10.1016/j.jpdc.2007.08.009>.

NAHRSTEDT, K.; YANG, Z.; WU, W.; AREFIN, A.; RIVAS, R. Next generation session management for 3D teleimmersive interactive environments. *Multimedia Tools and Applications*, Springer Netherlands, v. 51, n. 2, p. 593–623, 2010.

NETEM. *netem*. 2016. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`. Último acesso em: março de 2016.

NETO, M. P.; DIAS, D. R. C.; TREVELIN, L. C.; PAIVA GUIMARÃES, M.; BREGA, J. R. F. Computational science and its applications – iccsa 2015: 15th international conference, banff, ab, canada, june 22-25, 2015, proceedings, part v. In: ____. Cham: Springer International Publishing, 2015. cap. Unity Cluster Package – Dragging and Dropping Components for Multi-projection Virtual Reality Applications Based on PC Clusters, p. 261–272. ISBN 978-3-319-21413-9. Disponível em: <http://dx.doi.org/10.1007/978-3-319-21413-9_19>.

OKUDA, M.; KARUBE, Y.; MATSUKURA, R.; WATANABE, M.; MATSUZAWA, T. Development of a vizgrid volume communications environment. In: KOYAMADA, K.; TAMURA, S.; ONO, O. (Ed.). *Systems Modeling and Simulation*. [S.l.]: Springer Japan, 2007. p. 381–385. ISBN 978-4-431-49021-0.

OLIVEIRA, J. C. d.; YU, S. J.; GEORGANAS, N. D. Synchronized world embedding in virtual environments. *IEEE Comput. Graph. Appl.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 24, n. 4, p. 73–83, jul. 2004. ISSN 0272-1716. Disponível em: <http://dx.doi.org/10.1109/MCG.2004.18>.

OPIYO, E.; HORVÁTH, I. A conceptual structure for heterogeneous remote product data visualization. In: . [S.l.: s.n.], 2010. v. 1, p. 551–564. Cited By (since 1996)0.

PAPE, D. *Mirage 5000 CAVE - Sneak Preview*. 2013. `http://www.matrox.com/graphics/en/products/gxm/`.

PICK, S.; GEBHARDT, S.; WEYERS, B.; HENTSCHEL, B.; KUHLEN, T. A 3d collaborative virtual environment to integrate immersive virtual reality into factory planning processes. In: *Collaborative Virtual Environments (3DCVE), 2014 International Workshop on*. [S.l.: s.n.], 2014. p. 1–6.

PIMENTEL, M.; FUKS, H. *Sistemas colaborativos*. 1. ed. [S.l.]: Elsevier, 2011.

PONTONNIER, C.; DUVAL, T.; DUMONT, G. Collaborative virtual environments for ergonomics: embedding the design engineer role in the loop. In: *Collaborative Virtual Environments (3DCVE), 2014 International Workshop on*. [S.l.: s.n.], 2014. p. 1–5.

QUANTA. *Quanta*. 2016. `https://www.evl.uic.edu/cavern/quanta/index.html`. Último acesso em: março de 2016.

RAFFIN, B.; SOARES, L.; NI, T.; BALL, R.; SCHMIDT, G.; LIVINGSTON, M.; STAADT, O.; MAY, R. Pc clusters for virtual reality. In: *Virtual Reality Conference, 2006*. [S.l.: s.n.], 2006. p. 215–222. ISSN 1087-8270.

REYNOLDS, P. A. *Using Causal Paths to Improve Performance and Correctness in Distributed Systems*. Tese (Doutorado), Durham, NC, USA, 2006. AAI3265281.

RHEE, S.; ZIEGLER, R.; PARK, J.; NAEF, M.; GROSS, M.; KIM, M. Low-cost telepresence for collaborative virtual environments. *IEEE transactions on visualization and computer graphics*, v. 13, n. 1, p. 156–166, 2007. Cited By (since 1996)9.

RIZZO, A.; REQUEJO, P.; WINSTEIN, C. J.; LANGE, B.; RAGUSA, G.; MERIANS, A.; PATTON, J.; BANERJEE, P.; AISEN, M. Virtual reality applications for addressing the needs of those aging with disability. *Studies in health technology and informatics*, v. 163, p. 510–516, 2011.

ROBERTS, D.; WOLFF, R.; OTTO, O.; STEED, A. Constructing a gazebo: Supporting teamwork in a tightly coupled, distributed task in virtual reality. *Presence: Teleoper. Virtual Environ.*, MIT Press, Cambridge, MA, USA, v. 12, n. 6, p. 644–657, dez. 2003. ISSN 1054-7460. Disponível em: <http://dx.doi.org/10.1162/105474603322955932>.

RODELLO, I. A. *VRMol – um ambiente virtual distribuído para visualização de análise de moléculas de proteínas*. Tese (Doutorado) — Instituto de Física de São Carlos, Universidade de São Paulo, 2003.

RONNINGEN, L.; PANGGABEAN, M.; TAMER, O. Toward futuristic near-natural collaborations on distributed multimedia plays architecture. In: *Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on*. [S.l.: s.n.], 2010. p. 102–107.

RUNESON, P.; HOST, M.; RAINER, A.; REGNELL, B. *Case Study Research in Software Engineering: Guidelines and Examples*. [S.l.]: Wiley Blackwell, 2012. ISBN 1118104358.

SCHAEFFER, B.; GOUDESEUNE, C. Syzygy: native pc cluster vr. In: *Virtual Reality, 2003. Proceedings. IEEE*. [S.l.: s.n.], 2003. p. 15–22. ISSN 1087-8270.

SCHIRSKI, M.; GERNDT, A.; REIMERSDAHL, T. van; KUHLEN, T.; ADOMEIT, P.; LANG, O.; PISCHINGER, S.; BISCHOF, C. Vista flowlib - framework for interactive visualization and exploration of unsteady flows in virtual environments. In: *Proceedings of the Workshop on Virtual Environments 2003*. New York, NY, USA: ACM, 2003. (EGVE '03), p. 77–85. ISBN 1-58113-686-2. Disponível em: <http://doi.acm.org/10.1145/769953.769963>.

SCHROEDER, R. Being there together and the future of connected presence. *Presence: Teleoper. Virtual Environ.*, MIT Press, Cambridge, MA, USA, v. 15, n. 4, p. 438–454, ago. 2006. ISSN 1054-7460.

SCTP. *Stream Control Transmission Protocol*. 2016. `hhttp://tools.ietf.org/html/rfc4960`. Último acesso em: março de 2016.

SECONDLIFE. *Second Life Official Site – Virtual Worlds, Avatars, Free 3D Chat*. 2016. `http://secondlife.com/`. Último acesso em: março de 2016.

SEMENTILLE, A. C. *A utilização da arquitetura CORBA na construção de ambientes virtuais colaborativos*. Tese (Doutorado) — Instituto de Física de São Carlos, Universidade de São Paulo, 1999.

SHAHAB, Q.; KWON, Y.-M.; KO, H. Collaborative virtual experience based on reconfigurable simulation. In: . [S.l.: s.n.], 2006. v. 6391. Cited By (since 1996)0.

SHAW, M.; GARLAN, D. *Software Architecture. Perspectives on an Emerging Discipline*. [S.l.]: Prentice Hall, 1996.

SHERMAN, W. R.; COMING, D.; SU, S. Freevr: honoring the past, looking to the future. In: . [s.n.], 2013. v. 8649, p. 864906–864906–15. Disponível em: <http://dx.doi.org/10.1117/12-.2008578>.

SINGHAL, S.; ZYDA, M. *Networked virtual environments: design and implementation*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999. ISBN 0-201-32557-8.

SOARES, L. P. *Um Ambiente de multiprojeção totalmente imersivo baseado em aglomerados de computadores*. Tese (Doutorado) — USP, São Paulo, 2005.

SPEAKMAN, T.; CROWCROFT, J.; GEMMELL, J.; FARINACCI, D.; LIN, S.; LESHCHINER, D.; LUBY, M.; MONTGOMERY, T.; RIZZO, L.; TWEEDLY, A.; BHASKAR, N.; EDMONSTONE, R.; SUMANA-SEKERA R., V. L. *PGM reliable transport protocol specification*. 2013. Disponível em: <http://www.elook.org/computing/rfc/rfc3208-.html>.

STEINBACH, E.; HIRCHE, S.; KAMMERL, J.; VITTORIAS, I.; CHAUDHARI, R. Haptic data compression and communication. *Signal Processing Magazine, IEEE*, v. 28, n. 1, p. 87–96, jan. 2011. ISSN 1053-5888.

STEINER, M.; BIERSACK, E. DDC: A dynamic and distributed clustering algorithm for networked virtual environments based on P2P networks. In: *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*. [S.l.: s.n.], 2006. p. 1–6. ISSN 0743-166X.

SUGENO, M.; YASUKAWA, T. A fuzzy-logic-based approach to qualitative modeling. *Fuzzy Systems, IEEE Transactions on*, v. 1, n. 1, p. 7–, 1993. ISSN 1063-6706.

SUNG, M. Y.; YOO, Y.; JUN, K.; KIM, N.-J.; CHAE, J. Experiments for a collaborative haptic virtual reality. In: *Artificial Reality and Telexistence–Workshops, 2006. ICAT '06. 16th International Conference on*. [S.l.: s.n.], 2006. p. 174–179.

TANENBAUM, A. S.; STEEN, M. v. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0132392275.

TANENBAUM, A. S.; VIEIRA, D. *Redes de computadores*. 5.ed.. ed. São Paulo: Pearson, 2011. ISBN 9788543008585.

TUMANOV, A.; ALLISON, R.; STUERZLINGER, W. Variability-aware latency amelioration in distributed environments. In: *Virtual Reality Conference, 2007. VR '07. IEEE*. [S.l.: s.n.], 2007. p. 123–130.

UBIK, S.; NAVRATIL, J.; ZEJDL, P.; HALAK, J. Real-time stereoscopic streaming of medical surgeries for collaborative elearning. In: *CDVE*. [S.l.: s.n.], 2012. p. 73–77.

VIEL, C. C.; MELO, E. L.; GODOY, A. P.; DIAS, D. R. C.; TREVELIN, L. C.; TEIXEIRA, C. A. C. Multimedia presentation integrating interactive media produced in real time with high performance processing. In: *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2012. (WebMedia '12), p. 115–122. ISBN 978-1-4503-1706-1. Disponível em: <http://doi.acm.org/10.1145/2382636.2382664>.

VISTA. *ViSTA VR toolkit download | SourceForge.net*. 2016. `https://sourceforge.net/projects/libglass/?source=directory`. Último acesso em: março de 2016.

VRJUGGLER. *VR Juggler*. 2016. `http://vrjuggler.org/`. Último acesso em: março de 2016.

WANG, Y.; LI, Z.; ZHANG, W. A fully distributed P2P communications architecture for network virtual environments. In: *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. [S.l.: s.n.], 2011. p. 1–4. ISSN 2161-9646.

WATANABE, M. c.; OKUDA, M. b.; KARUBE, Y.; MATSUKURA, R.; MATSUZAWA, T. Collaborative environment with visualizing medical volume data by virtual reality. In: . [S.l.: s.n.], 2007. p. 347–352. Cited By (since 1996)1.

YIN, R. K. *Case Study Research: Design and Methods*. Beverly Hills, CA: Sage Publications, 2013. (Applied social research methods series). Disponível em: <http://www.amazon.com-/Case-Study-Research-Methods-Applied/dp/1452242569>.

YOU, Y.; SUNG, M.; KIM, N.; JUN, K. An experimental study on the performance of haptic data transmission in networked haptic collaboration. In: *Advanced Communication Technology, The 9th International Conference on*. [S.l.: s.n.], 2007. v. 1, p. 657–662. ISSN 1738-9445.

YOU, Y.; SUNG, M. Y. Haptic data transmission based on the prediction and compression. In: *Communications, 2008. ICC '08. IEEE International Conference on*. [S.l.: s.n.], 2008. p. 1824–1828.

YOU, Y.; SUNG, M. Y.; JUN, K. An integrated haptic data transmission in haptic collaborative virtual environments. In: *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*. [S.l.: s.n.], 2007. p. 834–839.

YUAN, Q.; LU, D. A latency-adaptive communication architecture for inter-networked virtual environments. In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. [S.l.: s.n.], 2004. v. 7, p. 6296–6301vol.7. ISSN 1062-922X.

ZADEH, L. A. *Fuzzy Sets, Information and Control*. [S.l.: s.n.], 1965. 338–353 p.

Apêndice A

REVISÃO SISTEMÁTICA

Este apêndice apresenta as etapas realizadas durante a atividade de revisão sistemática. São descritas as diretrizes propostas por Kitchenham and Brereton (KITCHENHAM *et al.*, 2009), que são divididas em três fases.

A.1 Planejamento da revisão sistemática

Esta revisão foi realizada sistematicamente com base nas diretrizes propostas por Kitchenham and Brereton (KITCHENHAM *et al.*, 2009). De acordo com eles, a condução de uma revisão sistemática é composta por três fases: (*i*) planejamento da revisão, (*ii*) condução da revisão e (*iii*) relato da revisão. Além disso, esta seção apresenta uma técnica de clusterização de artigos, o que facilitou a identificação dos artigos que foram revisados, sem que todos fossem lidos na íntegra. A Visual Text Mining (VTM) utiliza algoritmos de mineração de textos combinados com visualização interativa. A seguir são apresentados detalhes sobre como cada fase foi realizada.

A primeira tarefa a ser realizada é a definição do protocolo da revisão. O protocolo contém: (*i*) as questões de pesquisa, (*ii*) a estratégia de busca, (*iii*) os critérios de inclusão e exclusão e (*iv*) a extração de dados e método de síntese.

As questões de pesquisa devem estar relacionadas ao propósito da revisão, de modo que as respostas a essas questões estejam relacionadas ao escopo da revisão. Durante a realização desta etapa da revisão é importante definir o **escopo da revisão**. Kitchenham and Brereton (KITCHENHAM *et al.*, 2009) afirmam que o escopo deve ser definido utilizando-se o critério PICO. O escopo é composto por **População**, **Intervenção** e **Resultados de Relevância**.

A **População**, neste caso, são os possíveis problemas e soluções relacionados ao uso de redes de dados na implementação de 3DCVEs. A **Intervenção** é a literatura relacionada à 3DCVEs. Os **Resultados de Relevância** é o estado da arte sobre os estudos que foram conduzidos sobre 3DCVEs nos últimos anos, enfatizando os estudos primários que apresentam as possíveis soluções e problemas relacionados à área de pesquisa. A partir dos dados obtidos é apresentada a frequência de publicações na área e tendências futuras.

Os objetivos desta revisão são dois: (*i*) **encontrar os principais desafios relacionados à concepção de 3DCVE**; e (*ii*) **quais as soluções que tem sido utilizadas para resolver tais problemas**. Para que os objetivos fossem alcançados foram definidas algumas questões de pesquisa (QP):

QP$_1$ Quais são os protocolos de comunicação utilizados na intercomunicação entre 3DCVEs?

QP$_2$ Quais são as topologias de rede utilizadas na intercomunicação entre 3DCVEs?

QP$_3$ Quais são os desafios de implementação de 3DCVEs?

QP$_4$ Quais técnicas de avaliação (estratégias empíricas) tem sido empregadas na avaliação dos estudos encontrados? Por exemplo, estudos de caso, experimentos entre outros.

Depois de definidas as QPs uma *String* de Busca foi definida e também as escolha das bases de dados eletrônicas que seriam utilizadas. A *String* de Busca foi criada com as seguintes palavras-chave: *virtual reality, virtual environment, Cave Automatic Virtual Environment, CAVE, multiple projection, multiple screen, multi-projection, collaborative, collaboration, networked, distributed, network, topology and protocol*. Ela foi construída com operadores booleanos, como *AND* e *OR*. A Figura A.1 ilustra a *string* elaborada. A pesquisa abrangeu as bases dados eletrônicas consideradas como sendo as fontes científicas mais relevantes. As bases de dados eletrônicas selecionadas foram: *IEEE*, *ACM*, *Scopus*, *Springer*, *Science Direct* and *Web of Science*.

```
(("virtual reality") OR ("virtual environment")) AND (("Cave Automatic Virtual
Environment") OR ("CAVE") OR ("multiple projection") OR ("multiple screen") OR
("multi-projection")) AND (("collaborative") OR ("collaboration") OR ("networked")
OR ("distributed")) AND (("network") OR ("topology") OR ("protocol"))
```

Figura A.1: *String* de busca.

Com o intuito de determinar quais estudos primários foram relevantes para que os questionamentos de pesquisas fossem respondidos, critérios de inclusão e exclusão foram aplicados. Os critérios de inclusão aplicados foram:

(a) **O estudo primário apresenta ao menos uma solução para 3DCVE:** apresenta soluções à implementação de 3DCVE, tais como protocolos e topologias de rede, por exemplo.

(b) **O estudo primário apresenta ao menos um tipo de avaliação de verificação de performance e eficiência de 3DCVE:** sem resultados de avaliação não é possível realizar comparações entre as diferentes soluções encontradas.

Nem todos os critérios de inclusão devem estar presente nos estudos primários. Contudo, ao menos o item (a) deve estar presente. Se o item (b) também fosse obrigatório, a quantidade de estudos primários seria significamente menor.

Os critérios de exclusão aplicados foram:

(a) **O estudo primário não apresenta nenhum tipo de inovação em 3DCVE:** estudos primários que apresentam apenas caso de uso, sem apresentar detalhes relacionados à arquitetura ou processo de desenvolvimento.

(b) **O estudo primário é um *short paper*:** estudos com duas páginas ou menos não são considerados, visto que este tipo de estudo não apresenta informação suficiente.

A extração dos dados foi feita com o intuito de registrar as informações obtidas pelos pesquisadores com precisão. Um formulário padrão de extração de dados foi criado com os seguintes requisitos: *(i)* tipos de ambientes virtuais distribuídos encontrados; *(ii)* data de aquisição dos dados,; *(iii)* título, autores, journal e detalhes do estudo; e *(iv)*, um pequeno resumo sobre o estudo, sendo este realizado pelo pesquisador responsável pela leitura do mesmo.

Todos os estudos foram obtidos em portais digitais, tornando assim, a revisão replicável e extensível para outros pesquisadores.

A.2 Realização da revisão sistemática

Nesta fase, primeiramente, foram identificados os estudos nos portais digitais de pesquisa. O portal da IEEE foi o que retornou mais estudos relacionados, 449. Os portais ACM, Scopus, Springer LNCS e Science Direct retornaram 181, 212, 394 e 267, respectivamente. A base que retornou menor quantidade de artigos foi a Web of Science, apenas 15. Um dos possíveis motivos para a grande quantidade de estudos retornados está relacionado a indexação de estudos redundantes nos portais utilizados. No total, foram contabilizados 1518 estudos, sendo classificados como estudos candidatos 451. Contudo, depois da leitura dos títulos e resumos, e

aplicação dos critérios de inclusão e exclusão, a quantidade de estudos selecionados para leitura na íntegra foi de 132.

Tabela A.1: Estudos primários retornados de cada base de dados digital, total de estudos candidatos e o total de estudos primários identificados

Base de Dados Eletrônicas	Quantidade de Estudos Primários
IEEE	449
ACM	181
Scopus	212
Springer LNCS	394
Science Direct	267
Web of Science	15
Total	1518
Candidatos	451
Estudos Primários	132

A.3 Validação

Na fase de validação da seleção dos estudos primários foi utilizada uma abordagem que implementa a técnica *Visual Text Mining* (VTM), que faz uso de algoritmos e métodos de mineração de textos com visualização interativa. A ferramenta utilizada foi a *Projection Explorer* (PEx), que aplica decisões de inclusão e exclusão (MALHEIROS *et al.*, 2007).

A Figura A.2 apresenta um **mapa de documento** gerado a partir da PEx. Este mapa é composto por todos os estudos primários analisados durante a revisão sistemática, utilizando-se diferentes tons de cinza para diferenciar as fases em que o estudo foi removido da revisão. Os ponto brancos representam os estudos primários que foram removidos no primeiro estágio; os pontos cinza claro são os estudos primários que foram excluídos no segundo estágio; e os pontos pretos são os estudos incluídos.

A exploração do mapa de documento é realizada de duas maneiras: (*i*) primeiro, um algoritmo de clusterização é aplicado, com o intuito de criar grupos de estudos primários que sejam, de alguma forma, relacionados; (*ii*) posteriormente, os clusters resultantes são analisados em termos de: **Pure Clusters** - todos os documentos pertencentes a um cluster tem a mesma classificação (todos são incluídos ou excluídos, independentemente do estágio de exclusão); **Mixed Clusters** - representa os documentos com diferentes classificações no mesmo cluster. Estes casos são dicas para o revisor, isto é, os estudos primários contidos neste cluster devem ser revisados pelos revisores. De modo à facilitar a visualização, na Figura A.2 apenas cinco cluster gerados pela PEx são apresentados. Exemplos de **pure cluster** (todos excluídos) são identifi-

cados por meio do rótulo ”(a)” e, portanto, não precisaram ser revisados. Exemplos de **mixed cluster** (contendo pontos pretos (incluídos), e branco ou cinza (excluídos)) são identificados por meio do rótulo ”(b)”, os quais tiveram seus estudos primários revisados da forma tradicional.

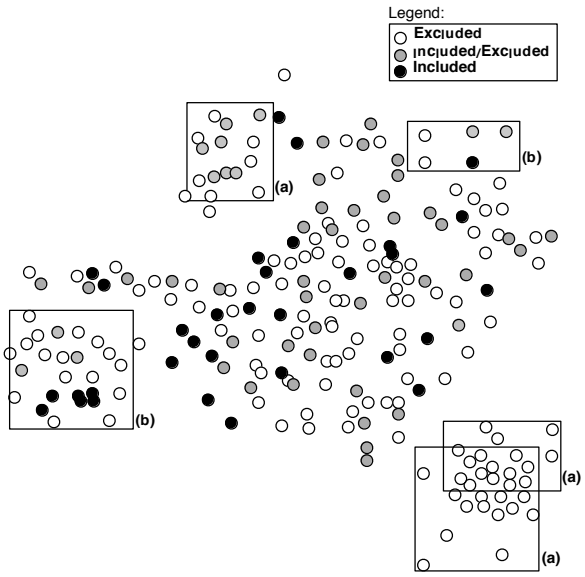


Figura A.2: Mapa de documento colorido com os estudos inclusos e excluídos.

A.3.1 Ameaças à validade

Seleção de estudos primários: a fim de assegurar um processo de seleção imparcial foram definidas questões de pesquisa e critérios de inclusão e exclusão com antecedência. Acredita-se que as questões e critérios foram detalhados o suficiente para fornecer uma avaliação de como o conjunto final de estudos primários deveria ser obtido. No entanto, não se pode descartar ameaças de uma perspectiva de avaliação de qualidade. Além disso, foi imposto um limite, em que apenas os estudos que foram publicados nos últimos 12 anos poderiam ser selecionados (2002-2015);

A não inclusão de estudos primários importantes: a busca por estudos primários foi realizada em vários motores de busca, embora seja possível que alguns estudos importantes tenham sido ignorados. No entanto, esta ameaça foi atenuada, selecionando motores de busca que são considerados como sendo as fontes científicas mais relevantes na área acadêmica; e

Extração de dados: outra ameaça para esta revisão refere-se a forma como os dados foram extraídos das bibliotecas digitais, uma vez que nem todas as informações eram apresentadas de forma explícita nos estudos primários. Assim, alguns dados tiveram de ser interpretados. Por conseguinte, a fim de garantir a validade, foram analisadas várias fontes de dados, isto é, artigos, relatórios técnicos e *white papers*.

A.4 Considerações finais

Este apêndice apresentou as diretrizes utilizadas no processo da revisão sistemática. No entanto, as discussões e respostas as questões de pesquisas são apresentadas no Capítulo 2.

Foram apresentadas as três etapas da revisão: planejamento da revisão, condução da revisão e relato da revisão. Além disso, foi utilizado um algoritmo de clusterização de estudos, com o intuito de auxiliar o processo de escolha dos estudos primários.

Apêndice B

REGRAS DO SISTEMA DIFUSO

Este apêndice apresenta a definição do conjunto de regras do sistema difuso que foi utilizado na realização do estuo de caso relacionados à verificação dinâmica das conexões inter–aglomerados gráficos distribuídos.

B.1 Regras FCL

```
1 FUNCTION_BLOCK dynamic-adaptive
2
3 VAR_INPUT
4   Delay: REAL;
5   TimeVariation: REAL;
6   Loss: REAL;
7 END_VAR
8
9 VAR_OUTPUT
10  DataPackages: REAL;
11  Protocol: REAL;
12  Buffer: REAL;
13  Predict: REAL;
14 END_VAR
15
16 FUZZIFY Delay
17  RANGE := (0.000 .. 600.000);
18  TERM notharmful := (0.000, 1.000) (30.000, 1.000) (60.000, 0.000);
19  TERM harmful := (55.000, 0.000) (120.000, 1.000) (140.000, 1.000)
        (200.000, 0.000);
```

```
20   TERM highlyharmful := (190.000, 0.000) (300.000, 1.000) (600.000,
         1.000);
21 END_FUZZIFY
22
23 FUZZIFY TimeVariation
24   RANGE := (0.000 .. 100.000);
25   TERM slightlyharmful := (0.000, 1.000) (4.000, 1.000) (5.000, 0.000);
26   TERM harmful := (4.000, 0.000) (6.000, 1.000) (8.000, 1.000)
         (10.000, 0.000);
27   TERM highlyharmful := (9.000, 0.000) (20.000, 1.000) (100.000,
         1.000);
28 END_FUZZIFY
29
30 FUZZIFY Loss
31   RANGE := (0.000 .. 20.000);
32   TERM withoutloss := (0.000, 1.000) (2.000, 1.000) (3.000, 0.000);
33   TERM low := (2.000, 0.000) (3.500, 1.000) (4.500, 1.000) (6.000,
         0.000);
34   TERM high := (5.000, 0.000) (10.000, 1.000) (20.000, 1.000);
35 END_FUZZIFY
36
37 DEFUZZIFY DataPackages
38   RANGE := (0.000 .. 1.000);
39   TERM small := (0.000, 1.000) (0.200, 1.000) (0.300, 0.000);
40   TERM medium := (0.280, 0.000) (0.350, 1.000) (0.450, 1.000) (0.600,
         0.000);
41   TERM big := (0.550, 0.000) (0.700, 1.000) (1.000, 1.000);
42   METHOD : COG;
43   ACCU : MAX;
44   DEFAULT := nan;
45 END_DEFUZZIFY
46
47 DEFUZZIFY Protocol
48   RANGE := (0.000 .. 1.000);
49   TERM udp := (0.000, 1.000) (0.200, 1.000) (0.300, 0.000);
50   TERM tcpudp := (0.250, 0.000) (0.350, 1.000) (0.450, 1.000) (0.600,
         0.000);
51   TERM sctp := (0.550, 0.000) (0.700, 1.000) (1.000, 1.000);
52   METHOD : COG;
53   ACCU : MAX;
54   DEFAULT := nan;
55 END_DEFUZZIFY
56
```

```
57 DEFUZZIFY Buffer
58  RANGE := (0.000 .. 1.000);
59  TERM noBuffer := (0.000, 1.000) (0.200, 1.000) (0.300, 0.000);
60  TERM level1 := (0.280, 0.000) (0.450, 1.000) (0.600, 1.000) (0.700,
       0.000);
61  TERM level2 := (0.680, 0.000) (0.800, 1.000) (1.000, 1.000);
62  METHOD : COG;
63  ACCU : MAX;
64  DEFAULT := nan;
65 END_DEFUZZIFY
66
67 DEFUZZIFY Predict
68  RANGE := (0.000 .. 1.000);
69  TERM noPredict := (0.000, 1.000) (0.200, 1.000) (0.300, 0.000);
70  TERM level1 := (0.250, 0.000) (0.350, 1.000) (0.450, 1.000) (0.600,
       0.000);
71  TERM level2 := (0.550, 0.000) (0.700, 1.000) (1.000, 1.000);
72  METHOD : COG;
73  ACCU : MAX;
74  DEFAULT := nan;
75 END_DEFUZZIFY
76
77 RULEBLOCK
78  AND : MIN;
79  OR : MAX;
80  ACT : MIN;
81  RULE 1 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is withoutloss then Buffer is noBuffer
82  RULE 2 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is withoutloss then Predict is noPredict
83  RULE 3 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is withoutloss then Protocol is udp
84  RULE 4 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is withoutloss then DataPackages is small
85  RULE 5 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is low then Buffer is level1
86  RULE 6 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is low then Predict is level1
87  RULE 7 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is low then Protocol is udp
88  RULE 8 : if Delay is notharmful and TimeVariation is slightlyharmful
       and Loss is low then DataPackages is small
```

```
89  RULE 9 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is withoutloss then Buffer is level1
90  RULE 10 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is withoutloss then Predict is noPredict
91  RULE 11 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is withoutloss then Protocol is udp
92  RULE 12 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is withoutloss then DataPackages is small
93  RULE 13 : if Delay is harmful and TimeVariation is harmful and Loss
        is withoutloss then Buffer is level1
94  RULE 14 : if Delay is harmful and TimeVariation is harmful and Loss
        is withoutloss then Predict is noPredict
95  RULE 15 : if Delay is harmful and TimeVariation is harmful and Loss
        is withoutloss then Protocol is udp
96  RULE 16 : if Delay is harmful and TimeVariation is harmful and Loss
        is withoutloss then DataPackages is small
97  RULE 17 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is low then Buffer is level1
98  RULE 18 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is low then Predict is level1
99  RULE 19 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is low then Protocol is tcpudp
100 RULE 20 : if Delay is harmful and TimeVariation is slightlyharmful
        and Loss is low then DataPackages is medium
101 RULE 21 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is low then Buffer is noBuffer
102 RULE 22 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is low then Predict is noPredict
103 RULE 23 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is low then Protocol is tcpudp
104 RULE 24 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is low then DataPackages is small
105 RULE 25 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is high then Buffer is level2
106 RULE 26 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is high then Predict is noPredict
107 RULE 27 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is high then Protocol is sctp
108 RULE 28 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is high then DataPackages is small
109 RULE 29 : if Delay is highlyharmful and TimeVariation is
        highlyharmful and Loss is withoutloss then Buffer is level2
```

```
110  RULE 30 : if Delay is highlyharmful and TimeVariation is
         highlyharmful and Loss is withoutloss then Predict is noPredict
111  RULE 31 : if Delay is highlyharmful and TimeVariation is
         highlyharmful and Loss is withoutloss then Protocol is sctp
112  RULE 32 : if Delay is highlyharmful and TimeVariation is
         highlyharmful and Loss is withoutloss then DataPackages is small
113  RULE 33 : if Delay is highlyharmful and TimeVariation is harmful and
         Loss is withoutloss then Buffer is noBuffer
114  RULE 34 : if Delay is highlyharmful and TimeVariation is harmful and
         Loss is withoutloss then Predict is noPredict
115  RULE 35 : if Delay is highlyharmful and TimeVariation is harmful and
         Loss is withoutloss then Protocol is sctp
116  RULE 36 : if Delay is highlyharmful and TimeVariation is harmful and
         Loss is withoutloss then DataPackages is small
117 END_RULEBLOCK
118
119 END_FUNCTION_BLOCK
```