UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Luiz Soares dos Santos Baglie

Visualização da informação 3D interativa em navegadores web – histórico e design de solução genérica

São José do Rio Preto
2018

Luiz Soares dos Santos Baglie

Visualização da informação 3D interativa em navegadores web - histórico e design de solução genérica

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Financiadora: CAPES.

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Instituto de Biociências, Letras e Ciências Exatas

Programa de Pós-Graduação em Ciência da Computação

Orientador: Prof. Dr. José Remo Ferreira Brega

São José do Rio Preto

2018

Baglie, Luiz Soares dos Santos.
Visualização da informação 3D interativa em navegadores web – histórico e design de solução genérica / Luiz Soares dos Santos Baglie. -- São José do Rio Preto, 2018
182 p. : il. , tabs.

Orientador: José Remo Ferreira Brega
Dissertação (Mestrado) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Biociências, Letras e Ciências Exatas

1. Ciência da computação. 2. Interação homem-máquina. 3. Realidade virtual. 4. Imagem tridimensional. 5. Internet. I. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. II. Título.

CDU – 681.32

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Luiz Soares dos Santos Baglie

Visualização da informação 3D interativa em navegadores web - histórico e design de solução genérica

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação , junto ao Programa de Pós-Graduação em Ciência da Computação , da Faculdade de Ciências da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de Bauru.

Financiadora: CAPES.

Trabalho aprovado. Bauru, 06 de Julho de 2018:

Prof. Dr. José Remo Ferreira Brega
Faculdade de Ciências
Universidade Estadual Paulista - Bauru
Presidente da Banca

Prof. Associado Edgard Afonso Lamounier Júnior
Faculdade de Engenharia Elétrica
Universidade Federal de Uberlândia

Prof. Assistente Danilo Medeiros Eler
Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista - Presidente Prudente

Bauru

06 de Julho de 2018

Agradecimentos

Agradeço, primeiramente, a meu orientador, Prof. Dr. José R.F. Brega, por sua excelente orientação e todos os esforços e apoio dedicados a mim.

Agradeço também a todas as pessoas que me apoiaram, de alguma maneira ou de outra, na realização deste trabalho (em ordem alfabética, agrupados por razão):

- Alexys B. Alfonso, Dr, Francisco C. Lavarda, Dr, e Paulo N. Lisboa, Dr, por sugestões e orientações sobre como prosseguir com o estudo de caso;
- Coordenação de Aperfeicoamento de Pessoal de Nível Superior (CAPES), pela bolsa de estudos para realização do Mestrado;
- Diego R.C. Dias, Dr, e Marcelo d.P. Guimarães, Dr, por revisões, comentários e críticas sobre artigos escritos durante o curso;
- Família e amigos, por estarem disponíveis quando precisei;
- Glésio Garcia de Paiva, Msc, Kelen C. Leite, Dr, Luis Carlos Trevelin, Dr, por cederem e auxiliarem com configuração do MiniCAVE no LAVIIC;
- Ignez Caracelli, Dr, pela paciência e compreensão de um experimento falho;
- Mário P. Neto, Msc, por seu apoio quase como co-orientador;
- Participantes do experimento realizado, pela contribuição ao trabalho; e
- Todos envolvidos no Programa de Pós-Graduação em Ciência da Computação da UNESP, sem os quais este trabalho não seria realizado.

Também agradeço àqueles que me ajudaram e apoiaram, mas com os quais minhas memórias, no momento, falham.

Resumo

Visualização de Informação (VI) é uma disciplina focada na exploração, monitoramento e descoberta de informações, colaboração profissional e apresentações de dados compreensíveis. VI 3D pode ser usada para dados inerentemente tridimensionais ou em visualizações que requerem entendimento de formas (e.g., design de automóvel, moléculas). Além disso, interação por meio dispositivos de entrada além de mouse e teclado auxilia a investigação de novas consultas. VI 3D interativa (VI3DI) pode ser desenvolvida com técnicas e recursos da Realidade Virtual, que envolve aplicações 3D com ambientes virtuais, proporcionando imersão, interação e envolvimento. Imersão pode se dar por sistemas de multi-projeção, os quais usam clusters gráficos (CGs); interação pode fazer uso de dispositivos de entrada diversificados. Navegadores web são interessantes para VI3DI, devido a facilidade de execução, portabilidade e recursos disponíveis. O objetivo desta dissertação é, com base no entendimento dos domínios e tendências presentes nos trabalhos científicos em que multi-projeção e dispositivos de interação são utilizados na web, propor, implementar, aplicar e avaliar um design para aplicações VI3DI na web. É realizada uma revisão sistemática da literatura, revelando a falta de estudos e recursos sobre uso de ambos multi-projeção e dispositivos de interação diversificados na web. Assim, propõe-se um design de solução para aplicações VI3DI web, apoiando a multi-projeção por meio de CGs mestre-escravo e descentralizado e dispositivos de interação diversificados. O design é implementado em um framework/biblioteca JavaScript, Livvclib, para validá-lo e facilitar seu reuso. Livvclib é usada adicionar suporte a multi-projeção e dispositivos de interação a uma aplicação web de visualização molecular. A aplicação é avaliada por meio de um experimento com 11 participantes junto a sistema de multi-projeção e dispositivos de interação. Resultados mostram aceitação do desempenho do design proposto, de sua implementação, da possibilidade de visualização das moléculas com a multi-projeção e dos dispositivos de interação diversificados.

Palavras-chave: Web, Sistema de Multi-Projeção, Cluster Gráfico, Interação Homem-Máquina.

Abstract

Information Visualization (IV) is a discipline focused on exploring, monitoring, and discovering information, professional collaboration, and understandable data presentations. 3D IV can be used for inherently three-dimensional data or in visualizations that require understanding shapes (e.g., automotive design, molecules). In addition, interaction via input devices besides mouse and keyboard helps the investigation of new queries. Interactive 3D IV (I3DIV) can be developed with techniques and resources from Virtual Reality, which involves 3D applications with virtual environments, providing immersion, interaction, and engagement. Immersion can be provided by multi-projection systems, which use graphical clusters (GCs); interaction can make use of diversified input devices. Web browsers are interesting for VI3DI, due to ease of execution, portability and features available. The purpose of this dissertation is to, based on the understanding of the domains and tendencies present in the scientific works in which multi-projection and interaction devices are used on the web, propose, implement, apply, and evaluate a design for VI3DI applications on the web. A systematic review of the literature is conducted, revealing the lack of studies and resources on the use of both multi-projection and diversified interaction devices on the web. Thus, we propose the design of a solution for VI3DI web applications, supporting the multi-projection through master-slave and decentralized GCs, and diversified interaction devices. The design is implemented in a JavaScript framework, Livvclib, to validate it and facilitate its reuse. Livvclib is used to add support to multi-projection and interaction devices to a web application for molecular visualization. The application is evaluated in an experiment with 11 participants, a multi-projection system, and interaction devices. Results show acceptance of the performance of the proposed design, its implementation, the possibility of visualization of the molecules with the multi-projection and of the diversified interaction devices.

Keywords: Web, Multi-Projection System, Graphic Cluster, Human-Computer Interaction.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACM	*Association of Computing Machinery* (do inglês, "Associação de Maquinário de Computação")
API	*Application Programming Interface* (do inglês, "Interface de Programação de Aplicações")
CAC	*Controller Application Communication* (do inglês, "Comunicação Aplicação Controle")
CAVE	CAVE *Automatic Virtual Environment* (do inglês, "Ambiente Virtual Automático CAVE")
CG	Cluster Gráfico
DCC	*Digital Content Creation* (do inglês, "Criação de Conteúdo Digital")
Dimmol	*Distributed Immersive Multi-platform Molecular visualization* (do inglês, "visualização Molecular Multi-plataforma Imersiva Distribuída"
E/S	Entrada/Saída
fps	*frames-per-second* (do inglês, "quadros-por-segundo")
HMD	*Head-Mounted Display* (do inglês, "Tela Montada na Cabeça")
ICC	Interface Cérebro-Computador
ICE	*Interactive Connectivity Establishment* (do inglês, "Estabelecimento de Conectividade Interativa")
IEEE	*Institute of Electrical and Eletronics Engineering* (do inglês, "Instituto de Engenharia Elétrica e Eletrônica")
IETF	*Internet Engineering Task Force* (do inglês, "Força Tarefa de Engenharia da Internet")
LAVIIC	Laboratório de Visualização Interativa Imersiva Colaborativa
LIV	Laboratório de Interfaces e Visualização
Livvclib	*Laboratory of Interfaces and Visualization's Web Cluster Library* (do inglês, "Biblioteca de Aglomerados Web do Laboratório de Interfaces e Visualização")
MDE	*Multi-Display Environments* (do inglês, "Ambientes Multi-Exibição")
MINY	*Multimodality Is Nice for You* (do inglês, "Multimodalidade É Bom para Você")

NAN	*Native Abstractions for Node.js* (do inglês, “Abstrações Nativas para Node.js”)
NPM	*Node Package Manager* (do inglês, “Gerenciador de Pacotes do Node”)
P2P	*Peer-To-Peer* (do inglês, “Ponto-A-Ponto”)
RA	Realidade Aumentada
RSL	Revisão Sistemática da Literatura
RV	Realidade Virtual
SAGE2	*Scalable Adaptive Graphics Environment 2* (do inglês, “Ambiente de Gráficos Adaptativos Escaláveis”)
SCTP	*Stream Control Transmission Protocol* (do inglês, “Protocolo de Transmissão de Controle de *Stream*”)
SDK	*Software Development Kit* (do inglês, “Kit de Desenvolvimento de Software”)
STUN	*Session Traversal Utilities for* NAT (do inglês, “Utilitários de Tráfego de Sessão para NAT”)
SO	Sistema Operacional
TI	Tecnologia da Informação
TURN	*Traversal Using Relays around* NAT (do inglês, “Travessia Utilizando *Relays* ao redor de NAT”)
VET	*Virtual Experience Test* (do inglês, “Teste de Experiência Virtual”)
VI	Visualização da Informação
VI3DI	Visualização da Informação Tridimensional Interativa
VRPN	*Virtual Reality Peripheral Network* (do inglês, “Rede Periférica de Realidade Virtual”)
W3C	*World Wide Web Consortium* (do inglês, “Consórcio Web Mundial”)
WebVR	*Web Virtual Reality* (do inglês, “Realidade Virtual da Web”)
WebRTC	*Web Real-Time Communication* (do inglês, “Comunicação em Tempo-Real na Web”)

Sumário

1 Introdução

Visualização de Informação (VI) é uma disciplina que oferece algoritmos, designs interativos e processos analíticos para apoiar a exploração, monitoramento e descoberta de informações, colaboração profissional e apresentações compreensíveis (SHNEIDERMAN; PLAISANT; HESSE, 2013). VI tridimensional, especificamente, pode ser usada para dados inerentemente tridimensionais ou em visualizações que requerem o entendimento de formas (MUNZNER, 2014). Conforme as visualizações e conjuntos de dados se tornam maiores e mais complexos, a interação se torna crucial para ultrapassar os limites do sistema visual humano e seus dispositivos de exibição (e.g., monitores de vídeo, projetores), permitindo que eles modifiquem a visualização a medida que novas consultas são investigadas (MUNZNER, 2014). Interação pode ser melhor entendida como o grau com que os usuários de um meio podem influenciar a forma ou o conteúdo do ambiente mediado, em tempo real, sendo determinada pela estrutura tecnológica do meio (STEUER, 1992), i.e. os dispositivos utilizados para realizar entrada de dados. Existem muitos dispositivos de entrada pelos quais a interação pode se dar, incluindo mouse e teclado.

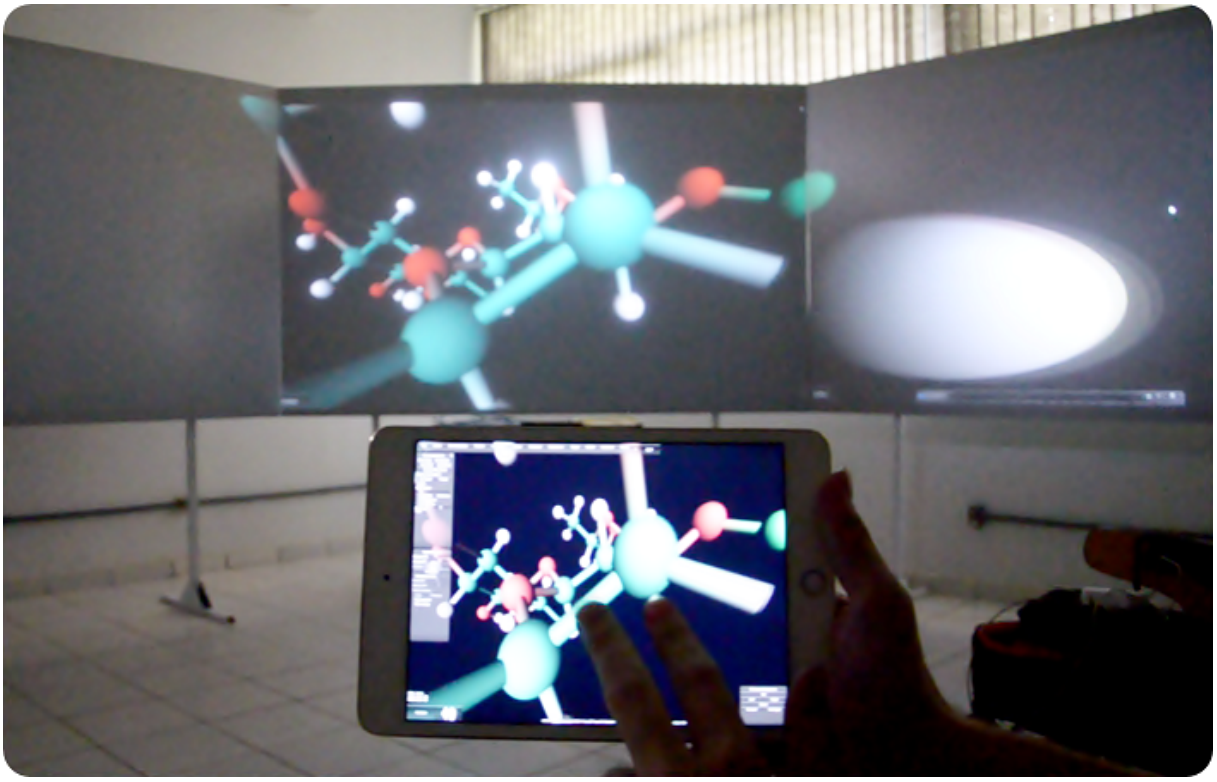
VI tridimensional interativa (abreviada nesta dissertação como "VI3DI") pode ser desenvolvida por meio de técnicas e recursos de Realidade Virtual (RV), disciplina que envolve aplicações 3D com ambientes virtuais, a fim de proporcionar imersão (captura da atenção do usuário), interação (responsividade às ações do usuário) e envolvimento (engajamento do usuário na atividade sendo desempenhada) (GUIMARÃES et al., 2018).

Entre as maneiras para proporcionar imersão, estão os sistemas de multi-projeção, dos quais o primeiro foi o CAVE *Automatic Virtual Environment* (CAVE; do inglês, "Ambiente Virtual Automático CAVE") (CRUZ-NEIRA et al., 1992). Esses sistemas são compostos por conjuntos de monitores de vídeo (ou telas) localizados em torno do(s) usuário(s); cada monitor mostra uma parte do ambiente virtual da aplicação, de forma que, combinados, eles apresentam uma visão ampla e contínua do ambiente virtual (NETO et al., 2015a). Uma das maneiras de utilizar sistemas de multi-projeção é por meio clusters gráficos, aglomerados de computadores interconectados através de uma rede a fim de renderizar alguma imagem, ou fluxo contínuo delas (CHEN et al., 2001a).

A Figura 1 mostra uma aplicação VI3DI para visualização de estruturas moleculares. No caso, a aplicação é exibida no sistema de multi-projeção MiniCAVE (DIAS et al., 2012) e executada no cluster gráfico associado a ele. A aplicação também permite utilizar o mouse e as telas sensíveis a toque de dispositivos móveis para manipular a visualização da estrutura molecular.

Os fatos de que a web e a capacidade de processar gráficos 3D constam, atualmente, tanto em *desktops* quanto em dispositivos móveis torna a integração desses elementos importante (CONGOTE et al., 2011). Os navegadores web são plataformas atraentes para aplicações VI3DI, pois o código-fonte de aplicações desenvolvidas para eles (que, necessariamente, se restringe as linguagens JavaScript, HTML e CSS) é, essencialmente, multi-plataforma e, também, devido a

Figura 1 – Aplicação VI3DI com sistema de multi-projeção e interação por meio de dispositivo móvel.



Fonte - Produzida pelo autor.

novas tecnologias, tais como *Application Programming Interfaces* (APIs; do inglês, "Interfaces de Programação de Aplicações") para JavaScript focadas na comunicação em tempo real entre duas páginas web (The Khronos Group, Inc, 2017) e, também, no processamento de gráficos 3D acelerado por hardware (World Wide Web Consortium, 2017b), sem necessidade de *plugins* ou componentes de terceiros.

1.1 Objetivos e Contribuição

Esta dissertação é resultante de pesquisa e desenvolvimento científico com o seguinte objetivo geral: com base no entendimento dos domínios e tendências presentes nos trabalhos científicos em que multi-projeção e dispositivos de interação são utilizados na web, propor, implementar, aplicar e avaliar um design para aplicações VI3DI na web.

Mais especificamente, visou-se realizar uma revisão sistemática da literatura sobre o uso de sistemas de multi-projeção e interação humano-computador juntos a web, a fim de se entender quais a tendências e práticas presentes nesses temas, a fim de se poder identificar recursos que permitam o desenvolvimento de aplicações VI3DI web. Com a identificação da falta desses recursos é, então, proposto um design de solução para aplicações VI3DI executadas em navegadores web, apoiando a utilização de clusters gráficos mestre-escravo e descentralizado (e, por consequência, multi-projeção) e, também, dispositivos diversificados de interação, usando a abordagem identificada como *dispositivo-servidor-navegador*. Visou-se implementar o design

proposto na forma de um *framework* JavaScript, a fim de validá-lo e promover sua incorporação em aplicações novas e ou existentes, permitindo, então, desenvolver uma aplicação VI3DI web de visualização molecular utilizando esta implementação. Com isso, é possível avaliar o desempenho e a qualidade do design proposto por meio, principalmente, de suas implementação (o *framework*) e aplicação, respectivamente.

A principal contribuição desta dissertação é fornecer recursos conceituais e de software que permitam melhor usufruir do poder dos navegadores web no desenvolvimento de aplicações VI3DI para web.

1.2 Organização da Dissertação

Esta dissertação está organizada em capítulos que, progressivamente, abordam os conceitos, desafios, propostas e implementações relacionados ao trabalho desenvolvido.

O Capítulo 2 Fundamentação apresenta os principais temas e conceitos sobre os quais esta dissertação se baseia, de maneira que o restante do trabalho possa ser melhor compreendido.

O Capítulo 3 Trabalhos Relacionados discute os resultados obtidos a partir de uma Revisão Sistemática da Literatura realizada sobre os temas e conceitos do capítulo anterior. O processo pelo qual essa revisão foi realizada é abordado no Apêndice A Revisão Sistemática da Literatura.

O Capítulo 4 Design de Solução para Aplicações Web de Visualização da Informação 3D Interativa apresenta a principal contribuição desta dissertação: o design proposto de solução para utilização de clusters gráficos e dispositivos de interação em aplicações VI3DI web.

O Capítulo 5 *Framework* para Aplicações Web de Visualização da Informação 3D Interativa apresenta a segunda contribuição principal desta dissertação: um *framework* JavaScript que implementa o design de solução apresentado no capítulo anterior, seu uso e avaliação de seu desempenho. As tecnologias utilizadas para tanto são melhor abordadas no Apêndice B Fundamentação Tecnológica.

O Capítulo 6 Estudo de Caso apresenta um estudo de caso, onde o *framework* desenvolvido do capítulo anterior é utilizado para adaptar uma aplicação já existente de visualização molecular, e discute um experimento realizado com a aplicação adaptada, bem como seus resultados. O roteiro do experimento consta no Apêndice D Roteiro do Experimento, enquanto o questionário utilizado para avaliação, no Apêndice E Questionário.

Por fim, o Capítulo 7 Conclusão encerra esta dissertação, apresentando observações finais, contribuições e trabalhos futuros.

2 Fundamentação

Este capítulo trata em maiores detalhes a base de temas, disciplinas e conceitos sobre a qual a pesquisa desta dissertação foi realizada. Mais especificamente, ele trata de Visualização da Informação, interação e dispositivos de interação, multi-projeção e clusters gráficos, finalizando com considerações sobre esses tópicos.

2.1 Visualização da Informação

Não há benefícios somente na coleta e armazenamento de dados por organizações e sistemas, pois essas ações não agregam valor aos dados; mais ainda, os enormes volumes de dados coletados e armazenados dificultam sua análise e compreensão pelo ser humano (BARIONI et al., 2002).

A capacidade de extrair informações desses dados, contudo, os tornam valiosos e, para isso, Visualização de Informação e Processos Analíticos Visuais são disciplinas que oferecem algoritmos, designs interativos e processos analíticos para apoiar exploração, monitoramento e descoberta de informações, colaboração profissional e apresentações compreensíveis (SHNEIDERMAN; PLAISANT; HESSE, 2013).

Os dados são manipulados e transformados, de maneira que representações interativas possam ser criadas para eles. Todavia, o propósito principal da VI não é o processamento dos dados ou de suas representações, mas, na verdade, a compreensão dos fenômenos descritos por eles, que pode ser fruto de um processo exploratório (CARR, 1999; CARD; MACKINLAY; SHNEIDERMAN, 1999; SARAIYA; NORTH; DUCA, 2005; SHNEIDERMAN, 2008; YI et al., 2008).

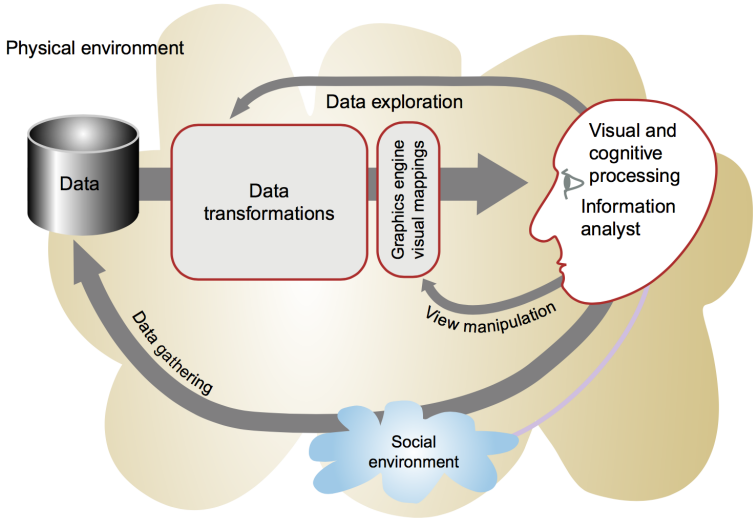
O processo empregado na VI inclui quatro etapas básicas, combinadas em vários ciclos de retroalimentação e ilustradas na Figura 2. As quatro etapas consistem em (WARE, 2012):

1. Coleta e armazenamento de dados;

2. Um estágio de pré-processamento projetado para transformar os dados em algo mais fácil de se manipular. Geralmente, há alguma forma de redução de dados para revelar aspectos selecionados. Exploração de dados é o processo de alterar o subconjunto que está sendo visualizado atualmente;

3. Mapeamento dos dados selecionados para uma representação visual, que é realizada por meio de algoritmos de computador que produzem uma imagem na tela. O usuário pode transformar os mapeamentos, destacar subconjuntos ou transformar a visualização. Geralmente isso é feito no próprio computador do usuário; e

Figura 2 – Processo de Visualização da Informação.



Fonte - Ware (2012).

4. Exploração dos dados por meio das representações gráficas geradas. Nesse momento, ocorre o processo de descoberta de conhecimento (por meio do sistema perceptivo e cognitivo humano), colaborando para a tomada de decisões.

Uma visualização, em si, é algo mais que uma representação gráfica dos dados ou conceitos, assim, sendo ela uma construção interna da mente, a visualização se torna um artefato externo que apoia a tomada de decisão (WARE, 2012). Visualizações também auxiliam na identificação de regiões de interesse e parâmetros para análises mais especializadas dos dados, ao possibilitar uma visão geral qualitativa de conjuntos grandes e complexos deles (GRINSTEIN; WARD, 2002).

Já representações dizem respeito a forma com que os dados são apresentados (e.g., linha, círculo, barra, etc.) (KIRK, 2012). Elas são baseadas em metáforas visuais, conceitos metafóricos entendidos e estruturados não apenas em termos próprios, mas, também, em termos de outros conceitos e que podem usadas para entender como diferenças sutis na forma da linguagem podem sugerir diferentes interpretações dos mesmos dados (LAKOFF; JOHNSON, 1980). As representações visam usufruir do sistema visual humano para análise dos dados, por meio da identificação de propriedades, tendências, padrões e discrepâncias presentes nos dados e nos relacionamentos entre eles (YI et al., 2008; CARR, 1999).

O sistema visual fornece um canal de alta largura de banda para nossos cérebros, sendo que uma quantidade significativa de processamento de informação visual ocorre em paralelo no nível pré-consciente (MUNZNER, 2014). A VI se aproveita disso ao possibilitar o relativo rápido entendimento de grandes quantidades de dados, por meio de suas representações; mais ainda, essa disciplina também apoia a percepção de propriedades desconhecidas anteriormente, identificação de problemas nos dados, facilidade no entendimento de características tanto de larga escala como de pequena escala, padrões de relacionamentos e formação de hipóteses (WARE, 2012).

Sendo assim, a VI procura facilitar a compreensão de dados como informações, a partir de representações visuais deles, o que lhe confere relevância a qualquer disciplina que envolva a

análise desses dados por um ser humano (ZORZAL et al., 2007).

2.1.1 Visualização da Informação Tridimensional

Devido, principalmente, aos mecanismos do sistema visual humano para reconhecimento de padrões, representações bidimensionais costumam ser particularmente bem sucedidas e oferecer técnicas mais maduras, além de poderem ser facilmente incluídas em mídia impressa (WARE, 2012). Todavia, o uso de uma terceira dimensão na Visualização da Informação também é estudado desde, pelo menos, o começo da década 1990 (Baroth; Chin; Curran, 1990). Estudos da época, tais como os de Robertson, Card e Mackinlay (1993), Carpendale, Cowperthwaite e Fracchia (1995), mostravam que versões 3D de representações tradicionais permitiam exibir muito mais conteúdo do que suas versões 2D, mas requeriam interações mais elaboradas para explorar todos os dados.

A VI 3D oferece maiores ganhos quando a tarefa requer, fundamentalmente, a compreensão da estrutura geométrica tridimensional de objetos ou cenas. Em quase todos esses casos, uma visualização 3D, com controles de navegação interativos para definir o ponto de vista 3D, permitem que os usuários construam um modelo mental útil da estrutura representada pelo conjunto de dados mais rapidamente do que simplesmente usando várias visualizações alinhadas em eixos 2D. Para essas tarefas, todos os custos de uso do 3D (e.g., falhas na percepção de profundidade, distorção perspectiva, oclusão (JOHN et al., 2001)) são superados pelo benefício de ajudar o visualizador a construir um modelo mental da geometria 3D (MUNZNER, 2014).

A maioria das tarefas que envolvem visualização de dados espaciais inerentemente 3D se beneficiam da VI 3D; alguns exemplos clássicos desses dados são o fluxo de um fluido sobre uma asa de avião, conjunto de dados de imagens médicas de tomografia do corpo humano ou interação molecular dentro de uma célula viva (MUNZNER, 2014). A VI 3D vem sendo usada para visualizar objetos complexos, como moléculas e até mesmo objetos abstratos, tal como a estrutura semântica de coleções de documentos (CARD; MACKINLAY; SHNEIDERMAN, 1999) e existem evidências experimentais de que a VI 3D supera a 2D em tarefas de compreensão de formas (JOHN et al., 2001).

Ambientes e técnicas de Realidade Virtual e Realidade Aumentada (RA) podem proporcionar formas de navegação e interação mais intuitivas para visualizações tridimensionais, o que desperta o interesse dos usuários na VI 3D pelas formas de interações possíveis, além do impacto visual, aumentando o envolvimento do usuário (ZORZAL et al., 2007).

2.1.2 Considerações

A VI 3D não é objetivamente melhor que a VI 2D, meramente por oferecer uma dimensão a mais na visualização; nem a VI 2D é melhor que a VI 3D, devido a maneira como o sistema visual humano reconhece padrões. Na realidade, existem *tipos* de problemas que podem ser melhor resolvidos através três dimensões, enquanto outros beneficiam-se de somente duas.

Certamente, os tipos de problemas melhor resolvidos pela VI 3D compartilham vários aspectos em comum, principalmente no que diz respeito a compreensões de forma e espaço, todavia,

isso não os torna menos importantes. De fato, alguns desses problemas são particularmente importantes para áreas científicas, tais como física e química, com a compreensão de estruturas moleculares, e medicina, com a compreensão de imagens obtidas de tomógrafos e visualizações gerais do corpo humano.

A VI 3D pode ser desenvolvida utilizando a RV (GUIMARÃES et al., 2018), todavia, é importante separá-las. O objetivo de um sistema RV é fornecer a melhor simulação possível de uma realidade artificial (MAGGIONI, 1993), e, para isso, a RV se utiliza de dispositivos e técnicas para trabalhar com ambientes virtuais 3D. Enquanto isso, a VI 3D pode utilizar subconjuntos das técnicas e recursos projetados para a RV, a fim de representar conjuntos de dados e facilitar a compreensão deles pelo visualizador.

2.2 Interatividade e Utilização de Dispositivos de Interação

A interatividade é crucial para a construção de ferramentas VI que lidam com a complexidade. Quando os conjuntos de dados são grandes o suficiente, as limitações de pessoas e dos sistemas de exibição impedem a exibição de todos os dados de uma só vez; nesses casos, a interação, onde as ações do usuário fazem com que a visualização seja alterada, se apresenta como uma solução. Além disso, uma única visualização estática pode mostrar apenas um aspecto de um conjunto de dados, o que, para algumas combinações de conjuntos de dados e tarefas simples, pode ser o suficiente; em contraste, uma visualização que muda de forma interativa suporta muitas consultas possíveis (MUNZNER, 2014).

A interação, em si, pode ser definida como "a extensão em que os usuários podem participar na modificação da forma e do conteúdo de um ambiente mediado em tempo real", e se comporta como uma variável motivada por estímulos, que pode ser determinada pela estrutura tecnológica do meio (STEUER, 1992).

As tarefas de interação são entradas de dados de baixo nível exigidas do usuário, tais como inserir uma cadeia de texto ou escolher um comando. Para cada tarefa, o designer da interação escolhe um dispositivo de entrada e uma técnica de interação apropriados. Uma técnica de interação é uma maneira de usar um dispositivo físico para realizar uma tarefa de interação (JACOB, 1996).

A entrada de dados no computador e a saída desses dados do computador estão cada vez mais se aproximando das interações do ser humano com o mundo real, o que implica na tentativa de tornar as ações do usuário que realizam a entrada de dados o mais próximo possível dos pensamentos que motivaram essas ações, a fim de reduzir a lacuna entre as intenções do usuário e as ações necessárias para inseri-las no computador. Isso permite explorar as habilidades adquiridas pelo ser humano através da evolução e da experiência; neste contexto, os modelos de interação da RV também ganham força, ao explorar as habilidades e expectativas pré-existentes do usuário (JACOB, 1996).

Em ambientes virtuais, tais como aqueles utilizados na VI 3D, controles de navegação que permitem ao usuário alterar o ponto de vista 3D invocam interativamente os mesmos mecanismos perceptuais para fornecer paralaxe de movimento. Em ambientes virtuais suficientemente

complexos, em que um único ponto de vista fixo não fornece informações suficientes sobre a estrutura do ambiente, a capacidade de navegação interativa é fundamental para entender a estrutura 3D (MUNZNER, 2014).

Técnicas de RV usadas na VI3DI, fazem uso de complexos modelos de interação (geralmente multimodais), o que, muitas vezes, dificulta sua gestão, pois entradas e saídas de dados podem ser muito complexas, dependendo de grande número de dispositivos de entrada (e.g., luvas de dados, rastreadores oculares, mouses 3D, controles de videogame, etc.) (NAVARRE et al., 2005).

Com isso, é importante que as aplicações que usam dispositivos para realizar a entrada de dados no computador possam abstrair ou generalizar a maneira como elas acessam esses dispositivos, pois, muitas vezes, eles possuem *Software Development Kits* (SDKs; do inglês, "Kits de Desenvolvimento de Software") e recursos próprios para isso, podendo ser, inclusive, produzidos por empresas em competição (e.g., Nintendo Wii Remote e Microsoft Kinect). Um modelo de acesso mais específico para um dispositivo em particular pode aproveitar melhor os recursos incomuns desse dispositivo; por outro lado, um modelo de acesso genérico torna a aplicação mais portátil e mais fácil de atualizar, manter e entender. Embora grandes mudanças (e.g., substituir mouse por luva em tarefa de interação) possam exigir reelaboração da interface do usuário, mudanças menores (e.g., trocar sistema de rastreamento por outro ou mudar entre modelos de um mesmo dispositivo de entrada), não devem exigir alterações na própria aplicação (BIERBAUM, 2000).

O suporte a dispositivos para interação, além de mouse e teclado, é particularmente importante para VI 3D, pois o oferecimento de modelos de interação mais avançados e/ou naturais para representações 3D permite maiores interação e, consequentemente, engajamento no sistema de visualização.

Contudo, diferentes representações e dados podem demandar diferentes tarefas de interação, que, por sua vez, podem demandar diferentes dispositivos de entrada. Portanto, recursos de software visando apoiar VI 3D devem restringir no mínimo possível a utilização desses dispositivos, a fim de não limitar as possíveis representações, nem criatividade.

É importante, também, ressaltar a diferença, feita por esta dissertação, entre "dispositivos de entrada" e "dispositivo de interação". Um dispositivo de entrada, como o nome indica, é um dispositivo que permite a entrada de dados (quaisquer que sejam) numa aplicação, com isso, esta dissertação define "dispositivo de interação" como um dispositivo de entrada cujos dados são usados em alguma interação com a aplicação, a fim de separar dispositivos que meramente fornecem dados e dispositivos usados para interagir. Assim, por exemplo, numa aplicação de video conferência, ambos mouse e *webcam* poderiam ser considerados dispositivos de entrada, mas somente o mouse seria considerado de interação, pois é usado para selecionar opções e outros detalhes; a *webcam* não seria considerada de interação porque seus dados não são usados para se interagir a aplicação, em si.

2.3 Sistemas de Multi-Projeção

A Visualização da Informação 3D Interativa pode utilizar subconjuntos de técnicas da RV para fornecer maior imersão do usuário na aplicação, além, também, de interação e envolvimento.

Assim como vários outros termos relacionados a RV (incluindo ela própria), o termo "imersão" pode ser entendido de várias maneiras. Uma delas é que ele consiste na substituição dos sentidos do usuário (em particular, a visão e audição) por estímulos sintéticos, gerados pela aplicação por meio de aparelhos especializados (BOWMAN; MCMAHAN, 2007). Outra maneira de entendê-lo é como a sensação de "estar lá", i.e. o usuário se sente dentro do ambiente virtual (PAUSCH; PROFFITT; WILLIAMS, 1997). Ainda outra maneira de entender o termo é como a vivacidade da representação dos dados pela aplicação (RYAN, 1999).

Em todo caso, há dois principais meios avançados de saída visual usados pela RV para proporcionar imersão (DISZ et al., 1995):

- *Head Mounted Display* (HMD; do inglês, "Tela Montada na Cabeça"), no qual o usuário utiliza um tipo especial de óculos com monitores de vídeo separados para cada olho, de maneira a criar o efeito de estereoscopia[1]. Exemplos atuais incluem Oculus Rift[2] e Google Cardboard[3]; e
- Sistema com Tecnologia de Projeção (ou, também, Sistema de Multi-Projeção), no qual as imagens são produzidas por projetores, sobre superfícies ao redor do(s) usuário(s). Sistemas desse tipo permitem múltiplos usuários simultaneamente dentro do sistema em execução. O exemplo principal é o CAVE (CRUZ-NEIRA et al., 1992).

Sistemas de multi-projeção (mais especificamente, aqueles projetados após o CAVE) podem ser considerados de última geração para visualização colaborativa imersiva (CORDEIL et al., 2017). O efeito de estereoscopia pode ser alcançado utilizando lentes polarizadas para os projetores, junto a óculos 3D, embora isso requira o rastreamento da posição da cabeça do usuário (CRUZ-NEIRA et al., 1992).

Aplicações projetadas para execução em sistemas de multi-projeção têm arquiteturas de renderização mais complexas, comparadas a aplicações comuns, já que seus gráficos devem que ser corretamente distribuídos entre várias saídas de vídeo que podem ter diferentes posições e orientações no mundo real. Muitos deles, incluindo o CAVE, realizam essa tarefa com o uso de clusters gráficos (CHEN et al., 2001a; CHEN et al., 2001b; DIAS et al., 2012; STAADT et al., 2003), que são compostos de vários computadores interconectados através de uma rede, cada um responsável por renderizar uma parte do ambiente virtual.

Sistemas de multi-projeção apresentam várias vantagens (SOARES, 2005):

- Em comparação com os HMDs, os óculos 3D polarizados são muito mais leves e não causam desconforto ou cansaço;

[1] Estereoscopia: técnica para criar efeito de profundidade.
[2] Oculus Rift: <https://www.oculus.com/rift/>.
[3] Google Cardboard: <https://vr.google.com/cardboard/>.

- Oferecem amplo campo de visão;
- Não há necessidade de representar o corpo do usuário no ambiente virtual; e
- Oferecem possibilidade de uso colaborativo com vários usuários ao mesmo tempo, sendo limitados apenas pelo espaço físico da sala.

2.3.1 MiniCAVE

O MiniCAVE é um sistema de multi-projeção de baixo custo composto pelos seguintes elementos (DIAS, 2011; DIAS et al., 2011):

- 3 telas de alto brilho medindo 2,5m de largura por 1,5m de altura, cada;
- 3 pares de projetores BenQ W100 de alta definição;
- 3 pares de lentes opostamente polarizadas;
- 6 PCs, cada um com processador Intel i7, 8GB de RAM e uma placa de vídeo NVIDIA FX 1800 758mb; e
- *Switch* 3Comm 1GB.

Cada PC está ligado a um dos projetores através da interface HDMI. Como todos eles estão conectados através do *switch* e são responsáveis por renderizar as imagens exibidas pelo MiniCAVE, eles criam um cluster gráfico (Seção 2.4 Cluster Gráficos) com seis nós.

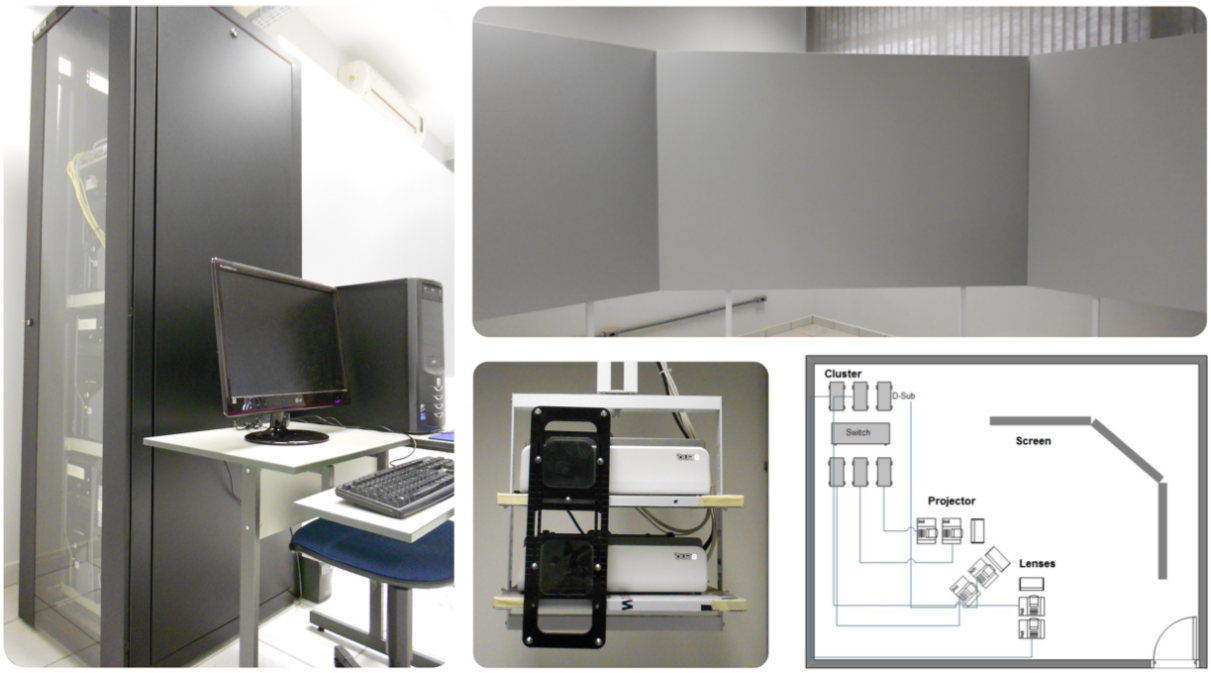
Cada par de projetores possui um par de lentes polarizadas e é apontado para uma das telas de alto brilho. Isso faz com que cada tela receba um par de imagens opostamente polarizadas que, junto aos óculos 3D polarizados, fornecem o efeito da estereoscopia.

Nesta dissertação, foi usado o exemplar do MiniCAVE montado no Laboratório de Interfaces e Visualização (LIV) da Faculdade de Ciências (FC) da Universidade Estadual Paulista (UNESP). A Figura 3 mostra um exemplar do MiniCAVE e cada um dos seus componentes.

2.3.2 Considerações

Quanto à maneira como imersão é proporcionada, esta dissertação tem foco específico no uso dos sistemas de multi-projeção, ignorando HMDs. A principal razão para isso é que, recentemente, o suporte a HMDs tem recebido uma grande quantidade de atenção, graças aos esforços e interesse de fabricantes e desenvolvedores. Em motores de jogo, especialmente, eles podem ser usados de maneira relativamente fácil, requerendo pequenos esforços e alterações no código-fonte da aplicação (Epic Games, Inc., 2017; Google, inc., 2018). Na web, o suporte a eles, embora ainda não totalmente maduro, progride graças à API JavaScript Web *Virtual Reality* (WebVR; “Realidade Virtual Web”). Por outro lado, sistemas de multi-projeção não são amplamente apoiados, dificultando seu uso na web.

Figura 3 – Sistema de multi-projeção MiniCAVE.



Fonte - Dias (2011).

2.4 Cluster Gráficos

Tradicionalmente, a renderização de aplicações em sistemas com múltiplas telas era feita, principalmente, através de um supercomputador gráfico ligado a vários dispositivos de exibição, capaz de utilizar recursos como memória compartilhada e multi-processamento, facilitando o desenvolvimento dessas aplicações; todavia, máquinas como essas eram muito caras, limitando a base de usuários (CHEN et al., 2001a; STAADT et al., 2003).

Com a evolução e barateamento do poder de processamento dos computadores, PCs destinados ao mercado consumidor passaram a oferecer capacidades de renderização cada vez melhores, tornando possível e viável a utilização de clusters (i.e. aglomerados) destes computadores para renderização de largas simulações de múltiplas telas, a relativos baixo custo e alto desempenho (CHEN et al., 2001a; STAADT et al., 2003).

Um cluster gráfico (CG), em si, consiste de computadores, chamados de *nós*, conectados através de uma rede, a fim de renderizar algum tipo de imagem, ou fluxo de imagens; cada nó está ligado a um dispositivo de exibição. O tipo da imagem a ser renderizada não é restrito, mas muitas pesquisas sobre renderização em CGs (incluindo esta dissertação) tratam da renderização de geometria poligonal distribuída pelo CG (STAADT et al., 2003). O propósito de clusters é tomar vantagem dos recursos adicionais de cada nó para aliviar gargalos do processamento (RAFFIN et al., 2006).

O fato de que processadores de um CG são conectados através de uma rede oferece maiores modularidade, flexibilidade e escalabilidade (CHEN et al., 2001a; RAFFIN et al., 2006), no entanto, o desenvolvimento de aplicações para supercomputadores de memória compartilhada (onde várias aplicações podem acessar os mesmos endereços de memória) é substancialmente

diferente do que para CGs. Novos desafios incluem definir, por exemplo, como as tarefas de renderização serão distribuídas pelo CG, quais dados serão sincronizados e como os nós do CG se conectarão (STAADT et al., 2003). Assim, ainda que existam vários recursos para o desenvolvimento de aplicações com multi-projeção por meio de CGs para diferentes plataformas, muitas vezes, o desenvolvimento é feito com o uso de APIs e bibliotecas de relativo baixo nível, o que dificulta a continuidade dos projetos e cria maiores dependências com o sistema operacional (SO) e/ou hardware (GUIMARÃES et al., 2018). Com isso, o desenvolvimento para CGs é complexo, tedioso, demorado e suscetível a vários erros, o que levou à criação de padrões como *Message Passing Interface*[4] e OpenPBS[5], além de várias bibliotecas para diferentes plataformas (STAADT et al., 2003) (e.g., Neto et al. (2015a), Lugrin et al. (2012), Sherman, Coming e Su (2013)).

2.4.1 Operação

Os nós de um CG podem trocar três tipos de dados, dependendo de como a execução da aplicação é sincronizada entre os nós (CHEN et al., 2001a):

- Controles: envolvem eventos da aplicação ou mudanças na visualização. São o tipo mais abstrato de dados trocados;
- Primitivas: instruções de renderização 2D ou 3D, que dizem o que os nós devem renderizar, mas não como; e
- Pixels: os próprios pixels que serão exibidos. Quando são enviados, isso indica que o nó transmissor já efetuou a parte bruta da renderização.

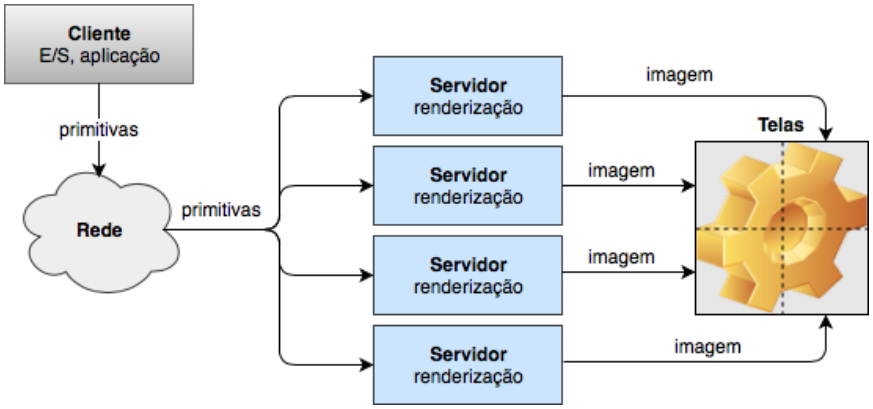
A maneira como os dados são distribuídos entre os nós também é de grande importância e pode ser implementada por meio de duas arquiteturas principais (CHEN et al., 2001a; CHEN et al., 2001b): cliente-servidor e mestre-escravo.

Na arquitetura cliente-servidor, os nós são classificados em *cliente* ou *servidor*. Só existe um nó cliente, com o qual o usuário interage e o qual é responsável por executar a aplicação e por processar a entrada/saída (E/S) de dados, mas não pela renderização dos gráficos. O nó cliente elabora os controles/primitivas e geometrias a serem renderizados e os transmite aos nós servidores, que são responsáveis exclusivamente por renderizá-los (CHEN et al., 2001a). Os controles/primitivas podem ser transmitidos a todo quadro desenhado, ou armazenados localmente pelos nós servidores, sendo que, nesse caso, o nó cliente só transmite as diferenças entre um quadro e outro. A Figura 4 apresenta um esquema de como funciona a arquitetura cliente-servidor; o nó cliente envia as primitivas de renderização, enquanto os nós servidores se encarregam de renderizá-las nas saídas de vídeo.

Essa abordagem é mais transparente para o desenvolvedor, pois permite que o processo de renderização na instância da aplicação do nó cliente seja substituído por outro, que transmita os dados da renderização para os nós servidores, sem alteração nas APIs (HASHIMOTO; ISHIDA;

[4] *Message Passing Interface*: <http://mpi-forum.org>.
[5] OpenPBS: <http://open.pbs.org/home/>.

Figura 4 – Arquitetura da renderização cliente-servidor.

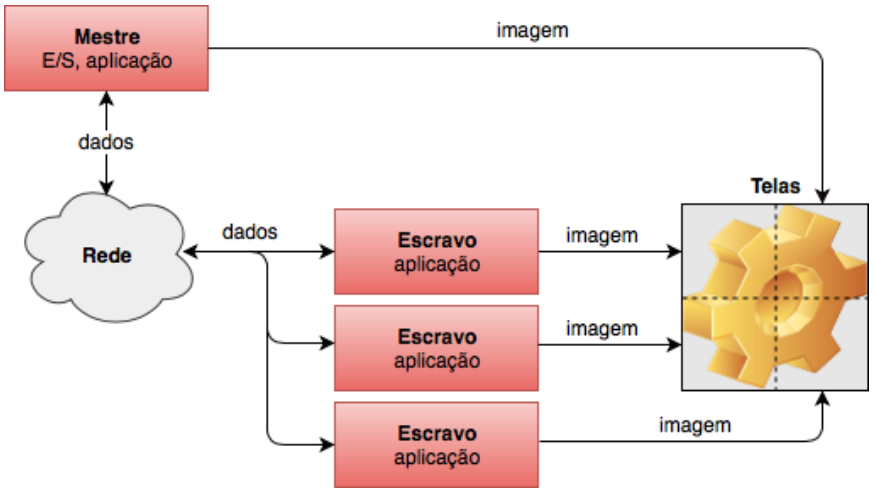


Fonte - Produzida pelo autor.

SATO, 2007). No entanto, a quantidade de dados trafegados pela rede, principalmente quando os nós servidores não tem armazenamento local, é razoavelmente grande, criando a possibilidade de que a rede se torne o gargalo no desempenho dessas aplicações (HASHIMOTO; ISHIDA; SATO, 2007; STAADT et al., 2003).

Na arquitetura mestre-escravo, os nós são classificados em *mestre* ou *escravo* e todos eles são responsáveis por executar e renderizar a aplicação. Só existe um nó mestre, com o qual o usuário interage. O nó mestre não envia instruções de renderiazação aos escravos, ao invés disso, ele sincroniza o seu estado, que é, então, replicado nos nós escravos, dessa maneira, por sua própria natureza, somente as mudanças nos dados são transferidas (HASHIMOTO; ISHIDA; SATO, 2007; STAADT et al., 2003). A Figura 5 apresenta um esquema de como funciona a arquitetura mestre-escravo; todos os nós executam a aplicação e tem seus estados sincronizados, porém somente o nó mestre recebe entrada do usuário.

Figura 5 – Esquema da renderização mestre-escravo.



Fonte - Produzida pelo autor.

Essa abordagem é deveras menos transparente para o desenvolvedor, pois é necessário tratar da sincronização do estado da aplicação nos nós (HASHIMOTO; ISHIDA; SATO, 2007; STAADT et al., 2003), que pode ser entendido como o conjunto de valores armazenados pelas

variáveis e objetos da aplicação. Mesmo com o uso de bibliotecas, muitas vezes, a arquitetura mestre-escravo necessita de alterações em código-fonte já existente e trabalho extra por parte do desenvolvedor, todavia, a troca de somente esses tipos de dados implica em menor carga na rede, o que torna a abordagem mais adequada a clusters de PCs, que geralmente trabalham em ambientes de banda não tão larga (HASHIMOTO; ISHIDA; SATO, 2007).

É importante notar que estas arquiteturas de renderização não devem ser confundidas com a arquitetura de comunicação das aplicações, que também pode usar dos nomes "cliente-servidor" e "mestre-escravo". É perfeitamente plausível a existência de aplicações para CGs com renderização mestre-escravo e comunicação cliente-servidor (e.g., Neto et al. (2015a)).

Como mostrado no Capítulo 5 *Framework* para Aplicações Web de Visualização da Informação 3D Interativa, o próprio *framework* desenvolvido nesta dissertação utiliza renderização mestre-escravo, com comunicação cliente-servidor. No caso, o *framework* é usado para estabelecer um CG mestre-escravo, onde os nós são páginas web (exibidas por navegadores web). As páginas web, em si, são carregadas de um servidor web usando comunicação cliente-servidor.

2.4.2 Recursos Existentes

Diversas bibliotecas e *plugins* existem atualmente para auxiliar o desenvolvimento de aplicações 3D de múltiplas telas usando CGs. Também existem extensões que podem ser adicionadas a motores de jogo ou gráficos, que permitem compilar/executar as aplicações em sistemas de múltiplas telas. Muitos desses recursos são voltados a RV, assim, estudá-los envolve também o estudo sobre bibliotecas, *plugins* e extensões de RV.

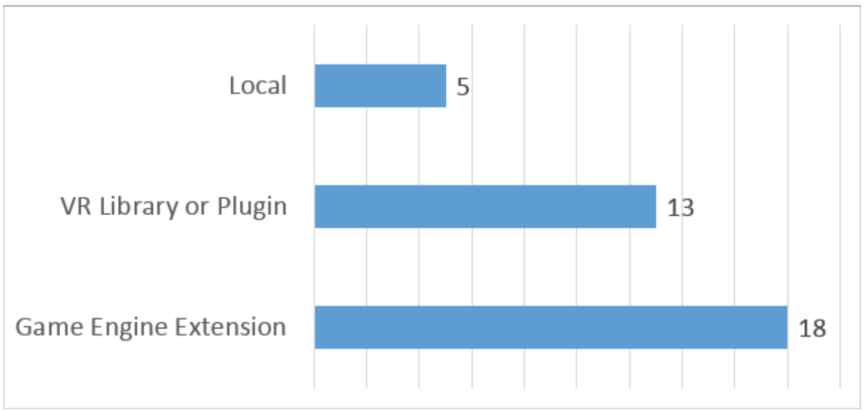
Esses recursos de RV podem envolver exibição tanto em HMDs quanto em sistemas de multi-projeção. Do lado da multi-projeção, existem aqueles que suportam somente uma arquitetura de renderização (cliente-servidor ou mestre-escravo) e outros que suportam ambas (SHERMAN; COMING; SU, 2013). O modelo mestre-escravo acaba sendo preferido, todavia, devido a apresentar uma taxa de *frames-per-second* (fps; do inglês, "quadros-por-segundo") maior, com menor latência de rede (comparado ao modelo cliente-servidor) (STAADT et al., 2003).

Dentre as bibliotecas de RV disponíveis, as duas mais utilizadas são FreeVR e D-vision; já quanto aos *plugins*, o mais usado é o MiddleVR, disponível para várias plataformas e motores (NETO; BREGA, 2015). Contudo, muitas outras opções existem, ao ponto de que é impraticável abordá-las todas nesta dissertação. A Figura 6 exibe as abordagens utilizadas para implementação de multi-projeção em 36 estudos revisados por Neto e Brega (2015). É possível observar que a grande maioria dos estudos utiliza extensões para motores a jogo, enquanto uma minoria prefere não utilizar nenhuma solução existente, implementando as suas próprias.

2.4.3 Considerações

Existem diversas estratégias para a implementação de aplicações VI3DI com multi-projeção por meio de CGs, contudo, a decisão mais importante para essas aplicações certamente é a escolha da arquitetura de renderização, pois ela influencia outros aspectos, tais como quais dados serão sincronizados e como isso será feito.

Figura 6 – Abordagens utilizadas para implementação de multiprojeção em 36 estudos pesquisados.



Fonte - Neto e Brega (2015).

No caso desta dissertação, o fator de maior peso na escolha da arquitetura de renderização é o foco proposto na execução das aplicações em navegadores web, que, por natureza, são plataformas visuais; adicionalmente, o JavaScript é executado numa única *thread*, causando a necessidade de que as instâncias das aplicações tomem tão pouco tempo quanto possível trocando dados, a fim de que mais tempo seja dedicado a renderização bruta. Com isso, a renderização cliente-servidor, na qual o cliente pode, para todos os efeitos, não apresentar módulo visual (algo que não pode ser evitado em um navegador web) e que apresenta relativa grande carga na rede, não é adequada.

Devido a maneira como operam e seu propósito de uso, CGs são similares a *multi-display environments* (MDEs; do inglês, "ambientes multi-exibição") e, portanto, podem ser confundidos com eles, assim, é importante, também, entender a distinção entre CGs e MDEs. MDEs são sistemas de aplicações que visam integrar vários dispositivos (geralmente incluindo dispositivos móveis, tais como smartphones e tablets) em um único ambiente interativo (SEYED et al., 2013). A principal diferença entre eles é que CGs distribuem e sincronizam o conteúdo de uma mesma aplicação para vários dispositivos de exibição, enquanto um MDE compreende um sistema de aplicações e dispositivos operando em simbiose. Recursos especializados para o desenvolvimento de aplicações para CGs tendem a ser particularmente pervasivos (como é o caso do design de solução proposto por essa dissertação), pois devem operar sob detalhes da aplicação, tais como sincronização de estado e motor de renderização. Já um MDE é composto por aplicações diferentes para cada dispositivo utilizado, podendo inclusive, incluir sistemas de multi-projeção (e.g., Moraes, Eler e Brega (2014)).

3 Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados a esta dissertação, resultados de uma revisão sistemática de trabalhos científicos que tratam de multi-projeção e de dispositivos de interação na web.

O meio principal escolhido para pesquisa de trabalhos relacionados a esta dissertação foi a Revisão Sistemática da Literatura (RSL), também chamada somente de "Revisão Sistemática". A RSL é um meio para identificar, avaliar e interpretar os trabalhos relevantes a uma questão de pesquisa; ela é considerada um tipo de estudo *secundário*, pois é construída sobre trabalhos individuais projetados para alavancar o estado-da-arte, que são considerados estudos *primários* (KITCHENHAM; CHARTERS, 2007).

O Apêndice A Revisão Sistemática da Literatura apresenta as ideias e procedimentos envolvidos numa RSL, bem como os parâmetros e definições projetados para esta em particular. Esta seção visa discutir os resultados obtidos pela RSL realizada, bem com apresentar em particular alguns dos estudos primários encontrados.

Como discutido na Subseção A.1.1 Objetivos e Questões, ainda que esta dissertação trate da VI3DI como a sinergia entre a imersão e a interação (e consequente envolvimento), a RSL foi realizada com duas buscas, uma focando em aplicações web projetadas para sistemas de multi-projeção e outra focando em aplicações web utilizando dispositivos de interação diversificados. Assim, os resultados dessas buscas são abordados, inicialmente, separadamente e, então, de maneira unificada.

3.1 Multi-Projeção em Navegadores Web

3.1.1 Visão Geral

A RSL realizada encontrou, ao todo, 13 estudos primários que utilizaram ou desenvolveram técnicas ou recursos de multi-projeção em navegadores web. Essa quantidade de estudos foi considerada extremamente baixa, pois corresponde a apenas cerca de 4% do total (294) de estudos primários retornados pela lógica de busca aplicada. Todavia, dada a abrangência da lógica de busca, acredita-se que isso seja um indicativo de que o tema ainda não é devidamente explorado pela literatura. Outros dados possivelmente reforçando esse ponto são os anos em que os estudos primários incluídos foram publicados e os domínios do conhecimento em que se focam.

Todos os estudos primários incluídos foram publicados a partir de 2014, ainda que a busca tenha limitasse os resultados a partir de 2010. Isso sugere que o interesse na utilização da web para multi-projeção é relativamente recente e que há várias questões ainda não respondidas. Para referência, as APIs JavaScript WebGL, que pode ser usada para renderização de gráficos 3D acelerados por hardware em navegadores web, e WebRTC, que pode ser usada para conexões *Peer-To-Peer* (P2P; do inglês, "Ponto-A-Ponto") entre navegadores web, tiveram seus primeiros versão e rascunho, respectivamente, publicados em 2011 (The Khronos Group, Inc, 2017; World

Wide Web Consortium, 2017b).

Em relação aos domínios do conhecimento, a frequência com que foi abordado o domínio "Desenvolvimento de Software" (MARRINAN et al., 2014; SCHWEDE; HERMANN, 2015; GEYMAYER; SCHMALSTIEG, 2016; SU et al., 2016; GRUBERT; KRANZ, 2017; GUIMARÃES et al., 2018), também vai de encontro a noção de que ainda existem várias questões não respondidas sobre o tema, especialmente no que diz respeito a implementação. Ainda nessa questão, a frequência do domínio "Visualização e Interação" (SCHWEDE; HERMANN, 2015; MARAI; FORBES; JOHNSON, 2016; REN et al., 2016; SU et al., 2016; GRUBERT; KRÄNZ, 2017; KIM et al., 2017) reforça a existência de benefícios na utilização de sistemas de multi-projeção para a VI interativa.

Quanto aos tipos de dispositivos de exibição utilizados nos estudos primários incluídos, curiosamente, observou-se que dispositivos móveis foram usados com relativa frequência (segunda maior, logo após projetores e painéis LCD, que empataram). Nos estudos em que isso ocorreu (KIM et al., 2017; GRUBERT; KRANZ, 2017; GRUBERT; KRÄNZ, 2017; RÄDLE et al., 2014), fatores elencados para tanto incluíram a presença de uma tela sensível a toque, a portabilidade e a mobilidade.

Nos softwares desenvolvidos nos estudos primários incluídos foram projetados, observou-se que a grande maioria deles (10 de 13) foi projetada para "vários" usuários, i.e. uma quantidade de usuários não especificada, limitadamente, possivelmente, unicamente pelo tamanho da sala. Dentre os três estudos que projetaram um software para somente um usuário (GRUBERT; KRANZ, 2017; GRUBERT; KRÄNZ, 2017; PATTANAKIMHUN; CHINTHAMMIT; CHOTIKAKAMTHORN, 2017), dois deles foram projeraram o software para exibir o conteúdo através de dispositivos móveis, o que sugere que a multi-projeção utilizando dispositivos móveis oferece maiores benefícios quando é destinada a ser utilizada por somente um usuário.

Dentre os estudos primários incluídos, há aqueles (CHUNG et al., 2014; MARRINAN et al., 2014; MARAI; FORBES; JOHNSON, 2016; REN et al., 2016; SU et al., 2016; PATTANAKIMHUN; CHINTHAMMIT; CHOTIKAKAMTHORN, 2017) em que são utilizadas aplicações separadas para interação do usuário (acessadas, geralmente, de notebook ou dispositivo móvel) e para o sistema de multi-projeção; mais ainda, alguns deles (CHUNG et al., 2014; REN et al., 2016) utilizam aplicações web somente para as aplicações acessadas pelo usuário.

A popularidade de abordagens descentralizadas para a implementação de CGs nos estudos primários incluídos (CHUNG et al., 2014; RÄDLE et al., 2014; GEYMAYER; SCHMALSTIEG, 2016; GRUBERT; KRANZ, 2017; GRUBERT; KRÄNZ, 2017) sugere que aplicações de multi-projeção para a web frequentemente tem como requisito a possibilidade de os usuários interagirem com qualquer um dos nós ou telas exibindo a aplicação, i.e. de que qualquer nó no CG possa sincronizar seu estado com todos os outros. Isso não era esperado, mas, entre as duas abordagens principais listadas na literatura para implementação de CGs (cliente-servidor e mestre-escravo; Seção 2.4 Cluster Gráficos), a abordagem mestre-escravo tem a vantagem, pois pode ser ajustada para fornecer descentralização com o que se julga ser relativa facilidade, como demonstrado na Subseção 5.2.1 Cluster Gráfico Mestre-Escravo.

Também é interessante observar que a abordagem cliente-servidor é popular na multi-projeção na web (MARRINAN et al., 2014; MARAI; FORBES; JOHNSON, 2016; REN et al., 2016; SU et al., 2016), uma vez que requer maior largura de banda que a abordagem mestre-escravo, potencialmente tendo desempenho mais severamente afetado, quando a aplicação é acessada remotamente pela internet (nos estudos primários incluídos, as aplicações são acessadas da rede local onde o servidor de aplicação está).

Já quanto a maneira como instâncias de uma aplicação de multi-projeção que utiliza CGs trocam dados, é surpreendente observar que somente um estudo primário incluído (GUIMARÃES et al., 2018) utiliza a tecnologia WebRTC (Seção B.1 WebRTC), uma vez que, considerando a possibilidade de estabelecer conexões P2P entre páginas web sem necessidade de plugins, se imagina que ela ofereça o melhor desempenho; nota-se, todavia, que WebRTC ainda é uma tecnologia não totalmente finalizada e suportada e que o estudo que a usa é o mais recente entre os incluídos, sugerindo, talvez, que a infrequência de seu uso seja mais provavelmente devido a (curta) passagem do tempo, do que de negligência dos pesquisadores.

Quanto a recursos de software reusáveis para o desenvolvimento de aplicações de multi-projeção para a web, dois principais encontrados nos estudos primários incluídos foram SAGE2 (MARRINAN et al., 2014) e HuddleLamp (RÄDLE et al., 2014). O fato de que a grande maioria dos outros estudos primários optou por implementar suas próprias soluções de multi-projeção indica a existência de requisitos nessas aplicações que esses dois recursos não são capazes de atender e/ou, também, que eles não são especialmente populares ou conhecidos entre pesquisadores.

Dentre os dois recursos, somente o SAGE2 foi utilizado em outros estudos primários incluídos (MARAI; FORBES; JOHNSON, 2016; SU et al., 2016). Juntos, esses estudos oferecem duas razões principais para usarem o SAGE2: as funcionalidades que ele oferece para o desenvolvimento de aplicações interativas de multi-projeção e a possibilidade de desenvolver aplicações de multi-projeção usando bibliotecas de JavaScript (e.g., D3.js[1]), uma característica que, espera-se, seja comum a qualquer recurso de software reusável para o desenvolvimento de aplicações de multi-projeção na web.

3.1.2 Estudos Primários Selecionados

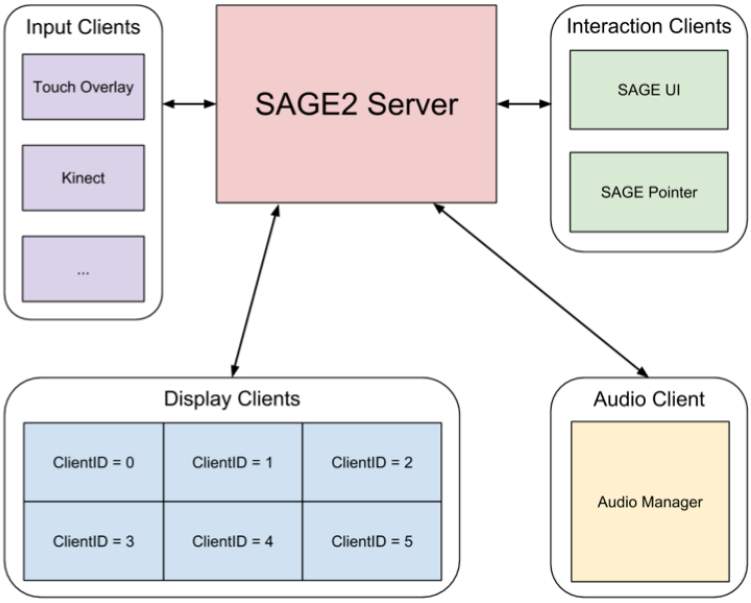
Esta subseção visa apresentar alguns dos estudos primários incluídos mais próximos desta dissertação.

O *Scalable Adaptive Graphics Environment 2* (SAGE2; do inglês, "Ambiente de Gráficos Adaptativos Escaláveis") (MARRINAN et al., 2014) é um servidor de aplicação customizado que fornece, a vários usuários, um ambiente operacional comum para acessar, exibir e compartilhar uma variedade de informações que exigem muitos dados. Ele exibe *streams* de pixels de dispositivos remotos, utilizando redes de alta velocidade para renderizar conteúdo, desde imagens e vídeos de alta definição a documentos PDF e telas de laptops. Ele oferece suporte a áudio e dispositivos de interação, além de recursos para o desenvolvimento de aplicações web que podem ser executadas nele. A Figura 7 ilustra a arquitetura do SAGE2; nela, todos os elementos, fora o servidor, são

[1] D3.js: <https://d3js.org>.

navegadores web que acessam endereços específicos servidos pelo SAGE2. Ainda que o SAGE2 seja código-aberto, o que, em tese, permite que ele seja fundido a servidores de aplicação já existentes ou seja usado no desenvolvimento de novos, o fato de que seu núcleo funcional reside justamente no lado servidor das aplicações web dificulta/desencoraja isso. Em termos de desempenho, ele suporta CGs cliente-servidor e a conexão entre os nós é intermediada pelo servidor, ambos fatores que podem comprometer o desempenho de aplicações, caso sejam acessadas de fora da rede local do servidor de aplicação.

Figura 7 – Arquitetura e o esquema de comunicação do SAGE2.

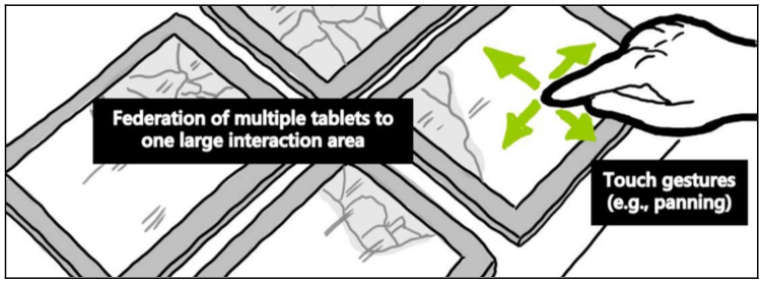


Fonte - Marrinan et al. (2014).

O HuddleLamp (RÄDLE et al., 2014) é uma luminária de mesa com uma câmera RGB-D integrada que monitora precisamente movimentos e posições de dispositivos móveis e das mãos sobre uma mesa. Isso permite o desenvolvimento de aplicações com reconhecimento espacial para colaboração ao redor de uma mesa e sem uma mesa interativa sensível a toque. O HuddleLamp fornece uma API JavaScript para o desenvolvimento de aplicações web que o utilizam. A Figura 8 ilustra a navegação síncrona por um mapa, usando o HuddleLamp; cada dispositivo recebe do servidor qual parte do mapa exibir com base em sua posição (calculada pelo HuddleLamp) e é atualizado quando a interação é detectada com algum deles. A principal barreira do HuddleLamp é ele próprio, uma vez que a necessidade de usar a lâmpada para determinar a posição e conteúdo de cada tela limita os dispositivos de exibição a dispositivos móveis e restringe o espaço disponível para a multi-projeção, não sendo escalável. Os nós não trocam dados entre si; ao invés disso, é o servidor (no qual o HuddleLamp está conectado) que calcula posições e conteúdos e os remete aos nós.

Geymayer e Schmalstieg (2016) propuseram uma infraestrutura *desktop* integrada para cognição e colaboração distribuídas. Essa infraestrutura permite a distribuição e manipulação de conteúdo web por um CG descentralizado, bem como o recebimento de dados de dispositivos de interação. A Figura 9 apresenta a arquitetura da infraestrutura proposta. Embora a infraestrutura proposta permita conexão P2P entre os nós, ela requer a instalação de um *plugin* no navegador

Figura 8 – Ilustração de navegação por mapa usando um ambiente de HuddleLamp.

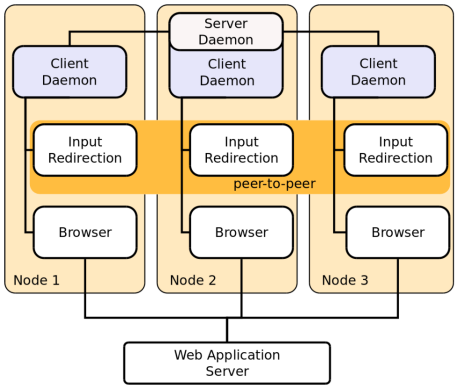


Fonte - Rädle et al. (2014).

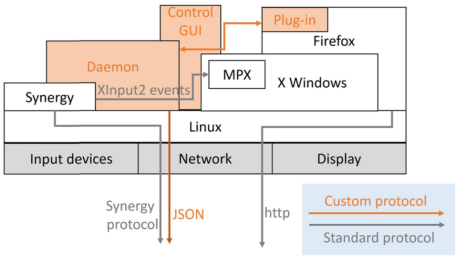
web, o qual, por sua vez, depende do SO GNU/Linux, diminuindo a acessibilidade de aplicações desenvolvidas para ela.

Figura 9 – Arquitetura da infraestrutura proposta por Geymayer e Schmalstieg (2016).

(a) Arquitetura de aplicação web com a infraestrutura.



(b) Arquitetura geral.



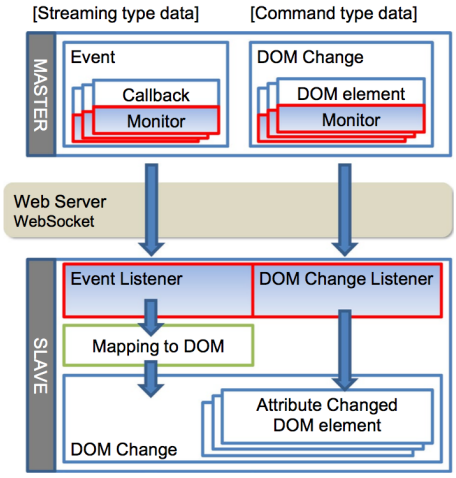
Fonte - Geymayer e Schmalstieg (2016).

Kim et al. (2017) propuseram um método de visualização distribuída de conteúdo X3D, através de navegadores web, utilizando HTML e o framework X3DOM[2]; o framework de distribuição dos dados nesse método é mostrado na Figura 10. Eles utilizaram CGs mestre-escravo para distribuir o conteúdo X3D por PCs desktop, sistemas de multi-projeção e dispositivos móveis e conseguiram carregar estruturas tridimensionais de até 72mb, com mais de 400 mil polígonos. O método proposto, contudo é especializado para visualização de conteúdo X3D, não permitindo outro tipo de conteúdo; os autores também optaram usar o servidor de aplicação para intermediar

[2] X3DOM: <https://www.x3dom.org>.

a troca de dados entre os nós, o que pode, possivelmente, afetar o desempenho em determinadas configurações dos nós.

Figura 10 – Framework de distribuição de dados em CG mestre-escravo exibindo conteúdo X3D.



Fonte - Kim et al. (2017).

Por fim, design de solução desenvolvido e proposto nesta dissertação se baseia amplamente no estudo de Guimarães et al. (2018), em que é estudada, elaborada e avaliada uma arquitetura para o desenvolvimento de aplicações RV imersivas e interativas para execução em navegadores web, por meio de CGs mestre-escravo; a arquitetura, em si, é mostrada na Figura 11. Os autores adaptaram uma aplicação que carrega documentos Virtual Reality Markup Language (do inglês, "Linguagem de Demarcação de Realidade Virtual") para execução no sistema de multi-projeção MiniCAVE, e validando a aplicação adaptada, obtiveram resultados considerados positivos, no que diz respeito ao processo de adaptação do código-fonte existente e do desempenho; nota-se que este é o único estudo primário incluído que usou WebRTC para troca de dados entre os nós. Entretanto, a arquitetura proposta não dá suporte a dispositivos de interação diversificados, a fim de permitir interações mais elaboradas. A arquitetura também não foi implementada na forma de um recurso de software reusável para aplicações existentes ou novas.

3.2 Dispositivos de Interação em Navegadores Web

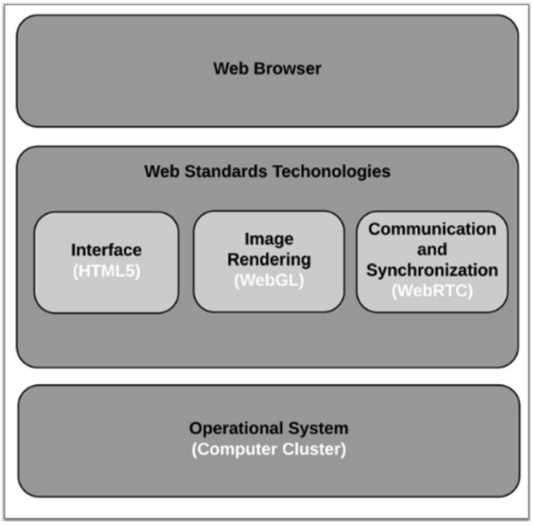
3.2.1 Visão Geral

A RSL realizada encontrou, ao todo, 26 estudos primários que utilizaram ou desenvolveram técnicas ou recursos para utilização de dispositivos de interação em navegadores web. Essa quantidade de estudos é considerada razoável, pois corresponde a cerca de 15% do total (167) de estudos primários retornados pela lógica de busca aplicada. A lógica de busca, em si, poderia ser mais abrangente, todavia, considera-se que ela permitiu a inclusão de quantidade e variedade suficiente de estudos primários para se obter um panorama adequado do tema.

A quantidade de estudos primários incluídos por ano flutuou bastante, sem tendência aparente[3]. Houve anos sem publicações no tema (2011) e, também, anos com relativamente

[3] Embora pareça haver uma tendência decrescente entre 2016-2018 na quantidade de publicações por ano, é

Figura 11 – Arquitetura de software para desenvolvimento de aplicações RV imersivas e interativas executadas em navegadores web.



Fonte - Guimarães et al. (2018).

bastante publicações (2014) (OAT; FRANCESCO; AURA, 2014; MAEDA; KOBAYASHI, 2014; GHATAK et al., 2014; RAHMAN et al., 2014; DO; CAI; JIANG, 2014; MARRINAN et al., 2014; WILSON; BROWN; BIDDLE, 2014). Isso sugere que o interesse na utilização de dispositivos de interação em navegadores web é intermitente, seja porque interações mais comuns já são tidas como suficientes, porque o tema apresenta dificuldades que desencorajam a pesquisa, ou, ainda, por outras questões desconhecidas; no segundo caso, considera-se importante que as dificuldades sejam estudadas e, se possível, ultrapassadas.

Quanto aos domínios do conhecimento, assim como analisado nos estudos relacionados a multi-projeção, a relativa alta frequência com que o domínio "Desenvolvimento de Software" (HAK; DOLEZAL; ZEMAN, 2012; GAO; OSMAN; SADDIK, 2013; LIU et al., 2013; MARRINAN et al., 2014; MORENO et al., 2015; ASSAL; CHIASSON; BIDDLE, 2016; ŞERBAN; CULIC, 2016; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017; SATHE et al., 2017; DIPIERRO, 2018) foi abordado sugere a existência de questões ainda não respondidas sobre o tema, enquanto a relativa alta frequência com o domínio "Visualização e Interação" (MOTTA; NEDEL, 2012; OAT; FRANCESCO; AURA, 2014; MAEDA; KOBAYASHI, 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; PAKKANEN et al., 2017; SEO; LEE; YOO, 2017; MEIXNER; KALLMEIER, 2016) foi abordado reforça a existência de benefícios na utilização de dispositivos de interação diversificados em aplicações web.

A manipulação dos atributos do conteúdo exibido no navegador foi o principal uso dado aos dispositivos de interação nos estudos primários incluídos (GHATAK et al., 2014; RAHMAN et al., 2014; DO; CAI; JIANG, 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; ASSAL; CHIASSON; BIDDLE, 2016; PAKKANEN et al., 2017; HAN et al., 2017; SANFILIPPO et al., 2017; DIPIERRO, 2018; SEO; LEE; YOO, 2017; MEIXNER; KALLMEIER, 2016; WILSON; BROWN; BIDDLE, 2014), o que era esperado, pois

necessário lembrar que a RSL diz respeito somente até 28 de Abril de 2018

este tipo de interação é o mais próximo da definição de "interação" apresentada na Introdução[4]. As outras modalidades principais de interação, seleção e navegação, também foram bastante utilizadas, demonstrando a importância desse tipo de ação.

Também de importância para esta dissertação foi a informação de que somente uma pequena parte (MARRINAN et al., 2014; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017; PALLEC et al., 2010) dos estudos primários incluídos dá suporte a múltiplos tipos de dispositivos de intração, o que reforça a importância de recursos reusáveis de software que atendam a esses requisitos e facilitem o desenvolvimento de aplicações web utilizando dispositivos de interação diversificados.

Já quanto aos tipos, em si, de dispositivos de interação usados para interação nos estudos primários incluídos, sensores de movimento, seja para o corpo inteiro (e.g., Microsoft Kinect, Asus Xtion) (MOTTA; NEDEL, 2012; MARRINAN et al., 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; MORENO et al., 2015; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017) ou para somente as mãos (Leap Motion) (DO; CAI; JIANG, 2014; RITTITUM; VATANAWOOD; THONGTAK, 2016; PAKKANEN et al., 2017; SEO; LEE; YOO, 2017) tem destaque, indicando que o reconhecimento de gestos é um importante requisito para aplicações mais interativas na web. É interessante, também, ver a utilização de dispositivos móveis (LIU et al., 2013; OAT; FRANCESCO; AURA, 2014; HAN et al., 2017), certamente devido a seus vários recursos para entrada de dados, tais como tela sensível a toque, acelerômetro, giroscópio, etc. Em todo caso, os dispositivos de interação usados mais peculiares foram Interface Cérebro-Computador (ICC) (PALLEC et al., 2010) e um roteador (ŞERBAN; CULIC, 2016); este último devido as capacidades particulares do modelo utilizado, que permitiam uso de microfone, câmera e outras modalidades de entrada.

Por uma grande margem, a abordagem mais popular para utilizar os dados de dispositivos de interação em navegadores web foi aquela a qual esta dissertação se refere como *dispositivo-servidor-navegador* (MOTTA; NEDEL, 2012; DO; CAI; JIANG, 2014; MARRINAN et al., 2014; MONSERRAT et al., 2015; MORENO et al., 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017; SANFILIPPO et al., 2017; SEO; LEE; YOO, 2017; PALLEC et al., 2010). Com ela, a maneira como os dados do dispositivo são servidos é variável, embora a opção mais comum nos estudos primários incluídos seja o uso de WebSockets. Essa abordagem é extremamente flexível e requer apenas recursos já disponíveis nos navegadores atuais, no entanto, ela também apresenta maior latência no recebimento dos dados e pode incorrer em mais trabalho de desenvolvimento, devido a necessidade de desenvolver um servidor de aplicação.

Outra abordagem interessante é hospedar um servidor de aplicação responsável por servir os dados do dispositivo no próprio dispositivo (LIU et al., 2013; ŞERBAN; CULIC, 2016). Ela tem limitações óbvias, pois somente dispositivos de interação com capacidades computacionais podem atuar como servidores, no entanto, é flexível (no que diz respeito aos dados do dispositivo específico) e permite acesso ao dispositivo sem intermediários.

[4] "Interação pode ser melhor entendida como o grau com que os usuários de um meio podem influenciar a forma ou o conteúdo do ambiente mediado, em tempo real, sendo determinada pela estrutura tecnológica do meio" (STEUER, 1992)

Contudo, percebe-se também que não há recursos devidamente consistentes para obtenção e transmissão dos dados de dispositivos de interação. Alguns estudos (MAEDA; KOBAYASHI, 2014; RAHMAN et al., 2014; ASSAL; CHIASSON; BIDDLE, 2016; WILSON; BROWN; BIDDLE, 2014) utilizaram dispositivos que foram mapeados como mouses no computador, eliminando, assim, a necessidade de desenvolver/trabalhar com maneiras elaboradas de se obter os dados desses dispositivos; já outros, utilizaram recursos fornecidos pelo HTML5 (GHATAK et al., 2014; DIPIERRO, 2018; MEIXNER; KALLMEIER, 2016) ou a API JavaScript WebVR (FITTKAU; KRAUSE; HASSELBRING, 2015; PAKKANEN et al., 2017; SATHE et al., 2017) para trabalhar com *webcams*/microfones e HMDs de RV, respectivamente. Essas abordagens facilitam o uso desses tipos de dispositivos específicos, porém, não dão suporte a outros tipos; isso, acredita-se, contribuiu para a popularidade da abordagem *dispositivo-servidor-navegador* para aquisição dos dados pelas aplicações web, todavia, mesmo assim, para a maioria dos dispositivos (MOTTA; NEDEL, 2012; MONSERRAT et al., 2015; MORENO et al., 2015; SANFILIPPO et al., 2017; SEO; LEE; YOO, 2017), foram desenvolvidos servidores de aplicação customizados para servir seus dados, ao invés de serem usados recursos já disponíveis, sugerindo a existência de requisitos que os recursos existentes não são capazes de atender e/ou que eles não são especialmente populares ou conhecidos entre pesquisadores.

Dada essa inconsistência nos recursos utilizados para o desenvolvimento dos estudos primários incluídos, há uma dificuldade em elencar as razões para escolha dos recursos para usar dispositivos de interação em navegadores, pois muitos estudos não necessitaram ou não usaram tais recursos ou simplesmente não elucidaram as razões para suas escolhas. Todavia, das razões que se pôde colher, as funcionalidades oferecidas pelo recurso (GHATAK et al., 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017) e o suporte a vários tipos de dispositivos (GAO; OSMAN; SADDIK, 2013; ŞERBAN; CULIC, 2016; MEIXNER; KALLMEIER, 2016) foram razões relativamente mais comuns; alternativamente, outra razão relativamente comum foi o fato de que o recurso utilizado era o único disponível (FITTKAU; KRAUSE; HASSELBRING, 2015; PAKKANEN et al., 2017; SATHE et al., 2017). Essas razões sugerem que, dado um recurso, a possibilidade de se trabalhar com vários tipos de dispositivos é importante não só para o desenvolvimento de tipos diferenciados de aplicação, como também, tipos diferenciados de interações.

3.2.2 Estudos Primários Selecionados

O estudo de Marrinan et al. (2014), em que é apresentado o recurso SAGE2, foi o único estudo primário incluído em ambas as buscas da RSL realizada. Além do suporte a multi-projeção na web, já discutido na Subseção 3.1.2 Estudos Primários Selecionados, ele também suporta a utilização de dispositivos de interação na web usando a abordagem *dispositivo-servidor-navegador*. No caso, como mostra a Figura 7, ele faz uso de "*Input Clients*" (do inglês, "Clientes de Entrada"), nós responsáveis por obter os dados dos dispositivos de interação, por meio da biblioteca Omicron[5], e repassá-los ao servidor SAGE2 (supondo que ele mesmo não atue como *Input Client*), o qual os repassa para os nós que processam a aplicação e exibem o conteúdo. Novamente, o SAGE2 é

[5] Omicron: <https://github.com/uic-evl/omicron>.

projetado para atuar como um servidor completo o que, ainda que seja código-aberto, dificulta seu uso em aplicações que já possuem infraestrutura do lado do servidor ou que desejam/necessitam utilizar tecnologias diferentes das usadas por ele.

Multimodality Is Nice for You (MINY; do inglês, "Multimodalidade É Bom para Você") (PALLEC et al., 2010) é um kit de recursos para implementação de barramento de eventos, permitindo o uso de dispositivos de interação remotos por aplicações em várias plataformas, incluindo navegadores web. Ele também opera no modelo *dispositivo-servidor-navegador*, onde os clientes se registram para eventos disparados pelo servidor, quando este obtém dados dos dispositivos. Único entre os recursos encontrados, o MINY é responsável tanto por capturar os dados dispositivos quanto por transmiti-los; ele suporta os seguintes dispositivos diferenciados, ICCs, leitor RFID, *webcams* e dispositivos no padrão X10[6]. O MINY, contudo, é composto por um servidor de aplicação, o que dificulta sua utilização, assim como acontece com o SAGE2; além disso, ele parece ter sido descontinuado - no site do MINY[7], a última atualização do código-fonte é de 2012.

Konstantinidis, Bamparopoulos e Bamidis (2017) usaram o *Controller Application Communication* (CAC; do inglês, "Comunicação Aplicação Controle") *Framework* (KONSTANTINIDIS et al., 2015) para desenvolver aplicações web de exercícios para pessoas idosas ou debilitadas. O CAC-*Framework* é, como o nome indica, um *framework* específico para transmissão de dados de dispositivos entrada diversificados para aplicações web. Ele suporta cinco tipos de dispositivos: Nintendo Wii Remote e Wii Balance Board, Microsoft Kinect, Neurosky Mindwave (ICC) e *smartwatches* Android. A Figura 12 mostra uma visão geral do design de solução proposto pelo CAC-*Framework*; observa-se que ele, assim como nos outros estudos primários selecionados, também utiliza a abordagem *dispositivo-servidor-navegador*. A figura também mostra que o CAC-*Framework* possui a mesma fraqueza dos recursos discutidos nos outros estudos primários selecionados: necessita de um servidor próprio, específico para seu funcionamento; mais ainda, o CAC-*Framework* parece ser código-fechado, o dificulta seu uso pela comunidade.

3.3 Considerações Finais

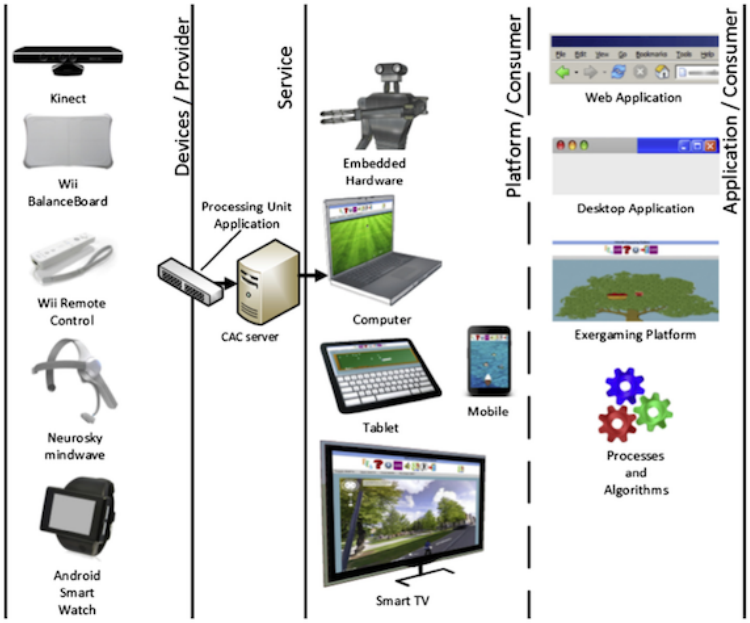
Com a avaliação dos estudos primários encontrados nas buscas realizadas pela RSL, é possível, então, responder as perguntas elaboradas a partir dos objetivos estabelecidos (Subseção A.1.1 Objetivos e Questões):

- **Como é aplicada a multi-projeção em navegadores web?**

  A multi-projeção na web pode ser aplicada de várias maneiras, entre as quais as mais populares são cluster descentralizado e cluster cliente-servidor. Nesses casos, os nós podem trocar dados por intermédio do servidor ou através de algum *plugin* adicionado ao navegador web. Outras maneiras menos populares para tanto incluem a utilização de cluster mestre-escravo e de múltiplas saídas de vídeo num único computador (criando, assim, um único desktop gigante).

[6] Padrão X10 para automação caseira: <http://kbase.x10.com>.
[7] MINY: <http://www.lifl.fr/miny>.

Figura 12 – Visão geral do CAC-*Framework*.



Fonte - Konstantinidis et al. (2015).

A maioria dos estudos sobre o tema foca em Desenvolvimento de Software, seja para auxiliar a utilização da multi-projeção na web ou para investigar novas maneiras de realizar certos de seus aspectos, e/ou em Visualização e Interação, investigando e implementando visualizações de dados que se beneficiam do uso da web para uso.

A Tabela 1 compila os principais dados coletados dos estudos primários incluídos para a busca a respeito de multi-projeção na web.

- **Como dispositivos de interação, além de mouse e teclado, podem ser utilizados em navegadores web?**

  Dispositivos de interação diversificados podem ser usados em navegadores web por meio de várias maneiras, entre as quais a abordagem mais popular é *dispositivo-servidor-navegador*, que consiste em usar um servidor de aplicação para servir os dados dos dispositivos a aplicações em navegadores web. Outra abordagem relativamente popular, embora um pouco limitada, é *dispositivo-navegador*, onde são usados recursos (tais como *drivers* do SO e recursos do HTML5) para que os dados do dispositivo sejam obtidos diretamente pela aplicação web.

  Assim como com a multi-projeção, a maioria dos estudos sobre o tema foca em Desenvolvimento de Software, seja investigando ou desenvolvendo maneiras para que os dispositivos de interação possam ser usados por aplicações web, e/ou em Visualização e Interação, investigando e implementando modelos de interação mais diversificados em aplicações web.

  A Tabela 2 compila os principais dados coletados para cada dispositivo de interação utilizado nos estudos primários incluídos para a busca a respeito de dispositivos de interação na web. Os dispositivos, em si, usados pelos estudos, foram omitidos dos dados, devido ao espaço e para permitir o foco nos dados que são exibidos.

- **Quais recursos possibilitam o desenvolvimento de aplicações VI3DI com multi-projeção para navegadores web?**

  Dentre os recursos reusáveis de software apresentados e desenvolvidos nos estudos primários incluídos, o único que permite o desenvolvimento de aplicações VI3DI com multi-projeção para navegadores web é o SAGE2 (MARRINAN et al., 2014). Ele apoia a multi-projeção por meio de cluster cliente-servidor e permite o uso de dispositivos de interação diversificados usando a abordagem *dispositivo-servidor-navegador*, onde os dados dos dispositivos são obtidos usando a biblioteca Omicron. Contudo, a abordagem cliente-servidor implica em maior carga de rede na aplicação, o que pode prejudicar seu desempenho, junto ao fato de que ele utiliza o servidor de aplicação para intermediar comunicação entre os nós do cluster; mais ainda, ele é disponibilizado como um servidor de aplicação, o que dificulta seu uso em infraestruturas existentes ou que usam tecnologias diferentes.

  Foram encontrados estudos e recursos que apoiam somente a multi-projeção na web (e.g., HuddleLamp (RÄDLE et al., 2014)) ou somente o uso de dispositivos de interação diversificados na web (e.g., CAC-Framework (KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017), MINY (PALLEC et al., 2010)), todavia, a possibilidade de usá-los em conjunto é limitada (devido a seus designs) ou desconhecida (devido a código-fonte fechado e/ou obsoleto).

Tabela 1 – Principais dados coletados dos estudos primários incluídos na busca por multi-projeção na web.

Estudo	Solução para Multi-Projeção	Meio de Troca de Dados	Recurso
Marrinan et al. (2014)	CG cliente-servidor	Pelo servidor (WebSockets)	SAGE2
Schwede e Hermann (2015)	Múltiplas saídas de vídeo	Não se aplica	Não se aplica
Ren et al. (2016)	CG cliente-servidor	P2P (TCP)	Customizado
Su et al. (2016)	CG cliente-servidor	Pelo servidor (WebSockets)	SAGE2
Marai, Forbes e Johnson (2016)	CG cliente-servidor	Pelo servidor (WebSockets)	SAGE2
Geymayer e Schmalstieg (2016)	CG descentralizado	P2P (WebSockets)	Customizado
Grubert e Kranz (2017)	CG descentralizado	P2P (WebSockets)	Customizado
Grubert e Kränz (2017)	CG descentralizado	P2P (WebSockets)	Customizado
Pattanakimhun, Chinthammit e Chotikakamthorn (2017)	Múltiplas saídas de vídeo	Não se aplica	Não se aplica
Chung et al. (2014)	CG descentralizado	Pelo servidor (WebSockets)	Customizado
Rädle et al. (2014)	CG descentralizado	Pelo servidor (WebSockets)	HuddleLamp
Kim et al. (2017)	CG mestre-escravo	Pelo servidor (WebSockets)	Customizado
Guimarães et al. (2018)	CG mestre-escravo	P2P (WebRTC)	Customizado

Fonte – Produzida pelo autor.

Dessa maneira, também se observa a falta de algum recurso comum (seja conceitual, na forma de um design a fim de solucionar os desafios, ou de software, na forma de *framework* ou

biblioteca de software) para o desenvolvimento de aplicações VI3DI com multi-projeção para navegadores web que requeiram exclusivamente os recursos oferecidos por padrão em navegadores web atuais para apoiar a multi-projeção e distribuição de visualização 3D por meio de cluster descentralizado (e, por extensão, mestre-escravo), com os nós se comunicando de maneira tão direta quanto possível, ao mesmo tempo em que dão suporte a diversos tipos de dispositivos de interação, usando a abordagem *dispositivo-servidor-navegador* ou algo ainda mais direto.

Tabela 2 – Principais dados coletados para cada dispositivo de interação utilizado nos estudos primários incluídos na busca por dispositivos de interação na web.

Estudo	Design da Solução	Recurso Dispositivo	Recurso Transmissão
Hak, Dolezal e Zeman (2012)	dispositivo-plugin-navegador	Formato de dados	Manitou
Motta e Nedel (2012)	dispositivo-servidor-navegador	Kinect SDK	Customizado
Gao, Osman e Saddik (2013)	dispositivo-plugin-navegador	HSHPlugin	Não se aplica
Liu et al. (2013)	dispositivo é servidor	AASMP	AASMP
Oat, Francesco e Aura (2014)	navegador-servidor-navegador	Customizado	Mocha
Maeda e Kobayashi (2014)	dispositivo-navegador	Dispositivo tipo-mouse	Não se aplica
Ghatak et al. (2014)	dispositivo-navegador	HTML5	Não se aplica
Rahman et al. (2014)	dispositivo-navegador	Dispositivo tipo-mouse	Não se aplica
Do, Cai e Jiang (2014)	dispositivo-servidor-navegador	Leap Motion SDK	Leap Motion SDK
Marrinan et al. (2014)	dispositivo-servidor-navegador	Omicron	SAGE2
	dispositivo-servidor-navegador	Omicron	SAGE2
	dispositivo-servidor-navegador	Omicron	SAGE2
Monserrat et al. (2015)	dispositivo-servidor-navegador	Não especificado	Customizado
	dispositivo-servidor-navegador	Não especificado	Customizado
	dispositivo-servidor-navegador	Não especificado	Customizado
Fittkau, Krause e Hasselbring (2015)	dispositivo-navegador	WebVR	Não se aplica
	Não especificado	Kinect SDK	Não especificado
Moreno et al. (2015)	dispositivo-servidor-navegador	OpenNI/NITE	Customizado
Rittitum, Vatanawood e Thongtak (2016)	dispositivo-servidor-navegador	Leap Motion SDK	Leap Motion SDK
Assal, Chiasson e Biddle (2016)	dispositivo-navegador	Dispositivo tipo-mouse	Não se aplica
Şerban e Culic (2016)	dispositivo é servidor	Wyliodrin STUDIO	Wyliodrin STUDIO
Konstantinidis, Bamparopoulos e Bamidis (2017)	dispositivo-servidor-navegador	Kinect SDK	CAC-Framework
	dispositivo-servidor-navegador	Não especificado	CAC-Framework
Pakkanen et al. (2017)	dispositivo-navegador	WebVR	Não se aplica
	Não especificado	Não especificado	Não especificado
	Não especificado	Não especificado	Não especificado
Sathe et al. (2017)	dispositivo-navegador	WebVR	Não se aplica
Han et al. (2017)	Não especificado	Não especificado	Não especificado
	Não especificado	Não especificado	Não especificado
Sanfilippo et al. (2017)	dispositivo-servidor-navegador	Não especificado	Customizado
DiPierro (2018)	dispositivo-navegador	HTML5	Não se aplica
Seo, Lee e Yoo (2017)	dispositivo-servidor-navegador	Não especificado	Customizado
Meixner e Kallmeier (2016)	dispositivo-navegador	HTML5	Não se aplica
Pallec et al. (2010)	dispositivo-servidor-navegador	MINY	MINY
	dispositivo-servidor-navegador	MINY	MINY
Wilson, Brown e Biddle (2014)	dispositivo-navegador	Dispositivo tipo-mouse	Não se aplica

Fonte – Produzida pelo autor.

4 Design de Solução para Aplicações Web de Visualização da Informação 3D Interativa

Com a RSL realizada, discutida no Capítulo 3 Trabalhos Relacionados, percebeu-se que, embora existam estudos que utilizem a multi-projeção na web e estudos que utilizem dispositivos de interação diversificados na web, não há um recurso (conceitual ou de software) particularmente popular para apoiar aplicações VI3DI multi-projetadas para navegadores web, com suporte a diferentes dispositivos de interação, ainda mais dando suporte a CGs descentralizados e mestre-escravo, abordagens que, juntas, formam a opção mais popular para distribuição dos dados por um CG.

Este capítulo, como uma resposta à situação percebida durante a RSL, propõe, no nível mais alto de abstração, um design de solução para aplicações VI3DI executadas em navegadores web, separando e explicando cada uma de suas funções necessárias.

Escolheu-se chamar a proposta apresentada neste capítulo de "design", pois, na visão do autor, seu nível de abstração não se enquadra adequadamente em definições popularmente utilizadas na Ciência da Computação para se referir a objetos de objetivo similar. Ela não é uma arquitetura de software, pois não define as estruturas envolvidas, somente os recursos que devem ser proporcionados por tais estruturas. Embora os recursos propostos possam ser estendidos pelos desenvolvedores, ela não um *framework* de trabalho, pois não faz propostas quanto ao modelo de realização das atividades de desenvolvimento, nem de software, pois não é software, razão pela qual ela também não é uma biblioteca. Assim, o termo "design", originário da língua inglesa e agora parte da portuguesa, mais se aproxima da intenção da proposta; o dicionário Oxford (em inglês) apresenta a seguinte definição[1], entre outras, para *design*: "**Um plano ou desenho produzido para mostrar a aparência e a função ou o funcionamento** de um edifício, peça de vestuário ou outro objeto antes que ele seja feito."

Adianta-se que o suporte à imersão ocorre por sistemas de multi-projeção e CGs. Como apresentado na Subseção 2.4.3 Considerações, uma das decisões mais importantes no que diz respeito ao uso de CGs é justamente a arquitetura de renderização (cliente-servidor, mestre-escravo ou, como revelado pela RSL, descentralizada); além disso, os requisitos específicos de aplicações executadas em navegadores web requerem que a troca de dados entre os nós de um CG sejam pequenas e rápidas, assim, foi escolhido usar a arquitetura mestre-escravo. O uso dela torna a adaptação de aplicações já existentes mais trabalhosa, pois, como visto, ela é menos transparente. A fim de mitigar essa questão, o design de solução proposto inclui recursos para observação e sincronização do estado da aplicação entre os nós, tratados em melhores detalhes nas Subseções 4.2.2 Sincronização de Estado e 5.2.2 Sincronização de Estado.

Este capítulo inicia apresentando a estrutura do design de solução proposto e algumas considerações sobre ela. Depois, são apresentados os seus elementos principais, abordando como

[1] Definição de *design* no dicionário Oxford: <https://en.oxforddictionaries.com/definition/design>.

se espera que operem.

O propósito deste capítulo inclui apresentar e discutir os desafios e requisitos envolvidos na implementação do design de solução proposto, a qual é abordada em mais detalhes no Capítulo 5 *Framework* para Aplicações Web de Visualização da Informação 3D Interativa.

4.1 Estrutura

O design de solução proposto para o desenvolvimento de aplicações VI3DI web distribui suas funcionalidades em dois lados principais, o cliente e o servidor.

Do lado cliente, há os componentes executados em navegadores web, que atuarão como os nós de CGs. Implementações do design proposto sempre usarão JavaScript para estes componentes, devido ao monopólio desta linguagem de programação na plataforma. Esse lado também apresenta a lógica própria da aplicação web desenvolvida com o design proposto, que pode conter código adicional em HTML e CSS.

Do lado servidor, estão os componentes que não são executados em navegadores web, e que, portanto, podem ser implementados usando qualquer tecnologia que permita, de algum modo, servir requisições HTTP através da internet.

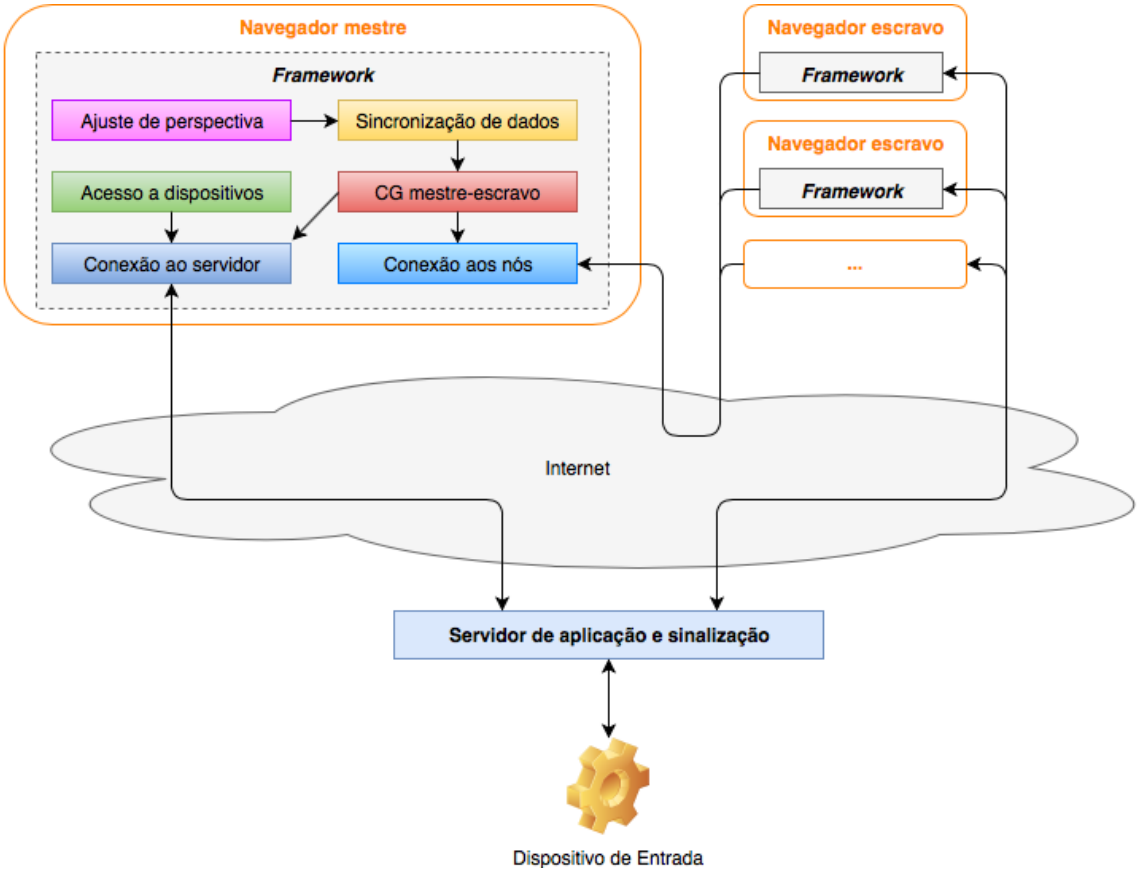
A Figura 13 apresenta as funcionalidades elencadas para o design de solução proposto e os relacionamentos entre elas. Ressalta-se as cores utilizadas para representar cada uma das funcionalidades; elas não possuem significado intrínseco, mas são usadas por esta dissertação para identificar cada uma das funcionalidades.

Sistemas de multi-projeção são apoiados, principalmente, por meio de CGs na arquitetura mestre-escravo, visando menor latência na comunicação entre os nós, devido ao menor impacto que esta arquitetura tem no tráfego da rede. O servidor de aplicação armazena uma referência para todos os CGs executando a aplicação no momento, com a relação de quais nós fazem parte de cada CG e quais os papéis desses nós; diferentes CGs são segregados e identificados por meio de "salas" virtuais, grupos lógicos de nós identificados por um nome qualquer.

O design proposto estabelece que a maneira como os nós trocam dados entre si seja abstraída, no entanto, como ele é voltado a aplicações web executadas em navegadores web, julga-se que a melhor opção para tanto, em termos de tecnologia, seja o uso das conexões estabelecidas pelo WebRTC, uma API JavaScript para comunicação em tempo real entre duas páginas web, por meio de conexões P2P (Seção B.1 WebRTC). Devido a isso, o design proposto também abrange o servidor de aplicação e sinalização[2], necessário para estabelecimento dessas conexões pelo WebRTC. Ainda mais, navegadores web, via de regra, não oferecem meios para acessar dados de dispositivos de interação, exceto mouse e teclado, assim, no design proposto, dispositivos de interação são apoiados por meio da abordagem *dispositivo-servidor-navegador*, em que os dispositivos são conectados ao servidor, que se encarrega de repassar os dados deles para os nós.

[2] "Sinalização", no caso, diz respeito ao processo pelo qual dois nós trocam dados até que estabeleçam conexão entre si, o qual necessita de um servidor de aplicação.

Figura 13 – Funcionalidades propostas para uma aplicação VI3DI web. Setas indicam que a funcionalidade de origem depende da de destino.



Fonte - Produzida pelo autor.

É importante observar que, como visto na Seção 2.4 Cluster Gráficos, o design de solução proposto faz uso simultâneo das arquiteturas cliente-servidor e mestre-escravo em sua operação. A arquitetura cliente-servidor é usada para carregar a aplicação web no navegador web. Os nós (navegadores web) atuam como clientes, realizando requisições ao servidor de aplicação para consultar os arquivos que devem carregar e quando precisam de sinalização para estabelecer conexão WebRTC entre si. Quando os nós executam a aplicação HTML/JavaScript desenvolvida com uma implementação do design de solução proposto, a arquitetura mestre-escravo é usada, uma vez que um dos nós atua como mestre e sincroniza seu estado diretamente com os escravos. Neste ponto, é como se o servidor que hospeda a aplicação não existisse mais (ele ainda pode ser usado, mas de forma transparente).

4.2 Composição

Como mostrado na Figura 13, o design de solução para aplicações VI3DI web proposto apoia as seguintes questões:

- Ajuste de perspectiva: ajuste da câmera virtual para aplicações de múltiplas telas;
- Sincronização de dados: sincronização de estado da aplicação do nó mestre para os escravos;

- CG mestre-escravo: estabelecimento e coordenação de um CG seguindo a arquitetura mestre-escravo e onde os nós são instâncias da aplicação carregadas por navegadores web;
- Acesso a dispositivos de interação: conexão e recebimento de dados de dispositivos de interação conectados ao servidor da aplicação;
- Conexão com nós: ligação e troca de dados entre os nós individuais envolvidos no CG;
- Conexão com servidor: ligação e troca de dados entre um nó e o servidor da aplicação; e
- Servidor de aplicação e sinalização: servidor da aplicação, que também é responsável por sinalizar conexões entre nós e operar os dispositivos de interação.

De maneira geral, pode-se considerar que, em comparação com o desenvolvimento de aplicações 3D web executadas em um único nó, os novos pontos críticos do desenvolvimento de aplicações VI3DI web com o design proposto envolvem:

- Controle de conexão dos nós em forma de CG;
- Determinação de quais valores de estado serão sincronizados no CG; e
- Correção de perspectiva do conteúdo exibido pelo nó;

A seguir, cada uma das funcionalidades apoiadas pelo design de solução proposto é apresentada e discutida.

4.2.1 Cluster Gráfico Mestre-Escravo

Esta é uma das funcionalidades mais importantes abordadas pelo design proposto. Ela é responsável por organizar e gerenciar um conjunto de nós executando a aplicação na forma de um CG, no lado cliente das aplicações.

Como o design proposto apoia a arquitetura mestre-escravo, é necessário que cada nó seja capaz de saber sua função no CG (mestre ou escravo) e o que lhe é permitido fazer.

A finalidade dessa funcionalidade é encapsular as ações do nó que envolvem o CG:

- União a sala virtual do CG no servidor;
- Manusear novo nó conectado ao CG;
- Manusear de mensagens entre o nó atual e outro ao sinalizar sua conexão;
- Enviar dados para outros nós do CG;
- Receber dados do nó mestre; e
- Manusear perda de conexão (com o CG ou com a internet).

Quando o nó tenta se conectar ao CG, é necessário que ele especifique qual o seu papel desejado, para que se possa definir se a união é permitida. Três papéis são propostos:

- Mestre: o nó assume controle do CG como nó mestre. Caso já exista outro nó com esse papel no CG, o acesso ao CG pode ser negado ao nó atual ou o nó mestre atual se torna um escravo e o nó atual o substitui como mestre;
- Escravo: o nó entra como escravo e solicita conexão ao nó mestre, para receber e sincronizar seus dados. Caso o nó mestre não faça parte da sala virtual, o acesso ao CG pode ser negado ao nó atual ou ele pode ficar na sala virtual, aguardando o mestre; e
- Qualquer: o papel real do nó é definido no momento em que ele se une a sala virtual do CG. Se existir um nó mestre no CG, então o nó atual se torna escravo, do contrário, se torna o mestre.

Propõe-se que o nó mestre mantenha uma conexão para cada nó escravo, enquanto os nós escravos mantém somente uma conexão, para o nó mestre. Ainda mais, propõe-se que os nós escravos que iniciem o processo de conexão ao nó mestre; dessa maneira, é usada uma abordagem baseada em eventos: o nó mestre não sai ativamente a procura dos escravos, são os escravos que solicitam ao mestre para se unir ao CG, ao se juntarem à sala virtual.

Como visto na Seção 3.1 Multi-Projeção em Navegadores Web, CGs descentralizados são opções comuns para a multi-projeção na web. Embora o design proposto projete o CG com a arquitetura mestre-escravo, ele pode ser facilmente adaptado para a arquitetura descentralizada, permitindo que os nós escravos enviem dados ao nó mestre, que os repassa para os outros escravos. Foi escolhida essa abordagem em forma de "estrela" para CGs descentralizados, devido a já comentada expectativa de que implementações do design de solução proposto usarão conexões P2P entre os nós do CG, assim, dado um CG com n nós, a arquitetura descentralizada requer $\sum_{i=1}^{n-1} i$ conexões entre os nós, enquanto a mestre-escravo, somente $n-1$ conexões; mais ainda, ao permitir que os nós escravos enviem dados ao mestre, é possível dar funcionamento similar ao descentralizado a um CG mestre-escravo, com relativa baixa perda de desempenho (uma vez que o nó mestre passa a atuar como intermediário entre os nós escravos).

4.2.2 Sincronização de Estado

O design de solução proposto visa a utilização de sistemas de multi-projeção por meio de CGs mestre-escravo, pois, como visto na Seção 2.4 Cluster Gráficos, ele é mais adequado ao paradigma dos navegadores web, apresenta relativo baixo uso da rede e permite que o mesmo código-fonte de aplicação seja executado em todos os nós.

O estado de uma aplicação pode ser entendido como os valores armazenados por ela em memória, assim, a abordagem mestre-escravo necessita de trabalho por parte do desenvolvedor para que esses valores (i.e. o estado da aplicação) sejam sincronizados do nó mestre com os nós escravos.

Existem várias maneiras e técnicas pelas quais valores e objetos podem ser sincronizados. O design proposto é focado na linguagem de programação JavaScript, onde a estrutura de objetos

é definida por *prototypes*, que podem ser alterados em tempo de execução, assim, implementações do design podem fazer uso dessa característica, permitindo que o programador meramente especifique quais atributos dos objetos deseja sincronizar; com isso, a implementação altera o *prototype* do objeto, substituindo o atributo especificado por uma propriedade computada que sincroniza qualquer alteração do valor do atributo com o CG.

É importante, também, notar que, embora o JavaScript permita o acesso e alteração de qualquer atributo em um objeto, isso não significa que todos os atributos devem ser acessados pela aplicação; alguns deles podem existir com a intenção de serem *internos*. Nesses casos, os valores dos atributos internos podem ser definidos por meio de métodos *públicos* do objeto. Sendo assim, implementações do design proposto também devem oferecer maneiras para sincronizar chamadas aos métodos de um objeto, de maneira que, quando o método for invocado no nó mestre, ele também o seja nos escravos.

O envio e recebimento dos dados de sincronização de atributos e métodos dos objetos do nó mestre para os nós escravos deve se dar por meio da comunicação entre os nós possibilitada pelo CG mestre-escravo.

4.2.3 Ajuste de Perspectiva

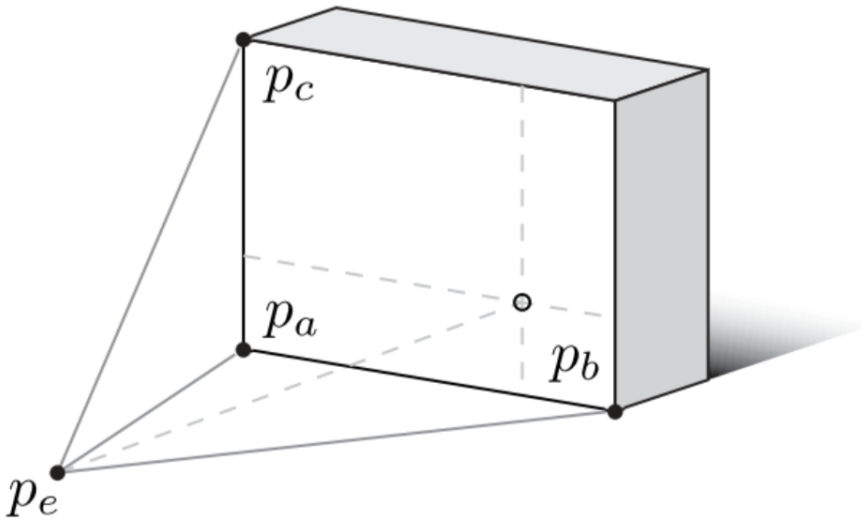
Em aplicações 3D, há objetos do tipo *câmera virtual* que são responsáveis por capturar um trecho do ambiente virtual e o repassar para a saída de vídeo da aplicação. Câmeras virtuais não são responsáveis pela renderização do conteúdo, em si, mas, sim, por sua seleção no ambiente.

A princípio, cada nó do CG a executar uma aplicação desenvolvida com base no design proposto estará vinculado a um dispositivo de exibição diferente, com posição e orientação próprias no mundo real, sendo assim, embora os nós processem essencialmente um mesmo ambiente virtual, cada um deles será responsável por exibir um trecho específico desse ambiente.

É necessário que a câmera virtual de cada nó leve em consideração as posição e orientação físicas do dispositivo de exibição ao qual o nó está ligado, para que saiba como oferecer a projeção perspectiva correta da visão para aquela saída de vídeo. Essa projeção perspectiva é computada utilizando uma *matriz de projeção*, que transforma os pontos capturados pela câmera no ambiente virtual do espaço global para o espaço da câmera. Isso dificulta a implementação de um controle único e geral para a câmera virtual da aplicação, pois cada nó terá seus próprios parâmetros para a projeção perspectiva. Adicionalmente, com o uso de visão estereoscópica, é necessário, também, considerar a posição da cabeça do usuário, para que seja aplicado o efeito adequado de paralaxe (KOOIMA, 2008).

Para solucionar essa questão, Kooima (2008) apresenta a *matriz de projeção perspectiva generalizada*, que usa três pontos no espaço tridimensional, p_a, p_b e p_c, para definir o tamanho, proporção, posição e orientação (em relação a exata frente do usuário, no ponto $(0, 0, 0)$) da saída de vídeo que exibirá a imagem capturada pela câmera virtual. Há, também, o ponto p_e, que representa a visão do usuário e é utilizado para correção de perspectiva em imagens estereoscópicas. A Figura 14 apresenta um retângulo (representando uma tela) e a relação entre os pontos p_a, p_b, p_c e p_e.

Figura 14 – Relação entre os pontos utilizados no cálculo da matriz de projeção perspectiva generalizada.



Fonte - Neto et al. (2015a).

Utilizando essa técnica, é possível que as câmeras virtuais de todos os nós de um CG tenham as mesmas posição e orientação no ambiente virtual da aplicação, enquanto apresentam imagens diferentes, relativas a maneira como suas matrizes de projeção foram configuradas.

As posição e orientação da câmera virtual do nó mestre são sincronizadas com os nós escravos por meio da própria sincronização de estado inclusa no design proposto.

4.2.4 Conexão com os Nós

Espera-se que a conexão entre os nós de CGs de aplicações desenvolvidas com o design proposto ocorra, principalmente, através do WebRTC. O processo de conexão entre dois nós utilizando WebRTC envolve vários passos; dessa maneira, implementações do design proposto devem abstrair a conexão entre nós através dos seguintes pontos:

- Iniciação de processo de conexão de um nó com outro;
- Criação e envio (através do do servidor de sinalização) de oferta de conexão entre dois nós;
- Manuseamento de oferta de conexão recebida do servidor de sinalização;
- Criação e envio (através do do servidor de sinalização) de resposta a oferta de conexão;
- Manuseamento de reposta a oferta de conexão recebida do servidor de sinalização;
- Manuseamento de erro na criação de resposta a oferta de conexão recebido do servidor de sinalização;
- Criação e envio (através do do servidor de sinalização) de dados de candidato de conexão;
- Manuseamento de dados de candidato de conexão recebidos do servidor de sinalização;

- Envio de dados de um nó para outro; e
- Encerramento da conexão entre dois nós.

Existem navegadores web nos quais o WebRTC ainda não está disponível, o que implica no suporte a outros modelos de conexão entre os nós. O conjunto apresentado de funcionalidades requeridas é abrangente o suficiente para que esses outros modelos possam ser suportados.

Em todo caso, nota-se que a existência de um servidor de sinalização (a conexão com o qual é abordada na Subseção 4.2.6 Conexão ao Servidor) é imprescindível para o estabelecimento da conexão entre os nós. O envio dos dados, em si, pode ou não depender do servidor, de acordo com a implementação. Dessa maneira, a conexão entre os nós pode ser tanto uma conexão P2P, propriamente dita, quanto uma conexão meramente lógica, onde os dados são roteados por um intermediário, tal como o servidor da aplicação e sinalização.

4.2.5 Acesso a Dispositivos de Interação

Em uma aplicação VI, a capacidade de interagir com os dados e a visualização é muito importante. Modelos de interação mais avançados podem usar dispositivos de entrada consideravelmente diferentes de mouse e teclado. Esses dispositivos, contudo, geralmente têm suas próprias SDKs, exigindo algum tipo de interface que permita que eles sejam usados com o mínimo de acoplamento com o restante da aplicação.

Devido às limitações dos navegadores web para acessar periféricos conectados ao computador, a conexão com os dispositivos de interação deve ser feita através do servidor de aplicação (Subseção 4.2.7 Servidor de Aplicação e Sinalização), para o qual são definidos eventos que permitem aos nós especificar de quais dispositivos eles precisam de dados e quais eventos desses dispositivos fornecem os dados. Mais especificamente, implementações dessa funcionalidade precisam fornecer acesso aos dados de dispositivos através dos seguintes pontos:

- Conexão a dispositivo de interação;
- Cadastro em evento do dispositivo de interação;
- Manuseamento de dados de evento do dispositivo de interação recebidos do servidor de aplicação;
- Descadastro de evento do dispositivo de interação; e
- Encerramento da conexão ao dispositivo de interação.

É importante notar, todavia, que o responsável por realizar, de fato, esses pontos não são os nós do CG, mas, sim, o servidor de aplicação, que tem acesso ao dispositivo. O propósito dessa funcionalidade é justamente fornecer aos nós uma interface que lhes permita acessar esses dispositivos de maneira transparente.

4.2.6 Conexão ao Servidor

Numa aplicação web, é natural a existência de um servidor de aplicação com a finalidade de servir a aplicação aos clientes. No design proposto, além do básico, o servidor de aplicação também possui, pelo menos, duas outras responsabilidades adicionais: sinalizar conexão entre dois nós e operar os dispositivos de interação, sob comando dos nós.

É necessário que o design inclua a maneira como os nós e o servidor de aplicação se comunicam, de modo que eles possam trocar dados de maneira bilateral em tempo real e que o nó não seja recarregado toda vez que isso ocorra (como acontece durante navegação comum).

Através da conexão nó-servidor, os nós enviam dados ao servidor, tais como CG ao qual desejam se unir, ofertas/respostas de conexão destinadas a outros nós e dispositivos de interação dos quais desejam dados; o servidor, por sua vez, responde a solicitações de participações em CGs, repassa ofertas/respostas de conexão aos nós de destino e também repassa dados obtidos de dispositivos de interação.

Existem, essencialmente, duas maneiras com as quais essa conexão pode ser implementada:

- WebSockets: a API de WebSockets resolve o problema de enviar dados diretamente para o cliente, permitindo que os nós mantenham uma conexão *socket* assíncrona com o servidor. Ao manter essa conexão, o nó pode enviar dados instantaneamente para o servidor sem precisar restabelecer uma conexão. Além disso, o servidor é capaz de enviar dados para os nós a qualquer momento enquanto a conexão permanecer aberta (WESSELS et al., 2011); e

- Ajax: conjunto de técnicas que permite aos nós enviar e recuperar dados do servidor de maneira assíncrona, sem interferir na exibição e no comportamento da página existente. Ao separar a troca de dados da apresentação deles, Ajax permite que os nós alterem seus dados dinamicamente, sem a necessidade de recarregar a página web inteira (GARRETT, 2005).

Do ponto de vista dos nós, implementações do design proposto devem abstrair a conexão nó-servidor através das seguintes funcionalidades:

- Estabelecimento da conexão com o servidor;

- Envio de dados para o servidor (identificados como *eventos de entrada*);

- Manuseamento de dados recebidos do servidor (identificados como *eventos de saída*); e

- Encerramento da conexão com o servidor.

Quanto ao ponto de vista do servidor, ele é dependente da implementação do servidor de aplicação e sinalização usado junto ao design proposto.

4.2.7 Servidor de Aplicação e Sinalização

O servidor de aplicação e sinalização não faz, completamente, parte do design proposto, contudo, como visto nas Subseções 4.2.4 Conexão com os Nós, 4.2.5 Acesso a Dispositivos de Interação e 4.2.6 Conexão ao Servidor, há a necessidade de certas interações com o servidor de aplicação (e.g., sinalizar conexão entre dois nós, operar dispositivo de interação sob ordens de um nó), que, por sua vez, estabelecem certos comportamentos necessários para o servidor de aplicação e sinalização.

Diferentemente das aplicações web executadas em navegadores web, que são, necessariamente, programadas com JavaScript, servidores de aplicação podem ser implementados usando uma infinidade de linguagens e tecnologias, assim, embora o design proposto não inclua o servidor de aplicação e sinalização, são estabelecidos *eventos* para ele, os quais implicam em ações específicas, seja no servidor ou nos nós. *Eventos de entrada* são aqueles disparados pelos nós no servidor, requisitando que o servidor realize algum processamento. Por outro lado, *eventos de saída* são aqueles disparados pelo servidor nos nós (individualmente ou em um grupo deles), requisitando que os nós realizem algum processamento. O processamento realizado pelo servidor devido a um evento de entrada muitas vezes implica na emissão de um evento de saída para o nó emissor ou um grupo de nós relacionado a ele; o inverso (evento de saída incorrer num de entrada), contudo, raramente é verdade.

A maneira, em si, como esses eventos são recebidos e emitidos depende tanto da implementação do servidor de aplicação e sinalização quanto da maneira pela qual os nós se conectam ao servidor em implementações do design proposto, que se comporta como a interface entre a aplicação VI3DI e os eventos de entrada e saída no servidor de aplicação e sinalização.

A seguir, são listados os eventos de entrada para cumprir o mínimo das funcionalidades do design proposto, no que diz respeito ao lado cliente da aplicação:

- *unir_cluster*: Nó deseja se unir ao um CG
  - Dispara: *uniu_cluster*.
  - Argumentos: identificador do CG, papel do nó.
- *sair_cluster*: Nó está saindo do cluster.
  - Dispara: *nodo_saiu*, nos outros nós do CG.
  - Argumentos: nenhum.
- *oferta*: Nó está ofertando conexão com outro.
  - Dispara: *oferta*, no outro nó.
  - Argumentos: identificador do outro nó, dados da oferta de conexão.
- *resposta*: Nó está respondendo a oferta de conexão.
  - Dispara: *resposta*, no nó que ofertou conexão.

  - Argumentos: identificador do nó que ofertou conexão, dados de resposta a oferta de conexão.

- *candidato*: Nó está enviando dados de candidato de conexão a outro.
  - Dispara: *candidato*, no outro nó
  - Argumentos: identificador do outro nó, dados de candidato.

- *conectar_dispositivo*: Nó requer conexão com dispositivo de interação.
  - Dispara: *conectado_dispositivo*.
  - Argumentos: identificador do dispositivo.

- *desconectar_dispositivo*: Nó requer encerramento de conexão com dispositivo de interação.
  - Dispara: nenhum.
  - Argumentos: identificador do dispositivo.

- *cadastrar_evento_dispositivo*: Nó requer recebimento de dados de um evento do dispositivo.
  - Dispara: *cadastrado_evento_dispositivo*.
  - Argumentos: identificador do dispositivo, nome do evento.

- *descadastrar_evento_dispositivo*: Nó requer parada no recebimento de dados de um evento do dispositivo.
  - Dispara: nenhum.
  - Argumentos: identificador do dispositivo, nome do evento.

Em contra-partida, esses eventos de entrada também acarretam no disparo dos seguintes eventos de saída:

- *uniu_cluster*: Nó foi unido a um CG.
  - Disparado por: *unir_cluster*.
  - Argumentos: dados do nó mestre.

- *oferta*: Repasse de oferta de conexão para o nó destinatário.
  - Disparado por: *oferta*, pelo nó ofertante.
  - Argumentos: identificador do nó ofertante, dados da oferta de conexão.

- *resposta*: Repasse de resposta a oferta de conexão para o nó ofertante.
  - Disparado por: *resposta*, pelo nó destinatário.
  - Argumentos: identificador do nó destinatário, dados de resposta a oferta de conexão.

- *candidato*: Repasse de dados de candidato de conexão de um nó para outro

  - Disparado por: *candidato*, pelo outro nó.
  - Argumentos: nome do dispositivo.

- *cadastrado_evento_dispositivo*: Servidor se cadastrou para receber dados de evento do dispositivo.
  - Disparado por: *cadastrar_evento_dispositivo*.
  - Argumentos: identificador do dispositivo, nome do evento.

- *disparou_evento_dispositivo*: Repasse de dados de evento do dispositivo.
  - Disparado por: disparo de evento do dispositivo no qual o servidor estava cadastrado.
  - Argumentos: identificador do dispositivo, nome do evento, dados do evento.

Observa-se que não há problema em eventos de entrada e saída com os mesmos nomes, pois eles representam coisas diferentes (um é disparado pelos nós no servidor, o outro é separado pelo servidor nos nós); todavia, a repetição do nome indica o relacionamento de causa e ação entre eles. Também se observa que o evento de saída *disparou_evento_dispositivo* é o único não disparado como resposta a um evento de entrada; isso acontece, pois, uma vez que o servidor tenha se cadastrado para um evento do dispositivo de interação, o recebimento de dados do dispositivo é independente de quaisquer eventos de entrada.

É importante notar, no entanto, que os eventos de entrada e saída listados são, assim como o restante deste capítulo, meras propostas que descrevem os mínimos recursos necessários para que implementações do design proposto funcionem como o projetado.

Implementações do design proposto (tal como a realizada durante para esta dissertação) podem redefinir os nomes dos eventos e as maneiras como alguns deles funcionam ou são chamados, bem como definir mais eventos (proporcionando, assim, mais funcionalidades e recursos).

4.3 Considerações Finais

O design de solução apresentado é formado por várias funcionalidades que visam, ao mesmo tempo, agilizar o desenvolvimento e abstrair certas dificuldades inerentes de aplicações VI3DI e, também, que fazem uso de CGs, em especial aquelas desenvolvidas para a web, para que sejam tanto de fácil utilização quanto de fácil manutenção/modificação/extensão.

Para o design proposto, os tipos de dispositivos que podem atuar como nós são limitados meramente pela necessidade de existência de navegadores web para eles, enquanto sua disposição física é completamente transparente. Dessa maneira, o CG formado por esses dispositivos não precisa ser, de fato, um CG, no sentido mais formal. Uma coleção heterogênea de PCs, tablets e smartphones pode muito bem executar em conjunto uma aplicação VI3DI desenvolvida com o design proposto

5 *Framework* para Aplicações Web de Visualização da Informação 3D Interativa

Este capítulo trata da implementação do design de solução para aplicações VI3DI web apresentado, na forma de um *framework* JavaScript. Inicia-se apresentando o *framework* desenvolvido. Em seguida, é discorrido sobre o funcionamento e uso de suas interfaces públicas para os lados cliente e servidor de uma aplicação. O capítulo, então, é encerrado com algumas considerações.

É importante observar que a implementação faz uso de várias tecnologias já disponíveis. Essas tecnologias são abordadas em maiores detalhes no Apêndice B Fundamentação Tecnológica, e, resumidamente, são:

- WebRTC: API JavaScript que permite o estabelecimento de conexões P2P entre duas páginas web;
- Three.js: Biblioteca JavaScript que abstrai o uso do WebGL, uma API JavaScript para gráficos 3D acelerados por hardware em navegadores web;
- Node.js: Ambiente de *runtime* JavaScript, que permite o desenvolvimento de servidores de aplicação usando JavaScript; e
- VRPN: Biblioteca C/C++ que abstrai o funcionamento e operação de dispositivos de entrada.

5.1 Laboratory of Interfaces and Visualization's Web Cluster Library

O recurso reusável de software resultante desta dissertação consiste na *Laboratory of Interfaces and Visualization's Web Cluster Library* (Livvclib; do inglês, "Biblioteca de Aglomerados Web do Laboratório de Interfaces e Visualização").

Livvclib[1] é um framework/biblioteca JavaScript de código-aberto que reúne as tecnologias apresentadas no Apêndice B Fundamentação Tecnológica a fim de implementar o design de solução para aplicações VI3DI web, apresentado no Capítulo 4 Design de Solução para Aplicações Web de Visualização da Informação 3D Interativa, i.e. um *framework* para o desenvolvimento de aplicações VI3DI com suporte a visualização distribuída por multi-projeção, com CGs mestre-escravo, e dispositivos de interação diversificados.

Com a Livvclib, os gráficos 3D de cada nó são renderizados pela biblioteca Three.js, através de uma extensão de sua câmera virtual perspectiva. A comunicação entre os nós se dá, principalmente, por meio de conexões P2P do WebRTC, utilizando sua interface de transmissão de dados. Até que esta conexão P2P seja estabelecida, no entanto, mensagens entre os nós

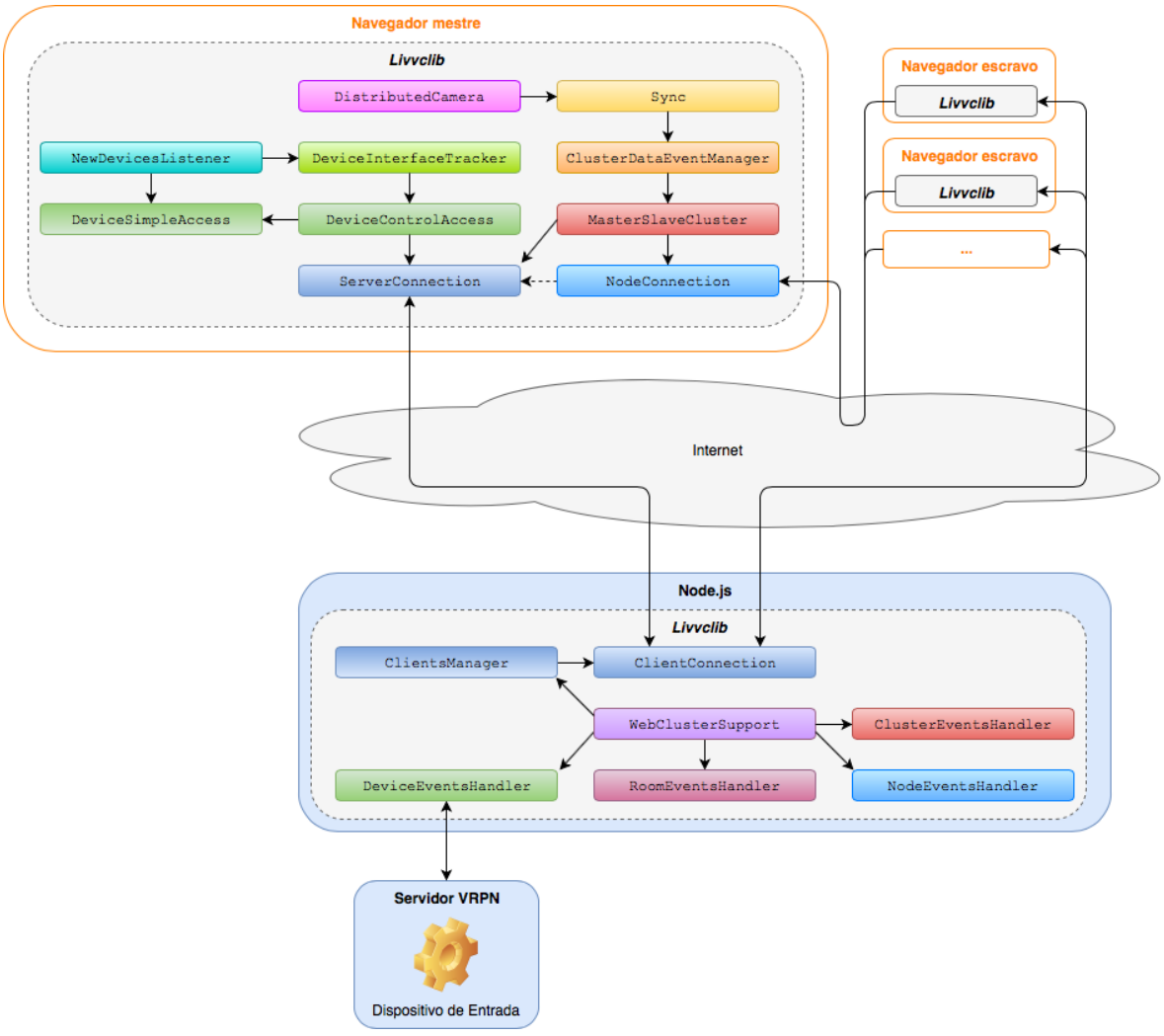
[1] Código-fonte da Livvclib: <https://bitbucket.org/livunespbauru/livvclib>.

passam por um servidor de aplicação e sinalização implementado com o Node.js. A utilização de dispositivos de interação é proporcionada por meio das abstrações de tipos de dados fornecidas pelo VRPN.

A versão 0.7.0 da Livvclib (a mais recente até o momento da escrita deste documento, em 16 de Abril de 2018) é composta por 36 classes, 5 funções e 14 enumeradores que visam auxiliar o desenvolvimento de aplicações VI3DI web tanto do lado cliente, quanto do lado servidor (supondo que o servidor seja desenvolvido com Node.js).

A Figura 13 apresentou as principais funcionalidades requisitadas pelo design de solução apresentado. Levando em conta os componentes públicos da Livvclib, é possível redesenhá-la como mostra a Figura 15, onde cada funcionalidade, agora, corresponde a uma interface fornecida pela Livvclib e usada pelo desenvolvedor para implementar VI3DI; além disso, as cores dos elementos nas duas figuras representam correspondência direta entre as funcionalidades propostas e as interfaces implementadas (cores novas indicam funcionalidades adicionais apresentadas pela Livvclib).

Figura 15 – Componentes públicos da Livvclib. Setas indicam dependências.



Fonte - Produzida pelo autor.

Comparando as Figuras 13 e 15, observa-se a existência de muito mais interfaces na

Livvclib do que funcionalidades propostas. A razão para isso é que as implementações de algumas das funcionalidades requerem ou se beneficiam de interfaces adicionais, que abstraiam certos comportamentos para o desenvolvedor.

As interfaces *públicas* da Livvclib, em ambos lados cliente e servidor, são todas implementadas em JavaScript, embora existam componentes *internos* desenvolvidos em C++. A Livvclib é formada pelas seguintes interfaces públicas principais:

- Do lado cliente:
  - `MasterSlaveCluster`: abstração do CG mestre-escravo;
  - `ClusterDataEventManager`: gerenciador de eventos intra-cluster;
  - `DistributedCamera`: câmera virtual para aplicações de múltiplas telas;
  - `Sync`: objeto utilitário para sincronização de dados;
  - `NodeConnection`: interface da conexão entre dois nós;
  - `DeviceControlAccess`: manipulação e gestão da conexão com um dispositivo de interação;
  - `DeviceSimpleAccess`: simulação de conexão com dispositivo de interação;
  - `NewDevicesListener`: recebe notificações de quando outros nós solicitaram conexão com dispositivo de interação;
  - `DeviceInterfaceTracker`: rastreia eventos de uma interface de um dispositivo de interação; e
  - `ServerConnection`: interface da conexão entre um nó e o servidor de aplicação e sinalização.
- Do lado servidor:
  - `ClientConnection`: interface da conexão entre o servidor de aplicação e sinalização e um nó.
  - `ClientManager`: gerenciador de nós em CGs;
  - `WebClusterSupport`: gerenciador dos tratamentos de eventos de entrada/saída; e
  - `ClientEventsHandler`: interface para tratamento de conjunto de eventos de entrada/saída (na Figura 15, os elementos `...EventsHandler` são implementações dela).

O restante deste capítulo traz um aprofundamento sobre cada uma dessas interfaces. Dada a natureza aberta do código-fonte da Livvclib, bem como a inclusão de exemplos em tal código, as abordagens se concentram no funcionamento, nas responsabilidades e nos relacionamentos de cada uma das interfaces, que julga-se mais propícios para esta dissertação, evitando maiores detalhes sobre quais os métodos, propriedades e atributos apresentados por elas, os quais podem ser conferidos no próprio código-fonte.

5.2 Lado Cliente

A Livvclib é projetada tendo em foco aplicações executadas em navegadores web (os nós), portanto a maior parte de seus componentes e complexidade se concentram em apoiar o desenvolvimento do lado cliente das aplicações web.

As funcionalidades do lado cliente apoiadas pela Livvclib são aquelas apresentadas para o design de solução apresentado (Subseção 4.2 Composição):

- Estabelecimento e gestão de cluster gráfico com a arquitetura mestre-escravo;
- Ajuste da perspectiva de cada nó;
- Sincronização do estado da aplicação entre os nós;
- Conexão entre os nós;
- Acesso a dispositivos de interação; e
- Conexão entre os nós e o servidor.

5.2.1 Cluster Gráfico Mestre-Escravo

Como visto, o design de solução apresentado apoia sistemas de multi-projeção através de CGs com a arquitetura mestre-escravo, o que requer uma interface entre a aplicação e o CG, a fim de facilitar sua utilização.

No lado cliente, a Livvclib suporta CGs mestre-escravo através, principalmente, da classe `MasterSlaveCluster`. Ela é responsável por abstrair (salvo alguns detalhes) todas as funções propostas para um CG mestre-escravo, agindo como uma interface entre o nó e o (restante do) CG:

- No nó mestre:
  - Manuseamento de oferta de conexão recebida de nó escravo;
  - Envio de resposta a oferta de conexão com nó escravo;
  - Envio de dados de candidato de conexão com nó escravo;
  - Manuseamento de dados de candidato de conexão com nó escravo;
  - Envio de dados de aos nós escravos;
  - Recebimento e repasse de dados dos nós escravos; e
  - Encerramento da conexão (e, por consequência, do CG).
- Nos nós escravos:
  - Iniciação de processo de conexão com o nó mestre;
  - Envio de oferta de conexão entre para o nó mestre;
  - Manuseamento de reposta a oferta de conexão recebida;

- Envio de dados de candidato de conexão com nó mestre;
- Manuseamento de dados de candidato de conexão com nó mestre;
- Recebimento de dados do nó mestre;
- Envio de dados para o nó mestre; e
- Encerramento da conexão com o nó mestre (saída do CG).

Usando `MasterSlaveCluster`, os nós podem se unir a um CG das seguintes maneiras (definidas no enumerador `ClusterJoiningMode`):

- `MASTER`: o nó assume controle do CG como mestre. Caso já exista um nó mestre, a solicitação de união falha;
- `SLAVE_NOW`: o nó assume o papel de escravo no CG e requer que o nó mestre já tenha se unido. Caso o nó mestre ainda não tenha se unido ao CG, a solicitação de união falha;
- `SLAVE_WAIT`: o nó assume o papel de escravo no CG. A solicitação de união tem sucesso mesmo que o nó mestre ainda não tenha se unido ao CG. Quando o nó mestre se unir ao CG, os nós escravos com este papel são notificados por um evento de saída do servidor; e
- `ANY`: o nó assume o papel de mestre, se não há nó mestre atualmente unido ao CG. Do contrário, o nó assume papel de escravo.

O funcionamento dessa classe assume que o servidor de aplicação e sinalização armazena as informações sobre os CGs executando a aplicação e que cada CG possui um nome usado na criação de uma sala virtual que segrega seus nós de outros CGs.

O processo de união do nó atual ao CG começa com a instância de `MasterSlaveCluster` usando uma instância da classe abstrata `ServerConnection` (Subseção 5.2.6 Conexão com o Servidor) para tentar se unir ao CG de uma sala virtual especificada pelo desenvolvedor. Caso o CG da sala virtual especificada cumpra os requisitos da maneira escolhida para o nó se unir ao CG, o nó é notificado do sucesso na união ao CG e do seu papel com ele.

Nesse ponto, se o nó assume o papel de mestre, então as instâncias de `MasterSlaveCluster` nos nós escravos que aguardavam o mestre são notificadas sobre a união do mestre e imediatamente começam a sinalizar o estabelecimento de conexão com o nó mestre. Quando nó assume o papel de escravo, se o nó mestre já fizer parte do CG, então o escravo inicia a sinalização para se conectar ele; do contrário, o escravo fica inativo, aguardando o mestre.

A Subseção 5.2.4 Conexão com os Nós aborda a classe abstrata `NodeConnection`, que abstrai como uma conexão nó-nó se dá; é possível configurar em `MasterSlaveCluster` um objeto responsável por determinar qual a modalidade de conexão mais apropriada entre o nó atual e outro.

Quando o nó mestre se desconecta do CG, seja isso programado pelo desenvolvedor ou devido a fatores tais como perda de conexão com a internet, os nós escravos são automaticamente

desconectados do CG. Quando um nó escravo se desconecta, o CG não é afetado, embora seja notificado.

Instâncias de `MasterSlaveCluster` contém um vetor de `NodeConnection`s, onde cada elemento representa a conexão do nó atual com algum outro no CG. Para o nó mestre, esse vetor terá um elemento para cada nó escravo, afinal, todos os nós escravos sinalizam para se conectar ao nó mestre. Para os nós escravos, esse vetor tem um único elemento, a conexão com o nó mestre. Assim, observa-se que o nó mestre se comunica com todos os escravos, porém, os escravos se comunicam exclusivamente com o mestre.

`MasterSlaveCluster` permite enviar dados para os nós com os quais o atual está conectado, iterando sobre o vetor de `NodeConnection`s, embora seja possível especificar somente um deles. Para o nó mestre, os dados são, por padrão, enviados a todos os escravos, enquanto, para os nós escravos os dados são enviados somente para o nó mestre. O nó mestre pode repassar os dados recebidos de um escravo para os outros, permitindo, assim, o funcionamento de um cluster descentralizado, ainda que, claramente, exista uma nó principal entre aqueles conectados ao CG.

Os dados, em si, são valores ou objetos JavaScript. Para o envio, o valor/objeto é automaticamente colocado dentro de um "pacote", que visa possibilitar controles adicionais sobre os dados enviados. O pacote é um objeto JavaScript com os seguintes atributos:

- `id`: identificador único para o dado enviado;
- `sender`: identificador do nó que envia o dado;
- `data`: o valor/objeto JavaScript que se deseja enviar; e
- `recipient`: identificador do nó que deve receber o dado (depende de argumento opcional; por padrão, não consta).

Antes de enviar o pacote com os dados, ele é serializado em JSON, um formato de texto. Devido a isso, não é possível enviar objetos JavaScript contendo ciclos de referência, pois estes não podem ser serializados em JSON; caso o objeto tenha métodos, estes são perdidos na serialização.

Este é o principal meio utilizado pelo nó mestre para enviar dados, quaisquer que sejam, aos nós escravos. `MasterSlaveCluster` permite que múltiplas funções *callback* sejam executadas quando dados são recebidos e, com isso, os nós escravos recebem, essencialmente, todos os dados do nó mestre através de um único canal, o que evita a criação de várias conexões entre os nós para trocar diferentes dados e, assim, facilita o gerenciamento das conexões existentes.

5.2.1.1 Segmentação de Dados Trocados

Existem, certamente, tratamentos para os dados recebidos por um nó que são indiferentes quanto ao tipo do dado, tais como *logs* ou cálculos de métricas. Contudo, considerando os diferentes tipos de dados que podem ser trocados por uma aplicação, bem como os usos específicos desses dados, a existência, em cada nó, de um único canal pelo qual os dados do CG são enviados e recebidos acarreta em duas necessidades:

- Dados enviados de um nó para outros devem apresentar alguma propriedade que permita, mais tarde, identificá-los; e

- Cada função *callback* cadastrada para o recebimento de dados de nós deve identificar os dados recebidos, para determinar quais dados devem ser processados e quais não.

Embora essas necessidades possam ser facilmente atendidas usando construtos de linguagem do JavaScript, o problema reside na repetição dos processos que as atendem, algo que pode ser facilmente encapsulado.

A Livvclib fornece a classe `ClusterDataEventManager` que utiliza "eventos" para segmentar os dados trocados entre nós através de `MasterSlaveCluster`. Utilizando essa classe, todos os dados enviados por um nó para o CG tomam a forma de eventos, os quais são formados por um nome único e argumentos adicionais. Alternativamente, ela também permite que sejam cadastradas funções *callback* executadas somente quando o nó recebe eventos específicos de outros nós, ao invés de toda vez que o nó recebe um dado.

O que `ClusterDataEventManager` faz, essencialmente, é definir seus próprios eventos (cada um com sua lista própria de funções *callback*) e cadastrar uma função *callback* para quando o nó receber um dado qualquer. O que essa função faz é verificar, para cada dado recebido, se ele corresponde a um evento e, caso positivo, chama os *callbacks* relativos a esse evento.

A Figura 16 ilustra o funcionamento de `ClusterDataEventManager`. Nela, os componentes da Livvclib responsáveis pelo envio e recebimento dos dados, além de `MasterSlaveCluster`, são omitidos, em prol da brevidade.

No envio (Figura 16a), o dado a ser enviado é colocado dentro de um objeto "pacote", o qual descreve o evento e o dado sendo enviados. Quando esse pacote é recebido (Figura 16b), `ClusterDataEventManager` identifica o evento enviado e repassa o dado somente para as funções *callback* cadastradas para ele (*Objeto X* e *Objeto Z*, no caso); observa-se que a função *callback* *Objeto T* também recebe o pacote de dados, pois está cadastrada para receber todos os dados enviados, sem qualquer filtro.

Se `ClusterDataEventManager` recebe um dado qualquer, sem estar no formato de "pacote" que ele processa (Figura 16c), isso significa que o dado não é evento algum e, portanto, nenhum de seus *callbacks* recebe o dado; mais uma vez, observa-se que *Objeto T* recebe o dado.

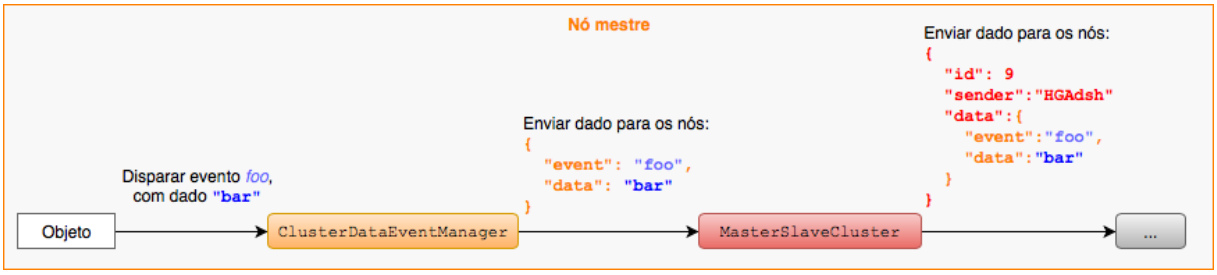
5.2.1.2 Armazenamento de Dados Enviados

Aplicações VI3DI que utilizam CG e que foram desenvolvidas com a Livvclib podem apresentar cenários onde os nós escravos são dinâmicos, i.e. eles podem surgir ou desaparecer dinamicamente. Nesses casos, quando um novo nó escravo se conecta ao mestre em meio a execução da aplicação, é interessante que esse nó escravo tenha meios para atualizar seu estado, de modo que possa acompanhar a execução do nó mestre sem problemas.

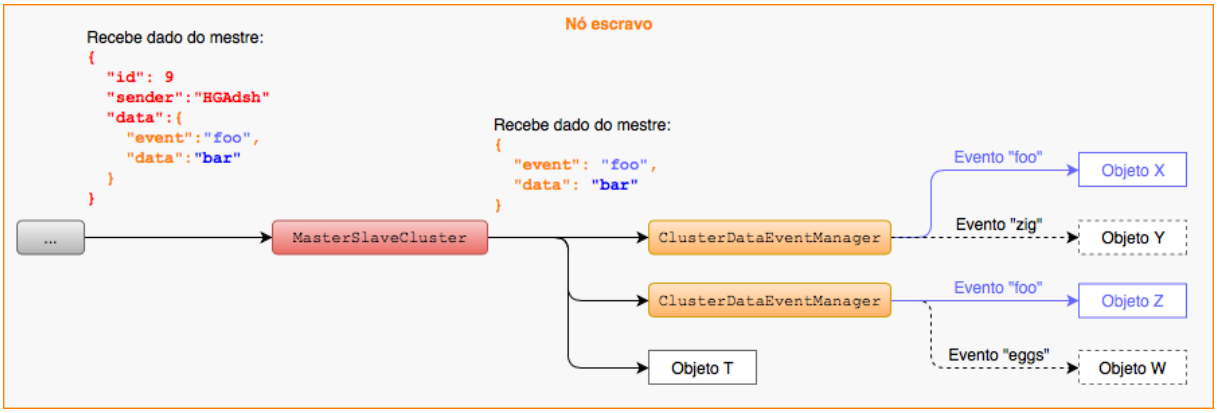
`BufferedMasterSlaveCluster` é uma especialização de `MasterSlaveCluster` que possibilita ao nó mestre armazenar os dados enviados; mais tarde, quando novos nós escravos se

Figura 16 – Envio e recebimento de dados por meio de `ClusterDataEventManager`.

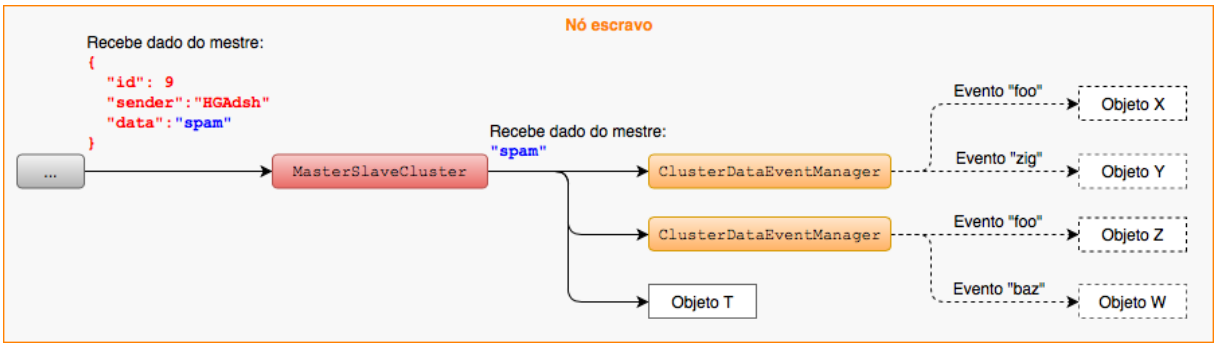
(a) Envio de dados com `ClusterDataEventManager`



(b) Recebimento de dados enviados por `ClusterDataEventManager`



(c) Recebimento de dados enviados sem `ClusterDataEventManager`



unem ao CG, eles podem consultar ao nó mestre quais foram dados enviados desde o início da aplicação e, assim, atualizar seus estados para o atual, junto com os outros nós.

Ao usar uma instância de `BufferedMasterSlaveCluster` para enviar dados para o CG, é possível especificar qual a maneira com que o dado será armazenado (definidas no enumerador `BufferMode`):

- `NONE`: o dado não é armazenado para posterioridade. Opção padrão, caso nenhuma seja especificada;
- `SINGLE`: o dado é armazenado num dicionário, sendo identificado por uma chave. Caso outro dado já esteja armazenado com a mesma chave, ele é substituído; e
- `NEW`: o dado é armazenado como um novo dado numa lista. Não interfere com outros dados

armazenados.

5.2.2 Sincronização de Estado

A utilização de CGs mestre-escravo requer que o nó mestre sincronize seu estado com os nós escravos. O estado do nó mestre em um dado momento é definido pelos valores armazenados pelo nó mestre, sejam eles diretos (em variáveis), ou indiretos (em atributos de objetos em variáveis). Assim, a sincronização de estado requer que, toda vez que algum valor for alterado no nó mestre, esse valor também seja alterado nos nós escravos, ou seja, quando o nó mestre alterar um valor, ele deve enviar essa alteração para os nós escravos, para que eles a repitam. Isso, a princípio, requer que todos os valores críticos da aplicação sejam identificados e, também, que, suas alterações sejam enviadas aos escravos, quando ocorrerem.

A fim de simplificar o ônus dessa sincronização do estado da aplicação, a Livvclib fornece a classe `Sync`, que permite rastrear o estado de objetos e sincronizá-los automaticamente quando forem alterados, de maneira transparente para o desenvolvedor.

`Sync` possui o método `trackAttribute()`, que é usado para sincronizar o valor de um atributo de um objeto. Esse método usa a biblioteca JavaScript WatchJS[2] para observar mudanças no valor do atributo; quando elas ocorrem, o atributo modificado e seu novo valor são repassados para os nós ao qual o original está conectado. Caso o nó original seja um escravo, ele repassará os dados da alteração para o mestre, que os repassará aos outros escravos, permitindo, assim, a emulação de um CG descentralizado.

Há, também, o método `trackMethod()`, usado para sincronizar invocações de um método de um objeto. Devido a natureza dinâmica do JavaScript, é possível alterar o *prototype* (i.e. conjunto de atributos e métodos) de um objeto durante a execução de uma aplicação; com isso, `trackMethod()` modifica o *prototype* do objeto observado, criando uma cópia do método observado, enquanto o método original é substituído por um novo, que invoca o original (através de sua cópia) e, em seguida, repassa o método invocado e os parâmetros utilizados para as instâncias de `Sync` correspondentes nos nós ao qual o original está conectado, que invocam o método em suas cópias locais do objeto observado.

Internamente, `Sync` usa uma instância de `ClusterDataEventManager` para enviar e receber os novos valores dos atributos e invocações de métodos de objetos, do nó mestre aos escravos.

`Sync` não apresenta requisito algum para os objetos observados. Isso facilita sua integração com aplicações e bibliotecas já existentes, pois não é necessário mudar tipos já existentes para se adequarem a padrão algum. Contudo, a inexistência de requisitos para os objetos observados também impede que seja estabelecida uma relação forte entre os objetos correspondentes em cada nó, de modo que, quando uma alteração é observada no nó mestre, seja sabido qual objeto ajustar nos nós escravos.

Devido a isso, cada instância de `Sync` deve ter um nome único e pré-determinado; dessa maneira, os dados de uma sincronização de atributo/método também incluem o nome da instância

[2] WatchJS: <https://github.com/melanke/Watch.JS>.

de `Sync` observando o objeto, o que permite as instâncias de `Sync` correspondentes nos nós escravos identificarem quando uma sincronização é para um objeto que elas observam ou não.

Com `Sync`, o trabalho de sincronização do estado da aplicação é quase totalmente transparente ao desenvolvedor. Sua única responsabilidade é identificar os objetos, atributos e métodos que necessitam ser sincronizados e apontá-los para instâncias de `Sync`, as quais se encarregam do trabalho de verdadeiramente sincronizar os objetos.

5.2.3 Ajuste de Perspectiva

Como visto na Subseção 4.2.3 Ajuste de Perspectiva, a princípio, cada nó de um CG executando uma aplicação desenvolvida com Livvclib estará vinculado a um dispositivo de exibição diferente, sendo assim, embora todos os nós possuam câmeras virtuais com as mesmas posição e orientação no ambiente virtual, cada um deles será responsável por exibir um trecho específico desse ambiente, relativo à posição e à orientação físicas do dispositivo de exibição ao qual está ligado.

`DistributedCamera` é uma extensão do tipo `THREE.PerspectiveCamera`, parte do Three.js (CABELLO, 2017), que implementa a matriz de projeção generalizada proposta por Kooima (2008). Os valores de p_a, p_b, p_c e p_e são atribuídos e acessados através de um atributo do tipo `Screen`, chamado `screenPoints`, o qual, por sua vez, possui atributos referentes a cada um desses pontos (`pA`, `pB`, `pC` e `pE`, respectivamente). Cada ponto é representado pelo tipo `THREE.Vector3`, um vetor tridimensional.

`DistributedCamera` sobrescreve alguns métodos de `THREE.PerspectiveCamera` e também implementa outros próprios. Todos os seus métodos e propriedades, contudo, são relativamente independentes uns dos outros. Isso é particularmente útil para casos de extensão de aplicações e bibliotecas já existentes, pois existe a possibilidade de que elas já façam uso de uma câmera virtual e não permitam sua troca por uma instância `DistributedCamera` (ver Capítulo 6 Estudo de Caso); nesses casos, graças a flexibilidade do JavaScript, é possível alterar o *prototype* da câmera utilizada, substituindo seus métodos pelas implementações de `DistributedCamera`.

As posição e orientação de instâncias de `DistributedCamera` são sincronizadas do nó mestre nos nós escravos utilizando uma instância de `Sync`. Dessa maneira, o desenvolvedor trabalha em sua aplicação como se as câmeras virtuais do nó mestre (aquele que recebe a entrada de dados do usuário) fossem as únicas, enquanto os escravos são sincronizados de acordo.

5.2.4 Conexão com os Nós

A Livvclib é projetada para que os nós de um CG troquem dados por meio de conexões P2P entre os nós, estabelecidas com os recursos oferecidos pela API WebRTC. No entanto, dependendo de fatores específicos da aplicação ou dos dispositivos utilizados, WebRTC pode não ser a melhor escolha, ou simplesmente não estar disponível. Sendo assim, é necessário que a troca de dados entre os nós seja abstraída, permitindo que os nós se conectem de diferentes maneiras.

Levando esses fatores em conta, `NodeConnection` é uma classe abstrata[3] que representa

[3] Ou o mais próximo possível que o JavaScript apresenta desse construto de linguagem.

a conexão entre dois nós do CG (mais precisamente, com o uso de `MasterSlaveCluster`, a conexão entre o nó mestre e um dos escravos). Seu propósito é abstrair as conexões e trocas de dados entre os nós, permitindo que o desenvolvedor implemente esses detalhes da maneira mais adequada para sua aplicação.

`NodeConnection` define vários métodos que visam cobrir todo o processo de estabelecimento da conexão entre dois nós e a maneira como eles enviam dados para, bem como tratam dados recebidos de, outros nós. Dependendo do tipo da conexão, todavia, as implementações de alguns desses métodos podem acabar vazias, pois o tipo de conexão pode não ter evento correspondente.

A Livvclib fornece duas implementações de `NodeConnection`: uma utilizando o WebRTC e outra utilizando o próprio servidor de aplicação e sinalização.

`WebRTCNodeConnection` é a implementação principal e, como seu nome indica, ela trabalha com a API WebRTC, utilizando a interface `RTCPeerConnection` para representar a conexão P2P entre os nós e a interface `DataChannel` para enviar e receber os dados.

`SignalingServerNodeConnection` é uma alternativa a `WebRTCNodeConnection`, na qual não existe conexão direta entre os nós. Para que a troca de dados seja realizada, o remetente envia os dados para o servidor de aplicação e sinalização por meio da instância de `ServerConnection` usada para sinalizar a conexão entre ele e os destinatários. O servidor, então, repassa esses dados para os destinatários, que os recebem através das suas instâncias de `ServerConnection` (Subseção 5.2.6 Conexão com o Servidor) usadas para sinalizar a conexão entre eles e o remetente. Essa alternativa não é ideal, pois adiciona um intermediário na troca de dados entre nós (o servidor), o que pode elevar a latência; no entanto, ela possibilita que a aplicação seja executada em dispositivos nos quais os navegadores web ainda não tem suporte a WebRTC, como é o caso de dispositivos móveis.

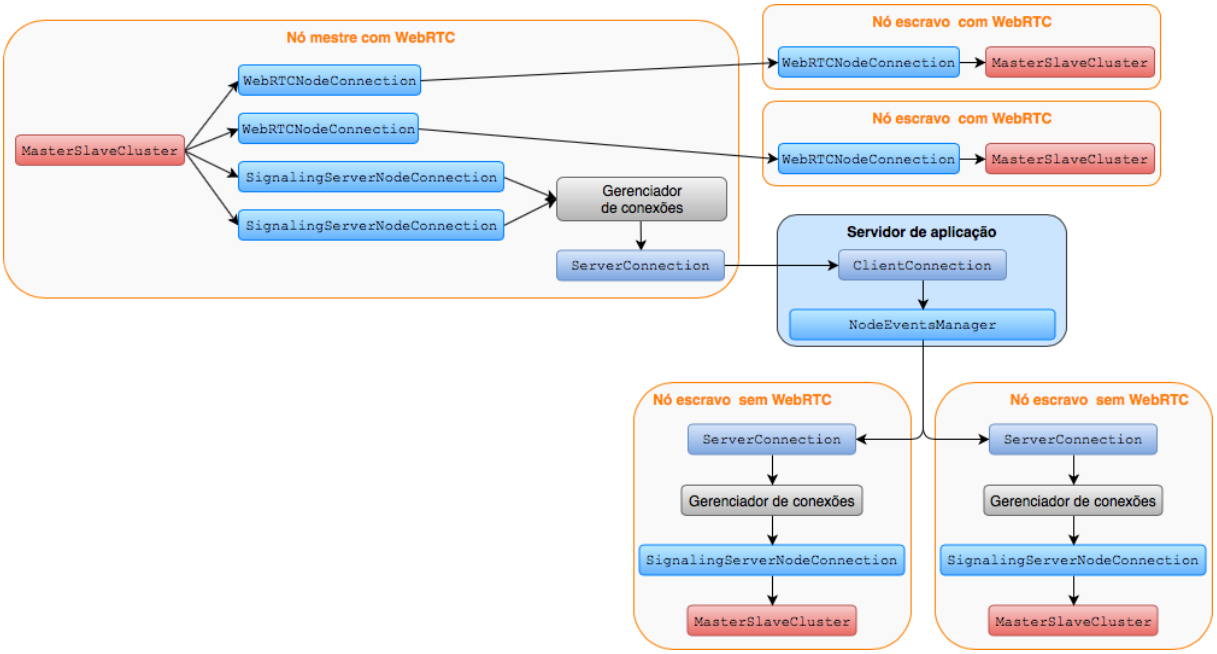
A Figura 17 mostra uma representação do funcionamento das implementações de `NodeConnection`. Nota-se que, mesmo que exista mais de um nó escravo conectado através `SignalingServerNodeConnection`, o nó mestre envia os dados para o servidor uma única vez, graças ao objeto "Gerenciador de Conexões", que é interno da Livvclib. A partir da sua conexão com o nó (Subseção 5.3.1 Comunicação com Lado Cliente), o servidor recebe o dado como um evento de entrada processado por `NodeEventsHandler` (Subseção 5.3.2 Tratamento de Eventos), que faz o *broadcast* do dado para os nós escravos, os quais, por sua vez, o recebem pelo caminho inverso do envio.

Como ambos os meios de conexão compartilham a mesma API, a maneira como a troca de dados entre os nós efetivamente acontece é transparente para as instâncias de `MasterSlaveCluster` abstraindo o CG.

5.2.5 Acesso a Dispositivos de Interação

A Livvclib também oferece meios para que aplicações executadas em navegadores web utilizem dispositivos de interação diferenciados de mouse e teclado, tais como controles de videogame, Kinect, etc. Os dispositivos são utilizados por meio da biblioteca C/C++ VRPN

Figura 17 – Representação das implementações de `NodeConnection`. Setas representam o dado sendo enviado.



Fonte - Produzida pelo autor.

(Subseção B.4 Virtual Reality Peripheral Network), que executa seu próprio servidor de aplicação nos computadores onde os dispositivos de interação estão ligados. Esses servidores, então, expõem para a internet as interfaces dos dispositivos de interação, para que aplicações cliente se conectem a elas e recebam dados dos dispositivos.

No lado cliente da Livvclib, a classe `DeviceControlAccess` abstrai os detalhes da conexão de uma aplicação com uma interface de um dispositivo de interação ligado a um servidor VRPN. Com ela, o nó pode se conectar a uma interface de um dispositivo de interação, configurá-la para ouvir determinados eventos do dispositivo e receber seus dados.

É importante notar que `DeviceControlAccess` não realiza essas operações, propriamente ditas. Essa classe, junto à `DeviceEventsHandler` (Subseção 5.3.2 Tratamento de Eventos), estabelece um conjunto de eventos de entrada e saída para o servidor de aplicação. Assim, quando o nó usa `DeviceControlAccess` para realizar alguma operação relativa a uma interface de um dispositivo de interação, o que ele realmente faz é disparar um evento de entrada, especificando a operação, para o servidor de aplicação. Esse evento é capturado por `DeviceEventsHandler`, que é quem, de fato, realiza a operação com a interface do dispositivo; quando a operação acaba, é, então, disparado um evento de saída para nó requisitante (e capturado por `DeviceControlAccess`) com o resultado da operação.

O processo de comunicação é transparente, pois é gerenciado por `ServerConnection`. Para o desenvolvedor, o foco deve ser conhecer o endereço do servidor VRPN ao qual o dispositivo está ligado e com quais interfaces do dispositivo ele pode se conectar.

5.2.5.1 Notificação de Outros Nós

O uso de `DeviceControlAccess` requer que o nó faça parte de uma sala virtual no servidor, no entanto, essa sala não precisa, necessariamente, corresponder a um CG, podendo ser um mero agregado de nós.

Caso essa sala virtual contenha vários outros nós que dependem dos dados da interface dispositivo de interação (seja porque estão executando uma determinada aplicação junto ao nó requisitante, em forma de um CG, ou porque meramente usam esses dados), o tempo entre o usuário da aplicação realizar uma ação com o dispositivo de interação e os dados dessa ação chegarem aos nós escravos pode ser demasiadamente longo, pois os dados primeiro trafegam do servidor VRPN para o servidor de aplicação, depois do servidor de aplicação para o nó requisitante e, finalmente, do nó requisitante aos demais.

A fim de diminuir essa possível latência, `DeviceControlAccess` e `DeviceEventsHandler` permitem configurar a conexão com a interface de um dispositivos de interação de modo que os dados de eventos dessa interface sejam enviados a todos os nós que fazem parte da sala virtual do nó requisitante, ao invés de somente para ele, reduzindo, assim, o caminho dos dados do dispositivo até os nós e, consequentemente, a latência.

Com esse *broadcast* dos dados do dispositivo, é necessária uma maneira de notificar os outros nós de que eles receberão dados de uma interface de dispositivo, para que possam preparar os tratamentos adequados. Isto é feito com a classe `NewDevicesListener`, que, através de uma instância de `ServerConnection`, é notificada quando um nó na mesma sala virtual que o atual requisita conexão uma interface de um dispositivo de interação. Ao receber essa notificação, `NewDevicesListener` cria uma instância da classe `DeviceSimpleAccess` para representar a conexão com a interface do dispositivo e, também, uma instância de `DeviceInterfaceTracker` (Subseção 5.2.5.2 Rastreio de Eventos da Interface do Dispositivo de Interação) adequada para a interface.

`DeviceSimpleAccess` é a *superclasse* de `DeviceControlAccess` e possui um conjunto limitado de operações com interfaces de dispositivos de interação, suficiente para simular, nos nós, a conexão com o dispositivo. O controle verdadeiro sobre a conexão com a interface do dispositivo de interação ainda é mantido no nó requisitante (via de regra, o mestre), que usou `DeviceControlAccess` para gerenciar a interface do dispositivo de interação.

5.2.5.2 Rastreio de Eventos da Interface do Dispositivo de Interação

Tecnicamente, a classe `DeviceSimpleAccess` é o suficiente para se trabalhar com dispositivos de interação nos nós. No entanto, todos os dados de eventos de uma interface de dispositivo de interação são recebidos por um único canal, de maneira similar aos dados trocados entre os nós por meio de `MasterSlaveCluster`.

O VRPN fornece abstrações para tipos de dados comuns que se pode obter de dispositivos de interação. Essas abstrações, então, podem ser aplicadas a interfaces dos dispositivos de interação, permitindo capturar, no dispositivo, o tipo de dado que especificam.

A fim, então, de criar canais especializados para o recebimento de dados de eventos específi-

cos das abstrações de tipos de dados fornecidas pelo VRPN, há a classe `DeviceIntefaceTracker`, que, por meio de uma instância de `DeviceSimpleAccess`, mapeia esses eventos para o JavaScript. Mais especificamente, ela permite que o nó:

- Especifique quais eventos de uma abstração devem ser capturados pelo servidor de aplicação (que, mais tarde, os repassa para os nós);
- Cadastre funções *callback* para executar quando um evento específico da abstração for capturado; e
- Cadastre validadores para os dados enviados num evento da abstração, que determinam se as funções *callback* são chamadas ou não.

Visando facilitar ainda mais o uso dessa classe, são fornecidas especializações dela para trabalhar com as abstrações *Analog* (classe `AnalogTracker`) e *Button* (classe `ButtonTracker`). As especializações já especificam quais os eventos específicos dessas abstrações e também cadastram validadores específicos para eles. O desenvolvedor, nesse caso, deve meramente escolher quais eventos da abstração capturar e cadastrar as funções *callback* adequadas para eles.

É importante notar, contudo, que `DeviceIntefaceTracker` e suas especializações dizem respeito somente ao eventos das abstrações de tipos de dados do VRPN. Elas não gerenciam a conexão da aplicação com o servidor VRPN que expõe as interfaces dos dispositivos de interação, pois isso já é feito por `DeviceControlAccess`.

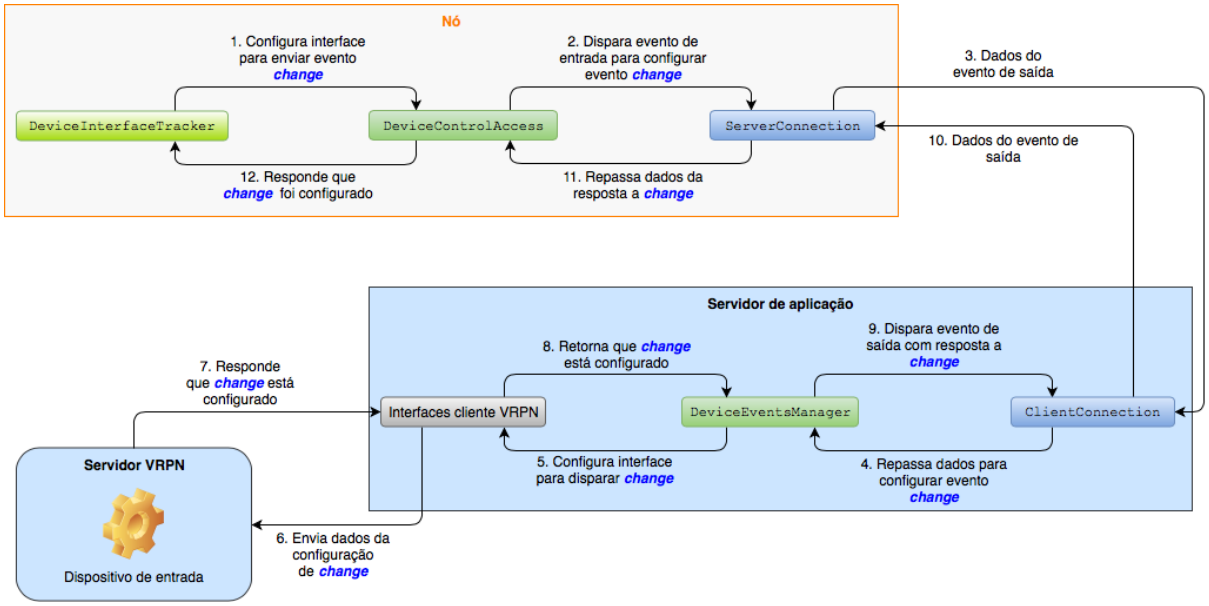
A Figura 18 demonstra o que realmente ocorre quando `DeviceInterfaceTracker` parece configurar uma interface de dispositivo de interação para que envie dados de um de seus eventos (no caso, o evento *change*). É usado `DeviceControlAccess` para disparar um evento de entrada no servidor de aplicação, que é processado por `DeviceEventsHandler`, que, por sua vez, utiliza as interfaces de cliente VRPN, internas na Livvclib, para se comunicar com o servidor VRPN no qual o dispositivo de interação está ligado. Quando o servidor VRPN responde, a resposta faz todo o caminho de volta a `DeviceInterfaceTracker`.

A Figura 19 mostra como os dados do evento anterior, *change*, saem do dispositivo e chegam ao nó. Percebe-se que, assim como `ClusterDataEventsManager` com os dados recebidos por `MasterSlaveCluster`, `DeviceInterfaceTracker` auxilia a segmentar os dados de eventos específicos para funções *callback* específicas; no caso, o *Objeto Y* é quem recebe esses dados. Como antes, também se observa que é possível cadastrar funções *callback* para receberem dados de todos os eventos, sem filtros (*Objeto X*, no caso).

5.2.6 Conexão com o Servidor

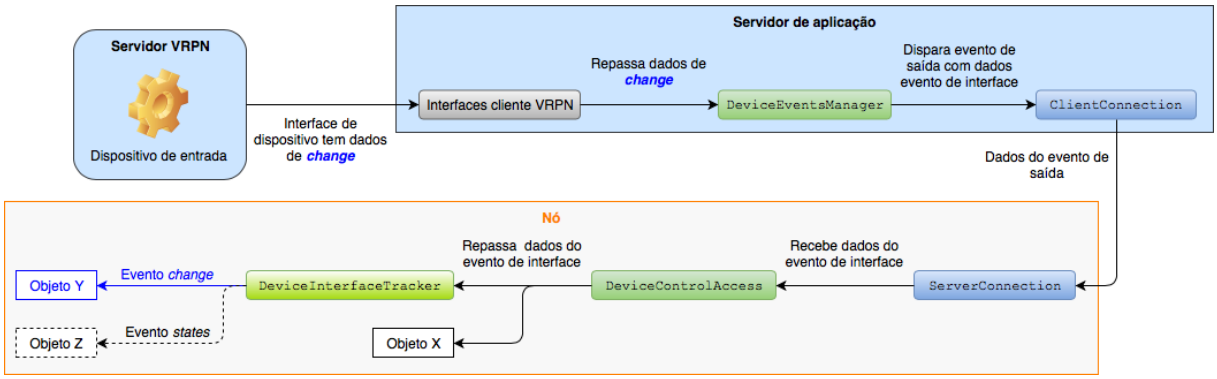
Independentemente da razão pela qual a Livvclib é utilizada (CG, dispositivos de interação, ambos), é necessário que ela tenha um canal constante de conexão com o servidor de aplicação e sinalização, por meio do qual troca eventos de entrada e saída para a sinalização da conexão entre dois nós, gestão dos dispositivos de interação, etc.

Figura 18 – Como `DeviceInterfaceTracker` configura uma interface de dispositivo de interação para enviar dados de determinado evento.



Fonte - Produzida pelo autor.

Figura 19 – Recebimento de dados de evento de interface de dispositivo de interação.



Fonte - Produzida pelo autor.

A fim de permitir que a lógica da aplicação, dos CGs e dos dispositivos de interação seja independente da maneira como o servidor de aplicação e sinalização é implementado, é necessário abstrair as conexões e trocas de dados dos nós com o servidor. Para tanto, é fornecida a classe abstrata `ServerConnection`, que representa o canal de comunicação de um nó com o servidor de aplicação e sinalização.

Ela define métodos que visam abranger a abertura do canal de comunicação nó-servidor, o disparo de eventos de entrada para o servidor e o recebimento de eventos de saída do servidor. Sua API é propositalmente reduzida, a fim de permitir maior flexibilidade, tanto na implementação do canal de comunicação nó-servidor, quanto do próprio servidor de comunicação e sinalização.

É fornecida uma implementação de `ServerConnection`, `SocketIOServerConnection`, que gerencia o canal de comunicação com o servidor utilizando a biblioteca Socket.IO[4] (disponível

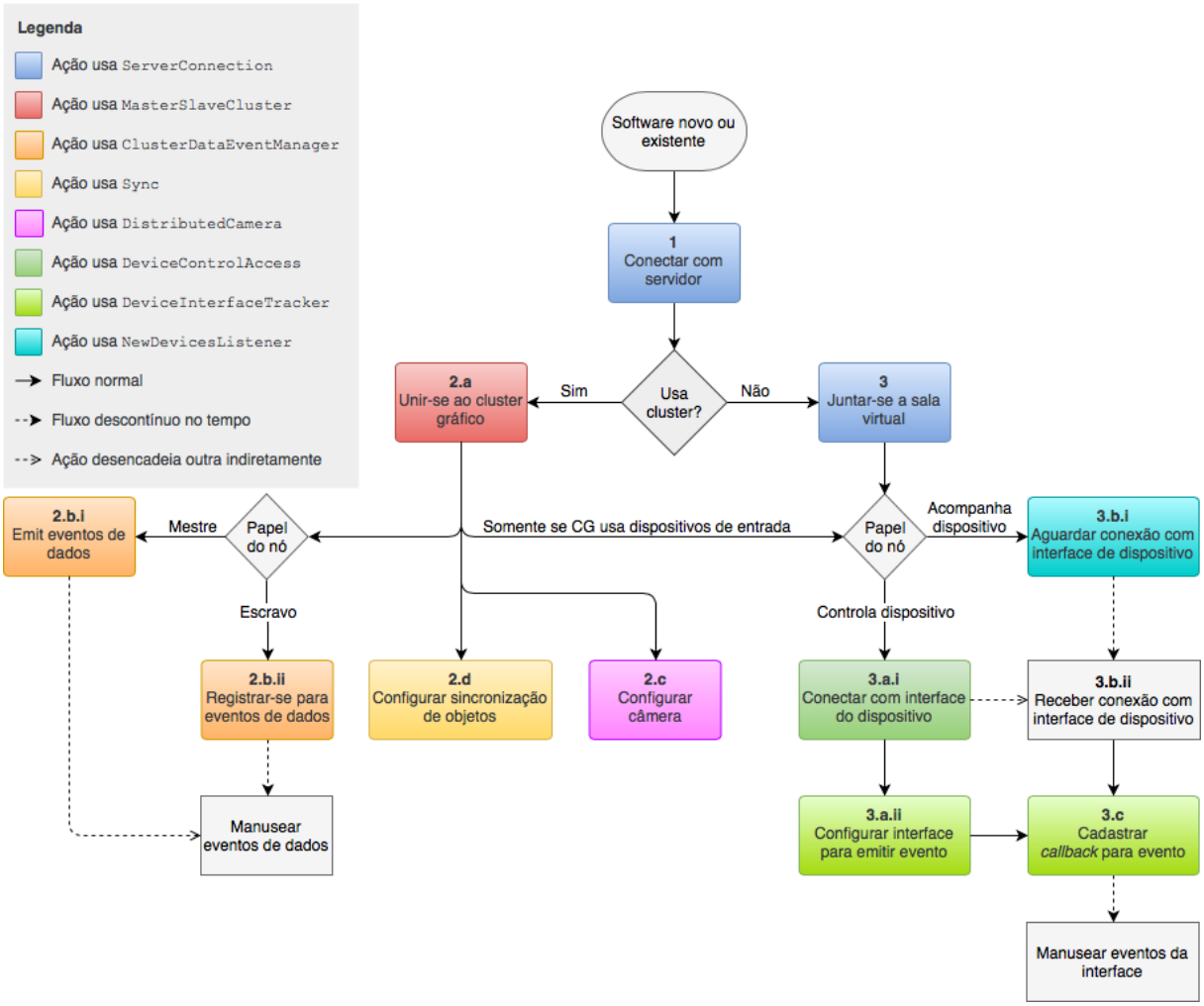
[4] Socket.IO: <https://socket.io>.

em várias linguagens), a qual é baseada em WebSockets e permite comunicação bidirecional em tempo-real baseada em eventos entre um servidor e um cliente. Graças a seu uso (e ao WebRTC), o processo de sinalização da conexão entre dois tem um fluxo idêntico ao apresentado na Figura 62.

5.2.7 Uso

A Figura 20 ilustra a maneira como os recursos oferecidos pela Livvclib para o desenvolvimento lado cliente de uma aplicação podem ser utilizados.

Figura 20 – Uso dos recursos da Livvclib para o lado cliente.



Fonte - Produzida pelo autor.

O uso da Livvclib começa a partir do código-fonte próprio da aplicação VI3DI web. Independentemente da razão pela qual a Livvclib é utilizada, o primeiro passo é criar uma instância de uma implementação de `ServerConnection` (1), a fim de que o nó possa se comunicar com o servidor de aplicação e sinalização; com isso, o nó já tem condições para se unir a um CG e/ou trabalhar com dispositivos de interação.

Com o nó conectado ao servidor, o uso pode seguir por dois caminhos, dependendo de se a aplicação utiliza CGs ou não.

Caso a aplicação trabalhe com CGs, o nó, então, utiliza a instância de `ServerConnection` para criar uma instância de `MasterSlaveCluster` que, por sua vez, é usada para uni-lo ao CG de alguma sala virtual (2.a).

Uma vez que o nó tenha se unido ao CG, caso ele seja o mestre, ele pode usar instâncias de `ClusterDataEventManager` para enviar eventos de dados aos nós escravos (2.b.i); caso o nó seja escravo, ele também pode usar instâncias de `ClusterDataEventManager`, mas, dessa vez, para cadastrar funções *callback* (2.b.ii) executadas ao receber eventos de dados do nó mestre.

Com o nó unido ao CG, independentemente de seu papel, ele também pode usar instâncias de `Sync` para definir os atributos e métodos de objetos chaves da aplicação que serão sincronizados (2.c). Além disso, é possível configurar a(s) câmera(s) virtual(is) da aplicação para que seja(m) instância(s) de `DistributedCamera` que sincroniza(m) sua(s) posição(ões) e orientação(ões) com os outros nós no CG (2.d).

Caso a aplicação não trabalhe com CGs, isso significa que a aplicação usa a Livvclib para acessar dispositivos de interação. Nesse caso, o nó deve utilizar a instância de `ServerConnection` para se unir a uma sala virtual qualquer (3) (de preferência, que não corresponda a um CG).

Com o nó fazendo parte de uma sala virtual (seja devido a um CG ou por conta própria), há dois caminhos a se prosseguir, dependendo de se o nó é quem deve controlar a operação dos dispositivos de interação na sala virtual ou não.

Caso o nó deva, de fato, controlar a operação dos dispositivos de interação, ele deve, então, usar a instância de `ServerConnection` para criar uma instância de `DeviceControlAccess` que, por sua vez, irá requisitar conexão com uma interface de um dispositivo (3.a.i) e, em seguida, requisitar para que a interface envie dados de um evento específico (3.a.ii). Caso o nó deva somente acompanhar os dados recebidos da interface do dispositivo, ele deve usar a instância de `ServerConnection` para configurar uma instância de `NewDevicesListener` (3.b.i) que, por sua vez, criará instâncias de `DeviceSimpleAccess` e `DeviceInterfaceTracker` quando for notificada da conexão com alguma interface de dispositivo de interação (3.b.ii).

Uma vez que o nó tenha acesso a conexão com a interface do dispositivo entrada (seja porque controla sua operação ou porque a acompanha), ele pode usar uma instância de `DeviceInterfaceTracker` para cadastrar funções *callback* (3.c) executadas quando a interface envia dados de algum evento.

5.3 Lado Servidor

O design de solução apresentado não inclui funcionalidades para o lado servidor de uma aplicação desenvolvida com ele, exceto pela definição geral de alguns eventos de entrada, aos quais o servidor deve responder, e de saída, que o servidor deve emitir aos nós.

A Livvclib, contudo, inclui componentes para o lado servidor, os quais foram desenvolvidos não só como parte do framework, mas, também, para ajudar no desenvolvimento de seus componentes principais, no lado cliente.

Em todo caso, os componentes do lado servidor da Livvclib não são essenciais, a fim

de permitir que o desenvolvedor implemente ou reuse um servidor de aplicação e sinalização utilizando tecnologias diferentes do Node.js. Para que um servidor de aplicação e sinalização possa ser utilizado com o lado cliente da Livvclib dois requisitos devem ser atendidos:

- O servidor deve ser capaz de se comunicar com alguma implementação de `ServerConnection` (seja a que já é fornecida com a Livvclib, ou outra, nova, implementada pelo desenvolvedor); e
- O servidor deve processar os eventos de entrada e emitir os eventos de saída especificados no Apêndice C Eventos de Entrada e de Saída da Livvclib.

Os componentes fornecidos pela própria Livvclib para atender a esses requisitos são abordados nas próximas subseções.

5.3.1 Comunicação com Lado Cliente

Como os nós de uma aplicação desenvolvida com a Livvclib precisam de um canal de comunicação para troca de eventos com o servidor de aplicação e sinalização, é necessário apoiar essa necessidade também do lado servidor.

A classe abstrata `ClientsManager` define o comportamento de um objeto responsável por gerir os nós executando a aplicação (não inclui gestão de CGs). Sua gestão é particularmente simples, limitando-se armazenar e consultar as referências aos canais de comunicação entre o servidor e os nós e, também, realizar *broadcast* de dados para uma sala virtual de nós.

Do lado servidor, os canais de comunicação nó-servidor são abstraídos pela classe abstrata `ClientConnection`. Inversamente a `ServerConnection`, `ClientConnection` representa o canal de comunicação do servidor com um dos nós, e ele deve ser capaz não só de receber eventos de entrada do nó e emitir eventos de saída para ele, como também unir o nó a uma sala virtual, algo essencial para o funcionamento dos CGs.

Assim como com `ServerConnection`, a API dessas classes abstratas é reduzida, a fim de possibilitar maior flexibilidade na maneira como o servidor se comunica com os nós.

A Livvclib fornece implementações para estas duas classes, `SocketIOClientsManager` e `SocketIOClientConnection`, as quais utilizam a biblioteca Socket.IO e foram projetadas para trabalhar junto a `SocketIOServerConnection` do lado cliente.

5.3.2 Tratamento de Eventos

A principal função dos componentes da Livvclib no lado servidor é tratar os eventos de entrada disparados pelos nós no servidor e disparar os eventos de saída do servidor nos nós.

Isso é feito por meio de implementações da classe abstrata `ClientEventsHandler`, as quais são notificadas toda vez que o canal de comunicação entre um nó e o servidor é aberto ou fechado. Na abertura do canal, a instância de `ClientEventsHandler` armazena a referência para o canal de comunicação com o nó, usando-o, a princípio, para cadastrar tratamentos a

determinados eventos de entrada possivelmente emitidos pelo nó e, mais tarde, para emitir eventos de saída ao nó ou a outros relacionados a ele.

A Livvclib fornece quatro implementações de `ClientEventsHandler` que apoiam o funcionamento das interfaces disponíveis no lado cliente a classe `WebClusterSupport` pode ser usada para automatizar a notificação de uma lista de instâncias de `ClientEventsHandler` toda vez que um canal de comunicação nó-servidor é aberto ou fechado. Para tanto, ela requer a instância de `ClientsManager` usada para gerenciar esses canais.

O suporte a CGs se dá por meio da classe `ClusterEventsHandler`, que trata eventos de entrada e emite eventos de saída relativos ao estabelecimento de um CG mestre-escravo, bem como apoia a sinalização durante a conexão entre dois nós (por WebRTC, pelo servidor, ou qualquer outro meio). Adicionalmente, também há a classe `NodeEventsManager`, que trata e emite eventos usados na troca de dados entre nós conectados por meio de `SignalingServerNodeConnection`.

Quanto aos dispositivos de interação, como visto na Subseção 5.2.5 Acesso a Dispositivos de Interação, numa aplicação desenvolvida com a Livvclib, o servidor de aplicação é o responsável por, de fato, manipular os dispositivos de interação, ainda que sob ordens dos nós. A classe `DeviceEventsHandler` é a responsável por tratar e emitir os eventos relacionados a tanto.

O manuseamento dos dispositivos de interação, propriamente dito, é feito por componentes *internos* da Livvclib, escritos em C++, que usam a API cliente do VRPN para se conectar a servidores VRPN, solicitar a conexão a interfaces de dispositivos de interação, se cadastrar para receber dados de eventos dessas interfaces e tratar esses dados, quando recebidos. Como a Livvclib é implementada, primariamente, em JavaScript, é necessária, então, uma "ponte" entre esses componentes C++ e o código JavaScript para o Node.js. Isso foi feito por meio da biblioteca NAN[5], que é usada para desenvolver extensões C/C++ para aplicações Javascript para Node.js, e nbind[6], um conjunto de cabeçalhos, classes e funções em C/C ++ que facilita *portar* código C++ existente para JavaScript usando NAN.

Por fim, como visto na Subseção 5.2.5.1 Notificação de Outros Nós, os nós precisam fazer parte de uma sala virtual para usarem os recursos da Livvclib que manipulam os dispositivos de interação; mais ainda, essa sala virtual não precisa, necessariamente, corresponder a de um CG.

Sendo assim, há a classe *RoomEventsManager*, que é responsável por tratar eventos de entrada e saída que visam meramente adicionar um nó a uma sala virtual e removê-lo de lá, para que possa usar os recursos relacionados a dispositivos de interação e sem qualquer consideração com CGs. Devido a simplicidade relacionada a estes eventos (não há sequer, eventos de saída emitidos), o lado cliente da Livvclib não apresenta controlador específico para eles, como acontece, por exemplo, com os eventos relacionados a CGs e dispositivos de interação.

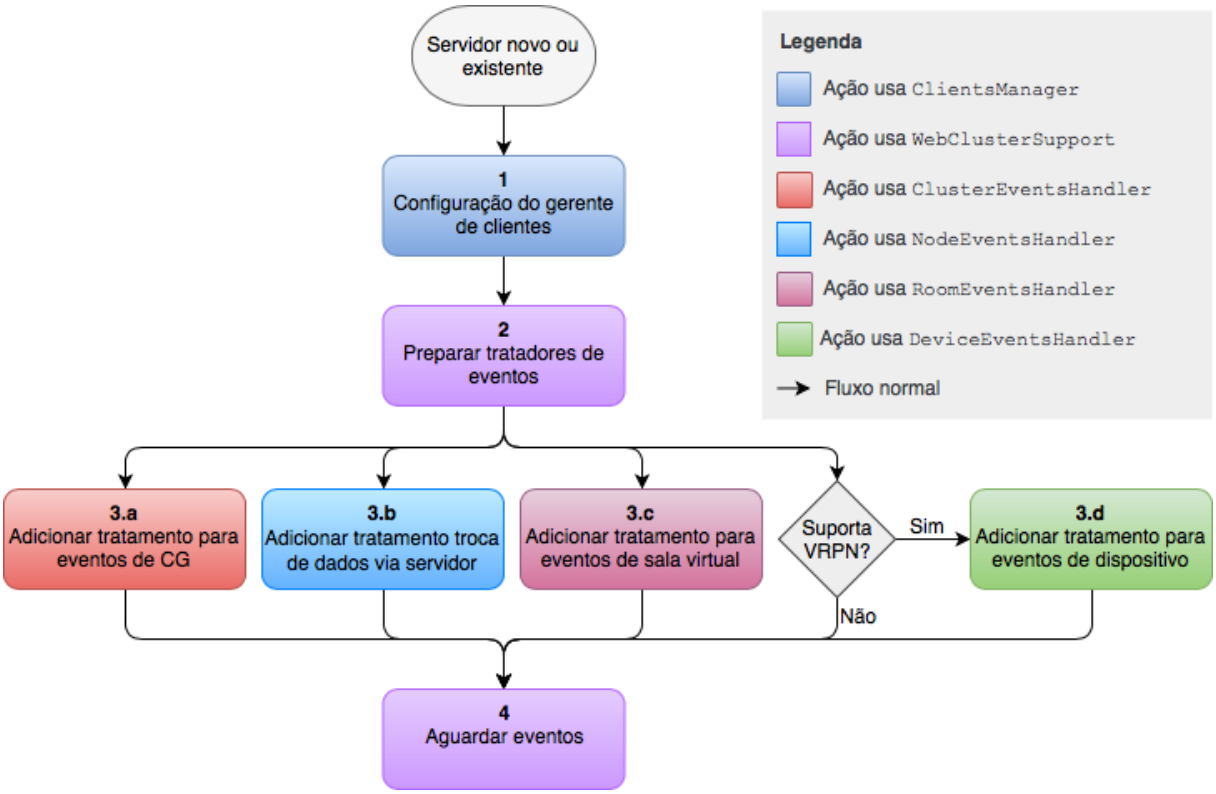
Os eventos de entrada e saída definidos pela Livvclib são apresentados em maiores detalhes no Apêndice C Eventos de Entrada e de Saída da Livvclib.

[5] NAN: <https://github.com/nodejs/nan>.
[6] nbind: <https://github.com/charto/nbind>.

5.3.3 Uso

A Figura 21 ilustra a maneira como os recursos oferecidos pela Livvclib para o desenvolvimento lado servidor de uma aplicação podem ser utilizados.

Figura 21 – Uso dos recursos da Livvclib para o lado servidor.



Fonte - Produzida pelo autor.

Observa-se que o uso da Livvclib no lado servidor é consideravelmente mais simples que no lado cliente, salientando o foco da Livvclib no cliente e o papel reduzido do servidor de aplicação e sinalização nas aplicações VI3DI desenvolvidas com ela.

Como antes, o uso da Livvclib começa com o código-fonte próprio da aplicação. A partir disso é criada uma instância de uma implementação de `ClientsManager` (1) para gerir os canais de comunicação entre os nós e o servidor.

Em seguida, a instância de `ClientsManager` é usada para instanciar `WebClusterSupport` (2), que facilita a execução dos tratamentos de eventos de entrada e saída no servidor.

Com a instância `WebClusterSupport`, são, então cadastradas nela as instâncias de `ClientEventsHandler`:

- `ClusterEventsHandler`, para sinalização e gerenciamento de CGs (3.a);
- `NodeEventsHandler`, para que nós possam trocar dados usando o servidor como intermediário, por meio de `SignalingServerNodeConnection` (3.b);
- `RoomEventsHandler`, para união de nós a salas virtuais, sem o uso de CGs (3.c); e

- `DeviceEventsHandler`, para receber ordens quanto aos dispositivos de interação (3.d), se o computador suportar o VRPN.

Para finalizar, a instância de `WebClusterSupport` é configurada para aguardar a chegada de eventos de entrada (4), dos quais o tratamento contribuirá para o funcionamento da aplicação VI3DI web no lado cliente.

É importante notar que, para os tratamentos relacionados a dispositivos de interação, o servidor VRPN com o dispositivo já deve estar no ar e operante.

5.4 Desempenho

Para avaliar o desempenho do Livvclib, vários experimentos foram realizados usando-a para trocar dados entre CGs e adquirir dados de dispositivos de entrada. A base para os experimentos foi medir, nos nós, os intervalos de tempo entre o recebimento de dados enviados consecutivamente a eles pelo nó mestre ou pelo servidor, usando Livvclib. Isso pode ser medido, compilado e comparado com relativa facilidade; quanto menores os intervalos medidos, melhor, porque significa que mais dados podem ser transferidos entre os nós em um determinado período de tempo.

Os experimentos foram conduzidos usando os seguintes equipamentos:

- Um MacBook Pro de 15", executando Windows 10 (processador Intel i7 2GHz, chip gráfico Intel Iris Pro 1536mb, 8GB de RAM), conectado via wi-fi;
- Um Acer Aspire 5 A515-51G-58VH (processador Intel i5 2.5GHz, placa gráfica GeForce 940MX 2GB, 8GB de RAM) conectado via wi-fi; e
- Um desktop com Windows 10 (Intel Pentium G3260 3.3GHz, chip gráfico Intel GD 2096mb, 4GB de RAM), conectado via ethernet.

Todos os computadores estavam conectados à mesma LAN. A rede wi-fi foi disponibilizada usando um roteador TP-Link Archer C25 (802.11ac). Em todos os experimentos, o MacBook atuou como o servidor de aplicação e sinalização, o qual foi implementado em JavaScript com Node.js, usando os recursos inclusos na Livvclib.

5.4.1 Experimentos com Clusters Gráficos

Nos experimentos com CGs, o nó mestre (Acer Aspire) enviou objetos JavaScript simples com os seguintes atributos para todos os nós escravos (todos eles abertos no Windows 10 PC): *idx* (índice do dado sendo enviado) e *data* (uma *string* com 100 caracteres, incluída unicamente para dar um "peso" ao objeto). Ao receber o objeto, cada nó escravo armazenava a hora atual usando a função `window.performance.now()`, que calcula a quantidade de milissegundos corrida desde a abertura da página web.

Em cada experimento, o nó mestre enviou consecutivamente 51 objetos de dados aos escravos. Depois disso, cada nó escravo calculou os 50 intervalos de tempo decorridos entre cada

objeto recebido e os enviou para o nó mestre, que então calculou a média e o desvio padrão dos intervalos. Escolheu-se calcular 50 intervalos porque foi o suficiente para permitir que algumas tendências aparecessem nos resultados e também para calcular valores mais precisos.

Metade dos experimentos foram realizados com os objetos de dados sendo enviados um imediatamente após o outro, enquanto a outra metade esperou 17 milissegundos antes de enviar cada objeto (neste caso, 17ms foram subtraídos de cada intervalo para contabilizar o atraso). A intenção por trás do atraso de 17ms era imitar uma aplicação sendo executada a ∼60fps, onde a renderização de cada quadro envolveria o nó mestre enviando alguns dados para os escravos ($1000/60 \approx 17$).

Três quantidades de nós escravos foram usadas para os experimentos: 1 escravo (que deveria fornecer os melhores resultados), 6 escravos (a quantidade de nós no MiniCAVE) e 30 escravos (um número relativamente alto de nós para um sistema de multi-projeção, mas não tanto para *wall displays*). Metade dos experimentos também envolveu nós trocando dados por WebRTC, enquanto a outra metade trocava dados por meio do servidor de aplicação e sinalização.

A Tabela 3 mostra os resultados dos experimentos com CGs. A coluna "Vazão" indica a quantidade de objetos que podem ser enviados do nó mestre para os escravos usando a Livvclib e considerando o intervalo de tempo médio entre um objeto e outro; essa coluna inclui o atraso entre cada objeto de dados enviado.

Tabela 3 – Intervalos de tempo entre o nó mestre enviar objetos JavaScript para os escravos e eles os receberem.

Nº de Escravos	**Atraso Entre Objetos**	**WebRTC**			**Servidor**		
		Média	***Desv. P.***	***Vazão***	***Média***	***Desv. P.***	***Vazão***
1	0ms	12.17ms	4.46	82.17	12.48ms	4.34	80.13
	17ms	10.49ms	4.64	36.38	11.45ms	4.49	35.15
6	0ms	12.49ms	4.4	80.06	13.09ms	4.41	76.39
	17ms	11.28ms	4.89	35.36	12.38ms	3.95	34.04
30	0ms	13.30ms	4.62	75.19	14.65ms	3.70	68.26
	17ms	12.71ms	4.30	33.66	14.23ms	1.40	32.02

Fonte – Produzida pelo autor.

Observa-se que os resultados são razoavelmente semelhantes em todas as condições experimentais, embora o intervalo médio aumente à medida que o número de escravos aumenta. Como esperado, os intervalos médios para os experimentos usando WebRTC foram ligeiramente melhores do que aqueles transferindo dados por meio do servidor de aplicação e sinalização.

5.4.2 Experimentos do Dispositivos de Entrada

Os experimentos com dispositivos de entrada foram similares àqueles com CGs. No lugar do nó mestre enviando o objeto JavaScript, era o servidor de sinalização e aplicação que enviava o mesmo objeto de antes, embora como se fosse um dado capturado de um dispositivo de entrada. A razão para enviar o mesmo objeto JavaScript de antes, em vez dos dados verdadeiros do dispositivo, foi simular um cenário mais neutro em relação ao tamanho dos dados enviados aos

nós.

Foi criada uma sala virtual para os nós (somente a sala, sem o CG; todos os nós abertos no desktop Windows 10), que solicitou a conexão com um mouse conectado ao desktop Windows 10 (onde era executado o servidor VRPN) e que se cadastrou para receber dados de seu evento "change". Sempre que o mouse era movido, o VRPN o capturava e enviava a atualização para o servidor de aplicação e sinalização (MacBook), que, por sua vez, enviava o objeto JavaScript para os nós (em vez dos dados do dispositivo). Ao receber o objeto JavaScript do servidor, cada nó escravo armazenava a hora atual da mesma maneira que em experimentos anteriores.

Em cada experimento, o servidor de aplicação e sinalização recebeu 51 atualizações do VRPN e, consequentemente, enviou 51 objetos JavaScript para os nós. Depois disso, os nós calcularam os intervalos entre cada objeto de dados recebido e os enviaram para o servidor, que calculou a média e o desvio padrão dos intervalos. Como antes, os experimentos foram realizados com três quantidades de nós (1, 6 e 30)

A Tabela 4 mostra os resultados das experiências. Como antes, os resultados são razoavelmente semelhantes em todas as condições experimentais, embora um número maior de nós forneça intervalos médios mais longos.

Tabela 4 – Intervalos entre o VRPN capturar o movimento do mouse e os nós o receberem do servidor de aplicação e sinalização.

Nº de Nós	**Média**	**Desv. P.**	**Vazão**
1	8.00ms	3.46	125.00
6	9.10ms	5.15	109.89
30	9.42ms	5.91	106.16

Fonte – Produzida pelo autor.

Para experimentos com foco exclusivamente no desempenho da VRPN, consultar o estudo de Taylor-II et al. (2001).

5.4.3 Discussão

Idealmente, os experimentos mediriam o tempo decorrido entre o nó mestre/servidor enviar o objeto JavScript e os nós (escravos) que o receberem. Um problema em experimentos de medição de tempo como este é a sincronização dos relógios que medem os tempos de início e término; neste estudo, como as operações são executadas em milissegundos ou até menos, até mesmo diferenças mínimas nesses relógios afetariam significativamente a medição.

O problema foi evitado obtendo os horários de início e término no mesmo computador. Isso tornou necessário que as métricas dos experimentos fossem os intervalos de tempo transcorridos nos nós (escravos) entre cada objeto recebido do nó mestre/servidor. O momento em que um nó (escravo) recebe um objeto JavaScript é o momento de início de um novo intervalo e o momento de término do intervalo iniciado quando objeto anterior foi recebido.

Os resultados dos experimentos se mostram promissores em relação ao desempenho da Livvclib, uma vez que até mesmo os experimentos com as piores condições (i.e. aqueles com maiores quantidades de nós e maiores taxas de transmissão por unidade de tempo) mostraram resultados com relativas baixa latência e alta vazão. Para as experiências com CGs, a pior condição (30 escravos, sem atraso, transferência pelo servidor) permite, em média, 68 trocas de dados por segundo, o que significa que as aplicações podem ter pelo menos uma troca de dados por renderização de quadro (supondo que se almeje 60fps). Para os experimentos com dispositivos de entrada, as piores condições (30 nós) podem receber, em média, 106 atualizações por segundo dos dispositivos de entrada, permitindo que os aplicativos sejam muito responsivos e sensíveis às interações por meio de dispositivos de entrada, além de mouse e teclado.

Ambos os conjuntos de experimentos indicam escalabilidade positiva da Livvclib, uma vez que os resultados médios permanecem amplamente semelhantes entre os experimentos, apesar de alguns deles envolverem 6 ou 30 vezes mais nós (escravos) do que outros; os desvios padrão também são razoavelmente baixos, indicando que não houve perda substancial de desempenho, mesmo nos piores casos.

Em relação aos experimentos com CGs, embora os resultados obtidos indiquem um desempenho razoável, são consideravelmente inferiores aos obtidos por Guimarães et al. (2018) sob condições supostamente similares. Acredita-se que isso seja devido a diferenças nos seguintes fatores:

- Equipamento usado (3 PCs Windows vs 2 máquinas virtuais GNU/Linux em um único PC GNU/Linux);
- Dados trocados (objeto JavaScript com número e 100 caracteres, mais cabeçalhos extras da Livvclib, vs apenas 100 caracteres); e
- Rede (LAN isolada vs emulador de rede).

Também deve ser notado que esses experimentos foram conduzidos isoladamente e que todos os computadores usados neles eram parte da mesma LAN. Em um cenário real, é possível que o servidor de aplicação e sinalização atenda a muitos CGs simultaneamente e que os nós e o servidor façam parte de redes diferentes; nesses casos, o desempenho de recursos que dependem do servidor (`SignalingServerNodeConnection` e tudo relacionado a dispositivos de entrada) provavelmente será pior.

5.5 Considerações Finais

A Livvclib é uma implementação do design de solução para aplicações VI3DI web apresentado nesta dissertação e abordado no Capítulo 4 Design de Solução para Aplicações Web de Visualização da Informação 3D Interativa. Ela implementa todas as funcionalidades elencadas e propostas, embora não exatamente como descritas. As descrições das funcionalidades na Subseção 4.2 Composição tem o propósito principal de apresentar seus desafios e requisitos, mas não consideram certas condições de implementações práticas, assim, as implementações de

algumas dessas funcionalidades na Livvclib são ligeiramente diferentes do originalmente descrito, para que fosse possível adequá-las as condições práticas do desenvolvimento.

Também foi dado um passo adiante, com o desenvolvimento de componentes adicionais da Livvclib, voltados a apoiar o desenvolvimento do servidores de aplicação e sinalização para aplicações VI3DI web utilizando a Livvclib. Esses componentes são específicos para uma tecnologia, Node.js, enquanto os servidores podem ser desenvolvidos usando uma infinidade delas; todavia, a arquitetura do lado cliente da Livvclib permite abstrair o funcionamento do servidor de aplicação e sinalização com o que se julga ser facilidade.

É importante notar a quantidade de elementos necessários para se trabalhar com a Livvclib. Em primeiro lugar, é necessário instalar o Node.js; mesmo que o desenvolvedor decida reimplementar o servidor de aplicação e sinalização, essa instalação trará junto o NPM, que é particularmente útil no desenvolvimento de aplicações web. Com isso, são necessárias quatro bibliotecas JavaScript (Three.js, WatchJS, Socket.IO e nbind), as quais podem ser instaladas através de um único comando, com o NPM. Adicionalmente, para o uso de dispositivos de interação, é requisito obter o código-fonte do VRPN e compilar ele e os componentes C++ da Livvclib, o que, por sua vez, requer um compilador C++ adequado para o SO.

Tendo em vista essa questão, a Livvclib é distribuída junto de suas dependências JavaScript e dos binários compilados[7] do VRPN e de seus componentes C++, a fim de diminuir as barreiras para seu uso.

[7] Para Windows 7/8/10, apenas.

6 Estudo de Caso

Este capítulo apresenta um estudo de caso onde o design de solução para aplicações VI3DI web apresentado foi utilizado (por meio de sua implementação, Livvclib) para o desenvolvimento de uma aplicação web de visualização molecular. Começa-se apresentando a aplicação desenvolvida; em seguida é abordado um experimento realizado com a aplicação; por fim, os resultados do experimento são apresentados e discutidos.

6.1 Visualização Molecular Distribuída, Imersiva e Multi-Plataforma

O processo de aprendizagem é uma questão consideravelmente complexa, exigindo esforço e dedicação do aluno. Assim, existe a necessidade de recursos e métodos educacionais que, de alguma forma, mantenham o aluno motivado ao longo do processo (VIRVOU; KATSIONIS; MANOS, 2005). Um fator particular que aumenta a motivação (e, portanto, a chance de assimilar o conteúdo estudado) é a paridade entre o estilo de aprendizagem do aluno e o modo como a informação é recebida (MIKROPOULOS; NATSIS, 2011).

Entre os estilos de aprendizagem natural, o visual-espacial é o que mais se destaca (GARDNER, 1983); além disso, entender as relações espaciais é um requisito em muitas áreas da ciência, como a química, onde a visualização é importante para entender os processos químicos (DAVIES et al., 2005) e a forma tridimensional das moléculas (MERCHANT et al., 2013). A falta de instrução espacial pode tornar desafiadora a aprendizagem de alguns conceitos (MERCHANT et al., 2013).

Espera-se, então, que os educadores busquem novos recursos educacionais que satisfaçam essas necessidades. Uma alternativa pode ser encontrada em softwares educacionais que utilizam ambientes virtuais desenvolvidos com a tecnologia RV, que é altamente motivadora para os alunos, captando e mantendo sua atenção e incentivando a participação ativa ao invés da passividade (PANTELIDIS, 2009).

De fato, desde a introdução da capacidade de criar ambientes virtuais usando RV, algumas de suas aplicações mais importantes estão focadas no aprendizado e treinamento (CASU et al., 2015). Um exemplo é a modelagem química, em que a aquisição de conhecimento sobre os modelos é facilitada pelo uso de modelos 3D e técnicas de RV (DIAS, 2011).

Como parte de um esforço contínuo no LIV para utilizar o MiniCAVE lá montado nas áreas de educação e colaboração e comunicação, e considerando o uso de RV na visualização científica de moléculas e na educação, foi desenvolvida a aplicação *Distributed Immersive Multi-platform Molecular visualization* (Dimmol; do inglês, "visualização Molecular Distribuída, Imersiva e Multi-plataforma")[1], a fim de prover novas maneiras e técnicas pelas quais educadores (e possivelmente pesquisadores) em áreas relacionadas à química visualizam estruturas moleculares e reações químicas com seus alunos e parceiros, e também permitir novas formas de coordenação

[1] Código-fonte Dimmol: <https://github.com/LuizSSB/dimmol>.

e colaboração para reduzir ou até mesmo eliminar barreiras na compreensão (BAGLIE; NETO; BREGA, 2016; BAGLIE et al., 2017).

Dimmol é um *fork*[2] do UnityMol (LV et al., 2013), uma aplicação para visualização de estruturas moleculares, tais como moléculas, cristais e proteínas, implementado no motor de jogo Unity (Unity Technologies, 2017). Em comparação com a versão 0.9.3 do UnityMol, o Dimmol apresenta as seguinte novas funcionalidades:

- Possibilidade de distribuição de visualização por vários dispositivos (o que permite a multi-projeção) através de uma arquitetura mestre-escravo;
- Exibição em HMDs compatíveis com a SDK do Google VR (Google, inc., 2018);
- Execução em dispositivos móveis iOS e Android; e
- Exibição de trajetórias de estruturas moleculares.

A Figura 1, na Introdução, mostra o Dimmol em execução, distribuindo uma mesma visualização pelo MiniCAVE e em um iPad Mini.

Com a apresentação, no Capítulo 4 Design de Solução para Aplicações Web de Visualização da Informação 3D Interativa, (e subsequente implementação, no Capítulo 5 *Framework* para Aplicações Web de Visualização da Informação 3D Interativa) do design de solução para aplicações VI3DI web e as vantagens oferecidas pela web (principalmente, natureza multi-plataforma das aplicações, renderização de gráficos 3D acelerados por hardware, comunicação em tempo-real), tornou-se mais adequado dar continuidade ao projeto Dimmol na forma de uma aplicação web, ao invés de uma aplicação compilada (como assim o era), ainda que multi-plataforma.

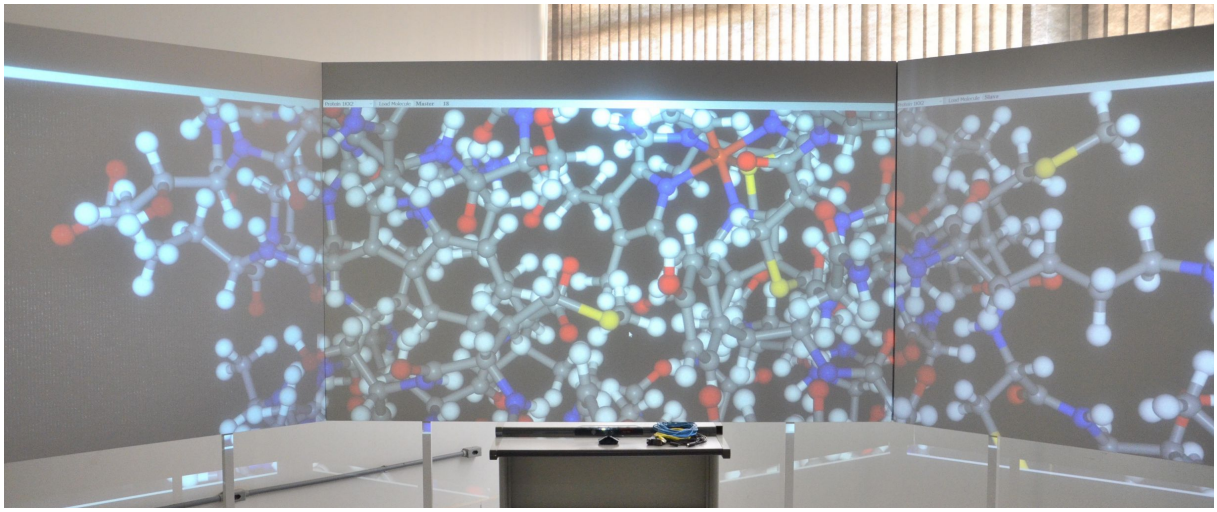
Foi, então, desenvolvido o Dimmol.js[3], baseado na aplicação NGL Viewer[4] (ROSE; HILDEBRAND, 2015; ROSE et al., 2016), para visualização molecular na web. Assim como o original, Dimmol.js permite a distribuição da visualização, a execução em dispositivos móveis (na verdade, em qualquer dispositivo com um navegador web com suporte a WebGL e WebSockets) e a exibição de trajetórias. Foi adicionado suporte a dispositivos de interação diversificados; mais especificamente, controles de video game no padrão XInput[5] e Nintendo Wii Remote. Por ser um *fork* do NGL Viewer, que oferece mais recursos que o UnityMol, o Dimmol.js naturalmente oferece mais recursos que o original, tais como a possibilidade de exibir mais de uma estrutura e representação de uma vez e filtragem dos átomos visíveis em cada representação para uma dada estrutura.

A Figura 22 mostra o Dimmol.js sendo executado no MiniCAVE. No caso, os navegadores web são exibidos em tela cheia, fazendo com que escondam suas barras de menu, de ferramentas e de endereço.

Graças ao NGL, no Dimmol.js, as estruturas moleculares podem ser carregadas de arquivos em uma variedade de formatos, incluindo PDF e MMTF, ou carregadas diretamente a

[2] Desenvolvimento separado e independente de um software a partir do código-fonte de outro já existente.
[3] Código-fonte Dimmol.js: <https://bitbucket.org/livunespbauru/dimmol.js>.
[4] NGL Viewer: <http://nglviewer.org/ngl>.
[5] XInput: <https://msdn.microsoft.com/en-us/library/windows/desktop/ee417001.aspx>.

Figura 22 – Dimmol.js em execução no MiniCAVE.



Fonte - Produzida pelo autor.

partir do *Worldwide Protein Data Bank*[6] (do inglês, "Banco de Dados de Proteínas Mundial"); mais de uma estrutura pode ser carregada por vez. Uma vez que uma estrutura seja carregada, é possível adicionar e configurar várias representações para ela (tais como bola-e-vareta, linhas ou alcaçuz), bem como trajetórias para a estrutura; trajetórias correspondem a arquivos que indicam como cada átomo varia ao longo de um conjunto de estados da estrutura. O usuário também pode manipular as posição e rotação da visualização toda, usando o mouse, ou as posição e orientação individuais de cada estrutura carregada, através de controles especiais.

O Dimmol.js oferece um menu "Dimmol", a partir do qual o usuário pode selecionar as opções relacionadas a distribuição da imagem e dispositivos de interação:

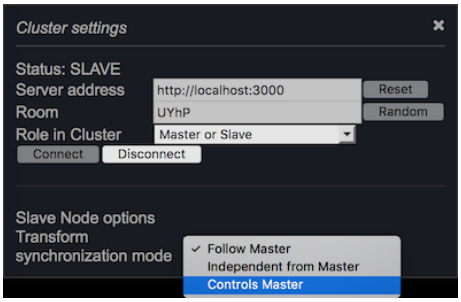
- *Cluster Options* (do inglês, "Opções de Cluster"; Figura 23a): opções relacionadas a conexão do nó a um CG e qual seu papel no CG. Para nós escravos, permite configurar o envio de dados para o nó mestre (que, então, os repassaria para os outros nós), possibilitando, assim, um cluster descentralizado;
- *Mobile Controls* (do inglês, "Controles Móveis"; Figura 23b): atalhos para controles de manipulação de posições e rotações de cada estrutura (para facilitar uso com dispositivos móveis);
- *Screen Options* (do inglês, "Opções de Tela"; Figura 23c): opções para configurar a matriz de projeção generalizada da tela para sistemas de multi-projeção; e
- *Devices* (do inglês, "Dispositivos"; Figura 23d): opções para conexão com dispositivos de interação (controle XInput ou Wii Remote).

A Figura 24 mostra como os dispositivos de interação suportados pelo Dimmol.js podem ser utilizados. No caso do Wii Remote, enquanto o botão A estiver pressionado, a visualização se orientará de acordo com orientação do Wii Remote no mundo físico.

[6] Worldwide Protein Data Bank: <https://www.wwpdb.org>.

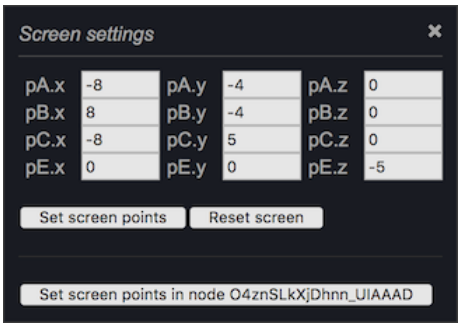
Figura 23 – Menus especializados do Dimmol.js. As opções em cada um são auto-explicativas.

(a) Opções relacionadas a CG (em escravo)

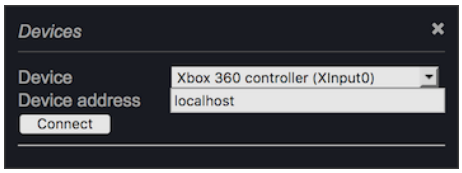


(b) Controles para dispositivos móveis (em escravo)

(c) Opções relacionadas a matriz de projeção (em mestre)



(d) Opções relacionadas a dispositivos de interação (em mestre)



Fonte - Produzida pelo autor.

Figura 24 – Funções dos botões dos dispositivos de interação suportados pelo Dimmol.js.

(a) Wii Remote

(b) Controle XInput/Xbox 360

Fonte - Produzida pelo autor.

O trabalho de adaptação somou, ao todo, mais de 3064 linhas de código; esse montante inclui declarações de tipos/variáveis/métodos/funções, chamadas de instruções, comentários e linhas vazias. Os recursos fornecidos pela Livvclib permitiram que as várias funcionalidades relativas ao design de solução apresentado fossem simplesmente utilizadas, ao invés de desenvolvidas, assim, a maior parte "útil" do código (i.e. declarações e chamadas) diz respeito a configurações do CG e dos dispositivos de interação e adaptação e controle do NGL Viewer para responder a dados do nó mestre e dos dispositivos de interação.

6.2 Experimento

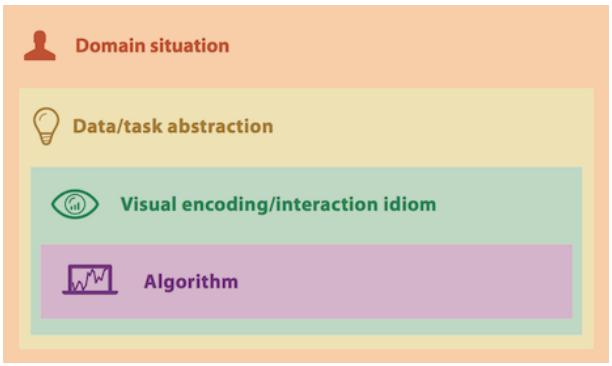
Foi elaborado e realizado um experimento envolvendo o uso do Dimmol.js por usuários alvo, estudantes de áreas relacionadas a Física e Química que possuem necessidade de algo do gênero, e por pessoas comuns, que não tem, obrigatoriamente, essa necessidade.

O objetivo do experimento constitui tanto a avaliação qualitativa do Dimmol.js quanto do desempenho do design de solução apresentado e de sua implementação.

6.2.1 Metodologia

De acordo com Munzner (2014), a validação de um sistema de visualização é feita em quatro níveis (Figura 25):

Figura 25 – Quatro níveis aninhados da validação de um sistema de visualização.



Fonte - Munzner (2014).

1. Situação do Domínio: identifica e entende a situação atual dos usuários e as suas questões que devem ser atendidas;

2. Abstração de Dados/Tarefas: abstrai as tarefas do domínio em algo mais independente, identificando quais dados são úteis na visualização e como eles devem ser transformados;

3. Codificação Visual e Interação: identifica as representações visuais (idiomas) e modelos de interação que melhor permitem cumprimento das tarefas; e

4. Algoritmo: procedimentos computacionais que transformam os dados, realizam as visualizações e processam as interações do usuário.

Com o Dimmol.js, foram avaliados dois pontos principais: a interação com a visualização molecular através de diferentes dispositivos de entrada e a responsividade dessa interação, bem como a sincronização de sua resposta visual, por um CG. Sendo assim, pode-se dizer que esta é uma avaliação orientada a técnica, onde se trabalha em um dos dois níveis inferiores, design de idioma ou de algoritmo, em que o objetivo é elaborar novos idiomas que melhor apoiem abstrações existentes ou novos algoritmos que melhor suportem idiomas existentes (MUNZNER, 2014). Neste estilo de trabalho, a avaliação começa diretamente em um dos dois níveis mais baixos, focando-se nele. Em todo caso, uma avaliação das visualizações oferecidas e da experiência de visualização molecular com um sistema de multi-projeção também é incorporada ao experimento.

Com base, então, em avaliar as camadas de Codificação Visual e Interação e Algoritmo, foram elaboradas três tarefas de visualização molecular, onde, dados um cristal de sal e um dispositivo de interação, o objetivo é manipular sua visualização a fim de encontrar certas regiões de interesse. Cada tarefa estabelece o tamanho do cristal, as regiões de interesse e o tipo de dispositivo de interação. Os dispositivos requeridos foram:

- Mouse: dispositivo suportado por padrão por navegadores web e pelo NGL Viewer, usado para obter dados de controle;
- Controle de video game: dispositivo acessado por meio do VRPN, usado para avaliar o suporte a dispositivos de interação diversificados; e
- Tablet: pode ser usado para manipular a visualização através do suporte a clusters descentralizados, uma vez que o tablet se conecta ao CG como um nó escravo. Usado para avaliar o suporte a CGs descentralizados na Livvclib.

O Apêndice D Roteiro do Experimento apresenta o roteiro do experimento realizado, para cada participante, incluindo as falas e ações projetadas para o coordenador do experimento e as reações esperadas dos participantes, bem como as tarefas específicas que deveriam ser realizadas por cada participante do experimento. Com exceção das tarefas, o roteiro não constitui uma representação perfeitamente exata da condução do experimento com cada participante, todavia, serve como forte guia para tanto.

Em suma, cada participante do experimento foi instruído sobre como usar mouse, controle XInput (DualShock 3) e tablet (iPad Mini) para manipular a visualização das estruturas moleculares no Dimmol.js; após isso, o participante realizou as três tarefas de visualização; para finalizar, foi dado um tempo de uso livre da aplicação para o participante, após o qual ele respondeu um questionário sobre a experiência.

6.2.2 Questionário

O questionário respondido pelos participantes do experimento é detalhado no Apêndice E Questionário. Ele é formado por 20 questões que visaram conhecer alguns dados sobre o usuário e avaliar sua experiência com o Dimmol.js na realização das três tarefas propostas. A maior parte das questões são, na verdade, afirmações, as quais o participante deve responder usando a escala

Likert, i.e. um valor de 1 a 5, onde 1 representa que ele discorda totalmente da afirmação e 5, que ele concorda totalmente.

As questões/afirmações foram elaboradas usando três critérios diferentes:

- Os níveis aninhados de avaliação propostos por Munzner (2014);
- As definições apresentadas pela ISO 9241 (ISO, 2001):
  - *Usuário*: pessoa que usa e interage com o produto;
  - *Usabilidade*: capacidade de um produto ser usado por usuários específicos para atingir objetivos específicos com eficácia, eficiência e satisfação em um contexto de uso específico;
  - *Eficiência*: precisão e completude com que os usuários atingem seus objetivos, em relação à quantidade de recursos gastos; e
  - *Satisfação*: conforto e aceitação do produto, medidos por métodos subjetivos e/ou objetivos.
- O questionário *Virtual Experience Test* (VET; do inglês, "Teste de Experiência Virtual") (CHERTOFF; GOLDIEZ; LAVIOLA, 2010), usado para avaliar experiências em um ambiente virtual, com base nas cinco dimensões do design experiencial: sensorial, cognitiva, afetiva, ativa e relacional.

A Tabela 5 lista as questões elaboradas/incluídas, junto aos níveis de validação aos quais se referem, as facetas da experiência que abordam e indicação de se foram elaboradas ou obtidas do VET. Somente um subconjunto das questões propostas pelo VET foram inclusas, a fim de evitar que o questionário ficasse demasiadamente longo e devido a algumas delas não serem adequadas para a experiência específica proporcionada pelo experimento.

6.2.3 Resultados e Discussão

O desenvolvimento do Dimmol.js permitiu validar a usabilidade do design de solução para aplicações VI3DI web apresentado, e de sua implementação, numa aplicação real voltada a Visualização Científica (ramo da VI que trata de informações científicas; no caso, estruturas moleculares), que tem parte de seu foco dedicado ao ensino e comunicação e colaboração, onde alunos, professores e pesquisadores podem visualizar e interagir com estruturas moleculares de maneira colaborativa, através de uma variedade de dispositivos.

O experimento foi conduzido no LIV, durante dois dias. No primeiro dia, participaram 3 alunos de mestrado em Física, e, no segundo dia, mais 8 alunos de mestrado em Ciência da Computação, totalizando, assim, 11 participantes, dentre os quais 1 era do sexo feminino e 10, do masculino. Os participantes tinham entre 25 e 31 anos de idade.

Durante o preparo do experimento, percebeu-se que alguns usuários, ao usarem o tablet, preferiam visualizar as estruturas moleculares com o tablet, para, então, confirmar a visualização no MiniCAVE. Com isso, a fim de evitar cansaço visual nos participantes (devido a troca de

Tabela 5 – Questões/afirmações para avaliar a experiência do participante com o Dimmol.js.

#	Questão	Níveis	Facetas	No VET?
1	Eu senti facilidade para manipular (arrastar, girar, zoom) a estrutura molecular na aplicação com o mouse.	Codificação	Usabilidade	Não
2	Eu senti que a manipulação com o mouse tinha resposta imediata e precisa.	Algoritmo	Usabilidade	Não
3	Eu senti facilidade para manipular a estrutura molecular na aplicação com o controle de video game.	Codificação	Usabilidade	Não
4	Eu senti que a manipulação com o controle de video game tinha resposta imediata e precisa.	Algoritmo	Usabilidade	Não
5	Eu senti facilidade para manipular a estrutura molecular na aplicação com um tablet.	Codificação	Usabilidade	Não
6	Eu senti que a manipulação com o tablet tinha resposta imediata e precisa.	Algoritmo	Usabilidade	Não
7	Qual maneira você julga melhor para manipular a estrutura molecular?	Codificação	Usabilidade	Não
8	Eu senti que as estruturas moleculares eram manipuladas e animadas na aplicação com rapidez e suavidade.	Algoritmo	Usabilidade	Não
9	Eu senti que as estruturas moleculares se movimentavam sem atrasos ou inconsistências entre as três as telas do MiniCAVE.	Algoritmo	Uabilidade	Não
10	Eu considero que o MiniCAVE contribuiu muito para o entendimento de estruturas moleculares.	Codificação	Eficiência	Não
11	Eu me senti imerso no ambiente virtual com as estruturas moleculares.	Codificação	Usabilidade	Não
12	Eu estaria disposto a visualizar mais estruturas moleculares usando a aplicação e o MiniCAVE.	Domínio	Satisfação	Não
13	Eu vejo benefício na utilização da aplicação e do MiniCAVE em tópicos/disciplinas que demandam visualização molecular.	Codificação	Usabilidade	Não
14	Eu considero o equipamento de exibição como de alta qualidade.			Sim
15	Eu considero o conteúdo visual do ambiente virtual como de alta qualidade.			Sim
16	Eu acredito que o ambiente virtual consegue suportar múltiplos usuários humanos ao mesmo tempo.			Sim
17	Eu considero interessantes as tarefas que consegui fazer no ambiente virtual.			Sim
18	Eu senti que o ambiente virtual me permitiu completar minhas tarefas de várias maneiras diferentes.			Sim
19	Eu senti que era capaz de reusar continuamente as técnicas aprendidas em tarefas anteriores para completar novas tarefas.			Sim
20	Espaço para comentários, observações, reclamações, sugestões, etc.			Não

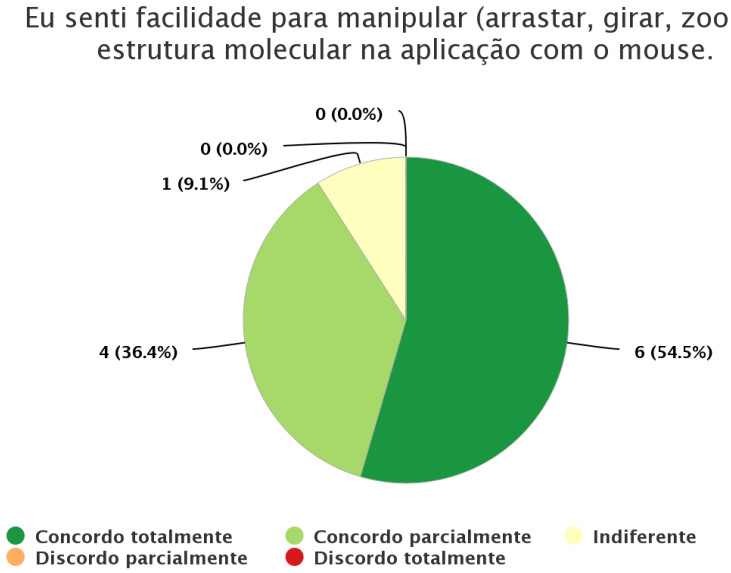
Fonte – Produzida pelo autor.

visualização estereoscópia, no MiniCAVE, para monoscópica, no tablet, e vice-versa), foi escolhido não utilizar a estereoscopia do MiniCAVE. Não houve maiores incidentes durante a condução do experimento.

As respostas dadas ao questionário foram compiladas numa planilha eletrônica[7] e são abordadas a seguir.

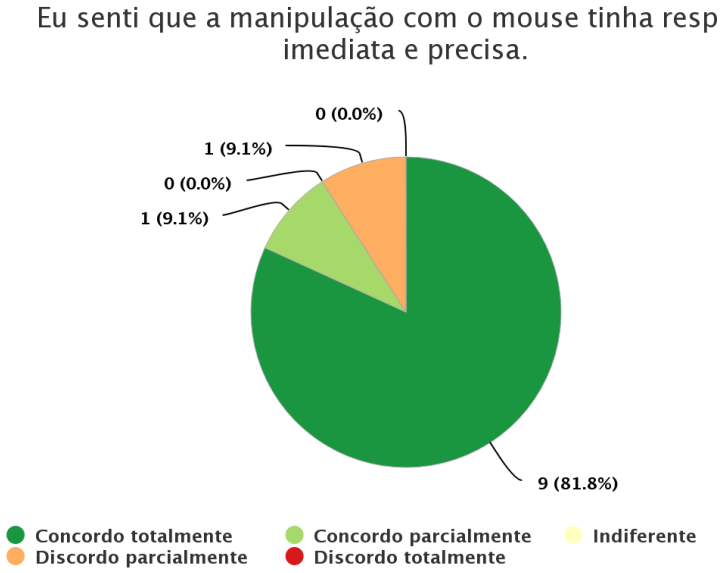
Em relação aos dispositivos de interação, os participantes, em geral, se mostraram mais receptivos ao mouse (Figuras 26-27 e 32) do que ao controle de video game (Figuras 28-29) ou ao tablet (Figuras 30-31). Segundo uma observação deixada no questionário por um dos participantes e comentários feitos por outro participante durante o experimento, as razões para tanto incluem:

Figura 26 – Respostas para a Questão 1.



Fonte - Produzida pelo autor.

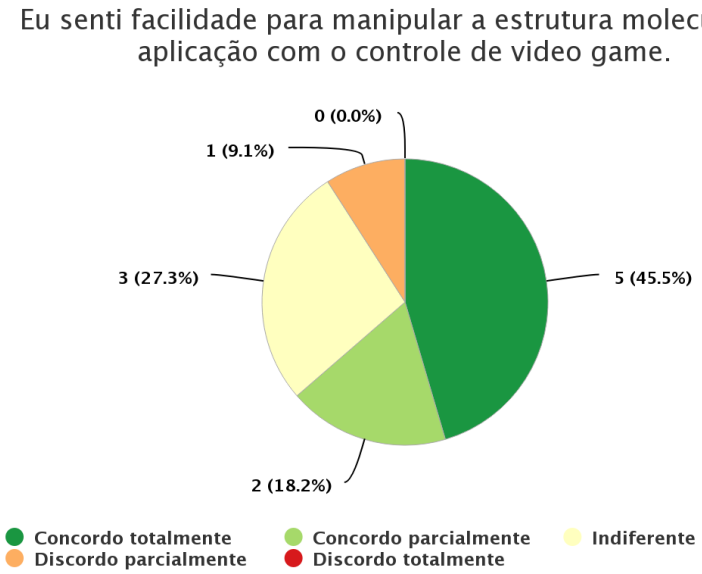
Figura 27 – Respostas para a Questão 2.



Fonte - Produzida pelo autor.

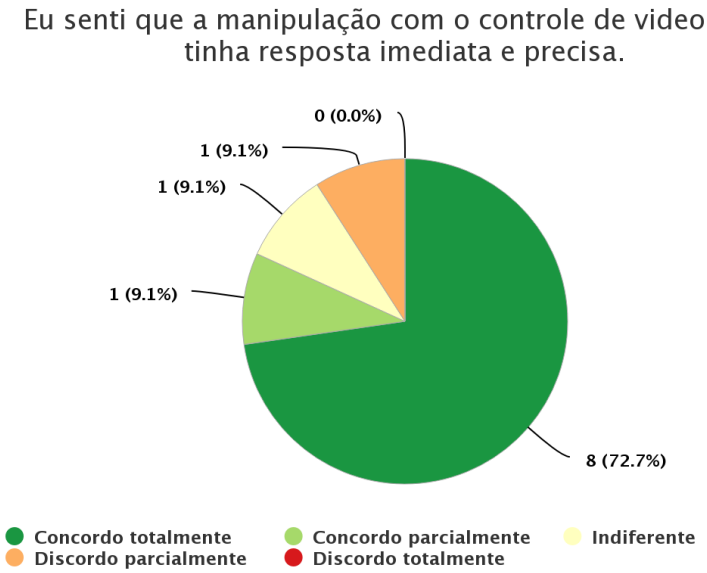
[7] Planilha eletrônica com respostas do questionário: <https://docs.google.com/spreadsheets/d/1MPzydKOajyuRMT7h7cBkarb4NrOq5hH-ngDzkBK0g9E>

Figura 28 – Respostas para a Questão 3.



Fonte - Produzida pelo autor.

Figura 29 – Respostas para a Questão 4.



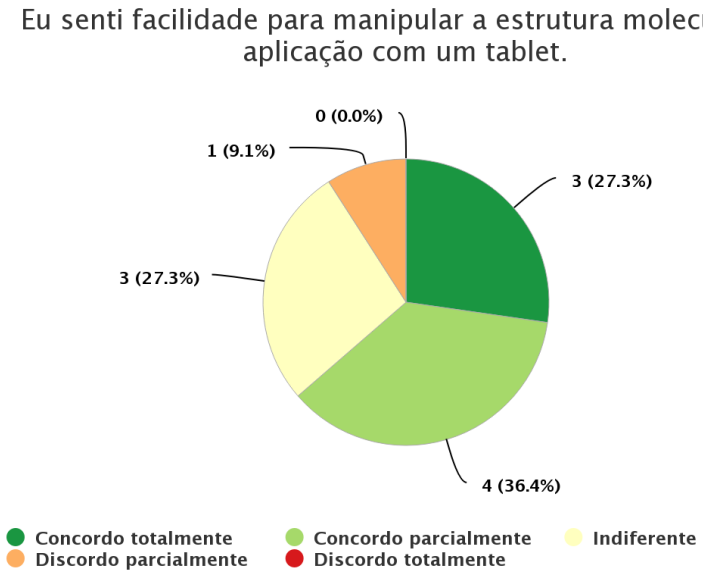
Fonte - Produzida pelo autor.

- Sensibilidade da manipulação (e.g., um curto e rápido deslize do dedo pela tela sensível a toque do tablet faz com que a estrutura molecular seja muito rotacionada no MiniCAVE). Isso pode ser tratado com relativa facilidade, alterando-se o trecho do código-fonte que determina o quanto a estrutura molecular gira, de acordo com o deslizar dos dedos sobre a tela sensível a toque do tablet ou o puxar do manete analógico esquerdo do controle de video game; e

- Familiaridade com a utilização do mouse.

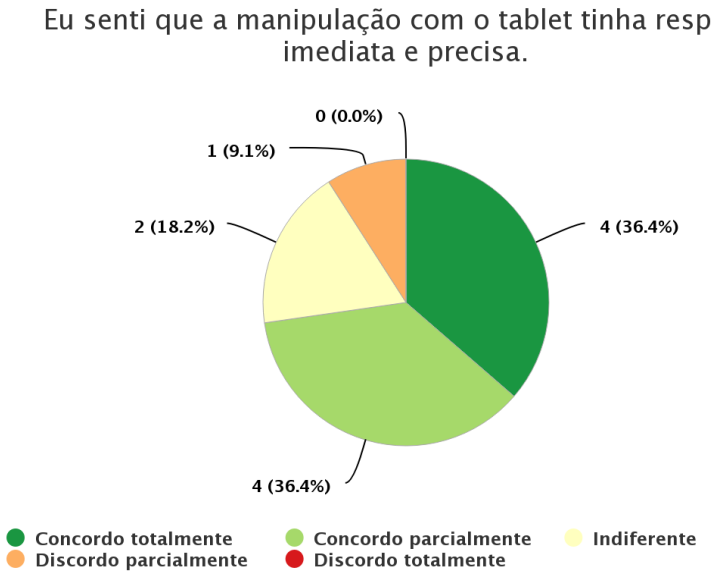
Em todo caso, as respostas as Questões 1-6 mostram que os dispositivos de interação

Figura 30 – Respostas para a Questão 5.



Fonte - Produzida pelo autor.

Figura 31 – Respostas para a Questão 6.



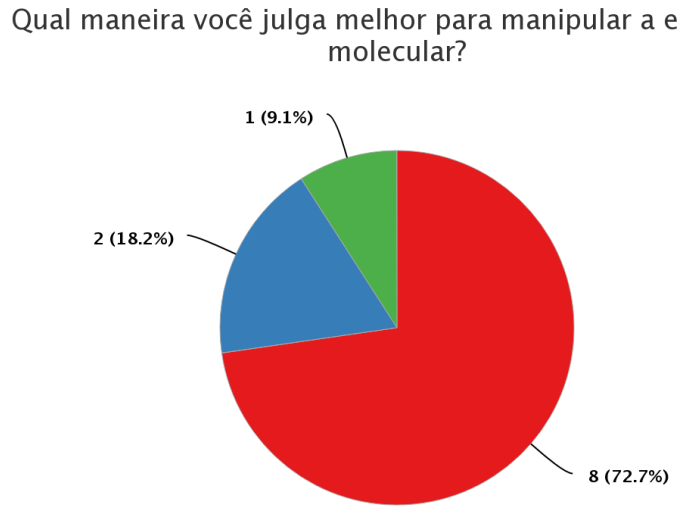
Fonte - Produzida pelo autor.

cumpriram seus papéis na realização das tarefas de maneira, em geral, pelo menos satisfatória, o que é importante para a avaliação do design de solução apresentado e da Livvclib, pois indica que seus desempenhos são adequados, no que trata os dispositivos de entrada e o CGs descentralizados.

As respostas as Questões 8 e 9 (Figura 33 e 33) indicam que o desempenho do suporte a CGs (e, consequentemente, da multi-projeção) também é adequado, dado que os usuários concordaram, em geral, com rapidez, suavidade da movimentação das estruturas moleculares pelas telas do MiniCAVE, bem como julgaram relativamente negligenciável a latência de sincronização entre as telas do MiniCAVE.

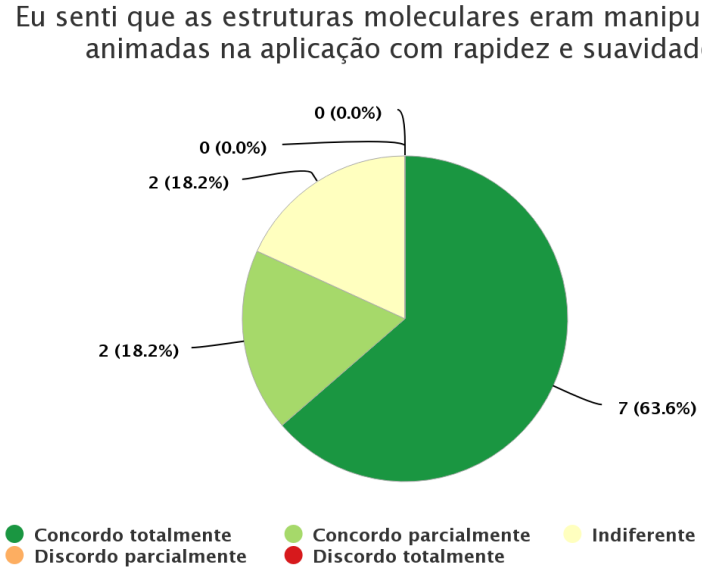
Nas respostas a Questão 10 (Figura 35), uma parcela substancial dos participantes se

Figura 32 – Respostas para a Questão 7.



Fonte - Produzida pelo autor.

Figura 33 – Respostas para a Questão 8.

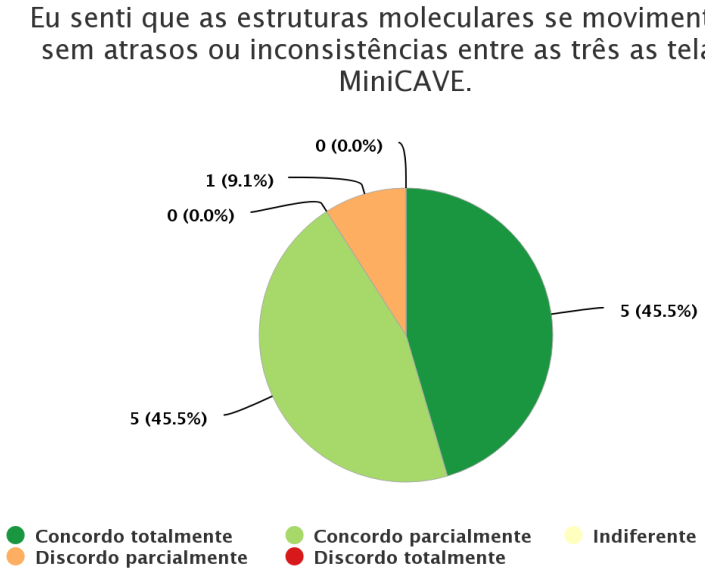


Fonte - Produzida pelo autor.

sentiu indiferente quanto a contribuição do MiniCAVE para a realização das tarefas e, de fato, durante a realização do experimento, percebeu-se que os participantes dificilmente giravam o pescoço para olhar para os lados, mantendo foco, principalmente, na tela central do MiniCAVE.

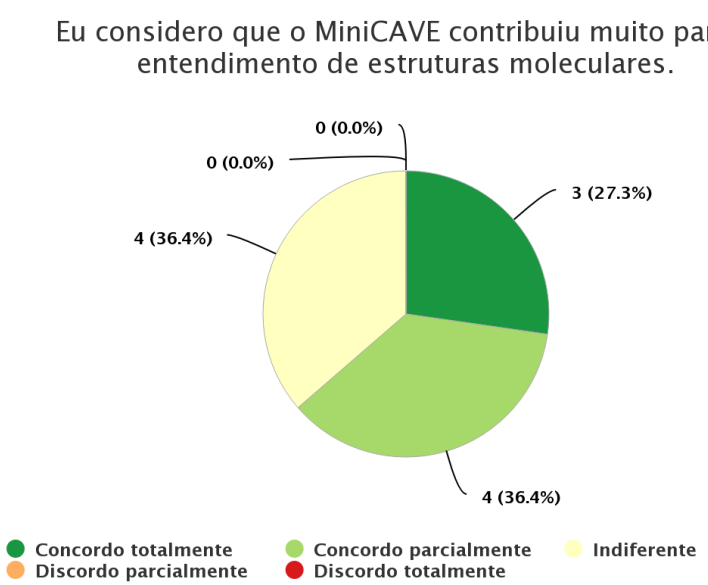
É interessante notar, no entanto, que a maioria dos participantes se se sentiu imersa com o MiniCAVE (Questão 11, Figura 36) e, também, que todos eles demonstraram interesse em visualizar mais estruturas moleculares utilizando o MiniCAVE (Questão 12, Figura 37) e concordaram com a existência de benefícios na utilização do MiniCAVE para este tipo de visualização (Questão 13, Figura 38), além de que a maioria o considerou, de uma maneira ou de outra, um aparato de exibição de alta qualidade (Questão 14, Figura 39). Isso sugere que, embora os participantes possam não ter usado todo potencial do MiniCAVE na execução das

Figura 34 – Respostas para a Questão 9.



Fonte - Produzida pelo autor.

Figura 35 – Respostas para a Questão 10.

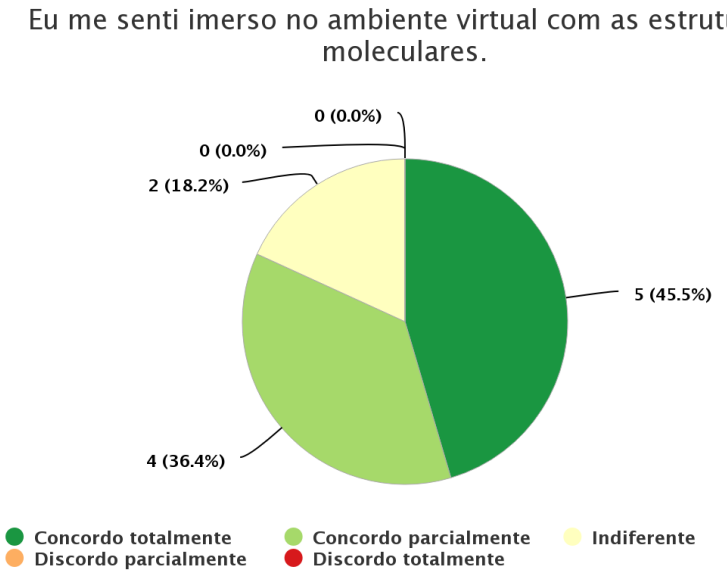


Fonte - Produzida pelo autor.

tarefas, eles tenham interesse na utilização deste aparato para visualização de estruturas mais complexas ou de outros tipos, uma vez, também, que a maioria concordou, de uma maneira ou de outra, que o conteúdo visualizado de alta qualidade (Questão 15, Figura 40).

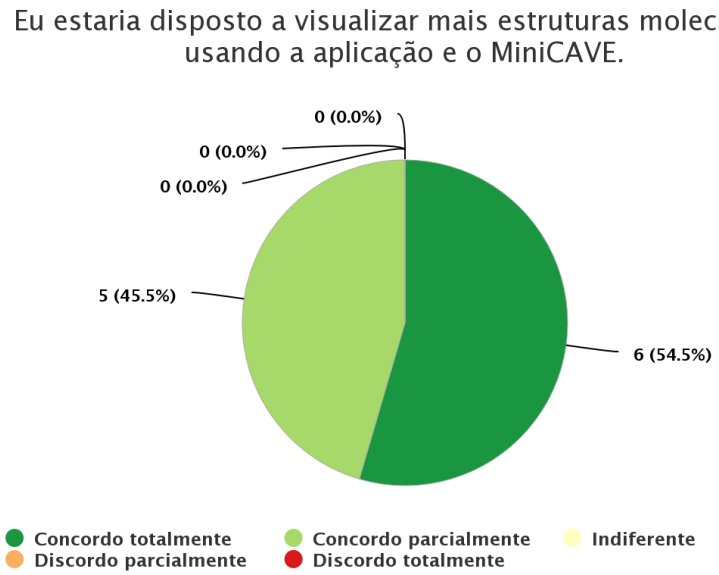
A Questão 16 (Figura 41), sobre o suporte a vários usuários simultaneamente no ambiente virtual, foi a que obteve a maior quantidade de participantes que discordaram ou se sentiram indiferentes, ainda que esse seja um diferencial de sistemas tipo-CAVE para aplicações RV. Nenhum comentário ou observação foi feito quanto a isso pelos participantes, o que sugere que o suporte a vários usuários simultâneos não é particularmente bem comunicado aos usuários do Dimmol.js. Embora exista a possibilidade de que esse suporte, na visão dos participantes, não

Figura 36 – Respostas para a Questão 11.



Fonte - Produzida pelo autor.

Figura 37 – Respostas para a Questão 12.



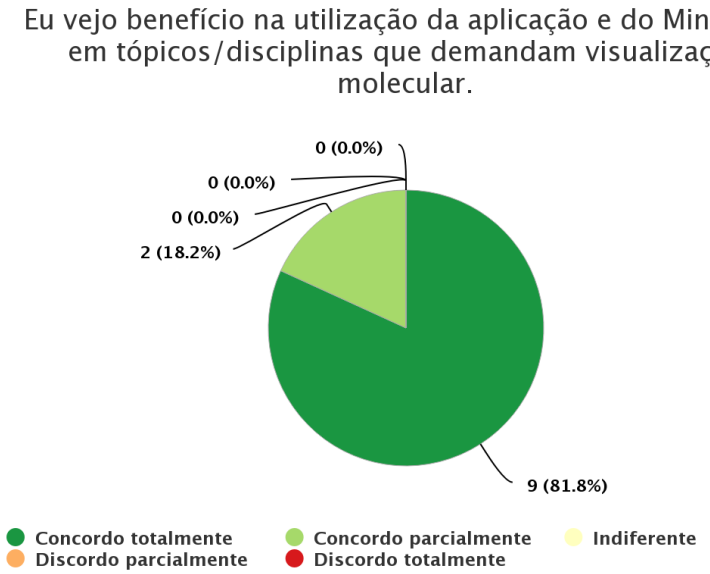
Fonte - Produzida pelo autor.

seja particularmente adequado para o Dimmol.js, nota-se que os três participantes estudantes de mestrado em Física (área que necessita de aplicações dessa variedade) responderam que concordam com a questão.

Os participantes também sentiram interesse nas tarefas realizadas no ambiente virtual (Questão 17, Figura 42), reforçando os pontos levantados por Pantelidis (2009) (ambientes virtuais da RV são altamente motivadores) e Zorzal et al. (2007) (RV desperta interesse dos usuários na VI 3D pelas formas de interações possíveis, aumentando o envolvimento).

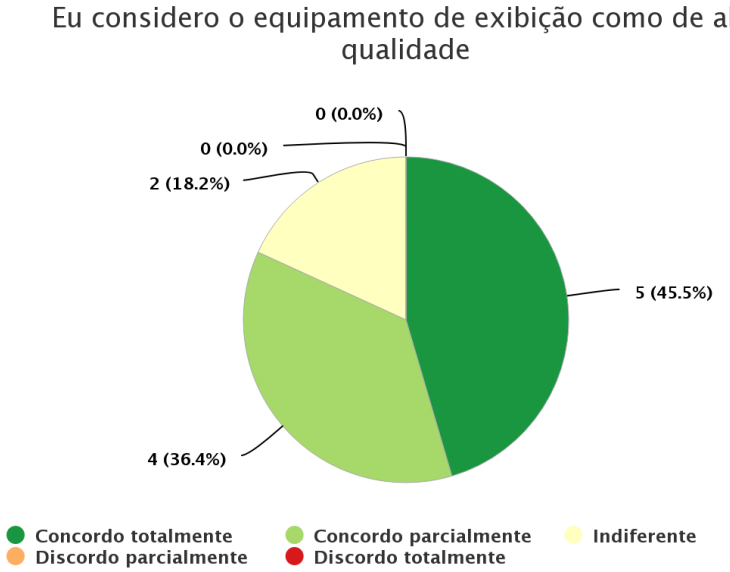
As grandes quantidade de usuários que concordaram, de uma maneira ou de outra, com as respostas as Questões 18 (Figura 43), sobre a realização das tarefas de diferentes maneiras, e

Figura 38 – Respostas para a Questão 13.



Fonte - Produzida pelo autor.

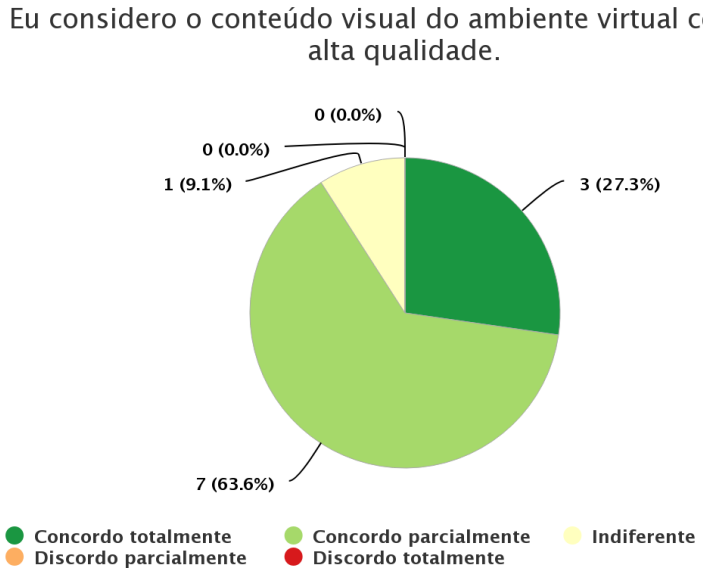
Figura 39 – Respostas para a Questão 14.



Fonte - Produzida pelo autor.

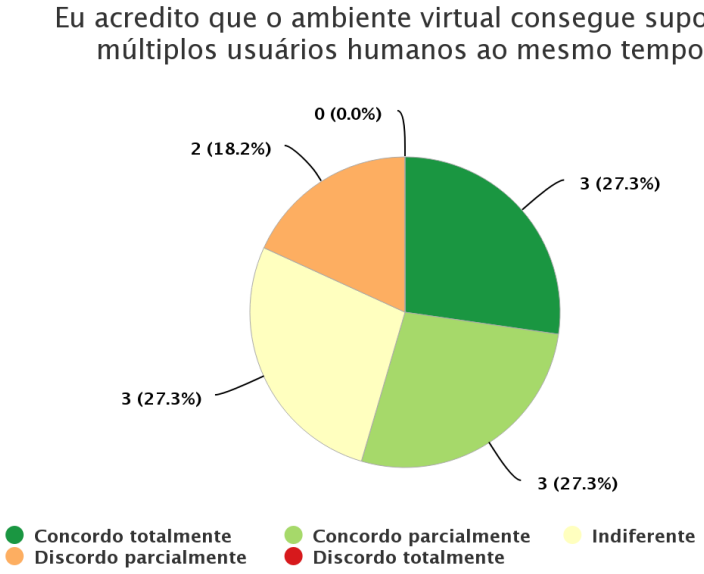
19 (Figura 44), sobre o reuso de técnicas aprendidas durante a realização das tarefas, indica que, apesar de possíveis dificuldades na manipulação da estrutura molecular (devido a sensibilidade dos controles e/ou falta de familiaridade com eles), os aparatos e interações fornecidos permitiram que os participantes melhorassem suas técnicas de visualização e manipulação da estrutura durante o experimento, o que é importante, pois, por sua vez, indica facilidade na utilização da aplicação por novos usuários.

Figura 40 – Respostas para a Questão 15.



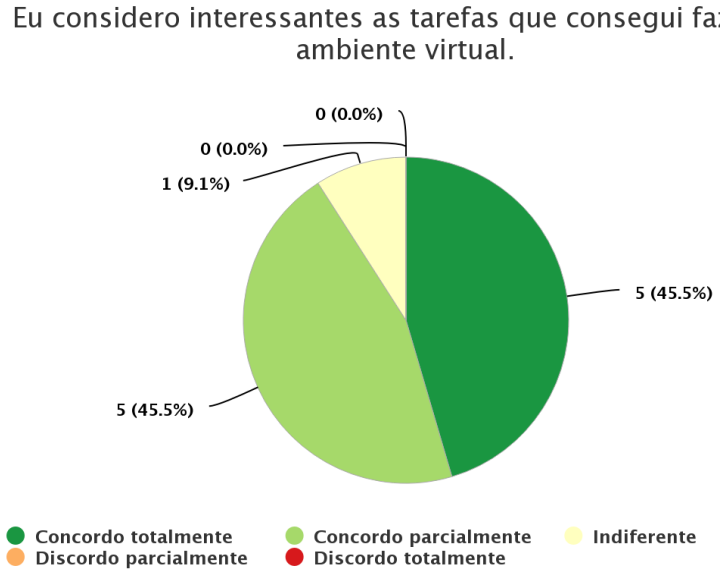
Fonte - Produzida pelo autor.

Figura 41 – Respostas para a Questão 16.



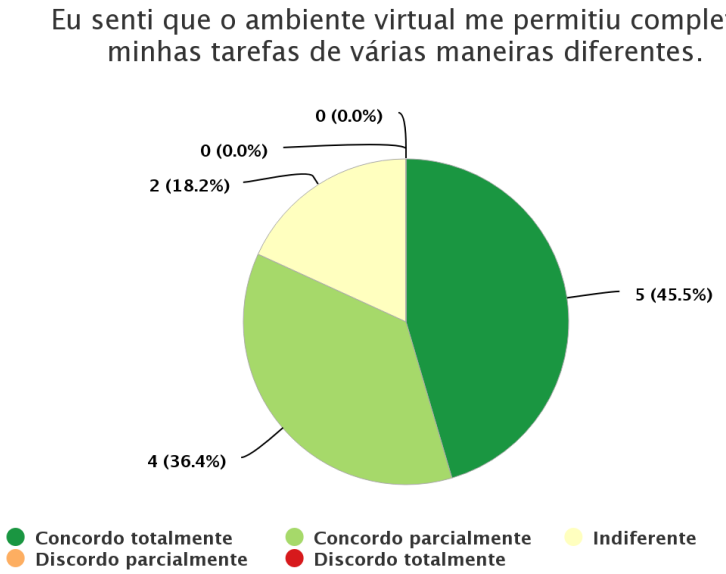
Fonte - Produzida pelo autor.

Figura 42 – Respostas para a Questão 17.



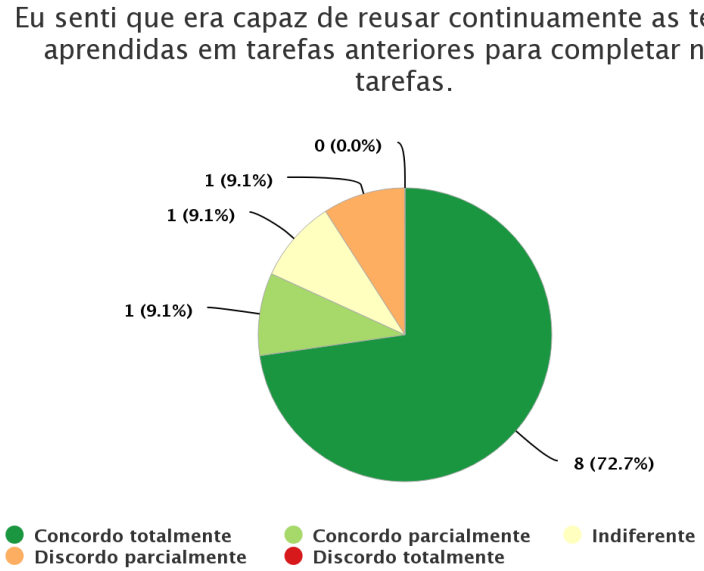
Fonte - Produzida pelo autor.

Figura 43 – Respostas para a Questão 18.



Fonte - Produzida pelo autor.

Figura 44 – Respostas para a Questão 19.



Fonte - Produzida pelo autor.

7 Conclusão

Este capítulo encerra o conteúdo principal desta dissertação. Inicia-se apresentando as observações finais, obtidas a partir da RSL realizada, da implementação do design de solução apresentado e do estudo de caso. Em seguida, são apresentados os trabalhos futuros pretendidos para os artefatos produzidos durante a pesquisa desta dissertação. Finaliza-se apresentando as publicações em que o autor participou durante o curso de mestrado.

7.1 Observações Finais

A VI 3D, especificamente, possibilita uma visualização mais efetiva de tipos de dados que, na VI 2D, seriam de difícil compreensão. Ao integrar a interação sobre a VI (2D ou 3D), então, permite-se que os dados visualizados sejam, principalmente, manipulados, selecionados, navegados.

Junto a isso, a adição de imersão na VI3DI visa um entendimento melhor de estruturas e formas tridimensionais, dada a sensação do usuário de estar frente-a-frente com os dados que visualiza, mesmo que estes assumam, para todos os efeitos, representações abstratas. Sistemas de multi-projeção são um dos principais meios de se proporcionar a imersão, no entanto, envolvem desafios no desenvolvimento das aplicações que os tornam de difícil uso.

O uso da web para desenvolvimento da VI3DI, com suporte a sistemas de multi-projeção e interação por meio dispositivos de entrada diversificados, permite maior facilidade no uso desas aplicações, pois, aos usuários, é necessário meramente acessar um endereço web nos navegadores, e portabilidade e manutenibilidade, pois o código-fonte da aplicação pode (em tese) ser executado em qualquer navegador web, sem necessidade de ajustes, e, também, porque aplicações web clientes só podem ser desenvolvidas com um limitado conjunto de tecnologias.

A RSL realizada mostrou que existem poucos estudos que abordam somente a multi-projeção na web ou a utilização de dispositivos de interação diversificados na web. Ao considerar simultaneamente esses dois tópicos, então, foi encontrado um único estudo, revelando, então, um espaço de pesquisa ainda não largamente explorado. Além das dificuldades citadas, isso também é fruto de limitações técnicas apresentadas por navegadores web e de recursos (conceituais e de software) apropriados para tanto.

Com isso, o design de solução para aplicações VI3DI web apresentado visa cobrir essas questões, mostrando os requisitos necessários para tanto e os relacionamentos entre elas. Este design foi concebido considerando aspectos técnicos da web, tais como o WebRTC, a fim de melhor se adaptar à realidade da plataforma; em todo caso, implementações dele ainda serão responsáveis por certos aspectos - em geral, aqueles mais dependentes de tecnologias com maiores tendências a mudanças. O design apoia a multi-projeção por meio de CGs mestre-escravo, os quais podem ser estendidos para possibilitar o estabelecimento de um CG descentralizado que, embora não ofereça desempenho ideal, necessita de relativamente poucas conexões.

A implementação do design de solução apresentado, a Livvclib, ocorreu semi-concorrentemente a elaboração do próprio design, de maneira iterativa, onde as decisões de design influenciaram a implementação e detalhes (e limitações) técnicos influenciaram decisões do design. Isso permitiu que a implementação validasse a possibilidade e a viabilidade do design elaborado, as quais foram refinadas graças às implementações prototípicas. Além de implementar o design de solução apresentado, a Livvclib também fornece certos recursos que visam facilitar o desenvolvimento de aplicações que a utilizam, tais como a segmentação de dados recebidos de outros nós e de dispositivos e a *bufferização* de dados enviados para CGs.

O design de solução apresentado concentra suas funcionalidades no lado cliente das aplicações, a fim de melhor aproveitar a facilidade e portabilidade da web; ao lado servidor são relegadas somente as funcionalidades que não podem, de outra maneira, ser executadas em navegadores web. A Livvclib mantém essa filosofia ao fazer com que o lado cliente seja independente da implementação do lado servidor, abstraindo seu funcionamento através da definição de eventos de entrada e saída que devem ser tratados e emitidos - *como* eles o são, fica a cargo do desenvolvedor. Isso permite que a Livvclib seja incorporada em infraestruturas já existentes ou que possuam requisitos tecnológicos. Em todo caso, a Livvclib também inclui recursos em uma tecnologia específica (Node.js) para o desenvolvimento do lado servidor da aplicação, permitindo que usuários possam simplesmente usá-la, caso suas restrições os permitam.

O estudo de caso mostrou que a Livvclib pode, de fato, ser usada na adaptação de aplicações existentes, o que, por sua vez, valida a eficiência do design de solução apresentado. A adaptação, em si, requisitou uma quantidade relativamente grande de linhas de código, todavia, isso não deve ser tomado como um indicador particularmente preciso do trabalho real de adaptação, devido ao estilo de código adotado, que requer a inclusão de várias linhas sem conteúdo, e a duplicação de classes da aplicação, a fim de minimizar mudanças diretas no código-fonte original da aplicação.

O experimento sobre a aplicação desenvolvida para o estudo de caso permitiu uma análise qualitativa do desempenho da Livvclib e da aplicação desenvolvida para o estudo de caso. A participação de estudantes de mestrado de Física e de Ciência da Computação forneceu uma amostra particularmente interessante, pois, de um lado, há especialistas sobre o conteúdo, que não, necessariamente, tem familiaridade com as técnicas e dispositivos, enquanto, de outro, há especialistas sobre as técnicas e dispositivos, que não, necessariamente, tem familiaridade com o conteúdo. O experimento, em todo caso, foi projetado para que qualquer pessoa o realizasse, independentemente de seu conhecimento sobre o conteúdo.

Julga-se que as respostas dadas ao questionário mostram aceitação da aplicação, de seu desempenho e dos aparatos de visualização e interação utilizados, embora alguns mais do que outros. Pontos de atenção se concentram nos controles de manipulação da visualização pelo controle de video game e pelo tablet, que não foram suficientemente adequados para serem preferíveis mouse, e comunicação (verbal, visual ou através da usabilidade) de que a aplicação pode ser usada em ambiente colaborativo.

7.2 Contribuições do Trabalho

O trabalho desenvolvido nesta dissertação forneceu as seguintes contribuições:

- Constatação de que estudos sobre multi-projeção na web são recentes e ainda poucos e de que a técnica mais comum para implementá-la são CGs descentralizados. Mais ainda, a multi-projeção na web com interação com dispositivos de entrada diversificados ainda é um campo largamente não explorado;

- Design de solução para aplicações VI3DI executadas em navegadores web, suportando CGs mestre-escravo e descentralizados e dispositivos de interação diversificados;

- Implementação do design apresentado na forma de um framework/biblioteca JavaScript, Livvclib, que pode ser incorporado a aplicações novas ou existentes. Esta implementação fornece recursos para ambos os lados cliente e servidor de uma aplicação web; e

- Adaptação de aplicação VI 3D web (mais especificamente, de visualização molecular) para permitir distribuição de seu conteúdo por um sistema de multi-projeção, interação por meio de dispositivos de entrada diversificados, tais como as telas sensíveis a toque de dispositivos móveis e controles de video game, e uso descentralizado.

7.3 Trabalhos Futuros

Tendo em vista a aplicação e avaliação do design de solução para aplicações VI3DI web apresentado e da Livvclib, pretende-se usá-los para adaptar a aplicação web VIMAPS (BREGA et al., 2015), que permite a visualização de eventos meteorológicos severos em mapas e de gráficos 3D para melhor entendê-los. O objetivo principal da adaptação é fazer com que o VIMAPS distribua seu conteúdo por um sistema de multi-projeção, possibilitando a visualização de grandes quantidades de dados simultaneamente, especialmente nos mapas com alto nível de *zoom*; novas modalidades de interação com a aplicação também serão investigadas.

Quanto à parte técnica do design de solução apresentado, duas frentes de aperfeiçoamento estão sendo investigadas. A primeira consiste na adição de suporte a HMDs RV na Livvclib, graças a API WebVR, com o propósito de oferecer mais uma maneira de imersão na aplicações. Dado que já existem recursos de software para facilitar o uso do WebVR, neste caso, o foco está na possibilidade de colaboração síncrona entre usuários com diferentes dispositivos de interação e meios de imersão. A segunda envolve estender o código-fonte da biblioteca VRPN, para que servidores VRPN utilizem os protocolos e padrões do WebRTC para transmitir os dados dos dispositivos de entrada. Dessa maneira, seria possível fazer com que as aplicações VI3DI web recebessem os dados dos dispositivos de entrada diretamente do computador em que estão conectados, sem passar pelo servidor de aplicação e sinalização, como ocorre atualmente, potencialmente diminuindo a latência na obtenção desses dados quando o servidor de aplicação e sinalização está em rede remota.

7.4 Publicações

No período entre Março/2016 e Junho/2018, o autor desta dissertação participou das seguintes publicações:

- Como primeiro autor:
  - BAGLIE, L.S.S.; NETO, M.P.; GUIMARAES, M.P.; BREGA, J.R.F.. Distributed, Immersive and Multi-Platform Molecular Visualization for Chemistry Learning. In: The 17th International Conference on Computational Science and Its Applications (ICCSA 2017), 2017, Trieste. Computational Science and Its Applications – ICCSA 2017, 2017. v. 17. p. 569-584.
  - BAGLIE, L.S.S; BREGA, J.R.F.. Relatório e Revisão Sistemática da Literatura sobre a utilização de dispositivos móveis simultaneamente a displays para a disciplina de Estudos Especiais. In: VI Workshop do Programa de Pós-Graduação em Ciência da Computação da UNESP. São José do Rio Preto, SP, Brasil: WPPGCC, 2016.
  - BAGLIE, L.S.S.; NETO, M.P.; BREGA, J.R.F.. Visualização imersiva e colaborativa de moléculas utilizando tecnologia de jogos. In: V Congresso Brasileiro de Informática na Educação (CBIE 2016), 2016, Uberlância. Anais dos Workshops do V Congresso Brasileiro de Informática na Educação (CBIE 2016), 2016. v. 5. p. 711-720.
- Como um dos autores:
  - BAPTISTA, F.Q.; NETO, M.P.; BAGLIE, L.S.S.; WEBER, S.A.T.; BREGA, J.R.F.. SPackageCreator3D - A 3D Content Creator to the Moodle Platform to Support Human Anatomy Teaching and Learning. In: The 17th International Conference on Computational Science and Its Applications (ICCSA 2017), 2017, Trieste. Computational Science and Its Applications – ICCSA 2017, 2017. v. 17. p. 605-618.
  - CARVALHO, B.A.; GUIMARAES, M.P.; DIAS, D.R.C.; BAGLIE, L.S.S.; NETO, M.P.; BREGA, J.R.F.. Embedding augmented reality applications into Learning Management Systems. In: The 17th International Conference on Computational Science and Its Applications (ICCSA 2017), 2017, Trieste. Computational Science and Its Applications – ICCSA 2017, 2017. v. 17. p. 585-594.

Referências

ANDERSON, E. F. et al. Choosing the infrastructure for entertainment and serious computer games-a whiteroom benchmark for game engine selection. In: IEEE. *Games and Virtual Worlds for Serious Applications (VS-GAMES), 2013 5th International Conference on*. [S.l.], 2013. p. 1–8. Citado na página 155.

ASSAL, H.; CHIASSON, S.; BIDDLE, R. Cesar: Visual representation of source code vulnerabilities. In: IEEE. *Visualization for Cyber Security (VizSec), 2016 IEEE Symposium on*. [S.l.], 2016. p. 1–8. Citado 8 vezes nas páginas 43, 45, 50, 137, 138, 140, 141 e 142.

Association for Computing Machinery. *ACM Digital Library*. 2018. Disponível em: <http://dl.acm.org>. Citado na página 128.

BAGLIE, L.; NETO, M. P.; BREGA, J. R. Visualização imersiva e colaborativa de moléculas utilizando tecnologia de jogos. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2016. v. 5, n. 1, p. 711. Citado na página 90.

BAGLIE, L. S. d. S. et al. Distributed, immersive and multi-platform molecular visualization for chemistry learning. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2017. p. 569–584. Citado na página 90.

BARIONI, M. C. N. et al. Data visualization in rdbms. In: *Information Systems and Databases*. [S.l.: s.n.], 2002. p. 264–269. Citado na página 25.

Baroth, E. C.; Chin, G. E.; Curran, P. S. *Information visualization techniques in a multi-mission operations environment*. [S.l.], 1990. Citado na página 27.

BIERBAUM, A. D. *VR Juggler: A Virtual Platform for Virtual Reality Application Development*. Dissertação (Mestrado) — Iowa State University, 2000. Citado na página 29.

BOWMAN, D. A.; MCMAHAN, R. P. Virtual reality: how much immersion is enough? *Computer*, IEEE, v. 40, n. 7, 2007. Citado na página 30.

BRADEN, R. *Requirements for Internet Hosts – Communication Layers*. [S.l.], 1989. Citado na página 147.

BREGA, J. R. F. et al. Visualização científica de descargas elétricas atmosféricas scientific visualization of atmospheric electrical discharges. In: SOCIEDADE BRASILEIRA DE COMPUTAçãO. *Workshop de Realidade Virtual e Aumentada (WRVA2015)*. [S.l.], 2015. p. 74–79. Citado na página 109.

CABELLO, R. *three.js / documentation*. 2017. Disponível em: <https://threejs.org/docs/>. Acesso em: mai. 2017. Citado 4 vezes nas páginas 72, 153, 154 e 155.

CARD, S. K.; MACKINLAY, J. D.; SHNEIDERMAN, B. *Readings in information visualization: using vision to think*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-533-9. Citado 2 vezes nas páginas 25 e 27.

CARPENDALE, M. S. T.; COWPERTHWAITE, D. J.; FRACCHIA, F. D. 3-dimensional pliable surfaces: For the effective presentation of visual information. In: ACM. *Proceedings of the 8th annual ACM symposium on User interface and software technology*. [S.l.], 1995. p. 217–226. Citado na página 27.

CARR, D. A. Guidelines for Designing Information Visualization Applications. Stockholm, Sweden, 1999. Citado 2 vezes nas páginas 25 e 26.

CASU, A. et al. RiftArt: Bringing Masterpieces in the Classroom through Immersive Virtual Reality. In: GIACHETTI, A.; BIASOTTI, S.; TARINI, M. (Ed.). *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. [S.l.]: The Eurographics Association, 2015. ISBN 978-3-905674-97-2. Citado na página 89.

CHANIOTIS, I. K.; KYRIAKOU, K.-I. D.; TSELIKAS, N. D. Is Node.js a viable option for building modern web applications? A performance evaluation study. *Computing*, v. 97, n. 10, p. 1023–1044, 2015. ISSN 0010-485X. Citado 5 vezes nas páginas 148, 149, 150, 151 e 152.

CHEN, H. et al. Data distribution strategies for high-resolution displays. *Computers & Graphics*, v. 25, n. 5, p. 811 – 818, 2001. ISSN 0097-8493. Mixed realities - beyond conventions. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0097849301001236>. Citado 4 vezes nas páginas 21, 30, 32 e 33.

CHEN, Y. et al. Software environments for cluster-based display systems. In: IEEE. *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*. [S.l.], 2001. p. 202–210. Citado 2 vezes nas páginas 30 e 33.

CHERTOFF, D. B.; GOLDIEZ, B.; LAVIOLA, J. J. Virtual experience test: A virtual environment evaluation questionnaire. In: IEEE. *Virtual Reality Conference (VR), 2010 IEEE*. [S.l.], 2010. p. 103–110. Citado na página 95.

CHUNG, H. et al. Visporter: facilitating information sharing for collaborative sensemaking on multiple displays. *Personal and Ubiquitous Computing*, Springer, v. 18, n. 5, p. 1169–1186, 2014. Citado 4 vezes nas páginas 38, 48, 133 e 135.

CONGOTE, J. et al. Interactive visualization of volumetric data with webgl in real-time. In: ACM. *Proceedings of the 16th International Conference on 3D Web Technology*. [S.l.], 2011. p. 137–146. Citado na página 21.

CORDEIL, M. et al. Immersive Collaborative Analysis of Network Connectivity: CAVE-style or Head-Mounted Display? *IEEE Transactions on Visualization and Computer Graphics*, v. 23, n. 1, p. 441–450, 2017. ISSN 10772626. Citado na página 30.

CRUZ-NEIRA, C. et al. The cave: Audio visual experience automatic virtual environment. *Commun. ACM*, ACM, New York, NY, USA, v. 35, n. 6, p. 64–72, jun. 1992. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/129888.129892>. Citado 3 vezes nas páginas 21, 30 e 143.

DAVIES, R. A. et al. Visualization of molecular quantum dynamics: A molecular visualization tool with integrated web3d and haptics. In: *Proceedings of the Tenth International Conference on 3D Web Technology*. New York, NY, USA: ACM, 2005. (Web3D '05), p. 143–150. ISBN 1-59593-012-4. Disponível em: <http://doi.acm.org/10.1145/1050491.1050512>. Citado na página 89.

DIAS, D. R. C. *Sistema Avançado de Realidade Virtual para Visualização de Estruturas Odontológicas*. Dissertação (Mestrado) — Universidade Estadual Paulista "Júlio de Mesquita Filho", 2011. Citado 3 vezes nas páginas 31, 32 e 89.

DIAS, D. R. C. et al. 3D Semantic Models for Dental Education. In: CRUZ-CUNHA, M. M. et al. (Ed.). *ENTERprise Information Systems*. [S.l.]: Springer Berlin Heidelberg, 2011, (Communications in Computer and Information Science, v. 221). p. 89–96. ISBN 978-3-642-24352-3. Citado na página 31.

DIAS, D. R. C. et al. Design and evaluation of an advanced virtual reality system for visualization of dentistry structures. In: *Virtual Systems and Multimedia (VSMM), 2012 18th International Conference on*. [S.l.: s.n.], 2012. p. 429–435. Citado 2 vezes nas páginas 21 e 30.

DIPIERRO, M. Toy vision-guided 3d robotic arm in javascript. *Computing in Science & Engineering*, IEEE, v. 20, n. 1, p. 43–49, 2018. Citado 7 vezes nas páginas 43, 45, 50, 137, 138, 140 e 142.

DIRKSEN, J. *Three.js Essentials*. Birmingham, Reino Unido: Packt Publishing, 2014. 198 p. Citado 4 vezes nas páginas 152, 153, 154 e 156.

DISZ, T. et al. Virtual reality visualization of parallel molecular dynamics simulation. In: *Society for Computer Simulation*. [S.l.: s.n.], 1995. p. 483–487. Citado na página 30.

DO, N. Q.; CAI, H.; JIANG, L. Leap studio–a virtual interactive 3d modeling application based on web gl. In: IEEE. *Digital Home (ICDH), 2014 5th International Conference on*. [S.l.], 2014. p. 374–379. Citado 7 vezes nas páginas 43, 44, 50, 137, 138, 139 e 140.

DOGLIO, F. *Pro REST API Development with Node.js*. La Paz, Uruguai: Apress, 2015. 191 p. Citado 2 vezes nas páginas 149 e 150.

Elsevier B.V. *Scopus | The largest database of peer-reviewed literature | Elsevier*. 2018. Disponível em: <https://www.elsevier.com/solutions/scopus>. Acesso em: mai. 2018. Citado na página 128.

Epic Games, Inc. *Virtual Reality Development | Unreal Engine*. 2017. Disponível em: <https://docs.unrealengine.com/latest/INT/Platforms/VR/index.html>. Acesso em: mai. 2017. Citado na página 31.

EVANS, A. et al. 3D graphics on the web: A survey. *Computers & Graphics*, v. 41, p. 43–61, 2014. ISSN 00978493. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0097849314000260>. Citado na página 153.

FITTKAU, F.; KRAUSE, A.; HASSELBRING, W. Exploring software cities in virtual reality. In: IEEE. *Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference on*. [S.l.], 2015. p. 130–134. Citado 9 vezes nas páginas 43, 44, 45, 50, 137, 138, 139, 140 e 142.

GAO, Y.; OSMAN, H. A.; SADDIK, A. E. Mpeg-v based web haptic authoring tool. In: IEEE. *Haptic Audio Visual Environments and Games (HAVE), 2013 IEEE International Symposium on*. [S.l.], 2013. p. 87–91. Citado 7 vezes nas páginas 43, 45, 50, 137, 138, 141 e 142.

GARDNER, H. E. *Frames Of Mind: The Theory Of Multiple Intelligences*. 10th. ed. Basic Books, 1983. Paperback. ISBN 0465025102. Disponível em: <http://www.worldcat.org/isbn/0465025102>. Citado na página 89.

GARRETT, J. J. *Ajax: A New Approach to Web Applications*. 2005. Disponível em: <http://www.adaptivepath.org/ideas/ajax-new-approach-web-applications/>. Acesso em: mai. 2017. Citado na página 59.

GEYMAYER, T.; SCHMALSTIEG, D. Collaborative distributed cognition using a seamless desktop infrastructure. In: IEEE. *Immersive Analytics (IA), 2016 Workshop on*. [S.l.], 2016. p. 7–12. Citado 9 vezes nas páginas 11, 38, 40, 41, 48, 132, 133, 134 e 135.

GHATAK, S. et al. Development of a keyboardless social networking website for visually impaired: Socialweb. In: IEEE. *Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS), 2014 IEEE*. [S.l.], 2014. p. 232–236. Citado 7 vezes nas páginas 43, 45, 50, 137, 138, 140 e 142.

Google, inc. *Google VR SDK for Unity | Google VR | Google Developers*. 2018. Disponível em: <https://developers.google.com/vr/unity/>. Acesso em: mai. 2017. Citado 2 vezes nas páginas 31 e 90.

GRINSTEIN, G. G.; WARD, M. O. Introduction to Data Visualization . In: *Information Visualization in Data Mining and Knowledge Discovery*. [S.l.]: Morgan Kaufmann Publishers Inc., 2002. ISBN 9781558606890. Citado na página 26.

GRUBERT, J.; KRANZ, M. mpcubee: Towards a mobile perspective cubic display using mobile phones. In: IEEE. *Virtual Reality (VR), 2017 IEEE*. [S.l.], 2017. p. 459–460. Citado 5 vezes nas páginas 38, 48, 132, 133 e 135.

GRUBERT, J.; KRÄNZ, M. Towards ad hoc mobile multi-display environments on commodity mobile devices. In: IEEE. *Virtual Reality (VR), 2017 IEEE*. [S.l.], 2017. p. 461–462. Citado 4 vezes nas páginas 38, 48, 133 e 135.

GUIMARÃES, M. d. P. et al. Immersive and interactive virtual reality applications based on 3d web browsers. *Multimedia Tools and Applications*, v. 77, n. 1, p. 347–361, Jan 2018. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-016-4256-7>. Citado 12 vezes nas páginas 21, 28, 33, 38, 39, 42, 43, 48, 86, 132, 133 e 135.

HAK, R.; DOLEZAL, J.; ZEMAN, T. Manitou: A multimodal interaction platform. In: IEEE. *Wireless and Mobile Networking Conference (WMNC), 2012 5th Joint IFIP*. [S.l.], 2012. p. 60–63. Citado 5 vezes nas páginas 43, 50, 137, 138 e 141.

HAN, Y. C. et al. Visualizing causes and effects of california sea lion unusual mortality event (ume). In: IEEE. *VIS Arts Program (VISAP), 2017 IEEE*. [S.l.], 2017. p. 1–7. Citado 4 vezes nas páginas 43, 44, 50 e 137.

HASHIMOTO, N.; ISHIDA, Y.; SATO, M. A self-distributing software environment for immersive multiprojector displays. *Systems and Computers in Japan*, v. 38, n. 2, p. 1–9, 2007. ISSN 08821666. Citado 2 vezes nas páginas 34 e 35.

Institute of Electrical and Electronics Engineering. *IEEE Xplore - About IEEE Xplore*. 2018. Disponível em: <http://ieeexplore.ieee.org/xpl/aboutUs.jsp>. Acesso em: mai. 2018. Citado na página 128.

Internet Engineering Task Force. *Rtcweb Status Pages*. 2017. Disponível em: <https://tools.ietf.org/wg/rtcweb/>. Acesso em: abr. 2017. Citado na página 145.

ISO. *Ergonomics of Human System Interaction*. Geneva, CH, 2001. v. 2000. Citado na página 95.

JACOB, R. J. Human-computer interaction: input devices. *ACM Computing Surveys (CSUR)*, ACM, v. 28, n. 1, p. 177–179, 1996. Citado na página 28.

JOHN, M. S. et al. The use of 2d and 3d displays for shape-understanding versus relative-position tasks. *Human Factors*, SAGE Publications Sage CA: Los Angeles, CA, v. 43, n. 1, p. 79–98, 2001. Citado na página 27.

JOHNSTON, A.; YOAKUM, J.; SINGH, K. Taking on webRTC in an enterprise. *IEEE Communications Magazine*, v. 51, n. 4, p. 48–54, 2013. ISSN 01636804. Citado 2 vezes nas páginas 145 e 146.

KIM, H. et al. Cluster rendering on large high-resolution multi-displays using x3dom and html. *Multimedia Systems*, v. 23, n. 2, p. 265–279, Mar 2017. ISSN 1432-1882. Disponível em: <https://doi.org/10.1007/s00530-015-0495-0>. Citado 7 vezes nas páginas 38, 41, 42, 48, 133, 134 e 135.

KIRK, A. *Data Visualization: A Sucessful Design Process*. [S.l.]: Packt Publishing Ltd, 2012. Citado na página 26.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007. Citado 7 vezes nas páginas 37, 125, 126, 127, 128, 129 e 131.

KONSTANTINIDIS, E. I. et al. A lightweight framework for transparent cross platform communication of controller data in ambient assisted living environments. *Information Sciences*, Elsevier, v. 300, p. 124–139, 2015. Citado 2 vezes nas páginas 46 e 47.

KONSTANTINIDIS, E. I.; BAMPAROPOULOS, G.; BAMIDIS, P. D. Moving real exergaming engines on the web: The webfitforall case study in an active and healthy ageing living lab environment. *IEEE journal of biomedical and health informatics*, IEEE, v. 21, n. 3, p. 859–866, 2017. Citado 11 vezes nas páginas 43, 44, 45, 46, 48, 50, 137, 138, 139, 140 e 142.

KOOIMA, R. Generalized perspective projection. *School of Elect. Eng. and Computer Science*, p. 1–7, 2008. Citado 2 vezes nas páginas 56 e 72.

LAKOFF, G.; JOHNSON, M. The metaphorical structure of the human conceptual system. In: *Cognitive Science*. [S.l.: s.n.], 1980. p. 195–208. Citado na página 26.

LAUER, H. C.; NEEDHAM, R. M. On the duality of operating system structures. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 13, n. 2, p. 3–19, abr. 1979. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/850657.850658>. Citado na página 148.

LIU, J.-H. et al. Aasmp–android application server for mobile platforms. In: IEEE. *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*. [S.l.], 2013. p. 643–650. Citado 5 vezes nas páginas 43, 44, 50, 137 e 138.

LORETO, S.; ROMANO, S. P. Real-time communications in the web: Issues, achievements, and ongoing standardization efforts. *IEEE Internet Computing*, v. 16, n. 5, p. 68–73, 2012. ISSN 10897801. Citado 3 vezes nas páginas 145, 146 e 147.

LUGRIN, J.-L. et al. Caveudk: A vr game engine middleware. In: *Proceedings of the 18th ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA: ACM, 2012. (VRST '12), p. 137–144. ISBN 978-1-4503-1469-5. Disponível em: <http://doi.acm.org/10.1145/2407336.2407363>. Citado na página 33.

LV, Z. et al. Game On, Science - How Video Game Technology May Help Biologists Tackle Visualization Challenges. *PLOS ONE*, Public Library of Science, v. 8, n. 3, p. 1–13, 03 2013. ISSN 19326203. Citado na página 90.

MAEDA, A.; KOBAYASHI, M. Trace select: acquiring remote targets with gesture guides generated from glyphs. *IEEE Transactions on Consumer Electronics*, IEEE, v. 60, n. 3, p. 453–460, 2014. Citado 7 vezes nas páginas 43, 45, 50, 137, 138, 141 e 142.

MAGGIONI, C. A novel gestural input device for virtual reality. In: IEEE. *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*. [S.l.], 1993. p. 118–124. Citado na página 28.

MARAI, G. E.; FORBES, A. G.; JOHNSON, A. Interdisciplinary immersive analytics at the electronic visualization laboratory: Lessons learned and upcoming challenges. In: IEEE. *Immersive Analytics (IA), 2016 Workshop on*. [S.l.], 2016. p. 54–59. Citado 6 vezes nas páginas 38, 39, 48, 133, 134 e 135.

MARRINAN, T. et al. Sage2: A new approach for data intensive collaboration using scalable resolution shared displays. In: IEEE. *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*. [S.l.], 2014. p. 177–186. Citado 16 vezes nas páginas 38, 39, 40, 43, 44, 45, 48, 50, 130, 132, 133, 134, 137, 138, 139 e 140.

MCCUNE, R. R. Node.js Paradigms and Benchmarks. *Striegel, Grad Os F*, v. 11, p. 1–6, 2011. Disponível em: <http://netscale.cse.nd.edu/twiki/pub/Edu/GradOSF11FinalProjects/final.pdf>. Citado 4 vezes nas páginas 148, 149, 150 e 151.

MEIXNER, B.; KALLMEIER, F. Speech control for html5 hypervideo players. In: *WSICC@ TVX*. [S.l.: s.n.], 2016. Citado 7 vezes nas páginas 43, 45, 50, 137, 138, 140 e 142.

MERCHANT, Z. et al. Exploring 3-D virtual reality technology for spatial ability and chemistry achievement. *Journal of Computer Assisted Learning*, v. 29, n. 6, p. 579–590, 2013. ISSN 02664909. Citado na página 89.

MIKROPOULOS, T. A.; NATSIS, A. Educational virtual environments: A ten-year review of empirical research (1999–2009). *Computers & Education*, v. 56, n. 3, p. 769 – 780, 2011. ISSN 0360-1315. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0360131510003052>. Citado na página 89.

MONSERRAT, P. F. et al. P4h: An example of successful use of serious games in telerehabilitation. In: IEEE. *Internet Technologies and Applications (ITA), 2015*. [S.l.], 2015. p. 261–265. Citado 8 vezes nas páginas 44, 45, 50, 138, 139, 140, 141 e 142.

MORAES, A. C.; ELER, D. M.; BREGA, J. R. Collaborative information visualization using a multi-projection system and mobile devices. In: IEEE. *Information Visualisation (IV), 2014 18th International Conference on*. [S.l.], 2014. p. 71–77. Citado na página 36.

MORENO, F. et al. An open source framework to manage kinect on the web. In: IEEE. *Computing Conference (CLEI), 2015 Latin American*. [S.l.], 2015. p. 1–9. Citado 10 vezes nas páginas 43, 44, 45, 50, 137, 138, 139, 140, 141 e 142.

MORGAN, T. B.; JARRELL, D.; VANCE, J. M. Poster: Rapid development of natural user interaction using kinect sensors and vrpn. In: *2014 IEEE Symposium on 3D User Interfaces (3DUI)*. [S.l.: s.n.], 2014. p. 163–164. Citado na página 158.

MOTTA, T.; NEDEL, L. Gestural interaction for manipulating graphs in a large screen using the kinect integrated to the browser. In: IEEE. *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*. [S.l.], 2012. p. 1–7. Citado 10 vezes nas páginas 43, 44, 45, 50, 137, 138, 139, 140, 141 e 142.

MUNZNER, T. What's vis, and why do it? In: _____. *Visualization Analysis & Design*. Boca Raton, FL, USA: CRC Press, 2014. p. 1–19. ISBN 978-1-4665-0893-4. Citado 8 vezes nas páginas 21, 26, 27, 28, 29, 93, 94 e 95.

NAVARRE, D. et al. A formal description of multimodal interaction techniques for immersive virtual reality applications. In: SPRINGER. *IFIP Conference on Human-Computer Interaction*. [S.l.], 2005. p. 170–183. Citado na página 29.

NAZAROV, R.; GALLETLY, J. Native browser support for 3d rendering and physics using webgl, html5 and javascript. In: GEORGIADIS, C. K.; KEFALAS, P.; STAMATIS, D. (Ed.). *BCI (Local)*. CEUR-WS.org, 2013. (CEUR Workshop Proceedings, v. 1036), p. 21–24. Disponível em: <http://dblp.uni-trier.de/db/conf/bci/bci2013l.html#NazarovG13>. Citado 3 vezes nas páginas 154, 155 e 156.

NETO, M. P.; BREGA, J. R. F. A survey of solutions for game engines in the development of immersive applications for multi-projection systems as base for a generic solution design. In: *2015 XVII Symposium on Virtual and Augmented Reality*. [S.l.: s.n.], 2015. p. 61–70. Citado 2 vezes nas páginas 35 e 36.

NETO, M. P. et al. Unity cluster package–dragging and dropping components for multi-projection virtual reality applications based on pc clusters. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2015. p. 261–272. Citado 4 vezes nas páginas 21, 33, 35 e 57.

NETO, M. P. et al. Tecnologias na integração de ambientes virtuais tridimensionais e a plataforma de ensino e aprendizagem moodle. *Revista Interdisciplinar de Tecnologias e Educação*, v. 1, p. 148–156, 2015. Citado na página 155.

Node.js Foundation. *Node.js*. 2017. Disponível em: <https://nodejs.org>. Acesso em: mai. 2017. Citado 2 vezes nas páginas 149 e 151.

NURMINEN, J. K. et al. P2p media streaming with html5 and webrtc. In: *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.: s.n.], 2013. p. 63–64. Citado 2 vezes nas páginas 145 e 146.

OAT, E.; FRANCESCO, M. D.; AURA, T. Mocha: Augmenting pervasive displays through mobile devices and web-based technologies. In: IEEE. *Pervasive computing and communications workshops (PERCOM workshops), 2014 IEEE international conference on*. [S.l.], 2014. p. 506–511. Citado 6 vezes nas páginas 43, 44, 50, 137, 138 e 141.

OJAMAA, A.; DUUNA, K. Assessing the security of Node.js platform. In: *2012 International Conference for Internet Technology and Secured Transactions*. [S.l.: s.n.], 2012. p. 348–355. ISBN VO -. Citado 3 vezes nas páginas 149, 150 e 151.

Open Source Virtual Reality. *OSVR Developer Portal*. 2018. Disponível em: <https://osvr.github.io/whitepapers/vrpn_in_osvr/>. Acesso em: mai. 2018. Citado na página 157.

OUSTERHOUT, J. Why threads are a bad idea (for most purposes). In: SAN DIEGO, CA, USA. *Presentation given at the 1996 Usenix Annual Technical Conference*. [S.l.], 1996. v. 5. Citado 2 vezes nas páginas 149 e 150.

PAKKANEN, T. et al. Interaction with webvr 360 video player: Comparing three interaction paradigms. In: IEEE. *Virtual Reality (VR), 2017 IEEE*. [S.l.], 2017. p. 279–280. Citado 9 vezes nas páginas 43, 44, 45, 50, 137, 138, 139, 140 e 142.

PALLEC, X. L. et al. A support to multi-devices web application. In: ACM. *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology*. [S.l.], 2010. p. 391–392. Citado 7 vezes nas páginas 44, 46, 48, 50, 137, 139 e 140.

PANTELIDIS, V. S. Reasons to Use Virtual Reality in Education and Training Courses and a Model to Determine When to Use Virtual Reality. *Themes in Science and Technology Education*, v. 2, n. 1-2, p. 59–70, 2009. ISSN 1792-8788. Citado 2 vezes nas páginas 89 e 102.

PATTANAKIMHUN, P.; CHINTHAMMIT, W.; CHOTIKAKAMTHORN, N. Evaluation of mobile phone interaction with large public displays. In: IEEE. *Computer Science and Software Engineering (JCSSE), 2017 14th International Joint Conference on*. [S.l.], 2017. p. 1–6. Citado 3 vezes nas páginas 38, 48 e 135.

PAUSCH, R.; PROFFITT, D.; WILLIAMS, G. Quantifying immersion in virtual reality. In: ACM PRESS/ADDISON-WESLEY PUBLISHING CO. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. [S.l.], 1997. p. 13–18. Citado na página 30.

RÄDLE, R. et al. Huddlelamp: Spatially-aware mobile displays for ad-hoc around-the-table collaboration. In: ACM. *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. [S.l.], 2014. p. 45–54. Citado 8 vezes nas páginas 38, 39, 40, 41, 48, 130, 133 e 134.

RAFFIN, B. et al. Pc clusters for virtual reality. In: IEEE. *Virtual Reality Conference, 2006*. [S.l.], 2006. p. 215–222. Citado na página 32.

RAHMAN, Y. A. et al. Helping-hand: a data glove technology for rehabilitation of monoplegia patients. In: IEEE. *Strategic Technology (IFOST), 2014 9th International Forum on*. [S.l.], 2014. p. 199–204. Citado 8 vezes nas páginas 43, 45, 50, 137, 138, 140, 141 e 142.

REN, D. et al. Evaluating wide-field-of-view augmented reality with mixed reality simulation. In: IEEE. *Virtual Reality (VR), 2016 IEEE*. [S.l.], 2016. p. 93–102. Citado 6 vezes nas páginas 38, 39, 48, 133, 134 e 135.

RITTITUM, P.; VATANAWOOD, W.; THONGTAK, A. Digital scrum board using leap motion. In: IEEE. *Computer and Information Science (ICIS), 2016 IEEE/ACIS 15th International Conference on*. [S.l.], 2016. p. 1–4. Citado 9 vezes nas páginas 43, 44, 45, 50, 137, 138, 139, 140 e 142.

ROBERTSON, G. G.; CARD, S. K.; MACKINLAY, J. D. Information visualization using 3d interactive animation. *Communications of the ACM*, ACM, v. 36, n. 4, p. 57–71, 1993. Citado na página 27.

ROSE, A. S. et al. Web-based molecular graphics for large complexes. In: *Proceedings of the 21st International Conference on Web3D Technology*. New York, NY, USA: ACM, 2016. (Web3D '16), p. 185–186. ISBN 978-1-4503-4428-9. Disponível em: <http://doi.acm.org/10.1145/2945292.2945324>. Citado na página 90.

ROSE, A. S.; HILDEBRAND, P. W. Ngl viewer: a web application for molecular visualization. *Nucleic Acids Research*, v. 43, n. W1, p. W576, 2015. Disponível em: <http://dx.doi.org/10.1093/nar/gkv402>. Citado na página 90.

RYAN, M.-L. Immersion vs. interactivity: Virtual reality and literary theory. *SubStance*, University of Wisconsin Press, v. 28, n. 2, p. 110–137, 1999. Citado na página 30.

SANFILIPPO, F. et al. A benchmarking framework for control methods of maritime cranes based on the functional mockup interface. *IEEE Journal of Oceanic Engineering*, IEEE, 2017. Citado 9 vezes nas páginas 43, 44, 45, 50, 137, 139, 140, 141 e 142.

SARAIYA, P.; NORTH, C.; DUCA, K. An insight-based methodology for evaluating bioinformatics visualizations. *Visualization and Computer Graphics, IEEE Transactions on*, v. 11, n. 4, p. 443–456, ago. 2005. ISSN 1077-2626. Citado na página 25.

SATHE, V. et al. Virtual reality websites (vr web). In: IEEE. *Electronics, Communication and Aerospace Technology (ICECA), 2017 International conference of*. [S.l.], 2017. v. 1, p. 647–652. Citado 7 vezes nas páginas 43, 45, 50, 137, 138, 140 e 142.

SAWICKI, B.; CHABER, B. Efficient visualization of 3D models by web browser. *Computing*, v. 95, n. SUPPL.1, 2013. ISSN 0010485X. Citado na página 154.

SCHWEDE, C.; HERMANN, T. Holor: Interactive mixed-reality rooms. In: IEEE. *Cognitive Infocommunications (CogInfoCom), 2015 6th IEEE International Conference on*. [S.l.], 2015. p. 517–522. Citado 5 vezes nas páginas 38, 48, 132, 133 e 135.

SEO, D.; LEE, Y.; YOO, B. Webizing interactive cad review system using super multiview autostereoscopic displays. In: SPRINGER. *International Conference on Human-Computer Interaction*. [S.l.], 2017. p. 62–67. Citado 9 vezes nas páginas 43, 44, 45, 50, 137, 139, 140, 141 e 142.

ŞERBAN, R.; CULIC, I. Configuring a cisco ir829gw as an internet of things device. In: IEEE. *RoEduNet Conference: Networking in Education and Research, 2016 15th*. [S.l.], 2016. p. 1–5. Citado 7 vezes nas páginas 43, 44, 45, 50, 137, 138 e 142.

SEYED, T. et al. From small screens to big displays: understanding interaction in multi-display environments. In: ACM. *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion*. [S.l.], 2013. p. 33–36. Citado na página 36.

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>. Citado 2 vezes nas páginas 33 e 35.

SHNEIDERMAN, B. Extreme visualization: squeezing a billion records into a million pixels. In: ACM. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. [S.l.], 2008. p. 3–12. Citado na página 25.

SHNEIDERMAN, B.; PLAISANT, C.; HESSE, B. W. Improving healthcare with interactive visualization. *Computer*, IEEE, v. 46, n. 5, p. 58–66, 2013. Citado 2 vezes nas páginas 21 e 25.

SINGH, K.; KRISHNASWAMY, V. A case for SIP in Javascript. *IEEE Communications Magazine*, v. 51, n. 4, p. 28–33, 2013. ISSN 01636804. Citado 3 vezes nas páginas 145, 146 e 152.

SOARES, L. P. *Um ambiente de multiprojeção totalmente imersivo baseado em aglomerados de computadores*. Tese (Doutorado em Sistemas Eletrônicos) — Escola Politécnica, Universidade de São Paulo, 2005. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-09112005-174258>. Citado na página 30.

STAADT, O. G. et al. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In: *Proceedings of the Workshop on Virtual Environments 2003*. New York, NY, USA: ACM, 2003. (EGVE '03), p. 261–270. ISBN 1-58113-686-2. Disponível em: <http://doi.acm.org/10.1145/769953.769984>. Citado 5 vezes nas páginas 30, 32, 33, 34 e 35.

STELZER, R. et al. Expanding vrpn to tasks in virtual engineering. In: AMERICAN SOCIETY OF MECHANICAL ENGINEERS. *Proceedings of the ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. [S.l.], 2014. p. 1–8. Citado na página 156.

STEUER, J. Defining virtual reality: Dimensions determining telepresence. *Journal of communication*, Wiley Online Library, v. 42, n. 4, p. 73–93, 1992. Citado 3 vezes nas páginas 21, 28 e 44.

SU, S. et al. High-resolution interactive and collaborative data visualization framework for large-scale data analysis. In: IEEE. *Collaboration Technologies and Systems (CTS), 2016 International Conference on*. [S.l.], 2016. p. 275–280. Citado 7 vezes nas páginas 38, 39, 48, 132, 133, 134 e 135.

SUMA, E. A. et al. Faast: The flexible action and articulated skeleton toolkit. In: *2011 IEEE Virtual Reality Conference*. [S.l.: s.n.], 2011. p. 247–248. ISSN 1087-8270. Citado na página 158.

TAKALA, T. M.; RAUHAMAA, P.; TAKALA, T. Survey of 3dui applications and development challenges. In: *2012 IEEE Symposium on 3D User Interfaces (3DUI)*. Costa Mesa, CA, EUA: IEEE, 2012. p. 89–96. Citado na página 130.

TAYLOR-II, R. M. *Home · vrpn/vrpn Wiki*. 2018. Disponível em: <https://github.com/vrpn/vrpn/wiki>. Acesso em: mai. 2018. Citado 4 vezes nas páginas 156, 157, 158 e 159.

TAYLOR-II, R. M. et al. Vrpn: A device-independent, network-transparent vr peripheral system. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. New York, NY, USA: ACM, 2001. (VRST '01), p. 55–61. ISBN 1-58113-427-4. Disponível em: <http://doi.acm.org/10.1145/505008.505019>. Citado 4 vezes nas páginas 85, 156, 157 e 158.

The Khronos Group, Inc. *WebGL - WebGL - OpenGL ES for the Web*. 2017. Disponível em: <https://www.khronos.org/webgl/>. Acesso em: mai. 2017. Citado 4 vezes nas páginas 22, 37, 38 e 152.

TILKOV, S.; VINOSKI, S. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, v. 14, n. 6, p. 80–83, 2010. ISSN 10897801. Citado 3 vezes nas páginas 149, 150 e 151.

Unity Technologies. *Unity - Store*. 2017. Disponível em: <https://store.unity.com>. Acesso em: mai. 2017. Citado 2 vezes nas páginas 90 e 153.

VIRVOU, M.; KATSIONIS, G.; MANOS, K. Combining software games with education: evaluation of its educational effectiveness. *Educational Technology & Society*, JSTOR, v. 8, n. 2, p. 54–65, 2005. Citado na página 89.

WARE, C. *Information Visualization: Perception for Design*. 3. ed. [S.l.]: Elsevier Science & Technology, 2012. (Interactive Technologies). ISBN 9780123814647. Citado 3 vezes nas páginas 25, 26 e 27.

WESSELS, A. et al. Remote data visualization through websockets. In: IEEE. *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*. [S.l.], 2011. p. 1050–1051. Citado na página 59.

WILSON, J.; BROWN, J. M.; BIDDLE, R. Ach walkthrough: A distributed multi-device tool for collaborative security analysis. In: ACM. *Proceedings of the 2014 ACM Workshop on Security Information Workers*. [S.l.], 2014. p. 9–16. Citado 8 vezes nas páginas 43, 45, 50, 137, 138, 140, 141 e 142.

World Wide Web Consortium. *HTML5*. 2017. Disponível em: <http://www.w3.org/TR/html5/single-page.html>. Acesso em: mai. 2017. Citado na página 152.

World Wide Web Consortium. *WebRTC 1.0: Real-time Communication Between Browsers*. 2017. Disponível em: <http://www.w3.org/TR/webrtc>. Acesso em: mai. 2017. Citado 4 vezes nas páginas 22, 37, 38 e 145.

YI, J. S. et al. Understanding and characterizing insights: how do people gain insights using information visualization? In: *Proceedings of the 2008 Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization*. New York, NY, USA: ACM, 2008. (BELIV '08), p. 4:1–4:6. ISBN 978-1-60558-016-6. Citado 2 vezes nas páginas 25 e 26.

ZORZAL, E. R. et al. Visualização de Informação com Realidade Virtual e Aumentada. In: *Realidade Virtual e Aumentada: Conceitos, Projeto e Aplicações*. [S.l.]: Editora SBC - Socidade Brasileira de Computação, Porto alegre, 2007. cap. 1, p. 2–21. ISBN 85-7669-108-6. Citado 2 vezes nas páginas 27 e 102.

Apêndices

APÊNDICE A – Revisão Sistemática da Literatura

Este apêndice apresenta a RSL realizada para identificar estudos relativos aos temas desta dissertação. Começa-se abordando o que é uma RSL e, em seguida, são apresentados o planejamento para realização dessa, os detalhes de condução da revisão e os resultados obtidos.

A Revisão Sistemática da Litetura é um meio para identificar, avaliar e interpretar os trabalhos relevantes a uma questão de pesquisa; ela é considerada um tipo de estudo *secundário*, pois é construída sobre trabalhos individuais projetados para alavancar o estado-da-arte, que são considerados estudos *primários* (KITCHENHAM; CHARTERS, 2007).

Ela é uma ferramenta adequada para o levantamento dos trabalhos relacionados aos temas de pesquisa desta dissertação, pois as três principais razões para sua realização vão de encontro com os objetivos deste estágio da pesquisa. Essas razões são (KITCHENHAM; CHARTERS, 2007):

- Resumir evidência existente a respeito de uma tecnologia (e.g., resumir soluções utilizadas para distribuição e sincronização de uma aplicação em um CG);
- Identificar lacunas nas últimas pesquisas, a fim de sugerir áreas para futura investigação; e
- Providenciar uma base para o posicionamento adequado de novas pesquisas.

A realização de uma RSL começa com a definição de um procolo de revisão que especifica a questão sendo abordada e os métodos utilizados para tanto, de modo que seja detectado o máximo possível de material relacionado a questão e que também seja possível avaliar o rigor empregado no processo, bem como sua reprodutibilidade (KITCHENHAM; CHARTERS, 2007). A partir das fases principais de uma RSL, apresentadas em Kitchenham e Charters (2007), a metodologia empregada nesta RSL pode ser apresentada como:

1. Especificação dos objetivos e questões da pesquisa;
2. Desenvolvimento do protocolo de revisão;
3. Avaliação do protocolo de revisão;
4. Identificação da pesquisa;
5. Seleção de estudos primários;
6. Avaliação da qualidade dos estudos;
7. Extração de dados;

8. Síntese dos dados; e
9. Formação do relatório (documento atual).

Apesar da ordem pré-estabelecida das fases, o processo tem um aspecto iterativo, onde, por vezes, torna-se necessário voltar a fases a anteriores para refinamento ou reformulação de parâmetros, e.g., a versão inicial protocolo de revisão inicial pode incorrer na seleção de estudos primários "falsos-positivos", mostrando a necessidade de ajustar os termos da busca para aumentar a taxa de seleção de estudos apropriados.

A.1 Planejamento

A definição do protocolo de revisão é, talvez, a parte mais importante do planejamento. O protocolo de revisão utilizado neste estudo estabelece os seguintes pontos:

- Objetivos da RSL e questões a serem respondidas;
- Bases científicas;
- Critérios de inclusão e exclusão e métodos de avaliação; e
- Dados a serem extraídos.

A.1.1 Objetivos e Questões

Kitchenham e Charters (2007) expõe que as questões da pesquisa são a parte mais importante da RSL, pois alavancam toda sua metodologia, estabelecendo meios para identificar os estudos primários que podem conter respostas (ou fragmentos delas) para as questões, o processo de extração dos dados necessários para responder as questões e a maneira como a análise de dados sintetizará os dados extraídos, oferecendo, então, as respostas para as questões.

Nesta dissertação, as questões estabelecidas, por sua vez, foram projetadas para atender objetivos previamente estabelecidos, assim, pode-se dizer que, nesta dissertação, a RSL visa responder questões e as respostas dessas questões satisfazem os objetivos de estudo.

Os objetivos estabelecidos foram:

- Identificar aplicações web (ou estudos sobre elas) que apresentam multi-projeção;
  - Avaliar maneiras como tal funcionalidade é usada por aplicações e estudos; e
  - Identificar recursos que permitam o desenvolvimento de tal funcionalidade.
- Identificar aplicações web (ou estudos sobre elas) que usam dispositivos de interação avançados e/ou diversificados;
  - Avaliar maneiras como tal funcionalidade é usada por aplicações e estudos; e
  - Identificar recursos que permitam o desenvolvimento de tal funcionalidade.

- Identificar soluções para o desenvolvimento de aplicações VI3DI para navegadores web.

A partir desses objetivos, foram derivadas três questões:

- **Como é aplicada a multi-projeção em navegadores web?**
- **Como dispositivos de interação, além de mouse e teclado, podem ser utilizados em navegadores web?**
- **Quais recursos possibilitam o desenvolvimento de aplicações VI3DI com multi-projeção para navegadores web?**

Embora a VI3DI seja tratada, nesta dissertação, como uma sinergia entre a imersão (proporcionada, no caso, pela multi-projeção) e a interação (proporcionada por dispositivos de interação diversificados), percebe-se, pelos objetivos e questões propostos, que a RSL realizada aborda estes temas, a princípio, separadamente, para, somente então, convergi-los sob o manto da VI3DI. Foi escolhida essa abordagem por duas razões principais:

- Tentativas iniciais de pesquisa pelos estudos primários mostraram, relativamente, poucos estudos se adequando a critérios de inclusão envolvendo ambas imersão e interação simultaneamente; e
- É possível utilizar recursos separados para implementação de imersão e interação, ainda que isso seja mais trabalhoso para o pesquisador/desenvolvedor.

Com isso, foram realizadas buscas individuais por estudos envolvendo a multi-projeção na web e por estudos envolvendo dispositivos de interação diversificados na web. Os resultados dessas buscas foram, então, analisados em conjunto a fim de responder a última questão estabelecida. Apesar da aparência pouco ortodoxa que a realização de duas buscas apresenta, Kitchenham e Charters (2007) não são explícitos nesse quesito, meramente especificando que é necessária uma *estratégia* de busca pelos estudos primários.

Para cada busca, foram definidos os seguintes conjuntos de termos-chaves:

- Termos-chave para multi-projeção na web: “multi projection”, “multi display”/“multiple display”, “multiple screen”, “graphic cluster”/“graphics cluster”/“graphical cluster”, “cave”, “powerwall”/“power wall”, “javascript” e “html5”/“html 5”; e
- Termos-chave para dispositivos de interação na web: “input device(s)”, “interaction device(s)”, “javascript” e “browser”.

A.1.2 Bases científicas

A pesquisa por estudos primários foi feita em três bases científicas: IEEE Xplore, ACM Digital Library e Scopus.

IEEE Xplorer (Institute of Electrical and Electronics Engineering, 2018) é uma biblioteca digital mantida pelo *Institute of Electrical and Eletronics Engineering* (IEEE; do inglês, "Instituto de Engenharia Elétrica e Eletrônica"). Ela apresenta quase 4 milhões de itens (entre artigos, livros, etc.) publicados pela própria IEEE ou seus parceiros. Grande parte dos itens fornecidos podem ser vistos como documentos PDF ou páginas web.

ACM Digital Library (Association for Computing Machinery, 2018) é uma biblioteca digital mantida pela *Association of Computing Machinery* (ACM; do inglês, "Associação de Maquinário de Computação"). Ela apresenta duas bases internas, a ACM *Full-Text Collection*, com mais de 400 mil publicações, e a ACM *Guide to Computing Literature*, com mais de 2 milhões de publicações entre livros, artigos, etc. A pesquisa foi realizada somente na base ACM *Full-Text Collection*, devido a questões de acesso.

Scopus (Elsevier B.V., 2018) é um banco de dados de resumos e citações da literatura revisada por especialistas: periódicos científicos, livros e anais de congressos. Ele não armazena o conteúdo, em si, meramente o referencia de seu local original, por isso, busca nele pode retornar resultados comuns as outras bases.

A.1.3 Critérios de Inclusão e Exclusão e Métodos de Avaliação

Os critérios de inclusão e exclusão devem ser utilizados para identificar quais estudos primários oferecem evidências para responder as questões da pesquisa (KITCHENHAM; CHARTERS, 2007). Para cada uma das buscas realizadas, foram estabelecidos os seguintes critérios:

- Critérios para multi-projeção na web:
  - Inclusão:
    - Aborda aplicação(ões) web que utiliza(m) multi-projeção ou distribuição de uma mesma visualização por várias telas, simultaneamente.
      - São consideradas telas: projetores, monitores de vídeo e dispositivos móveis.
      - Se não aborda um sistema em específico, então deve tratar de solução ou desafios relacionados a aplicações multi-projetadas na web.
  - Exclusão:
    - Tem foco em MDE.
- Critérios para dispositivos de interação na web:
  - Inclusão:
    - Aborda sistema(s) web que utiliza(m) dispositivo(s) de entrada diversificados para interação em aplicação web.
      - São "dispositivos de entrada diversificados" qualquer dispositivo que permita entrada de dados, além de mouse e teclado.

        - Se utiliza dispositivo móvel para interação, então conteúdo do dispositivo deve ser somente controles e opções para interações, não permitindo exibição de conteúdo adicional.
        - Se não aborda um sistema em específico, então deve tratar de solução ou desafios relacionados a utilização de dispositivos de interação diversificados em aplicações web.
    - Exclusão:
        - Dados do dispositivo de entrada não são usados para interação com aplicação ou dados (e.g., obtem imagem de *webcam* e meramente a reproduz no navegador).

- Critérios comuns as buscas:
    - Inclusão:
        - Publicado a partir de 2010.
    - Exclusão:
        - Não é artigo científico;
        - Não aborda aplicação/ferramenta web;
        - Estudo usa *web view*[1] para exibir conteúdo HTML/JavaScript; e
        - Estudo depende de *applet*, Adobe Flash ou Unity Web Player.

Embora o uso de *applets*, Adobe Flash e Unity *Web Player* em aplicações web ainda as mantenha com razoável suporte multi-plataforma (exceto em dispositivos móveis), foi escolhido excluir estudos que utilizam essas tecnologias, pois, com os novos recursos oferecidos pelo HTML5, elas estão se tornando obsoletas e, em alguns casos, depreciadas. Alternativamente, foi escolhido manter estudos que usam *plugins* de navegadores, pois considera-se que estes ainda são parte significativa da experiência de uso da web.

Quanto aos métodos de avaliação dos critérios de inclusão e exclusão para cada estudo primário encontrado, eles são, como sugerido em Kitchenham e Charters (2007), divididos em duas etapas. A primeira etapa consiste nos seguintes passos:

1. Verificar que termos-chave fazem parte do corpo do estudo[2];
2. Procura por imagens demonstrando que o estudo envolve multi-projeção/dispositivos de interação diversificados;
3. Leitura de resumo, introdução e conclusão.

Caso não seja possível incluir/excluir o estudo com base nestes passos, uma cópia completa dele é obtida e inicia-se a segunda etapa, que consiste na leitura completa do estudo primário,

[1] *Web view*: elemento visual que permite a exibição de conteúdo HTML/JavaScript em meio a uma aplicação compilada.

[2] Foi necessário verificar que os termos-chave constavam no corpo do estudo, pois foram encontrados muitos resultados falsos-positivos, onde alguns dos termos chaves apareciam somente nas seções de referências/bibliografias, não tendo relação com o texto principal.

procurando o contexto no qual os termos-chave são utilizados, a fim de determinar se o estudo deve ser incluído ou excluído.

A.1.4 Dados a serem extraídos

De cada estudo primário incluído segundo os critérios estabelecidos, foram extraídos os seguintes dados:

- Dados extraídos para multi-projeção na web:
  - Tipo de dispositivo de exibição usado para a multi-projeção;
  - Quantidade de usuários para a qual a aplicação foi projetada;
  - Design geral da solução para a multi-projeção;
  - Se usa CG, como os nós trocam dados;
  - Recurso utilizado para multi-projeção; e
  - Razão da escolha do recurso.
- Dados extraídos para dispositivos de intereção na web:
  - Uso do dispositivo de interação;
  - Se a solução e os recursos utilizados suportam tipos diferentes de dispositivo de interação;
  - Tipo de dispositivo de interação usado;
  - Design geral da solução para uso do dispositivo de interação no navegador;
  - Recurso utilizado para obter os dados do dispositivo;
  - Recurso utilizado para transmitir os dados ao navegador; e
  - Razão para escolha dos recursos.
- Dados extraídos comuns as buscas:
  - Base em que foi encontrado;
  - Ano da publicação; e
  - Domínios do conhecimento focados.

Para o item *Domínios do conhecimento focados*, foram considerados os mesmos domínios do conhecimento utilizados no estudo de Takala, Rauhamaa e Takala (2012), salvo pelos seguintes ajustes:

- O domínio "Visualização" foi estendido para "Visualização e Interação" para incluir estudos abordando visualizações, meios de interação, RV e RA; e
- Foi incluído o domínio "Desenvolvimento de Software", para estudos abordando novas técnicas, soluções ou recursos para o desenvolvimento de software (e.g., Rädle et al. (2014), Marrinan et al. (2014)).

A.2 Condução

A partir dos termos-chave definidos na Subseção A.1.1 Objetivos e Questões, duas lógicas de buscas sofisticadas podem ser formadas, através da utilização dos conectores lógicos `AND` e `OR` (KITCHENHAM; CHARTERS, 2007). Essas lógicas permitem realizar as buscas por estudos primários abordando a multi-projeção e o uso de dispositivos de interação na web.

Para a busca sobre o uso de multi-projeção por aplicações web, os termos-chave foram combinados na seguinte lógica de busca, denominada *SS1*[3]:

("multi-projection" OR "multi display" OR "multiple display" OR "multiple screen" OR "graphic cluster" OR "graphics cluster" OR "graphical cluster" OR cave OR powerwall OR "power wall") AND (javascript OR html5 OR "html 5").

Os dados e resultados dessa busca para cada base científica são mostrados na Tabela 6.

Tabela 6 – Dados e resultados para a lógica de busca *SS1*.

Base científica	Data	Encontrados	Inclusos
IEEE Xplore	28/04/2018	170	9
ACM Digital Library	28/04/2018	0	0
Scopus	28/04/2018	123	4
	Total	294	13

Fonte – Produzida pelo autor.

A respeito dos resultados na base Scopus, ao todo, foram encontrados, na verdade, 128 resultados; no entanto, 5 desses resultados também foram encontrados nas outras bases, portanto, eles foram deduzidos deste total.

Quanto a busca sobre o uso de dispositivos de interação diversificados por aplicações web, os termos-chave foram combinados na seguinte lógica de busca, denominada *SS2*[4]:

("input device" OR "input devices" OR "interaction device" OR "interaction devices") AND javascript AND browser

Os dados e resultados das buscas realizadas em cada base científica são mostrados na Tabela 7.

Novamente, a base Scopus retornou resultados já encontrados nas outras bases. No caso, foram encontrados, ao todo, 6 resultados, sendo que 2 deles já constavam nas outras bases.

[3] A função de busca da base ACM Digital Library possui uma sintaxe diferente das outras bases. Nesse caso, a lógica de busca *SS1* é traduzida para *+("multi-projection" "multi display" "multiple display" "multiple screen" "graphic cluster" "graphics cluster" "graphical cluster" cave powerwall "power wall") +(javascript html5 "html 5")*

[4] A função de busca da base ACM Digital Library possui uma sintaxe diferente das outras bases. Nesse caso, a lógica de busca *SS2* é traduzida para: *+("input device" "input devices" "interaction device" "interaction devices") +javascript +browser*

Tabela 7 – Dados e resultados para a lógica de busca *SS2*.

Base científica	Data	Encontrados	Inclusos
IEEE Xplore	28/04/2018	161	22
ACM Digital Library	28/04/2018	2	2
Scopus	28/04/2018	4	2
	Total	167	26

Fonte – Produzida pelo autor.

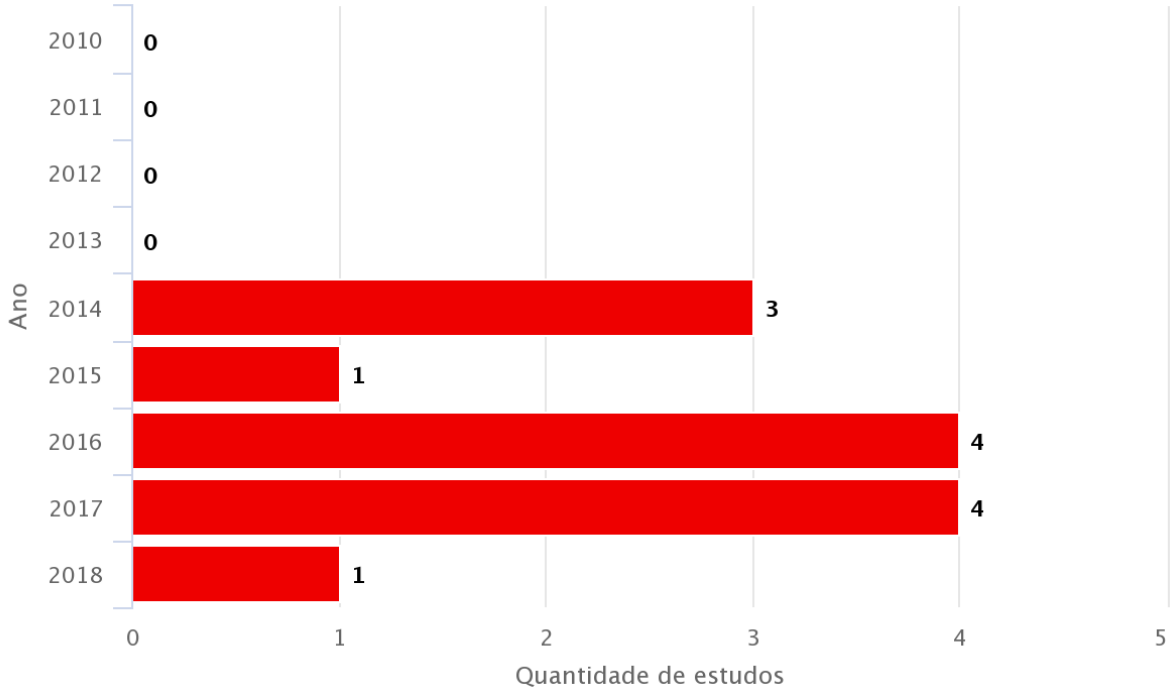
A.3 Resultados

Para melhor apresentação dos resultados, cada busca realizada será abordada em sua própria subseção.

A.3.1 Multi-Projeção em Navegadores Web

A Figura 45 mostra, para cada ano desde 2010, a quantidade de estudos primários encontrados que abordam a multi-projeção em navegadores web. Observa-se que, segundo a busca realizada, estudos sobre o tema só começaram em 2014 e que ainda há relativamente poucos deles.

Figura 45 – Anos com estudos primários sobre multi-projeção na web.

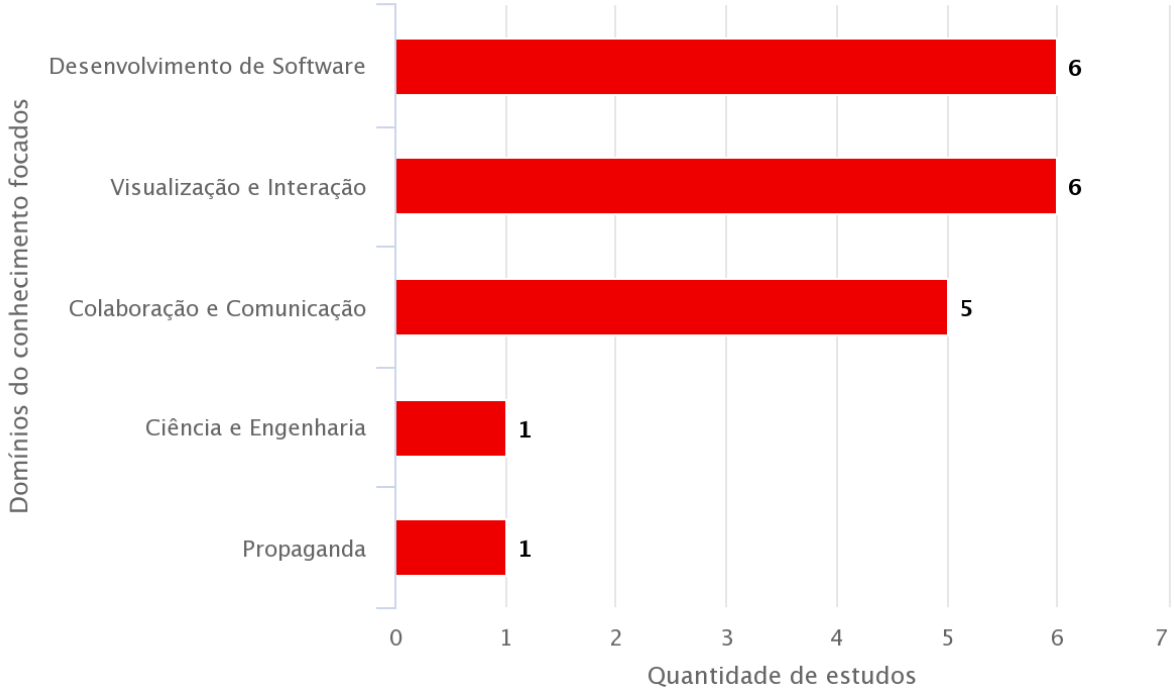


Fonte - Produzida pelo autor.

A Figura 46 mostra os domínios do conhecimento focados pelos estudos primários encontrados na busca; cada estudo focou em, pelo menos, um deles. Observa-se que os domínios mais frequentemente focados são “Desenvolvimento de Software” (MARRINAN et al., 2014;

SCHWEDE; HERMANN, 2015; GEYMAYER; SCHMALSTIEG, 2016; SU et al., 2016; GRUBERT; KRANZ, 2017; GUIMARÃES et al., 2018) e "Visualização e Interação" (SCHWEDE; HERMANN, 2015; MARAI; FORBES; JOHNSON, 2016; REN et al., 2016; SU et al., 2016; GRUBERT; KRÄNZ, 2017; KIM et al., 2017).

Figura 46 – Domínios do conhecimento focados pelos estudos que utilizam multi-projeção na web.



Fonte - Produzida pelo autor.

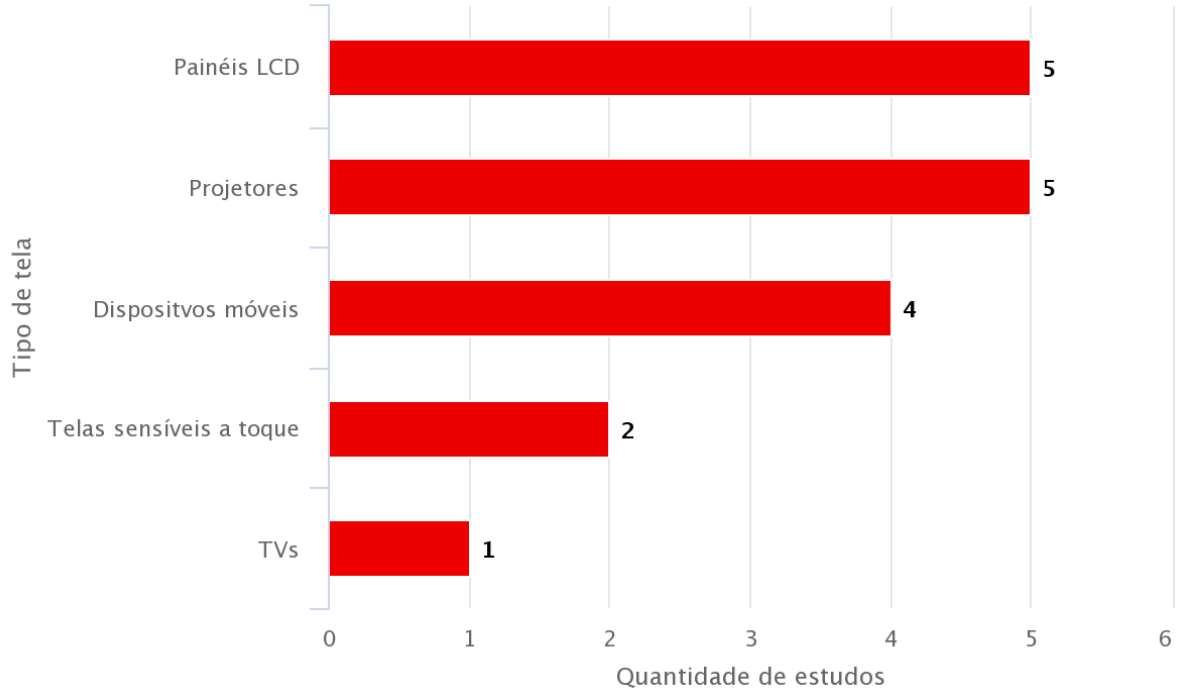
A Figura 47 mostra os tipos de dispositivos de exibição que os estudos primários encontrados utilizaram para multi-projeção; cada estudo utilizou, pelo menos, um desses tipos. Observa-se que projetores (CHUNG et al., 2014; SCHWEDE; HERMANN, 2015; REN et al., 2016; KIM et al., 2017; GUIMARÃES et al., 2018) e painéis LCD (MARRINAN et al., 2014; GEYMAYER; SCHMALSTIEG, 2016; MARAI; FORBES; JOHNSON, 2016; SU et al., 2016; KIM et al., 2017) são os tipos de exibição mais comuns, o que já era razoavelmente esperado.

A Figura 48 mostra as quantidades de usuários para os quais os sistemas/aplicações/recursos desenvolvidos nos estudos primários encontrados foram projetados. Dado o espaço ocupado por sistemas de multi-projeção, era esperado que a maioria dos estudos projetasse sistemas/aplicações/recursos para vários usuários, o que, de fato, acontece.

A Figura 49 mostra os meios (generalizados) utilizados pelos estudos primários encontrados para distribuir a imagem para o sistema de multi-projeção. Observa-se que a opção mais comum foi uma abordagem a qual esta dissertação se refere como "cluster descentralizado"[5] (CHUNG et al., 2014; RÄDLE et al., 2014; GEYMAYER; SCHMALSTIEG, 2016; GRUBERT; KRANZ, 2017; GRUBERT; KRÄNZ, 2017), seguida por clusters cliente-servidor (MARRINAN

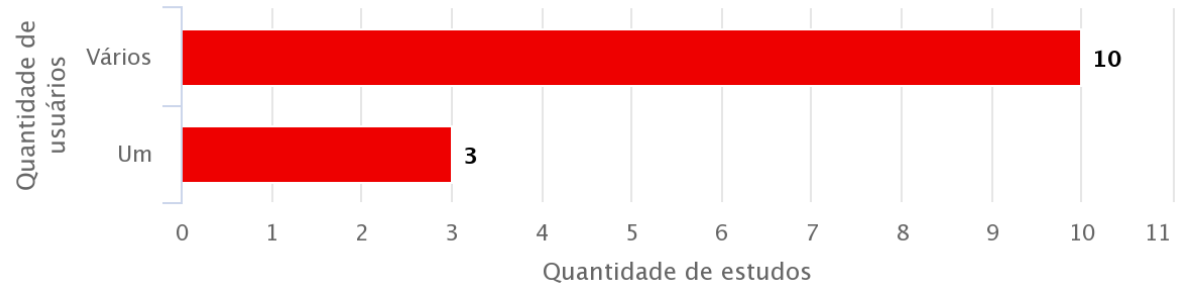
[5] Os estudos primários, em si, não usam especificamente a expressão "cluster descentralizado" para se referir a técnica que utilizaram, todavia, ela foi inferida com base nas descrições fornecidas para o funcionamento dos sistemas, que compartilhavam vários, senão todos, aspectos funcionais.

Figura 47 – Tipos de exibição utilizadas para multi-projeção na web.



Fonte - Produzida pelo autor.

Figura 48 – Quantidades de usuários para as quais foi usada multi-projeção na web.

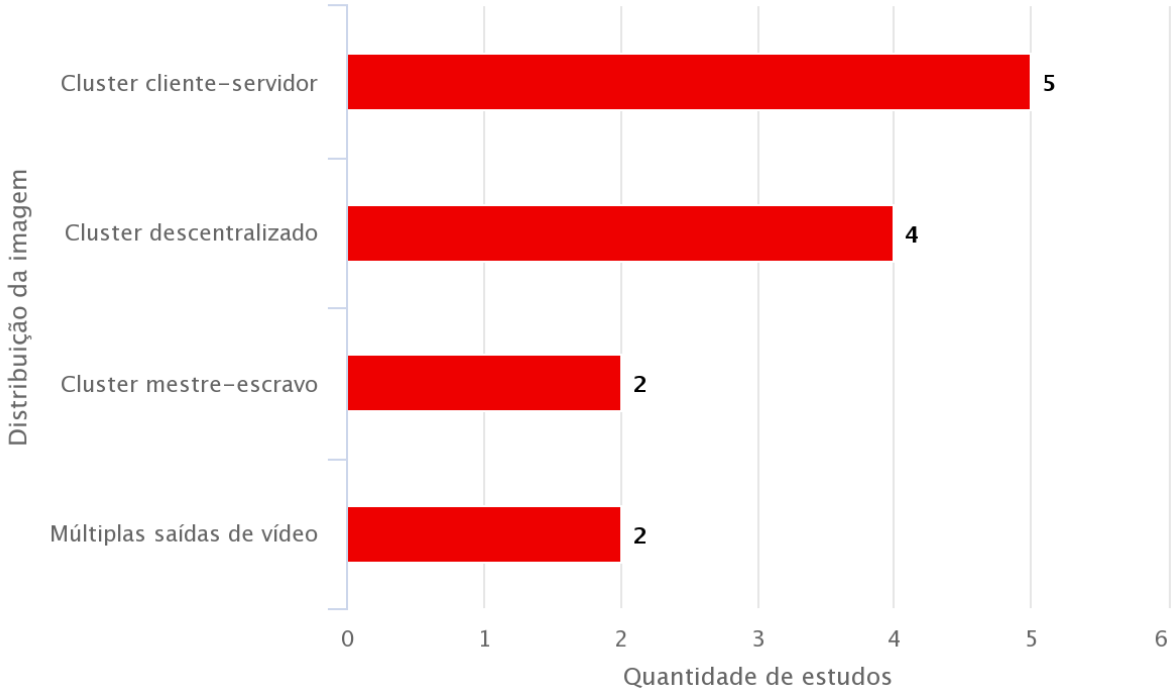


Fonte - Produzida pelo autor.

et al., 2014; MARAI; FORBES; JOHNSON, 2016; REN et al., 2016; SU et al., 2016). Um CG descentralizado tem funcionamento similar ao mestre-escravo, onde todos os nós executam a aplicação, porém, nesse caso, todos os nós também recebem entrada do usuário (ao invés de somente o mestre), o que permite que os usuários da aplicação interajam com qualquer um dos dispositivos computacionais (ou de exibição) exibindo a aplicação.

A Figura 50 mostra as abordagens utilizadas pelos estudos primários encontrados para trocar dados entre os nós executando a aplicação. Observa-se que a escolha mais comum é ter o servidor de aplicação intermediando a comunicação entre os nós, os quais se conectam com o servidor por WebSockets (MARRINAN et al., 2014; MARAI; FORBES; JOHNSON, 2016; SU et al., 2016; RÄDLE et al., 2014; KIM et al., 2017). Navegadores web não oferecem recursos para estabelecer conexões WebSockets entre duas páginas web, assim, no estudo em que isso acontece, o recurso desenvolvido requer um *plugin* no navegador (GEYMAYER; SCHMALSTIEG, 2016).

Figura 49 – Meios utilizados para distribuir imagem para sistema de multi-projeção na web.



Fonte - Produzida pelo autor.

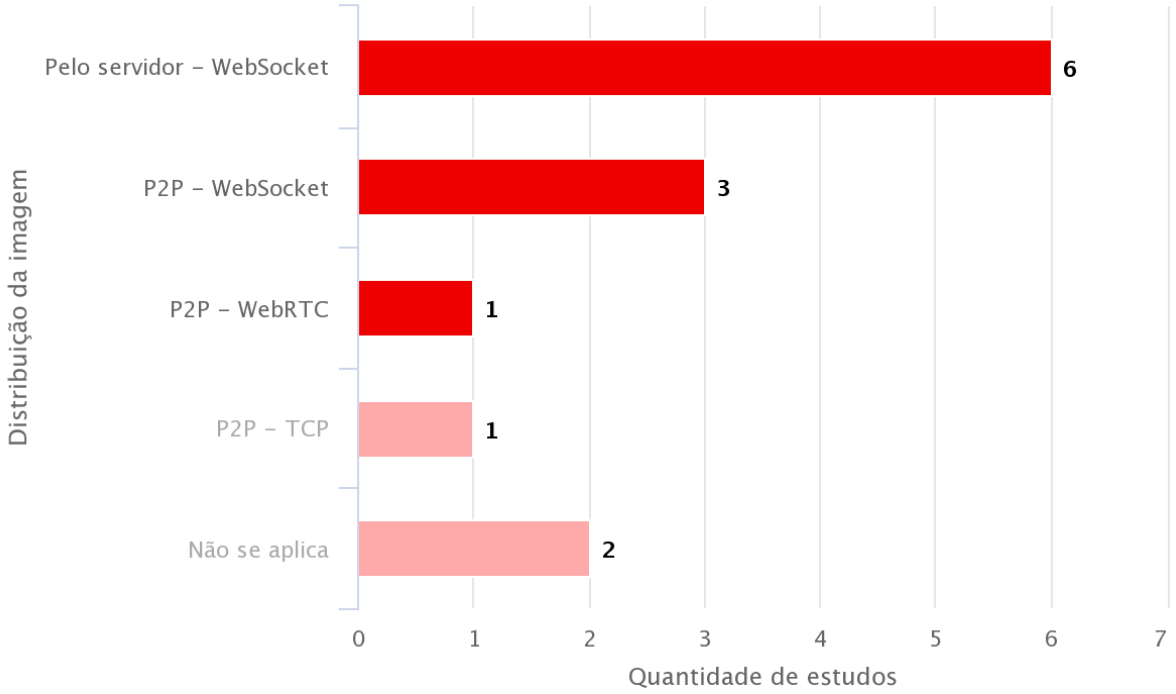
No caso de conexão P2P por TCP (REN et al., 2016), tem-se um cenário onde o usuário acessa a aplicação de um dispositivo próprio (e.g., notebook, tablet) e há um CG separado, implementado com outras tecnologias possivelmente não web, responsável por renderizar e exibir o conteúdo. Os casos "Não se aplica" se referem aos estudos que utilizaram múltiplas saídas de vídeo (SCHWEDE; HERMANN, 2015; PATTANAKIMHUN; CHINTHAMMIT; CHOTIKAKAMTHORN, 2017), onde havia uma única instância de navegador web esticada por todas os dispositivos de exibição ligados ao computador.

A Figura 51 mostra os recursos utilizados nos estudos primários encontrados para implementar a multi-projeção. A grande maioria (CHUNG et al., 2014; REN et al., 2016; GEYMAYER; SCHMALSTIEG, 2016; GRUBERT; KRANZ, 2017; GRUBERT; KRÄNZ, 2017; KIM et al., 2017; GUIMARÃES et al., 2018) não usou recurso específico algum, optando por implementar essa funcionalidade por si mesmos e não a tornando disponível a comunidade.

Dada a Figura 51, é de se esperar que sejam poucas as razões encontradas nos estudos primários encontrados para a utilização de algum recurso específico para implementar a multi-projeção em navegadores web. De fato, somente Marai, Forbes e Johnson (2016) e Su et al. (2016) ofereceram razões para a escolha do recurso utilizado, as quais podem ser resumidas, respectivamente, como:

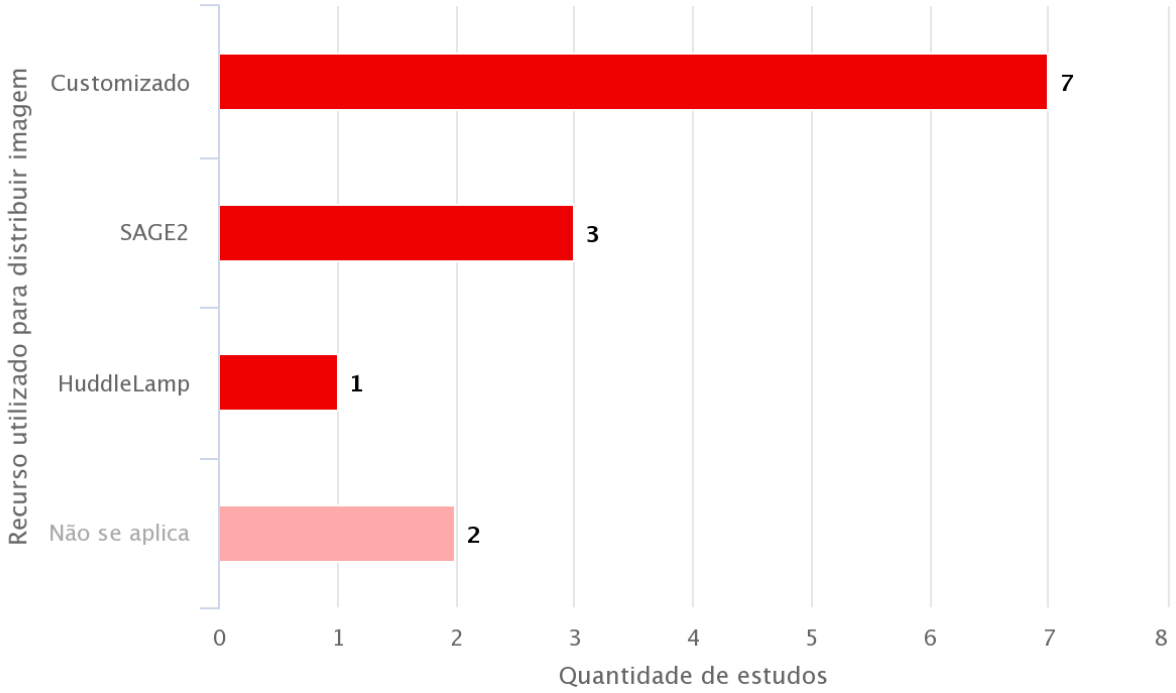
- Possibilidade de usar bibliotecas JavaScript em aplicações de multi-projeção; e
- Funcionalidades oferecidas pelo recurso, que facilitam a implementação.

Figura 50 – Meios utilizados para trocar dados em sistemas de multi-projeção na web.



Fonte - Produzida pelo autor.

Figura 51 – Recursos utilizados para trabalhar com multi-projeção na web.



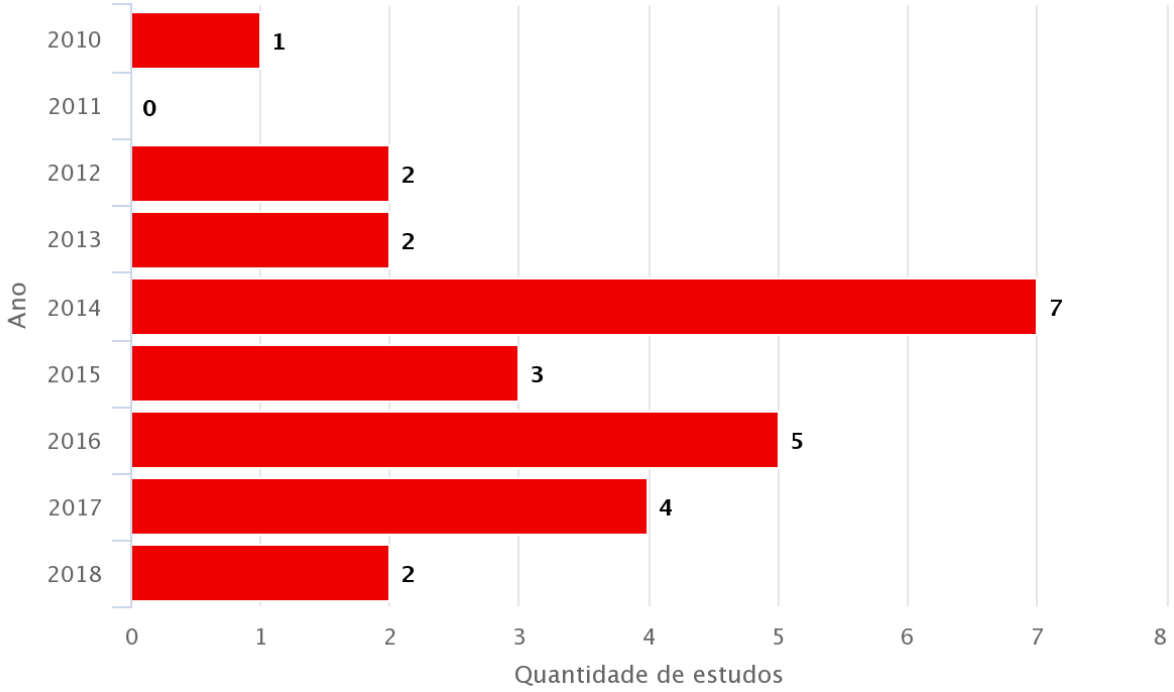
Fonte - Produzida pelo autor.

A.3.2 Dispositivos de Interação em Navegadores Web

A Figura 52 mostra, para cada ano desde 2010, a quantidade de estudos primários encontrados que abordam a multi-projeção em navegadores web. Observa-se que, segundo a busca

realizada, estudos sobre o tema existem, pelo menos, desde 2010 e que, desde então, embora o número de estudos tenha, no geral, crescido, ele também tem flutuado bastante.

Figura 52 – Anos com estudos primários sobre dispositivos de interação em navegadores web.

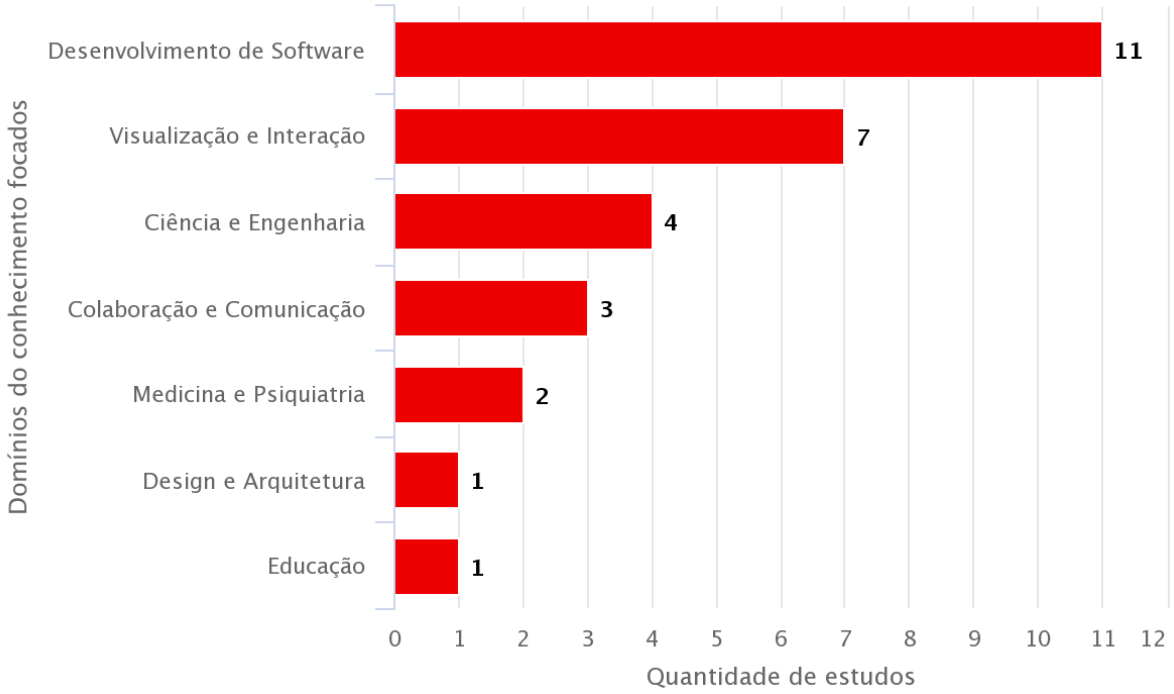


Fonte - Produzida pelo autor.

A Figura 53 mostra os domínios do conhecimento focados pelos estudos primários encontrados na busca; cada estudo focou em, pelo menos, um deles. Observa-se que, novamente, os domínios mais frequentemente focados são, em ordem, "Desenvolvimento de Software" (HAK; DOLEZAL; ZEMAN, 2012; GAO; OSMAN; SADDIK, 2013; LIU et al., 2013; MARRINAN et al., 2014; MORENO et al., 2015; ASSAL; CHIASSON; BIDDLE, 2016; ŞERBAN; CULIC, 2016; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017; SATHE et al., 2017; DIPIERRO, 2018) "Visualização e Interação" (MOTTA; NEDEL, 2012; OAT; FRANCESCO; AURA, 2014; MAEDA; KOBAYASHI, 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; PAKKANEN et al., 2017; SEO; LEE; YOO, 2017; MEIXNER; KALLMEIER, 2016).

A Figura 54 mostra os tipos de funções dadas aos dispositivos de interação pelos estudos primários incluídos; cada estudo deu, pelo menos, uma dessas funções ao(s) dispositivo(s) que usou. Observa-se que a função mais comum é "manipulação" (GHATAK et al., 2014; RAHMAN et al., 2014; DO; CAI; JIANG, 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; ASSAL; CHIASSON; BIDDLE, 2016; PAKKANEN et al., 2017; HAN et al., 2017; SANFILIPPO et al., 2017; DIPIERRO, 2018; SEO; LEE; YOO, 2017; MEIXNER; KALLMEIER, 2016; WILSON; BROWN; BIDDLE, 2014), que consiste no uso do dispositivo para alterar atributos do conteúdo sendo exibido no navegador. O uso "programável" se refere a estudos que permitem programar interações para os dados do dispositivo pelo recurso que apresentam (HAK; DOLEZAL; ZEMAN, 2012; GAO; OSMAN; SADDIK, 2013; LIU et al., 2013; MORENO et al., 2015; ŞERBAN; CULIC, 2016; PALLEC et al., 2010) (todos esses estudos

Figura 53 – Domínios do conhecimento focados pelos estudos que utilizam dispositivos de interação em navegadores web.



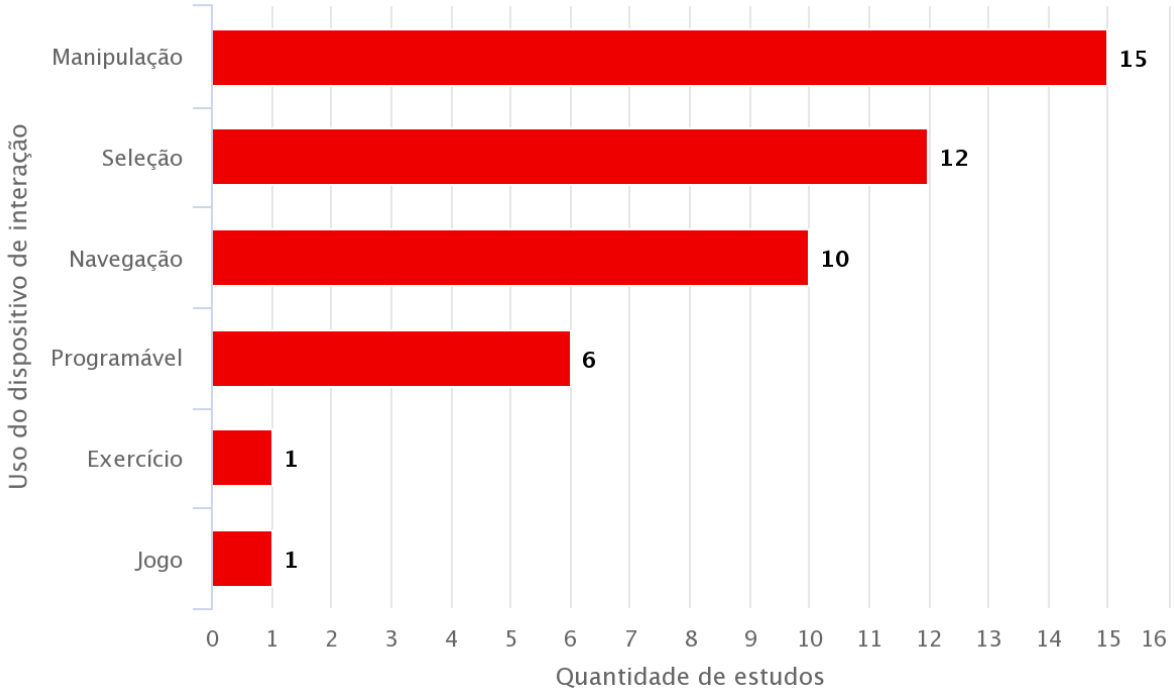
Fonte - Produzida pelo autor.

focam no domínio do conhecimento "Desenvolvimento de Software"). O uso "jogo" se refere ao uso do dispositivo para vencer desafios apresentados num jogo sério (MONSERRAT et al., 2015); o uso "exercício" é semelhante, embora foque em exercícios físicos acompanhados pela aplicação web (KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017).

A Figura 55 mostra o suporte dado pelos recursos utilizados/desenvolvidos pelos estudos primários incluídos para dispositivos de interação além daqueles nos quais focam. Observa-se que a grande maioria não suporta outros tipos de dispositivos de interação (HAK; DOLEZAL; ZEMAN, 2012; MOTTA; NEDEL, 2012; GAO; OSMAN; SADDIK, 2013; LIU et al., 2013; DO; CAI; JIANG, 2014; MONSERRAT et al., 2015; FITTKAU; KRAUSE; HASSELBRING, 2015; MORENO et al., 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; ASSAL; CHIASSON; BIDDLE, 2016; PAKKANEN et al., 2017; SATHE et al., 2017; WILSON; BROWN; BIDDLE, 2014). A opção "limitado" indica que o suporte é dado a tipos específicos de outros dispositivos (OAT; FRANCESCO; AURA, 2014; MAEDA; KOBAYASHI, 2014; GHATAK et al., 2014; RAHMAN et al., 2014; ŞERBAN; CULIC, 2016; DIPIERRO, 2018; MEIXNER; KALLMEIER, 2016) (e.g., estudo usa *webcam*, mas recursos suportam somente mais microfone).

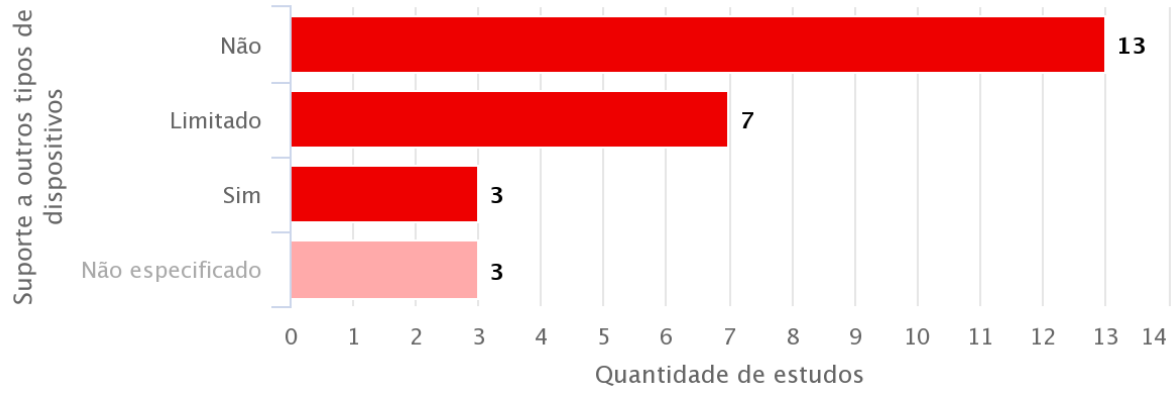
Nos 26 estudos primários encontrados que usam dispositivos de interação em navegadores web, foram utilizados, ao todo, 36 dispositivos, e a Figura 56 mostra quantas vezes cada tipo de dispositivo foi utilizado nos estudos. Observa-se que os dispositivos mais comumente usados usado nos estudos primários encontrados foram sensores de movimento (e.g., Microsoft Kinect, Asus Xtion) (MOTTA; NEDEL, 2012; MARRINAN et al., 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; MORENO et al., 2015; KONSTANTINIDIS; BAMPAROPOULOS;

Figura 54 – Usos dados para os dispositivos de interação em navegadores web.



Fonte - Produzida pelo autor.

Figura 55 – Suporte a múltiplos tipos de dispositivos de interação em navegadores web.

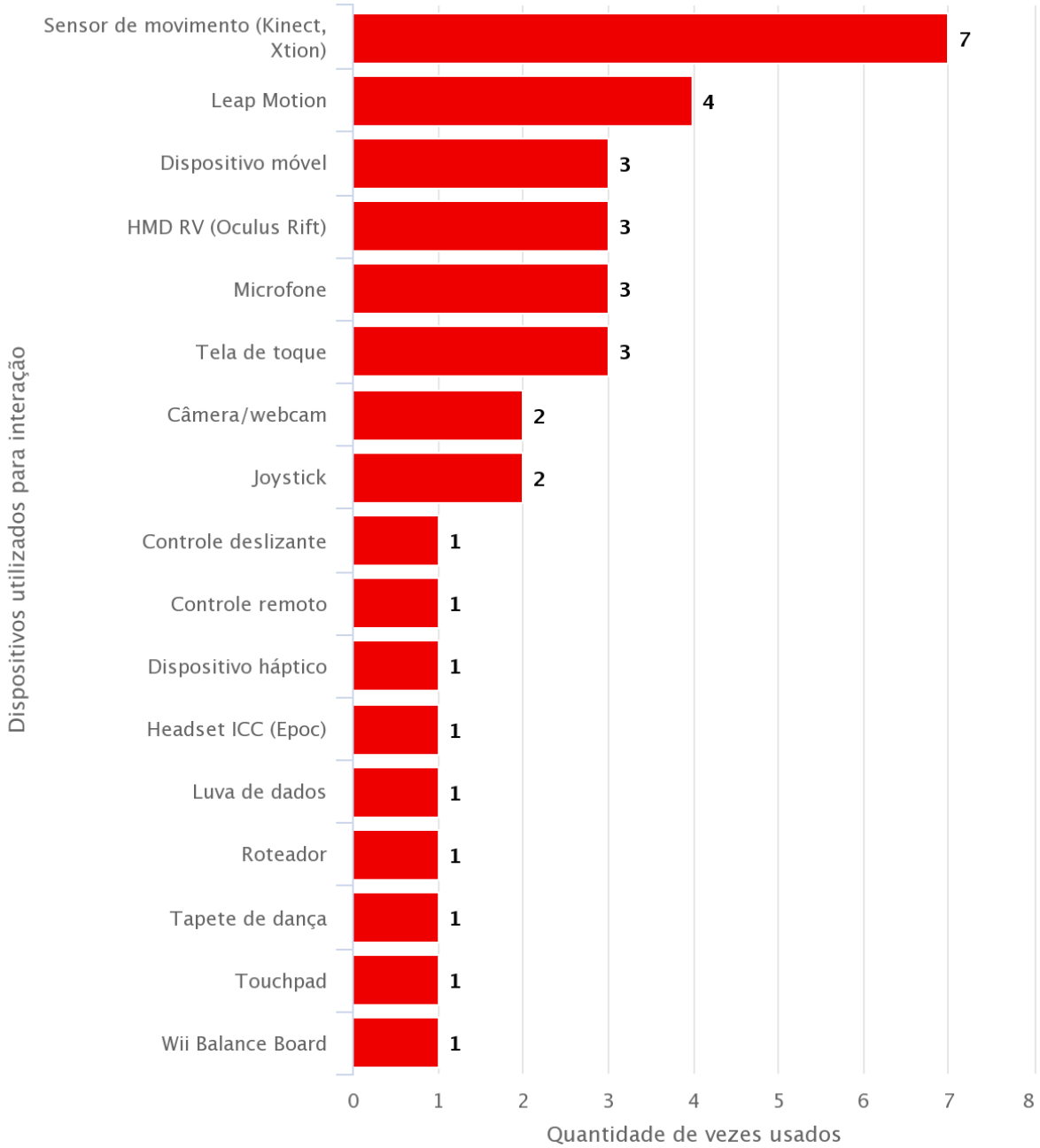


Fonte - Produzida pelo autor.

BAMIDIS, 2017) e o Leap Motion (DO; CAI; JIANG, 2014; RITTITUM; VATANAWOOD; THONGTAK, 2016; PAKKANEN et al., 2017; SEO; LEE; YOO, 2017), que, tecnicamente, também é sensor de movimento, embora, ao contrário dos outros, seja limitado as mãos, ao invés do corpo inteiro.

Em relação aos 36 dispositivos de interação usados, a Figura 57 mostra os meios (generalizados) utilizados pelos estudos primários incluídos para fazer com que aplicações web pudessem utilizar dados deles. Observa-se que, por uma margem substancial, a abordagem mais usada é *dispositivo-servidor-navegador* (MOTTA; NEDEL, 2012; DO; CAI; JIANG, 2014; MARRINAN et al., 2014; MONSERRAT et al., 2015; MORENO et al., 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017; SANFILIPPO

Figura 56 – Tipos de dispositivos de interação usados para interação em navegadores web.

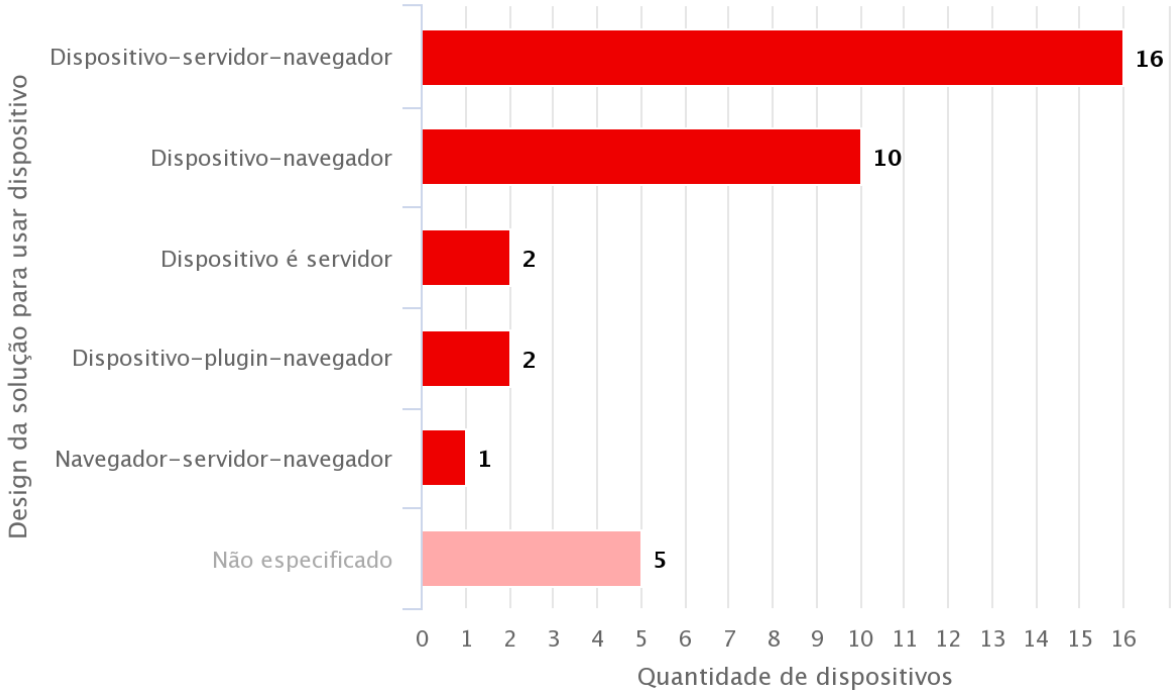


Fonte - Produzida pelo autor.

et al., 2017; SEO; LEE; YOO, 2017; PALLEC et al., 2010), que consiste em ter o dispositivo conectado a um computador em que é executado um servidor de aplicação; esse servidor obtém os dados do dispositivo e os serve para os clientes (navegadores web). As outras abordagens foram:

- *Dispositivo-navegador* (GHATAK et al., 2014; RAHMAN et al., 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; ASSAL; CHIASSON; BIDDLE, 2016; PAKKANEN et al., 2017; SATHE et al., 2017; DIPIERRO, 2018; MEIXNER; KALLMEIER, 2016; WILSON; BROWN; BIDDLE, 2014): o navegador é capaz de usar os dados do dispositivo dire-

Figura 57 – Designs das soluções para uso de dispositivos de interação em navegadores web.



Fonte - Produzida pelo autor.

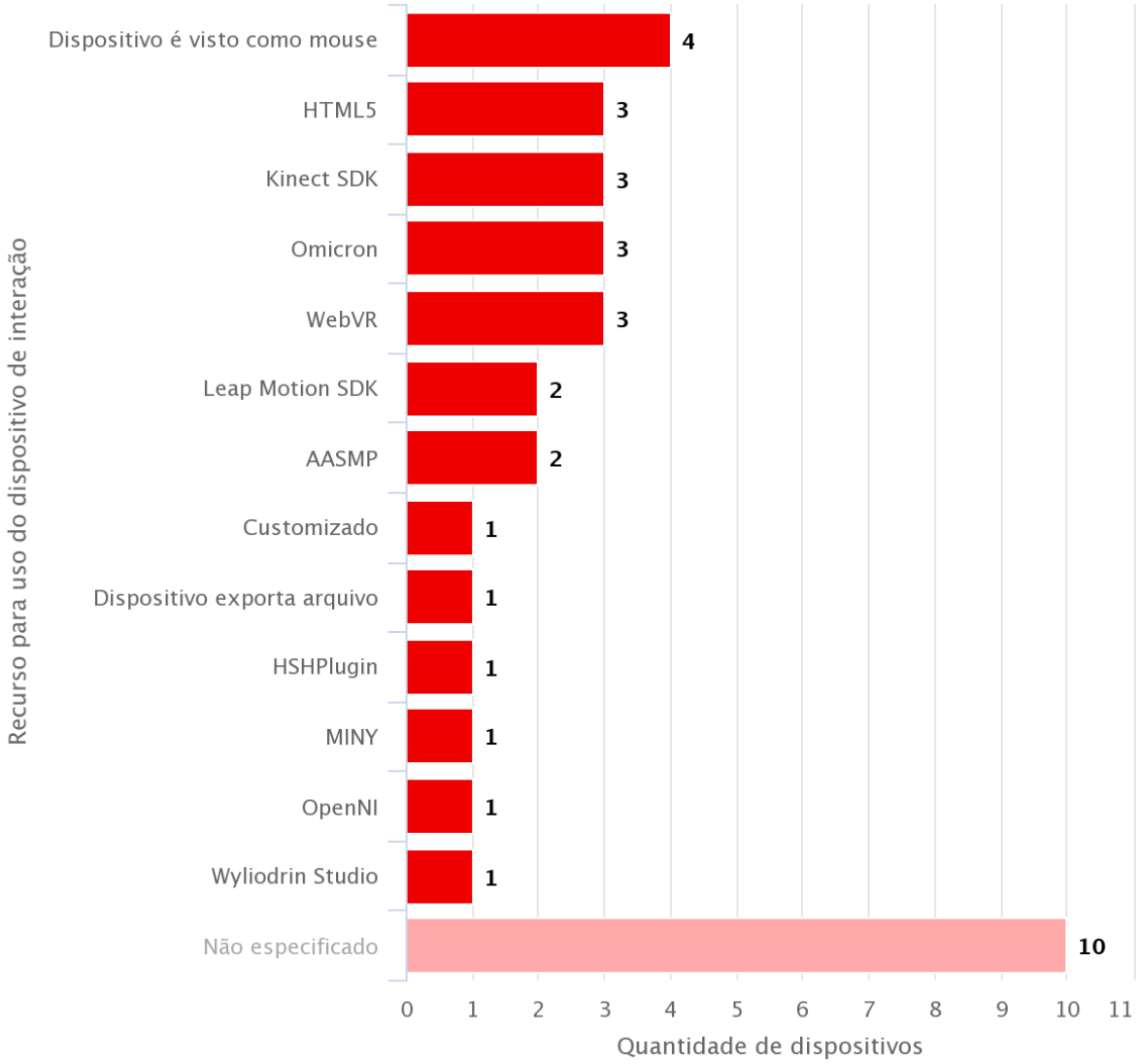
tamente, sem necessidade de intermediário;

- *Dispositivo-plugin-navegador* (HAK; DOLEZAL; ZEMAN, 2012; GAO; OSMAN; SADDIK, 2013): é adicionado *plugin* ao navegador, o qual lhe possibilita adquirir dados de um dispositivo de interação conectado ao computador;
- *Navegador-servidor-navegador* (OAT; FRANCESCO; AURA, 2014): dispositivo de interação é capaz de executar navegador web, com o qual envia seus dados para servidor de aplicação, que os serve para os clientes; e
- Dispositivo é servidor: o próprio dispositivo de interação é um servidor de aplicação, capaz de servir seus dados aos clientes.

A Figura 58 mostra os recursos utilizados pelos estudos primários incluídos para se obter os dados dos dispositivos de interação. Observa-se que a abordagem mais comum foi fazer com que o dispositivo fosse entendido como um mouse (MAEDA; KOBAYASHI, 2014; RAHMAN et al., 2014; ASSAL; CHIASSON; BIDDLE, 2016; WILSON; BROWN; BIDDLE, 2014), o qual é naturalmente suportado por SOs e navegadores web. A abordagem "dispositivo exporta arquivo" (HAK; DOLEZAL; ZEMAN, 2012) consiste na aplicação ser capaz de processar arquivos, gerados por ferramentas diversas, descrevendo uma entrada fornecida pelo dispositivo.

Por outro lado, a Figura 59 mostra os recursos utilizados para transmitir/repassar os dados obtidos pelos recursos anteriores para os navegadores web. Observa-se que a abordagem mais comum é o desenvolvimento de ferramentas próprias e customizadas (MOTTA; NEDEL, 2012; MONSERRAT et al., 2015; MORENO et al., 2015; SANFILIPPO et al., 2017; SEO; LEE;

Figura 58 – Recursos utilizados para obter os dados dos dispositivos de interação usados em navegadores web.

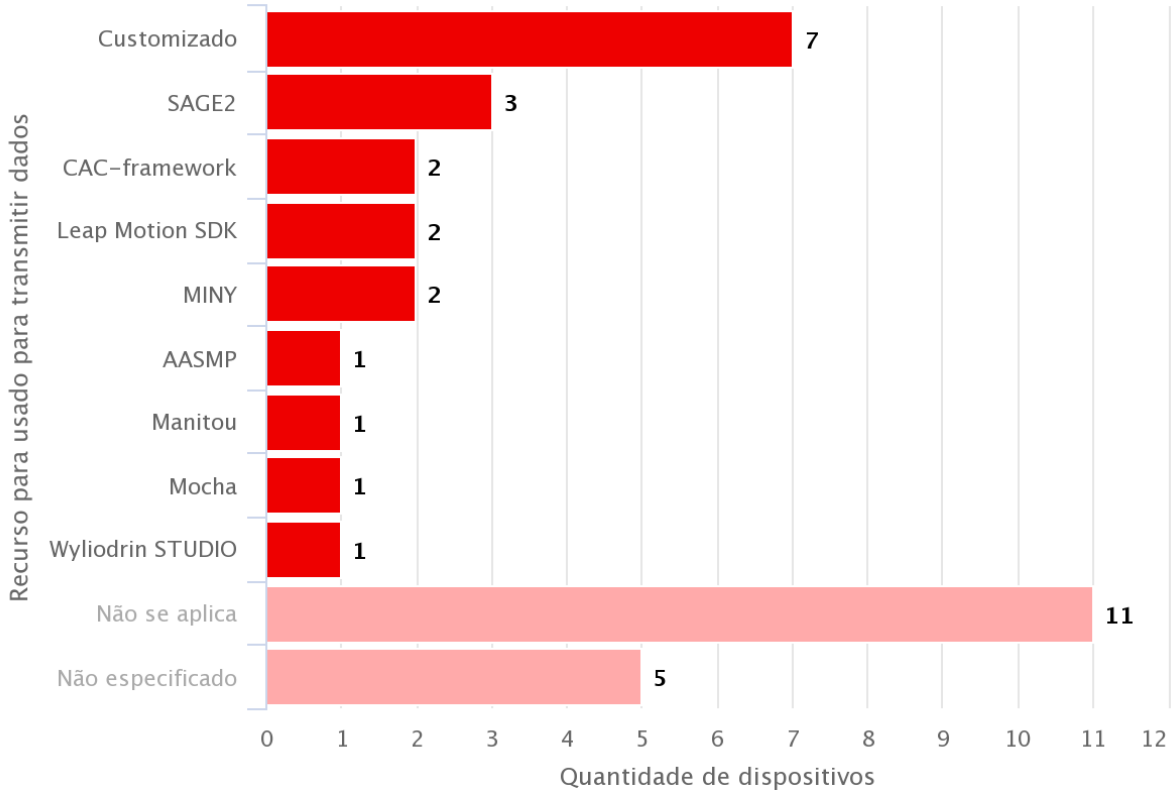


Fonte - Produzida pelo autor.

YOO, 2017). Em "não se aplica", estão inclusos casos em que os dados dos dispositivos são obtidos diretamente no navegador web (GAO; OSMAN; SADDIK, 2013; MAEDA; KOBAYASHI, 2014; GHATAK et al., 2014; RAHMAN et al., 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; ASSAL; CHIASSON; BIDDLE, 2016; PAKKANEN et al., 2017; SATHE et al., 2017; DIPIERRO, 2018; MEIXNER; KALLMEIER, 2016; WILSON; BROWN; BIDDLE, 2014)

Por fim, a Figura 60 mostra as razões citadas entre estudos primários incluídos para a escolha dos recursos que usaram; cada estudo pôde dar nenhuma, uma ou várias razões para o(s) recurso(s) utilizado(s). Observa-se que a razão mais comum envolve as funcionalidades oferecidas pelo recurso escolhido (GHATAK et al., 2014; FITTKAU; KRAUSE; HASSELBRING, 2015; RITTITUM; VATANAWOOD; THONGTAK, 2016; ŞERBAN; CULIC, 2016; KONSTANTINIDIS; BAMPAROPOULOS; BAMIDIS, 2017).

Figura 59 – Recursos utilizados para transmitir dados dispositivos de interação para os navegadores web.



Fonte - Produzida pelo autor.

A.4 Considerações Finais

Os dados de cada estudo primário foram compilados em uma planilha eletrônica[6].

Muitos dos estudos excluídos abordavam temas próximos ao procurado, porém diferentes (e.g., renderização de gráficos de aplicação móvel em PC desktop ou aplicação web que pode ser acessada tanto de PCs desktops quanto tablets) Já outros abordavam assuntos relacionados a biologia e geografia (no caso da lógica de busca *SS1*, certamente devido ao termo "cave" que trouxe estudo que usavam cavernas, ao invés do sistema de multi-projeção CAVE (CRUZ-NEIRA et al., 1992)).

Infelizmente, devido a aparente impossibilidade de articular as lógicas de busca de maneira que excluam estudos primários que abordam estes temas diversos sem excluir, também, estudos primários que abordam o tema procurado, isso já era esperado. Todavia, acredita-se que a quantidade de estudos primários incluídos seja suficiente para que as questões definidas possam ser respondidas.

[6] Planilha Eletrônica com dados coletados: <https://docs.google.com/spreadsheets/d/14rixCjqVNZM3a-3tVCAtMA5XvllzvnpN2roUF3Y2Dng>.

Figura 60 – Razões mencionadas para uso dos recursos para obter e transmitir dados de dispositivos de interação para navegadores web.

Fonte - Produzida pelo autor.

APÊNDICE B – Fundamentação Tecnológica

Este apêndice aborda em maiores detalhes a base tecnológica usada para a implementação do design de solução para aplicações VI3DI web apresentado. Mais especificamente, ele trata da API JavaScript WebRTC, da biblioteca JavaScript Three.js, do ambiente JavaScript de *runtime* Node.js e da biblioteca C/C++ VRPN.

B.1 WebRTC

A solução procurada por essa dissertação envolve o desenvolvimento de aplicações web com CGs, onde é necessário estabelecer comunicação entre vários navegadores web executando instâncias da aplicação. Para que essa comunicação seja eficiente e eficaz, ela deve ser implementada sobre uma base sólida de protocolos e conceitos. A mais recente tecnologia a atender esses requisitos o WebRTC.

No passado, para que comunicação por áudio e/ou vídeo em tempo real fosse possível em navegadores web, era necessária a utilização de *plugins* externos. Desde então, visando eliminar essa dependência, novos padrões surgiram, de maneira a suportar dispositivos, *codecs* e comunicação diretamente pelos navegadores web, por meio de novos recursos oferecidos pela especificação HTML5 (SINGH; KRISHNASWAMY, 2013; NURMINEN et al., 2013).

Esses padrões vem sendo trabalhados pelas duas principais entidades de padronização da Internet, *World Wide Web Consortium* (W3C; do inglês, "Consórcio Web Mundial") e *Internet Engineering Task Force* (IETF; do inglês, "Força Tarefa de Engenharia da Internet") (LORETO; ROMANO, 2012). O W3C trabalha numa API JavaScript que visa possibilitar que aplicações executadas em navegadores web, para qualquer tipo de dispositivo, enviem e recebam dados e mídias em tempo real, numa conexão tipo-P2P (World Wide Web Consortium, 2017b); nessa API, a interface com a rede é montada sobre protocolos de comunicação elaborados pela IETF (Internet Engineering Task Force, 2017).

A API JavaScript é denominada *Web Real-Time Communication* (WebRTC; do inglês, "Comunicação em Tempo-Real na Web"), nome compartilhado com o grupo dentro da W3C trabalhando em sua especificação. Ela visa estabelecer um caminho fim-a-fim para transmissão de mídia e dados entre duas páginas web, com pouca ou nenhuma interferência de outra entidade (JOHNSTON; YOAKUM; SINGH, 2013; SINGH; KRISHNASWAMY, 2013).

WebRTC fornece várias funções para permitir que aplicações sendo executadas nos navegadores web se conectem diretamente, sem necessidade de um servidor de aplicação intermediário. Essas funções devem tratar, por exemplo, de gestão da conexão P2P, capacidades de codificação/decodificação de mídia (áudio/vídeo), negociação, controle de mídia, *firewall* e outros detalhes (LORETO; ROMANO, 2012).

Com seu uso, o caminho da mídia flui diretamente no canal navegador-navegador, permitindo, assim, pela primeira vez, comunicação de dados P2P em aplicações para navegadores web. No entanto, isso acaba por tornar necessário algum tipo de canal de sinalização, que permita com que as instâncias dos navegadores web se encontrem (JOHNSTON; YOAKUM; SINGH, 2013), troquem informações sobre a natureza e qualidade da conexão e dos dados que serão trocados e outros detalhes (LORETO; ROMANO, 2012), antes de iniciar a conexão P2P, em si. A sinalização pode ser controlada por um servidor de aplicação comum (chamado, então, de *servidor de sinalização*), utilizando métodos adicionais de segurança (JOHNSTON; YOAKUM; SINGH, 2013).

WebRTC não apresenta especificações ou definições de como servidores de sinalização devem funcionar (JOHNSTON; YOAKUM; SINGH, 2013; SINGH; KRISHNASWAMY, 2013), deixando os desenvolvedores livres para criarem seus próprios esquemas de sinalização (utilizando, por exemplo, tecnologia de WebSockets[1]) ou utilizar protocolos que servem a necessidades comuns (como, por exemplo, o XMPP (LORETO; ROMANO, 2012)).

B.1.1 Detalhes Técnicos

A API do WebRTC oferece várias classes para permitir trabalhar com seus recursos.

A classe principal do WebRTC é `RTCPeerConnection`, que representa a conexão com um navegador web remoto. Inicialmente, a comunicação é coordenada por um canal de sinalização com o servidor de sinalização. Uma vez que a conexão seja firmada, os dados navegam diretamente entre os navegadores, ignorando o servidor (LORETO; ROMANO, 2012).

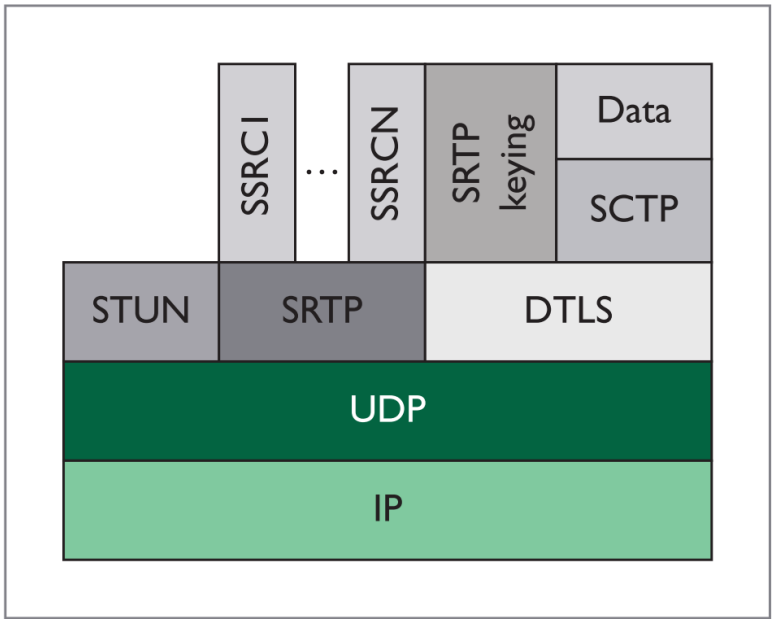
`RTCPeerConnection` utiliza o protocolo *Interactive Connectivity Establishment* (ICE; do inglês, "Estabelecimento de Conectividade Interativa") para descobrir informações a respeito da topologia da rede, a fim de determinar o melhor caminho para os dados (JOHNSTON; YOAKUM; SINGH, 2013); ICE também oferece segurança, prevenindo que os pares conectados troquem os dados com outras entidades (LORETO; ROMANO, 2012). Para transferir dados, são usados servidores *Session Traversal Utilities for NAT* (STUN; do inglês, "Utilitários de Tráfego de Sessão para NAT") e *Traversal Using Relays around NAT* (TURN; do inglês, "Travessia Utilizando *Relays* ao redor de NAT"), que ajudam a rotear os dados através de *firewalls* e caixas NAT (LORETO; ROMANO, 2012). A Figura 61 mostra os protocolos sobre os quais `RTCPeerConnection` opera.

Para o envio/recebimento de dados entre os navegadores, WebRTC oferece duas classes diferentes, `MediaStream` e `DataChannel`, que cuidam da troca de mídias e dados genéricos, respectivamente. No caso da pesquisa realizada nesta dissertação, são trocados apenas dados genéricos.

Dados genéricos tem valor no todo e perdas podem ser catastróficas; dessa maneira, eles necessitam de confiabilidade na entrega, para que a aplicação não tenha informações corrompidas. Como é possível ver na Figura 61, o WebRTC, a fim de diminuir a latência da conexão e oferecer um complemento a comunicação com TCP e HTTP (NURMINEN et al., 2013) (normalmente

[1] WebSockets: <https://tools.ietf.org/html/rfc6455>.

Figura 61 – Protocolos utilizados pelo WebRTC.



Fonte - Loreto e Romano (2012).

utilizada para troca de dados na web), transporta dados sobre o UDP, um protocolo que não oferece confiabilidade (BRADEN, 1989). Sendo assim, a troca de dados genéricos pela classe `DataChannel` permite confiabilidade na entrega dos dados usando protocolos das camadas mais altas da pilha; para tanto, a IETF definiu o uso do *Stream Control Transmission Protocol* (SCTP; do inglês, "Protocolo de Transmissão de Controle de *Stream*"), que permite ajustar o nível necessário de confiabilidade e transportes paralelos de mídia (LORETO; ROMANO, 2012).

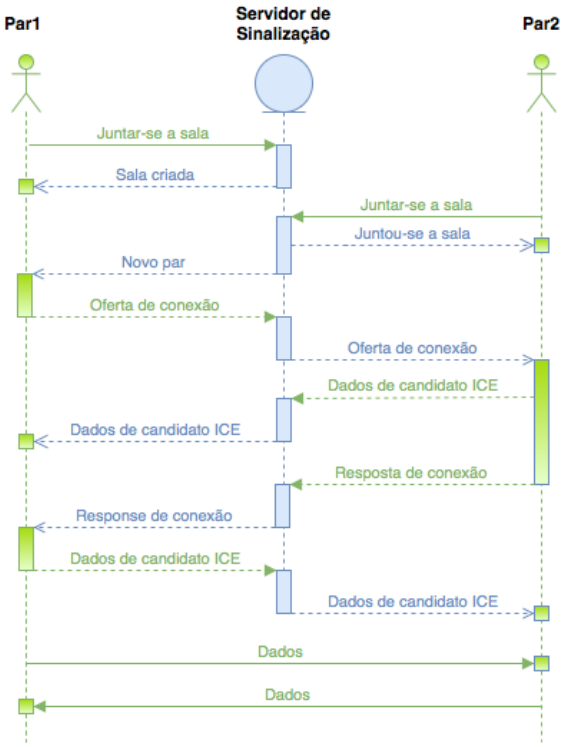
A Figura 62 mostra um diagrama de sequência de como seria a conexão e troca de dados entre dois navegadores web com WebRTC. Inicialmente, *Par1* se conecta ao servidor de sinalização e cria uma "sala" virtual, esperando os outros pares se conectarem. Após um tempo qualquer, *Par2* se junta a sala virtual e o servidor avisa *Par1* que existe outro par para com o qual se conectar. O *Par1* cria uma oferta de conexão e a envia para o servidor, que repassa ao *Par2*. O *Par2* transmite ao *Par1*, através do servidor, suas informações de ICE e, em seguida, a resposta a oferta de conexão, também pelo servidor. Por fim, o *Par1* passa a suas informações de ICE para o *Par2*, através do servidor, e, agora, ambos *Par1* e *Par2* estão conectados um ao outro, podendo trocar dados diretamente, sem intermediários.

É importante notar que o conceito de "sala virtual", na verdade, não é necessário para o estabelecimento da conexão pelo WebRTC. Todavia, ela é particularmente útil, pois permite segregar, no servidor, grupos de nós que estão estabelecendo conexão e, com isso, identificar, para cada um deles, quais outros são candidatos viáveis para conexão.

B.1.2 Considerações

Por WebRTC ainda ser relativamente novo, ainda existem navegadores web e dispositivos (em particular, de natureza móvel) que não o suportam. De fato, no momento, isso é algo a se considerar, pois limita os tipos de navegadores web e dispositivos nos quais as aplicações web

Figura 62 – Diagrama de sequência de dois pares trocando dados por WebRTC.



Fonte - Produzida pelo autor.

que fazem uso do WebRTC pode ser executadas.

Todavia, dadas as possibilidades de comunicação abertas pelo WebRTC, acredita-se que seja mera questão de tempo até que as entidades responsáveis pelos navegadores web e dispositivos sem suporte a WebRTC o implementem. Com essa visão, embora sua utilização neste momento não ofereça a melhor compatibilidade possível, ela ainda oferece benefícios únicos e que, acredita-se logo, não terão argumentos contra.

B.2 Node.js

Para o emprego adequado do WebRTC, é necessária a existência de um servidor de sinalização, coordenando a comunicação entre os nós, até que eles possam se comunicar sem intermediários.

Servidores de aplicação devem servir múltiplas requisições de maneira eficiente e concorrente (MCCUNE, 2011; CHANIOTIS; KYRIAKOU; TSELIKAS, 2015). Atender a essas requisições muitas vezes envolve operações bloqueantes de E/S ou de rede (e.g., abrir arquivo, aguardar dados de um *socket*) que independem da velocidade do processador e são consideravelmente lentas, assim, não é adequado tratar cada requisição individualmente e consecutivamente, a medida que chegam. Para tratar esse problema, é possível utilizar duas abordagens (LAUER; NEEDHAM, 1979): orientada a procedimento e orientada e evento.

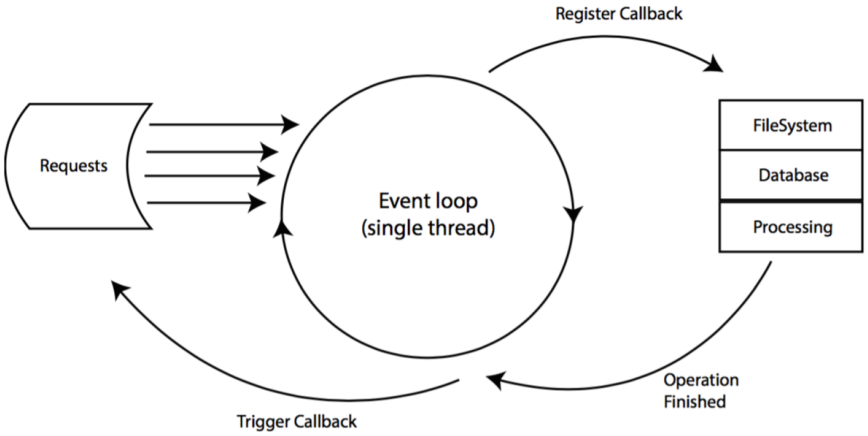
Na abordagem orientada a procedimento, há o equivalente ao uso de *threads*[2], onde o

[2] *Thread*: sequência de instruções gerenciadas independentemente.

servidor cria uma *thread* separada para atender e executar cada requisição (MCCUNE, 2011). Quando operações de E/S ou de rede são chamadas numa *thread*, o processador troca seu contexto de execução para outra *thread*, enquanto a operação é finalizada (TILKOV; VINOSKI, 2010). Embora essa seja uma solução óbvia e amplamente suportada para o problema (MCCUNE, 2011), desenvolver aplicações com múltiplas *threads* pode introduzir várias questões (e.g., sincronização de acesso aos dados e de dependências entre *threads*) de difícil compreensão e controle para os desenvolvedores e que, portanto, tornam a aplicação mais complexa (OUSTERHOUT, 1996).

A alternativa, orientação a eventos/mensagens, consiste na aplicação registrar seu "interesse" em determinados tipos de eventos. Quando o evento ocorre, uma mensagem é enviada a aplicação, para que ela possa tratar o evento (OUSTERHOUT, 1996). O processamento das mensagens ocorre em funções chamadas *callback*, e não bloqueia a execução de um recurso que já está em uso (MCCUNE, 2011). Essa abordagem permite um uso mais eficiente dos recursos, o que é traduzido como maior resistência a certos tipos de ataques (e.g., negação de serviço) (OJAMAA; DUUNA, 2012). A Figura 63 apresenta um esquema de como funciona o ciclo de eventos de um servidor web orientado a eventos.

Figura 63 – Abordagem orientada a evento.



Fonte - Doglio (2015).

A abordagem orientada a eventos é mais escalável e oferece maior controle, no entanto, ela ainda pode oferecer algumas dificuldades, no sentido de que nem toda operação de E/S ou de rede pode ser facilmente integrada no sistema de notificações (OUSTERHOUT, 1996) e pode ser complicado garantir que *callbacks* tenham acesso aos dados de contexto da aplicação (TILKOV; VINOSKI, 2010).

Node.js (Node.js Foundation, 2017) é um *framework*/ambiente de *runtime* para o desenvolvimento de aplicações na abordagem orientada a eventos (OJAMAA; DUUNA, 2012; MCCUNE, 2011). Ele é implementado em C/C++ e usa o motor JavaScript V8[3] (originalmente desenvolvido pela Google para dar suporte a JavaScript em seu navegador web Chrome) para permitir programação de aplicações em JavaScript fora de navegadores web. Um de seus usos mais comuns está no desenvolvimento de servidores de aplicações web (TILKOV; VINOSKI, 2010; MCCUNE, 2011; CHANIOTIS; KYRIAKOU; TSELIKAS, 2015; OJAMAA; DUUNA,

[3] Código-fonte V8: <https://chromium.googlesource.com/v8/v8.git>.

2012). Várias empresas de grande importância no mercado de Tecnologia da Informação (TI), tais como PayPal, eBay, Netflix e Yahoo!, tem utilizado Node.js (DOGLIO, 2015).

B.2.1 Detalhes Técnicos

Diferentemente de outros *frameworks*, Node.js não cria uma *thread* para tratar cada requisição recebida. Seguindo a abordagem orientada a eventos (OUSTERHOUT, 1996) e devido a utilizar JavaScript (que, por natureza, não suporta múltiplas *threads*), ele é executado sobre uma única *thread*, a de eventos (TILKOV; VINOSKI, 2010; CHANIOTIS; KYRIAKOU; TSELIKAS, 2015). Isso elimina concorrência no nível da aplicação, acabando com necessidades de sincronização de *threads* (OJAMAA; DUUNA, 2012).

Operações assíncronas são o padrão de funcionamento do Node.js (OJAMAA; DUUNA, 2012). Elas recebem como argumento uma função *callback*, que é invocada quando a operação encerra. Ao invocar uma operação de E/S ou de rede, o controle é imediatamente retornado, pois a operação, em si, é executada numa outra *thread*, obtida de um reservatório interno.

O programa, na verdade, é determinado pela funções chamadas explicitamente. Elas geralmente não realizam operações de E/S ou de rede, meramente registram os *callbacks* para essas operações. Sendo assim, no Node.js, por padrão, várias operações (e.g., salvar arquivo) não possuem meios para serem executadas de maneira síncrona; a única opção é cadastrar um *callback* para o evento (e.g., "arquivo salvo") (TILKOV; VINOSKI, 2010).

Dessa maneira, é possível entender como um servidor web implementado com Node.js pode atender a vários clientes simultaneamente, enquanto é executado numa única *thread*. As requisições web não são servidas no ciclo principal do programa; são eventos de E/S e de rede que dão início ao processamento real. Isso permite que mesmo computadores com recursos moderados possam atuar como servidores de alto desempenho (TILKOV; VINOSKI, 2010).

O uso do JavaScript pelo Node.js não é por acaso. JavaScript é uma linguagem com vários aspectos de linguagens funcionais e consegue facilmente dar suporte a tratamento de eventos. Nele, funções são objetos de primeira classe, i.e. são meramente um tipo especial de dado que pode ser tratado como qualquer outro. Adicionalmente, os problemas de acesso a dados do contexto são eliminados, pois as funções podem fazer referência a objetos definidos fora delas, encapsulando o contexto de execução e criando, assim, uma *closure* (CHANIOTIS; KYRIAKOU; TSELIKAS, 2015; OJAMAA; DUUNA, 2012).

As capacidades oferecidas pela especificação do HTML5 o tornam especialmente atraente para o desenvolvimento de aplicações web cliente. JavaScript tem monopólio como linguagem de programação para esta plataforma e, adicionalmente, vem sendo aperfeiçoado em termos de recursos e desempenho (OJAMAA; DUUNA, 2012) e integrado até mesmo em bancos de dados (TILKOV; VINOSKI, 2010). Assim, o uso de JavaScript também no desenvolvimento do servidor dessas aplicações permite que todos os aspectos de um sistema estejam escritos numa única linguagem de programação (CHANIOTIS; KYRIAKOU; TSELIKAS, 2015), aumentando a produtividade.

O Node.js vem acompanhado de APIs JavaScript para realizar tarefas básicas, tais

como hospedar um servidor HTTP, trabalhar com arquivos, compressão e outras tarefas comuns. Adicionalmente, ele também acompanha o *Node Package Manager* (NPM; do inglês, "Gerenciador de Pacotes do Node"), um utilitário de linha de comando que permite a gestão de pacotes de software (OJAMAA; DUUNA, 2012), não só para aplicações Node.js, mas para qualquer outro tipo. Também é possível que bibliotecas escritas em JavaScript para clientes de aplicações web sejam reutilizadas no servidor, se estiverem no padrão CommonJS[4].

O Node.js trabalha com um único núcleo do processador, o que pode oferecer um desempenho abaixo do ideal. Uma solução simples para tanto é a execução simultânea de várias instâncias do Node.js, todas escutando uma mesma porta e utilizando o SO como balanceador de carga (TILKOV; VINOSKI, 2010).

É importante notar, todavia, que existem alguns pontos de atenção no que diz respeito a segurança do Node.js. Mais notável, o uso de uma única *thread* por instância acaba por deixar a aplicação com um único ponto de falha (OJAMAA; DUUNA, 2012). Caso um *callback* leve muito tempo para executar, o processamento de outras requisições será atrasado; caso lance uma exceção, a aplicação toda é encerrada. *Hackers* malignos podem também explorar essa questão, fazendo com que valores inválidos sejam atribuídos a variáveis do contexto acessadas pelos *callbacks*, potencialmente corrompendo o processamento de outras requisições.

B.2.2 Desempenho

Em um estudo (antigo, mas ainda relevante) sobre o desempenho do Node.js para o desenvolvimento de servidores de aplicação, McCune (2011) comparou, em vários aspectos, servidores implementados utilizando o Node.js versão 0.4.12 (versão mais atual no momento da escrita deste documento é 9.11.1[5] (Node.js Foundation, 2017)), o *framework* EventMachine[6] versão 0.12.10 para Ruby e PHP com servidor Apache[7] versão 2.2.20.

No resultados (MCCUNE, 2011), com baixo nível de requisições concorrentes, não houve muita diferença entre o desempenho dos servidores, no entanto, a medida que as quantidades de requisições, totais e concorrentes, aumentou, a implementação em Node.js apresentou melhores resultados que os outras, sendo a única que conseguiu lidar de maneira consistente com altos volumes de requisições totais e concorrentes.

Em outro estudo sobre desempenho do Node.js, Chaniotis, Kyriakou e Tselikas (2015), compararam diversos aspectos de servidores de aplicação utilizando Node.js versão 0.10.21, PHP com servidor Apache versão 2.2.22 e PHP com servidor Nginx[8] versão 1.4.3.

Nos resultados (CHANIOTIS; KYRIAKOU; TSELIKAS, 2015), foram encontradas evidências que suportam os achados de McCune (2011), mas também alguns pontos contrários. Quanto a operações de E/S, foi encontrado que o servidor Node.js teve melhor desempenho que os outros, sendo aproximadamente 3 vezes mais rápido que o servidor Apache. Em termos de uso de memória, o servidor Node.js também apresentou melhores resultados que os outros. Embora

[4] CommonJS: <http://www.commonjs.org>.
[5] Verificação realizada em 5 de Abril de 2018
[6] EventMachine: <http://rubyeventmachine.com>.
[7] Apache Server: <https://httpd.apache.org>.
[8] Nginx: <http://nginx.org>.

o servidor Node.js tenha apresentado melhor velocidade do que os outros servidores em PHP, ele também apresenta um maior uso de CPU, cerca de 4 vezes mais do que o servidor Nginx. O servidor Nginx também apresentou melhor desempenho para servir páginas web.

É interessante notar, em todo caso, que ambos os servidores Nginx e Node.js utilizam a abordagem orientada a eventos, enquanto o Apache, a abordagem orientada a procedimento. Quanto ao uso de CPU, o servidor Apache teve o pior desempenho, com os autores atribuindo isso ao fato de que ele gasta recursos para criar e encerrar processos filhos para tratar requisições (CHANIOTIS; KYRIAKOU; TSELIKAS, 2015).

B.2.3 Considerações

Dados o propósito particularmente específico e uso limitado do servidor de sinalização, seu projeto e desempenho não possuem tanta importância para a solução proposta, em respeito a renderização dos gráficos ou a comunicação entre os nós do CG.

Dessa maneira, a simplicidade e o desempenho oferecidos pelo Node.js permitem que o servidor de sinalização seja desenvolvido de maneira relativamente rápida e fácil (até mesmo compartilhando código-fonte com o *framework* desenvolvido para o design de solução apresentado), enquanto executa com um mínimo de recursos. Além disso, seu gerenciador de pacotes, o NPM, foi utilizado para gerenciar as dependências do projeto desenvolvido e definir scripts para "compilação" e validação.

Embora o Node.js possua um ponto de falha centralizado, que pode representar uma falha de segurança, julga-se que os benefícios oferecidos tem superioridade.

B.3 Three.js

Nos últimos anos, houve uma rápida e crescente adoção da especificação HTML5 (a qual traz o elemento `<canvas>`, que permite a renderizar gráficos 2D e 3D (World Wide Web Consortium, 2017a)) pela maioria dos navegadores web (SINGH; KRISHNASWAMY, 2013). Com isso, o consórcio The Khronos Group, Inc. elaborou a API JavaScript *Web Graphics Library* (WebGL; do inglês, "Biblioteca de Gráficos da Web"), que permite a exibição de gráficos 3D acelerados por hardware em navegadores web, dentro do elemento `<canvas>`, baseada na biblioteca OpenGL, para a linguagem C, e agora suportada pelos maiores navegadores web do mercado (The Khronos Group, Inc, 2017).

O WebGL é projetado como uma biblioteca de baixo nível (The Khronos Group, Inc, 2017), portanto sua utilização não é trivial, assim, outras bibliotecas JavaScript foram desenvolvidas por terceiros, a fim de abstrair a complexidade do WebGL para algo mais intuitivo e fácil de se utilizar (DIRKSEN, 2014).

A Tabela 8 apresenta uma comparação grosseira da popularidade de bibliotecas para WebGL, com código-aberto e disponíveis no repositório online de código-fonte GitHub[9]. Para esta comparação, foram selecionadas as dez bibliotecas listadas na *wiki* do WebGL[10] e com maior

[9] GitHub: <https://github.com>.
[10] Wiki WebGL: <https://www.khronos.org/webgl/wiki/User_Contributions>.

número de estrelas no GitHub; cada estrela equivale a recomendação do projeto por um usuário (com um máximo de uma estrela por usuário, por projeto). Para efeitos adicionais de comparação, também são mostradas as quantidades de resultados retornados pela busca dessas bibliotecas nas bases digitais IEEE Xplore, ACM Digital Library e Science Direct. O dados apresentados foram consultados em 26 de Maio de 2017.

Tabela 8 – Comparação de popularidade entre bibliotecas para WebGL.

Biblioteca	Estelas GitHub	IEEE Xplore	Science Direct	ACM Digital Library
Babylon.js	4.269	6	0	0
CubicVR.js	446	2	0	0
GLGE	375	12	3	0
J3D	701	0	1	1
lightgl.js	824	1	0	0
OSG.JS	409	2	12	0
PhiloGL	701	2	3	0
SceneJS	519	8	7	0
Three.js	31.640	115	81	4
X3DOM	421	18	21	45

Fonte – Produzida pelo autor.

Observa-se que a biblioteca Three.js (CABELLO, 2017), originalmente desenvolvida por Ricardo Cabello, é, por uma grande margem, a mais popular, tanto na aprovação dos usuários quanto no âmbito científico.

É importante notar, todavia, que alguns motores de jogos, tal como Unity3D (Unity Technologies, 2017), permitem que jogos/aplicações desenvolvidos para eles sejam exportados para WebGL, permitindo que esses jogos/aplicações sejam executados diretamente do navegador web. No entanto, o suporte a WebGL de vários desses motores de jogo não dá acesso as suas APIs de rede, forçando o jogo/aplicação a funcionar sem capacidades multi-jogador, i.e. de se comunicar com outras máquinas na rede. Além disso, esses motores de jogo também não possuem suporte próprio e completo aos recursos do WebRTC.

Sendo assim, implementar o design de solução para aplicações VI3DI web apresentado utilizando a biblioteca Three.js oferece vários benefícios: maior alcance (e, possivelmente, aceitação), possibilidade de usar de vários componentes e APIs, desenvolvidas pela comunidade usuária da biblioteca, e acesso a comunicação P2P de baixa latência, através do suporte a WebRTC nos navegadores web.

B.3.1 Detalhes Técnicos

O Three.js[11] é uma biblioteca JavaScript para a exibição de gráficos 3D em navegadores web. Aplicações que o utilizam criam grafos de cenas 3D, dentro dos quais vários elementos e tipos de objetos são interligados e configurados (EVANS et al., 2014; DIRKSEN, 2014):

[11] Three.js: <https://github.com/mrdoob/three.js>.

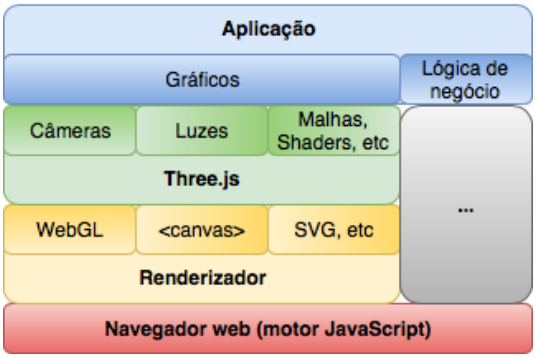
- Câmeras virtuais: responsáveis por capturar determinado trecho do ambiente virtual representado na cena e exibi-lo na tela;
- Luzes: iluminam a cena de diferentes maneiras;
- Malhas: são compostas por geometrias e materiais e servem propósitos diversos (o mais comum sendo a representação de objetos físicos), de acordo com a aplicação; e
- Outros: inclui animações e *shaders*.

A renderização da cena, em si, é feita através de um componente identificado como *Renderer*, que determina a maneira como os gráficos serão desenhados na tela. O Three.js já inclui dois desses componentes: `WebGLRenderer` (padrão) renderiza a cena utilizando WebGL e possui suporte a aceleração por hardware, enquanto `CanvasRenderer` (depreciado) renderiza a cena utilizando puramente o elemento `<canvas>` do HTML5 (CABELLO, 2017; SAWICKI; CHABER, 2013). Também é possível criar outros tipos de *Renderers*, como, por exemplo, um que gere os gráficos utilizando o formato de imagens SVG (SAWICKI; CHABER, 2013) ou outro que utilize CSS 3D (DIRKSEN, 2014), embora, nesses casos, pode haver limitações (e.g., impossibilidade de utilizar luzes).

É possível, também, a utilização de *shaders*, scripts pré-compilados executados pelo WebGL que permitem melhorias na qualidade visual dos objetos 3D gerados (NAZAROV; GALLETLY, 2013).

A Figura 64 mostra a arquitetura de uma aplicação HTML/JavaScript que utiliza Three.js para renderização de seus gráficos. Pela figura, nota-se que o desenho dos gráficos e funcionamento dos objetos são transparentes para o desenvolvedor, sendo que este deve se focar em como manusear os objetos.

Figura 64 – Arquitetura de aplicação HTML/JavaScript utilizando Three.js.



Fonte - Produzida pelo autor.

A Figura 65 apresenta um simples programa desenvolvido com Three.js; o programa meramente cria um cubo vermelho no centro da tela e o faz girar continuamente em torno do vetor $(1, 1, 1)$.

Em (a), é exibido o código-fonte JavaScript do programa. A ordem das ações é:

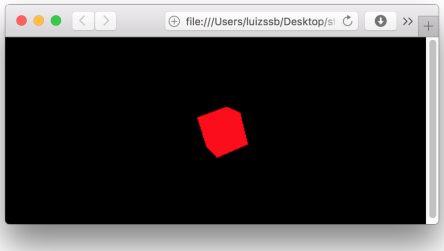
1. Criação do *Renderer* para WebGL (linhas 1-4);

Figura 65 – Exemplo de programa desenvolvido com Three.js.

(a) Código-fonte JavaScript.

```
1  const renderer = new THREE.WebGLRenderer();
2  renderer.setSize(window.innerWidth, window.innerHeight);
3  document.getElementById('canvasContainer')
4      .appendChild(renderer.domElement);
5
6  const scene = new THREE.Scene();
7  const camera = new THREE.PerspectiveCamera(
8      45, window.innerWidth / window.innerHeight, 1, 100
9  );
10 camera.position.set(0, 0, 10);
11 scene.add(camera);
12
13 var boxGeometry = new THREE.BoxGeometry(1.5, 1.5, 1.5);
14 var boxMaterial = new THREE.MeshBasicMaterial(
15     { color:new THREE.Color(0xFF0000) }
16 );
17 boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);
18 scene.add(boxMesh);
19
20 animateScene();
21
22 function animateScene() {
23     boxMesh.rotateOnAxis(
24         new THREE.Vector3(1, 1, 1).normalize(), 0.075
25     );
26     requestAnimationFrame(animateScene);
27     renderer.render(scene, camera);
28 }
```

(b) Execução no navegador web Safari.



Fonte - Produzida pelo autor.

2. Criação da cena (linha 6) e, então, da câmera virtual em perspectiva (linhas 7-11);
3. Criação da malha de um cubo (linha 13), de um material vermelho para ser aplicado ao cubo(linhas 14-16) e, então, do próprio cubo (linhas 17-18);
4. Início da renderização (linha 20);
5. Giro do cubo (linhas 23-25); e
6. Requisição para desenho de novo quadro (linhas 26-27).

Em (b), é possível ver o programa em execução no navegador web Safari. Nota-se que, neste programa, nenhuma luz foi adicionada a cena e, mesmo assim, o cubo está visível, devido a iluminação ambiente, que afeta igualmente todos os pontos do ambiente virtual. A falta de luz, contudo, faz com que não seja possível discernir perfeitamente as arestas e vértices desse cubo.

Observa-se que, no Three.js, o ciclo de atualização dos gráficos é controlado pelo próprio programa, e não internamente. Isso permite maior controle, de maneira que o programa pode decidir quando o processamento gráfico (no caso, o giro do cubo) será feito, ou não.

Three.js oferece malhas padronizadas para objetos primitivos (e.g., cubo e esfera), porém, é possível que o programa, em si, crie suas próprias malhas, definindo os vértices e faces em um objeto `Geometry` (CABELLO, 2017). Adicionalmente, é possível importar malhas preparadas em aplicações de *Digital Content Creation* (DCC; do inglês, "Criação de Conteúdo Digital"), tais como Maya[12] e Blender[13] (NAZAROV; GALLETLY, 2013), e exportadas para formatos interoperáveis, tal como COLLADA (NETO et al., 2015b).

Ao contrário de motores de jogos, que buscam fornecer toda infraestrutura para desenvolvimento dos jogos (e outras aplicações gráficas) (ANDERSON et al., 2013), Three.js é uma

[12] Maya: <http://www.autodesk.com/products/autodesk-maya/overview>.
[13] Blender: <https://www.blender.org>.

biblioteca para programação de gráficos apenas, não oferecendo recursos adicionais para outras tarefas, e.g., reprodução de áudio, cálculo de física ou outros. Essas tarefas podem ser realizadas utilizando recursos como a API de áudio do HTML5 (DIRKSEN, 2014) (no caso do áudio), bibliotecas de física desenvolvidas por terceiros, tais como Ammo.js[14] ou Phisijs[15] (NAZAROV; GALLETLY, 2013) (no caso da física) ou outros recursos disponíveis.

B.3.2 Considerações

Three.js é uma biblioteca particularmente poderosa no que diz respeito ao desenvolvimento de aplicações 3D para a web, com várias abstrações e flexibilidade suficientes para o desenvolvimento de vários tipos de aplicações interativas. Além disso, ela se mostrou extremamente popular com a comunidade desenvolvedora. Tudo isso indica que, se o objetivo é atender, de qualquer forma, o desenvolvimento de aplicações web 3D, Three.js é uma escolha de grande impacto.

O fato de ser completamente gratuito e código-aberto também tem relevância em sua escolha, pois permite fácil adoção e possibilidade de ajustar detalhes específicos de seu funcionamento, caso a sua arquitetura original não dê suporte ao que se deseja realizar.

B.4 Virtual Reality Peripheral Network

A fim de proporcionar maiores imersão, interatividade e/ou envolvimento, aplicações podem se utilizar de uma variedade de dispositivos de interação. Esses dispositivos, no entanto, se comunicam de diferentes maneiras com computadores e aplicações. Sendo assim, é necessário algum tipo de interface que permita seu uso generalizado, abstraindo suas particularidades.

Virtual Reality Peripheral Network (VRPN; do inglês, "Rede Periférica de Realidade Virtual") é essa interface generalizada e comum entre os dispositivos periféricos e a aplicação que os utiliza (STELZER et al., 2014). VRPN é "um conjunto de classes dentro de uma biblioteca e um conjunto de servidores que são projetados para implementar uma interface, transparente a nível de rede, entre aplicações e um conjunto de dispositivos físicos usados num sistema de Realidade Virtual" (TAYLOR-II, 2018).

Originalmente proposto por Taylor-II et al. (2001), o VRPN, em si, não é uma biblioteca RV, mas, sim, uma biblioteca complementar, que permite fácil interação com dispositivos de entrada diversos, que podem (ou não) estar fora do alcance do computador executando a aplicação.

Para permitir acesso aos dados dos dispositivos, o VRPN executa servidores de aplicação nos computadores nos quais eles estão conectados. Esses servidores possuem interfaces com os *drivers* dos dispositivos, permitindo que capturem seu dados de entrada. Aplicações cliente podem se conectar, de maneira transparente para o desenvolvedor, aos servidores VRPN e se registrar para serem notificadas quando novos dados de entrada estiverem disponíveis (TAYLOR-II et al., 2001).

[14] ammo.js: <https://github.com/kripken/ammo.js>.
[15] Physijs: <https://github.com/chandlerprall/Physijs>.

A utilização da arquitetura cliente-servidor para acesso aos dados foi concebida devido a diversos fatores (TAYLOR-II et al., 2001): distribuição física dos dispositivos de entrada (que podem estar distantes um do outro e/ou do computador executando a aplicação), suporte aos dispositivos pelos computadores servidor/cliente (e.g., aplicação é executada em GNU/Linux, porém dispositivo possui *drivers* somente para Windows) e a necessidade de certos dispositivos de estarem ligados e colhendo dados por um tempo bem maior que a execução da aplicação.

Foi escolhido usar o VRPN nesta dissertação sobre alternativas, tal como o Omicron, mesmo que ele não dê suporte a sensores de movimento, tais como Microsoft Kinect e Leap Motion, que, como visto na Subseção 3.2 Dispositivos de Interação em Navegadores Web, são bastante usados em aplicações web que se utilizam de dispositivos de interação diversificados. Há três razões para isso:

- O VRPN é um software maduro, desenvolvido desde 2001, e também relativamente popular no repositório online de código-fonte GitHub;
- O projeto OSVR, que visa homogenizar o uso de dispositivos de interação para RV (e suporta os sensores de movimento, além de outros dispositivos não suportados pelo VRPN), usa e extende o VRPN (Open Source Virtual Reality, 2018), assim, não se acredita que seja particularmente desafiante substituir um pelo outro; e
- Os autores possuem mais familiaridade e tempo de uso com o VRPN, permitindo desenvolvimento mais rápido e eficiente.

B.4.1 Detalhes Técnicos

O VRPN possui uma camada de abstração para os dados de entrada capturados pelos dispositivos que suporta, permitindo que dispositivos de funções similares sejam manuseados através das mesmas interfaces. As abstrações para dispositivos de entrada suportadas atualmente são (TAYLOR-II, 2018; TAYLOR-II et al., 2001):

- *Tracker*: relata poses (posição e orientação) e suas velocidade e aceleração;
- *Button*: relata eventos de pressionar e soltar botões;
- *Analog*: relata um ou mais valores analógicos;
- *Dial*: relata rotações incrementais;
- *ForceDevice*: permite aos clientes especificar superfícies e campos de força em espaço tridimensional;
- *Imager*: relata imagens capturadas por uma câmera;
- *Sound*: relata som capturado por um microfone; e
- *Text*: relata texto.

Novas abstrações de dispositivos também podem ser implementadas pelo desenvolvedor, como, por exemplo, uma curva (TAYLOR-II et al., 2001) ou um sistema ósseo (SUMA et al., 2011). Além disso, abstrações já existentes podem ser estendidas, para fornecer dados adicionais ou terem seu comportamento modificado de acordo com dispositivos específicos (TAYLOR-II, 2018); caso a abstração seja utilizada para um dispositivo que não a suporta, o VRPN falha silenciosamente, a fim de impedir o encerramento da aplicação por erros/exceções não tratadas (TAYLOR-II et al., 2001).

Cada abstração apresenta, pelo menos, um evento. Os eventos são utilizados para reportar dados capturados pelo dispositivo. Um *Analog*, por exemplo, possui somente um evento, *change*, usado para reportar mudança nos canais analógicos do dispositivo; já um *Button*, possui dois eventos: *states*, que informa o estado dos botões do dispositivo e é chamado logo que que a conexão é estabelecida, e *change*, que informa que um dos botões foi pressionado/despressionado.

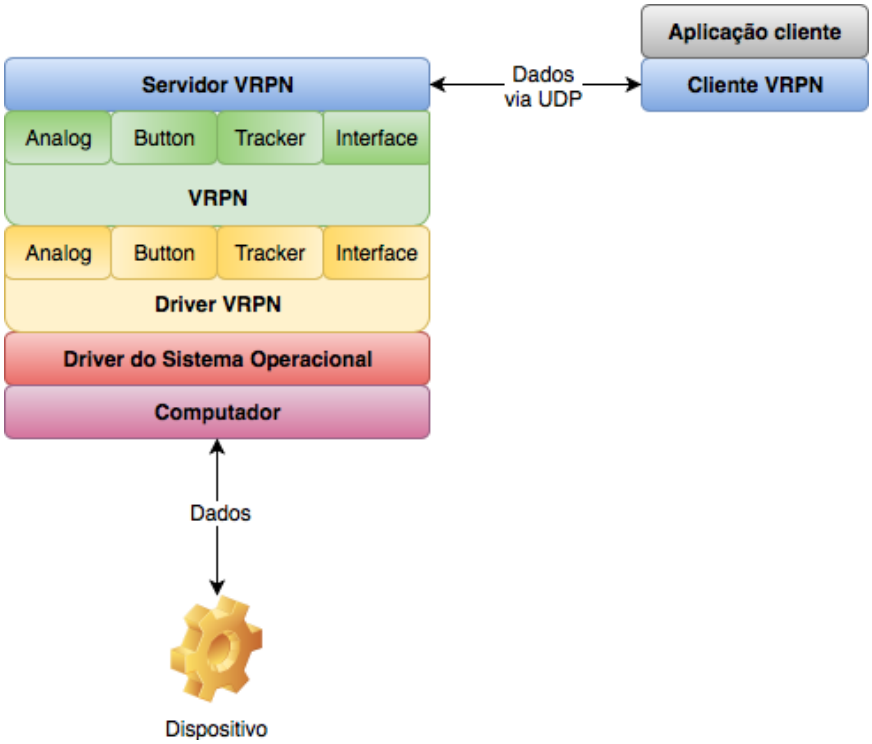
Os dispositivos, em si, podem fornecer interfaces para mais de uma abstração de dados, permitindo acesso a todos os seus controles de entrada e, também, abstrações diversas para um mesmo controle (TAYLOR-II et al., 2001). Um mouse comum, por exemplo, possui duas interfaces: *Analog* com dois canais (posição (x, y) do cursor) e *Button* com três índices (cliques esquerdo, direito e da roda); já um teclado comum tem somente uma interface, *Button* com um índice para cada tecla.

Todas essas interfaces de um dispositivo são mapeadas para uma mesma conexão do VRPN, a fim de maior eficiência quanto ao uso da rede (TAYLOR-II et al., 2001); o desenvolvedor meramente especifica quais interfaces do dispositivo deseja usar através de um arquivo de configuração. Por exemplo, para capturar os cliques do mouse (identificado como "Mouse0"), o usuário adiciona a instrução `vrpn_Button Mouse0` no arquivo de configuração; para capturar a posição do cursor desse mouse, o usuário adiciona a instrução `vrpn_Analog Mouse0`.

As conexões entre servidores e clientes do VRPN trabalham com os protocolos de transporte TCP e UDP (TAYLOR-II et al., 2001). Inicialmente, o cliente abre uma conexão UDP com o servidor, indicando uma porta para que seja estabelecida uma conexão TCP. O servidor, então, abre a conexão TCP com o cliente na porta indicada, pela qual eles fazem verificação de versão, sincronização dos *clocks* e estabelecimento de um canal UDP não-confiável, através do qual os dados serão transmitidos. Posteriormente, caso esse canal se torne indisponível (i.e. a conexão foi rompida), o VRPN automaticamente tenta reestabelecê-lo. Em todo caso, mesmo com essa abordagem cliente-servidor para acessar o dispositivo, o VRPN ainda oferece uma latência razoavelmente baixa (MORGAN; JARRELL; VANCE, 2014).

A Figura 66 apresenta a arquitetura de um servidor VRPN. O dispositivo está ligado a um computador no qual executa um SO. Nesse computador está instalado o *driver* do fabricante para dispositivo, sobre o qual é desenvolvido o *driver* do VRPN, que expõe as funcionalidades do dispositivo através de interfaces comuns (*Button*s, *Analog*s, *Tracker*s, etc.) ou implementa suas próprias. O núcleo do VRPN utiliza essas interfaces para permitir acesso aos dados pelo servidor VRPN, que serve os dados ao cliente VRPN, o qual é utilizado pela aplicação.

Figura 66 – Arquitetura de um servidor VRPN.



Fonte - Produzida pelo autor.

B.4.2 Considerações

Embora não seja necessário para aplicações 3D que meramente fazem uso de múltiplas telas, o VRPN ainda é importante para aplicações altamente interativas, tais como as de VI3DI. Assim, o suporte a ele complementa o *framework* implementado, permitindo que ofereça mais recursos a desenvolvedores.

O VRPN é implementado em C/C++ e existem recursos para ele que permitem que seja utilizado com outras linguagens de programação, tais como Java, Python, e C# (TAYLOR-II, 2018). Contudo, nenhum recurso existe atualmente para o uso do VRPN através de JavaScript. Acredita-se que isso se dê devido ao fato de que código JavaScript em navegadores web é executado em um ambiente fechado, impedindo maior contato com recursos de fora do navegador web. No entanto, como mostra a Subseção 5.2.5 Acesso a Dispositivos de Interação, existem alternativas a essa questão.

APÊNDICE C – Eventos de Entrada e de Saída da Livvclib

Este apêndice apresenta os eventos de entrada e de saída estabelecidos para o funcionamento da Livvclib. Os eventos estão agrupados de acordo com as funcionalidades que eles apoiam.

Como visto na Subseção 5.3.2 Tratamento de Eventos, no lado servidor, os eventos são tratados por meio de implementações da classe abstrada `ClientEventsHandler`. Todos os eventos tratados e emitidos por uma instância de `ClientEventsHandler` devem possuir, pelo menos, um argumento, `data`, um objeto JavaScript. Por padrão, esse objeto carrega os seguintes atributos:

- `sender`: identificador do nó que disparou o evento de entrada;
- `room`: sala virtual da qual o nó que disparou o evento de entrada faz parte; e
- `numberOfParticipants`: quantidade de nós unidos a sala virtual da qual o nó que disparou o evento de entrada faz parte.

Outros atributos também pode serm adicionados, de acordo com o necessário.

Como esse argumento é padronizado em todos os eventos, as listas de eventos de entrada e saída das próximas subseções o omitem.

C.1 Suporte a Clusters Gráficos

No lado cliente, estes eventos são emitidos e tratados por `MasterSlaveCluster` e componentes internos que conectam dois nós. No lado servidor, estes eventos são tratados e emitidos por `ClusterEventsHandler`.

Os eventos para suportar clusters gráficos são:

- Eventos de Entrada:
  - *joinCluster*: Nó deseja se unir a um CG.
    - Dispara: se sucesso, então *joinCluster*, do contrário, *joinClusterError*. Se o nó se uniu como mestre, também dispara *masterJoin* nos nós escravos.
    - Argumentos próprios:
      1. Nome da sala virtual do CG (string);
      2. Método pelo qual o nó prefere ser conectado com outros nós (enumerador `NodeConnectionMethod`); e

3. Modo de união do nó ao CG (enumerador `ClusterJoiningMode`).

- *leave*: Nó está deixando o CG.
  - Dispara: *peerGone*, nos nós restantes.
  - Argumentos próprios: nenhum.
- *peek*: Consulta situação de CG em uma sala virtual.
  - Dispara: *peek*.
  - Argumentos próprios:
    1. Nome da sala virtual do CG (string).
- *offer*: Oferta de conexão de um nó para outro.
  - Dispara: *offer*, no outro nó.
  - Argumentos próprios:
    1. Identificador do outro nó (string); e
    2. Dados da oferta (objeto).
- *answer*: Resposta a oferta de conexão com outro nó
  - Dispara: *answer*, no outro nó.
  - Argumentos próprios:
    1. Identificador do outro nó (string); e
    2. Dados da resposta (objeto).
- *answerFailed*: Ocorreu erro na criação da resposta a oferta de conexão.
  - Dispara: *answerFailed*, no outro nó.
  - Argumentos próprios:
    1. Identificador do outro nó (string); e
    2. Dados do erro (objeto).
- *candidate*: Dados de candidato de conexão.
  - Dispara: *candidate*, no outro nó.
  - Argumentos próprios:
    1. Identificador do outro nó (string); e
    2. Dados de candidato (objeto).

- Saída:
  - *joinCluster*: Nó se uniu ao CG.
    - Disparado por: *joinCluster*.
    - Argumentos próprios:
      1. Dados do nó mestre (objeto).
  - *joinClusterError*: Ocorreu um erro ao unir o nó ao CG.
    - Disparado por: *joinCluster*.

    - Argumentos próprios: nenhum.

- *peerGone*: Nó do CG se desconectou.
    - Disparado por: *leave*.
    - Argumentos próprios:
        1. Indicação se nó desconectado era mestre ou não (boolean).

- *masterJoin*: Nó mestre se uniu a sala.
    - Disparado por: *joinCluster*.
    - Argumentos próprios:
        1. Dados do nó mestre (objeto).

- *peek*: Retorna consulta da situação de um CG.
    - Disparado por: *peek*.
    - Argumentos próprios:
        1. Dados dos nós no CG (objeto).

- *offer*: Oferta de conexão com outro nó.
    - Disparado por: *offer*.
    - Argumentos próprios:
        1. Identificador do outro nó (string); e
        2. Dados da oferta (objeto).

- *answer*: Resposta a oferta de conexão com outro nó
    - Disparado por: *answer*.
    - Argumentos próprios:
        1. Identificador do outro nó (string); e
        2. Dados da resposta (objeto).

- *answerFailed*: Ocorreu erro na criação da resposta a oferta de conexão.
    - Disparado por: *answerFailed*, no outro nó.
    - Argumentos próprios:
        1. Identificador do outro nó (string); e
        2. Dados do erro (objeto).

- *candidate*: Dados de candidato de conexão com o outro nó.
    - Disparado por: *candidate*.
    - Argumentos próprios:
        1. Identificador do outro nó (string); e
        2. Dados de candidato (objeto).

C.2 Suporte a Troca de Dados pelo Servidor

No lado cliente, estes eventos são emitidos e tratados por componentes internos que gerenciam a conexão entre dois nós por meio do servidor de aplicação. No lado servidor, estes eventos são tratados e emitidos por `NodeEventsHandler`

Os eventos para suportar a troca de dados entre os nós por meio do servidor são:

- Entrada:
  - *node:data*: Envio de dado de um nó para outros na mesma sala virtual.
    - Dispara: *node:data*, nos outros nós.
    - Argumentos próprios:
      1. Dado a enviar (objeto).
  - *node:quit*: Equivalente a *leave*. Usado por nós que se conectam sem usar um CG.
    - Dispara: *node:gone*, no outro nó.
    - Argumentos próprios: nenhum.
- Saída:
  - *node:data*: Dados enviados de um nó para outros na mesma sala virtual.
    - Disparado por: *node:data*.
    - Argumentos próprios:
      1. Dados enviados (string).
  - *node:gone*: Equivalente a *peerGone*. Usado por nós que se conectam sem usar um CG.
    - Disparado por: *node:quit*.
    - Argumentos próprios:
      1. Identificador do nó que se desconectou (string).

C.3 Suporte a Dispositivos de Interação

No lado cliente, estes eventos são emitidos e tratados por `DeviceControlAccess`. No lado servidor, estes eventos são tratados e emitidos por `DeviceEventsHandler`.

Os eventos para o suporte a dispositivos de interação são:

- Entrada:
  - *device*: Requisição de conexão a uma interface de um dispositivo de interação.
    - Dispara: se sucesso, então *device:connect:<name>:<address>:<interface>*, do contrário, *device:connectError:<name>:<address>:<interface>*. Se sucesso, também dispara *device:new* nos outros nós da sala virtual.
    - Argumentos próprios:

1. Nome do dispositivo (string);
2. Endereço IP do servidor VRPN (string); e
3. Nome da interface do dispositivo a qual se deseja conectar (string).

– *device:disconnect*: Requisição de encerramento da conexão a uma interface um dispositivo de interação;

* Dispara: *device:lost:<name>:<address>:<interface>*, em todos os nós na sala virtual.
* Argumentos próprios:
  1. Nome do dispositivo (string);
  2. Endereço IP do servidor VRPN (string); e
  3. Nome da interface do dispositivo a qual se deseja conectar (string).

– *device:query*: Consulta as interfaces de dispositivos as quais o servidor está conectado.

* Dispara: *device:query*.
* Argumentos próprios: nenhum

– *device:subscribe:<name>:<address>:<interface>*: Requisição para receber dados de evento da interface do dispositivo.

* Dispara: se sucesso, então *device:subscribe:<name>:<address>:<interface>*, do contrário, *device:subscribe:<name>:<address>:<interface>*.
* Argumentos próprios:
  1. Nome do evento da interface (string); e
  2. Indicação se os dados do dispositivo devem ser repassados para todos os nós na sala virtual ou não (boolean).

– *device:unsubscribe:<name>:<address>:<interface>*: Requisição para parar de receber dados de evento da interface do dispositivo.

* Dispara: nada.
* Argumentos próprios:
  1. Nome do evento da interface (string).

- Saída:

– *device:new*: Aviso de que um dos nós na sala virtual requisitou conexão a uma interface de um dispositivo de interação

* Disparado por: *device*.
* Argumentos próprios:
  1. Nome do dispositivo (string);
  2. Endereço IP do servidor VRPN (string); e
  3. Nome da interface do dispositivo a foi conectado (string).

– *device:query*: Retorna consulta de dispositivos de interação aos quais o servidor está conectado.

    * Disparado por: *device:query*.
    * Argumentos próprios:
        1. Vetor com os dados de cada dispositivo ao qual o servidor está conectado.
- *device:connect:<name>:<address>:<interface>*: Servidor conectou a interface de dispositivo de interação.
    * Disparado por: *device*.
    * Argumentos próprios: nenhum.
- *device:connectError:<name>:<address>:<interface>*: Servidor falhou em se conectar a interface de dispositivo de interação.
    * Disparado por: *device*.
    * Argumentos próprios: nenhum.
- *device:subscribe:<name>:<address>:<interface>*: Cadastrou servidor para receber e repassar dados de evento da interface do dispositivo de interação.
    * Disparado por: *device:subscribe:<name>:<address>:<interface>*.
    * Argumentos próprios:
        1. Nome do evento da interface (string).
- *device:subscribeError:<name>:<address>:<interface>*: Falhou em cadastrar servidor para receber dados de evento da interface do dispositivo de interação.
    * Disparado por: *device:subscribe:<name>:<address>:<interface>*.
    * Argumentos próprios:
        1. Nome do evento da interface (string).
- *device:lost:<name>:<address>:<interface>*: Servidor perdeu conexão a interface de dispositivo de interação.
    * Disparado por: *device:disconnect* ou perda súbita da conexão ao dispositivo de interação (e.g., servidor VRPN ficou fora do ar).
    * Argumentos próprios: nenhum.
- *device:event:<name>:<address>:<interface>*: Evento de interface do dispositivo de interação. Pode ser disparado somente no nó que originalmente solicitou conexão a interface ou em todos os nós de sua sala virtual, dependendo do valor usado para o segundo argumento do evento *device:subscribe:<name>:<address>:<interface>*.
    * Disparado por: recebimento de dados do servidor VRPN.
    * Argumentos próprios:
        1. Nome do evento da interface (string); e
        2. Dados do evento (objeto).

Observa-se que esses eventos apresentam um aspecto dos de outras funcionalidades: alguns deles são definidos dinamicamente, de acordo com as interfaces de dispositivos de interação com as quais se requisitam conexões. Esses eventos incorporam em seus nomes o nome do dispositivo, o endereço do servidor VRPN e o nome da interface com a qual se foi conectado, o que apresenta duas vantagens:

- O próprio nome do evento já oferece informações suficientes para identificar a qual interface de dispositivo de interação ele se refere; e
- São criados canais específicos para o recebimento de dados de cada interface de dispositivo de interação, tanto no servidor de aplicação e sinalização quanto nos nós.

Dessa maneira, objetos, dos dois lados, que desejam receber notificações, dados e eventos relativos a uma interface de dispositivo de interação específica podem cadastrar funções *callback* para seus eventos específicos, ao invés de se cadastrar para algum evento generalizado, no qual seu tratamento necessita identificar qual a interface que o disparou.

C.4 Suporte a Sala Virtual

No lado cliente, não há interface específica para emitir e tratar esses eventos. No lado servidor, esses eventos são tratados e emitidos por `RoomEventsHandler`.

O suporte a sala virtual não possui eventos de saída, somente de entrada:

- *room:join*: Nó deseja se unir a sala virtual no CG.
  - Dispara: nada.
  - Argumentos próprios:
    1. Nome da sala virtual (string).
- *room:leave*: Nó deseja sair da sala virtual no CG a que pertence.
  - Dispara: nada.
  - Argumentos próprios: nenhum.

APÊNDICE D – Roteiro do Experimento

D.1 Introdução

Foi desenvolvida aplicação web (i.e. que é acessada pelo navegador web) para a visualização distribuída, imersiva e interativa de estruturas moleculares.

Essa aplicação pode ser executada no MiniCAVE, um sistema de multi-projeção, o que permite uma visão ampliada e abrangente das estruturas. A aplicação também suporta manipular (arrastar, girar, zoom) a estrutura molecular usando mouse, controles de video game e dispositivos móveis.

Demonstração:

- Versão antiga: <https://www.youtube.com/watch?v=dxPBcE5LjtY>; e
- Versão atual: <https://www.youtube.com/watch?v=42oMiIuq6aU>.

D.2 Objetivo

Avaliar a aplicação web, a utilização do MiniCAVE para visualização molecular e a interação por mouse, controle de video game e dispositivos móveis, através de experimento com pessoas envolvidas em áreas que necessitem visualizar estruturas moleculares, tais como química, física e biotecnologia.

D.3 Metodologia

A aplicação será executada e configurada no MiniCAVE montado no Laboratório de Interfaces e Visualização montado na Faculdade de Ciências da Universidade Estadual Paulista, campus de Bauru. A execução do experimento tem como coordenador o aluno de mestrado Luiz S.S. Baglie.

Participantes são tratados individualmente pelo coordenador, que começa com a intenção de contextualizar o participante sobre o que está sendo realizado:

- **Coordenador lê:**
  - *Este é um experimento para a validação de uma aplicação web de visualização distribuída, imersiva e interativa de estruturas moleculares.*

    A aplicação será executada no MiniCAVE, o sistema de multi-projeção neste laboratório, o que permite uma visão ampliada e abrangente das estruturas. A aplicação também permite que estruturas moleculares sejam manipuladas usando mouse, controle

de videogame e a tela sensível a toque de dispositivos móveis.

O experimento, em si, consiste no uso da aplicação, do MiniCAVE e dos aparatos de interação para realizar tarefas envolvendo a visualização de estruturas moleculares. Durante todo o processo, o coordenador, eu, estará disponível para solucionar quaisquer dúvidas e ajudar com o necessário.

A seguir, serão carregadas algumas estruturas moleculares na aplicação, para que você possa treinar a manipulação delas com diversos dispositivos de interação.

Alguma dúvida até o momento?

- **Participante faz:**
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.

Em seguida, é iniciada a fase de treinamento:

- **Coordenador faz:**
  - Carrega na aplicação (e, por consequência, no MiniCAVE) arquivo .pdb contendo nanotubo de carbono e cadeia de quitosana.
- **Coordenador lê:**
  - *No momento, há uma estrutura carregada na aplicação, um nanotubo de carbono junto a uma cadeia de quitosana.*

    Durante o experimento, seu foco estará na parte central da aplicação, onde está a visualização das estruturas moleculares, enquanto o coordenador será responsável por operar os outros aspectos da aplicação.

    Alguma dúvida?
- **Participante faz:**
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.

Com a introdução do treinamento, o participante é, então, instruído sobre como usar o mouse para manipular as estruturas moleculares:

- **Coordenador lê:**

  - *A seguir, então, você será instruído sobre como manipular as estruturas moleculares. Comece segurando o mouse.*

- **Coordenador faz:**
  - Entrega mouse ao participante.
- **Coordenador lê:**
  - *Segure o clique com o botão esquerdo do mouse sobre a visualização e o arraste. Isso faz com que a visualização seja rotacionada.*
- **Participante faz:**
  - Segura o clique com o botão esquerdo do mouse sobre a visualização e o arrasta.
  - Vê visualização girar.
- **Coordenador lê:**
  - *Segure o clique com o botão direito do mouse sobre a visualização e o arraste. Isso faz com que a visualização seja movida.*
- **Participante faz:**
  - Segura o clique com o botão direito do mouse sobre a visualização e o arrasta.
  - Vê visualização ser movida.
- **Coordenador lê:**
  - *Posicione o cursor sobre a visualização e usar a roda do mouse. Isso faz com que a visualização seja ampliada ou reduzida.*
- **Participante faz:**
  - Posiciona cursor sobre a visualização e usa a roda do mouse.
  - Vê visualização ser ampliada e/ou reduzida.
- **Coordenador lê:**
  - *Alguma dúvida?*
- **Participante faz:**
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.

Em seguida, o participante é instruído sobre como usar um controle de videogame para a manipulação:

- **Coordenador lê:**

  - *Agora, então, você será instruído sobre como manipular a visualização usando um controle de videogame. Segure este controle de videogame.*

- **Coordenador faz:**
  - Entrega controle de videogame ao participante.
  - Configura controle de videogame para poder ser usado com a aplicação.
- **Coordenador lê:**
  - *Use o analógico esquerdo para rotacionar a visualização. Em seguida, use o analógico direito para mover a visualização.*
- **Participante faz:**
  - Usa analógicos esquerdo e direito do controle de videogame;
  - Vê visualização ser rotacionada e movida.
- **Coordenador lê:**
  - *Use as setas direcionais cima e baixo para controlar o ampliamento e redução da visualização. Em seguida, use as setas direcionais esquerda e direita, para "tombar" a visualização.*
- **Participante faz:**
  - Pressiona setas direcionais do controle de videogame.
  - Vê visualização ser ampliada/reduzida e tombada para esquerda/direita.
- **Coordenador lê:**
  - *Alguma dúvida?*
- **Participante faz:**
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.

Para finalizar o treinamento, o participante é instruído sobre como usar um tablet para manipulação.

- **Coordenador lê:**
  - *Para finalizar as instruções, você será instruído sobre como manipular a visualização utilizando um tablet. Segure este tablet.*
- **Coordenador faz:**
  - Entrega tablet ao participante.

  - Configura tablet para poder ser usado com a aplicação.

- **Coordenador lê:**

  - *Deslize o dedo sobre a visualização para rotacioná-la. Em seguida, deslize dois dedos sobre a visualização para movê-la.*

- **Participante faz:**

  - Desliza dedos sobre a visualização, na tela do tablet.
  - Vê visualização ser rotacionada e movida.

- **Coordenador lê:**

  - *Realize o gesto de "pinça" abrindo sobre a visualização para ampliá-la; em seguida, realize a "pinça" fechando para reduzi-la.*

- **Participante faz:**

  - Realiza gestos de "pinça" sobre a visualização, na tela do tablet.
  - Vê visualização ser ampliada e reduzida.

- **Coordenador lê:**

  - *Alguma dúvida?*

- **Participante faz:**

  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.

Com o participante instruído sobre a utilização da aplicação, prossegue-se, então, para a parte principal do experimento, as tarefas, que são:

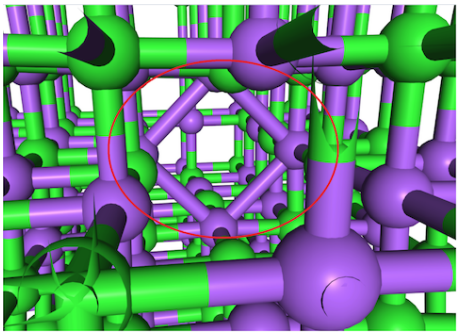
1. Usar mouse para encontrar e colocar em foco a vacância existente na estrutura task1.pdb (Figura 67a);
2. Usar controle de video game para encontrar e colocar em foco os átomos de boro e de nitrogênio existentes na estrutura task2.pdb (Figura 67b); e
3. Usar tablet para encontrar e colocar em foco os átomos de boro e de oxigênio existentes na estrutura task3.pdb (Figura 67c).

As tarefas são realizadas da seguinte maneira:

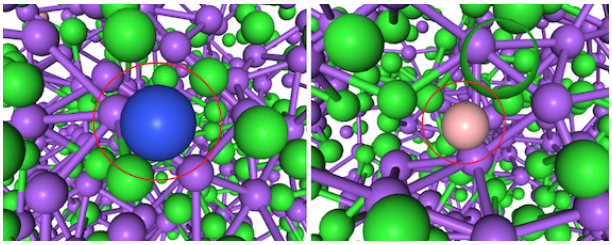
- **Coordenador lê:**

Figura 67 – Possíveis respostas para as tarefas propostas.

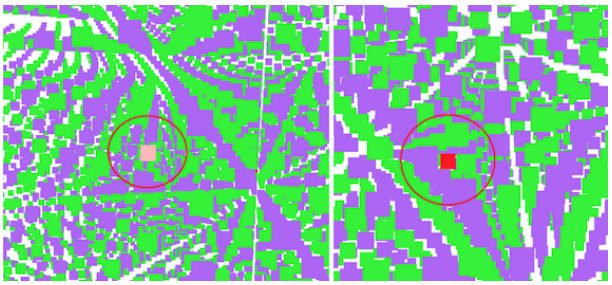
(a) Tarefa 1



(b) Tarefa 2



(c) Tarefa 3.



Fonte - Produzida pelo autor.

- *Com a compreensão de como usar a aplicação, você deve agora realizar quatro tarefas.*

  Após finalizar cada tarefa, você deve comunicar o coordenador, para que ele possa preparar a próxima tarefa. Durante o processo, caso deseje adicionar ou substituir representações ou controlar a visibilidade dos ligantes das estrutura (caso existam), solicite ao coordenador.

  Alguma dúvida?

- **Participante faz:**
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.
- **Coordenador faz:**
  - Carrega e configura task1.pdb.
- **Coordenador lê:**
  - *Na primeira tarefa, você deve usar o mouse para encontrar e colocar em foco a vacância existente no seguinte cristal de sal. A vacância é uma região do cristal que se assemelha a um "rombo", onde deveriam haver átomos, mas não há. Os átomos do cristal se aglutinam mais em torno da vacância, por isso, tente identificar para onde os átomos estão se deslocando.*

- **Coordenador faz:**
  - Entrega controles da aplicação ao participante.
  - Começa a medir tempo.
- **Participante faz:**
  - Tira dúvidas e pede ajuda (opcional).
  - Realiza tarefa.
  - Comunica coordenador.
- **Coordenador faz:**
  - Termina de medir tempo e o anota.
  - Dá nota a resposta.
  - Carrega e configura estrutura task2.pdb.
  - Configura controle de videogame.
- **Coordenador lê:**
  - *Na segunda tarefa, você deve usar o controle de videogame para encontrar e focar nos átomos de boro e de nitrogênio presentes no seguinte cristal de sal. No cristal, átomos de cloro são verdes e os átomos de sódio, roxos. Os átomos de boro e de nitrogênio que você deve encontrar são rosa claro e azul, respectivamente.*
- **Coordenador faz:**
  - Entrega controles da aplicação ao participante.
  - Começa a medir tempo
- **Participante faz:**
  - Tira dúvidas e pede ajuda (opcional).
  - Realiza tarefa.
  - Comunica coordenador.
- **Coordenador faz:**
  - Termina de medir tempo e o anota.
  - Dá nota a resposta.
  - Configura manipulação por dispositivo móvel.
  - Carrega e configura estrutura task3.pdb.
- **Coordenador lê:**

  - *Na terceira tarefa, você deve usar o tablet para encontrar para encontrar e focar nos átomos de boro e de nitrogênio no seguinte cristal de sal. Mais uma vez, os átomos de cloro e sódio, são, verdes e roxos, respectivamente, enquanto os átomos de boro e de oxigênio que você deve encontrar são rosa claro e vermelho, respectivamente.*

- **Coordenador faz:**
  - Entrega controles da aplicação ao participante.
  - Começa a medir tempo
- **Participante faz:**
  - Tira dúvidas e pede ajuda (opcional).
  - Realiza tarefa.
  - Comunica coordenador.
- **Coordenador faz:**
  - Termina de medir tempo e o anota.
  - Dá nota a resposta.

Após o participante finalizar as quatro tarefas, é dado um tempo para que ele possa usar a aplicação livremente, explorando o que desejar:

- **Coordenador lê:**
  - *Com as tarefas completadas, a partir de agora, você pode usar a aplicação para visualizar o que quiser, como quiser, por um período de no máximo 10 minutos.*
- **Coordenador faz:**
  - Entrega controles ao participante.
- **Participante faz:**
  - Tira dúvidas e pede ajuda (opcional).
  - Usa aplicação por até 10 minutos.
  - Avisa coordenador que finalizou uso (opcional).

Ao término dos 10 minutos, ou a indicação de parada pelo participante, a parte principal do experimento é encerrada.

Para finalizar, o participante deve responder ao questionário sobre sua experiência[1]:

- **Coordenador lê:**

[1] O questionário pode ser acessado no link: <https://docs.google.com/forms/d/e/1FAIpQLScImEisMLtLl01SvWgXqslFrrJgHUgYh0uitWiQy-S0nAvyuA/viewform>

  - *A parte principal do experimento está encerrada. No próximo passo, você deve responder a um questionário sobre experiência de uso da aplicação e dos equipamentos durante a realização das tarefas. Após isso, o experimento estará encerrado. Alguma dúvida?*

- **Participante faz:**
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.
- **Coordenador faz:**
  - Entrega questionário ao participante.
- **Participante faz:**
  - Responde questionário.
  - Tira dúvidas (opcional).
  - Sinaliza para prosseguir.
- **Coordenador lê:**
  - *Com isso, este experimento está encerrado. Muito obrigado por sua participação, ela significa muito para nós. Alguma dúvida final?*
- **Participante faz:**
  - Tira dúvidas (opcional).
  - Despede-se e vai embora.

APÊNDICE E – Questionário

Questionário sobre a utilização da aplicação web Dimmol.js junto ao aparato MiniCAVE, realizada no Laboratório de Interfaces e Visualização da Unesp.

O questionário é composto por 20 itens e mais um espaço de texto livre para comentários, observações, etc.

Para afirmações (18 itens), selecione o número que melhor identifica o grau em que você concorda com a afirmação, sendo que 1 (um) indica que você discorda totalmente da afirmação, enquanto 5 (cinco) indica que você concorda totalmente com a afirmação.

Para a questão , selecione a opção que melhor se aplica a realidade. Não há respostas corretas ou erradas.

1. Eu senti facilidade para manipular (arrastar, girar, zoom) a estrutura molecular na aplicação com o mouse.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

2. Eu senti que a manipulação com o mouse tinha resposta imediata e precisa.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

3. Eu senti facilidade para manipular a estrutura molecular na aplicação com o controle de video game.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

4. Eu senti que a manipulação com o controle de video game tinha resposta imediata e precisa.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

5. Eu senti facilidade para manipular a estrutura molecular na aplicação com um tablet.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

6. Eu senti que a manipulação com o tablet tinha resposta imediata e precisa.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

7. Qual maneira você julga melhor para manipular a estrutura molecular?

   ( ) Mouse

   ( ) Controle de video game

( ) Tablet

8. Eu senti que as estruturas moleculares eram manipuladas e animadas na aplicação com rapidez e suavidade.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

9. Eu senti que as estruturas moleculares se movimentavam sem atrasos ou inconsistências entre as três as telas do MiniCAVE.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

10. Eu considero que o MiniCAVE contribuiu muito para o entendimento de estruturas moleculares.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

11. Eu me senti imerso no ambiente virtual com as estruturas moleculares.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

12. Eu estaria disposto a visualizar mais estruturas moleculares usando a aplicação e o MiniCAVE.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

13. Eu vejo benefício na utilização da aplicação e do MiniCAVE em tópicos/disciplinas que demandam visualização molecular.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

14. Eu considero o equipamento de exibição como de alta qualidade

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

15. Eu considero o conteúdo visual do ambiente virtual como de alta qualidade.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

16. Eu acredito que o ambiente virtual consegue suportar múltiplos usuários humanos ao mesmo tempo.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

17. Eu considero interessantes as tarefas que consegui fazer no ambiente virtual.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

18. Eu senti que o ambiente virtual me permitiu completar minhas tarefas de várias maneiras diferentes.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

19. Eu senti que era capaz de reusar continuamente as técnicas aprendidas em tarefas anteriores para completar novas tarefas.

	1	2	3	4	5	
Discordo totalmente	( )	( )	( )	( )	( )	Concordo totalmente

20. Espaço para comentários, observações, reclamações, sugestões, etc.

    - Texto livre.

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

TERMO DE REPRODUÇÃO XEROGR□FICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 20/07/2018

Assinatura do autor