Heleno Murilo Campeão Vale

**METODOLOGIA DE INTEGRAÇÃO ON-LINE DE DADOS REAIS DE
MANUFATURA COM AMBIENTES DE REALIDADE VIRTUAL IMERSIVOS**

Tese apresentada à Escola de Engenharia de
São Carlos da Universidade de São Paulo,
como parte dos requisitos para a obtenção do
Título de Doutor em Engenharia Mecânica.

Área de concentração: Manufatura.

Orientador: Prof. Dr. Arthur José Vieira Porto.

São Carlos

2012

ESTE EXEMPLAR TRATA-SE
DA VERSÃO CORRIGIDA.
A VERSÃO ORIGINAL
ENCONTRA-SE DISPONÍVEL
JUNTO AO DEPARTAMENTO
DE ENGENHARIA MECÂNICA
DA EESC-USP

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

V149m Vale, Heleno Murilo Campeão
METODOLOGIA DE INTEGRAÇÃO ON-LINE DE DADOS REAIS DE
MANUFATURA COM AMBIENTES DE REALIDADE VIRTUAL
IMERSIVOS. / Heleno Murilo Campeão Vale; orientador
Arthur José Vieira Porto. São Carlos, 2012.

Tese (Doutorado) - Programa de Pós-Graduação em
Engenharia Mecânica e Área de Concentração em
Manufatura -- Escola de Engenharia de São Carlos da
Universidade de São Paulo, 2012.

1. Realidade Virtual Imersiva. 2. Manufatura
Virtual. 3. Integração On-Line de Dados Reais com
Ambientes Virtuais. I. Título.

FOLHA DE JULGAMENTO

Candidato: Bacharel **HELENO MURILO CAMPEÃO VALE**.

Título da tese: "Metodologia de integração on-line de dados reais de manufatura com ambientes de realidade virtual imersivos".

Data da defesa: 04/10/2012

Comissão Julgadora:

Resultado:

Prof. Titular **Arthur José Vieira Porto (Orientador)**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Prof. Titular **Eduardo Vila Gonçalves Filho**
(Escola de Engenharia de São Carlos/EESC)

Aprovado

Prof. Associado **Jonas de Carvalho**
(Escola de Engenharia de São Carlos/EESC)

Aprovado

Profa. Dra. **Jandira Guenka Palma**
(Universidade Estadual de Londrina/UEL)

Aprovado

Prof. Dr. **José Marcio Machado**
(Universidade Estadual Paulista "Júlio de Mesquita Filho"/UNESP – Campus de São José do Rio Preto)

APROVADO

Coordenador do Programa de Pós-Graduação em Engenharia Mecânica:
Prof. Associado **Marcelo Areias Trindade**

Presidente da Comissão de Pós-Graduação:
Prof. Titular **Denis Vinicius Coury**

À minha família.

Agradecimentos

À minha família, por toda paciência, apoio, carinho e incentivo.

Ao professor Dr. Arthur José Vieira Porto, pelos ensinamentos e pela confiança em mim depositada.

À professora Dra. Jandira Guenka Palma e ao professor Dr. Alexandre José Tiberti, por todos os atalhos sugeridos.

Aos amigos Claiton de Oliveira e Fábio Mariotto, pelas sugestões, ideias e apoio.

À querida Beth, pela amizade e prontidão.

Ao professor Dr. Mário Luiz Tronco, pelas orientações no programa PAE.

À Ana Paula, pela prontidão e eficiência.

Ao pesquisador André Del Bianco Giuffrida, pelos conhecimentos compartilhados.

À CAPES pela bolsa concedida.

Por fim, a todos os meus outros queridos amigos e colegas de laboratório, com os quais foram compartilhadas muitas risadas e bons momentos: Luciano Lulio, Rubens Tabile, Wellington Lopes, Robson (Galã), Heráclito Jaguaribe, David Sena, Hilano Carvalho, Fábio Perfetto, Antônio Santana e José Hamilton Gorgulho Júnior.

“Your quest is your purpose. Go on.”

Tobias Sammet

Resumo

VALE, H. M. C. **Metodologia de Integração on-line de Dados Reais de Manufatura com Ambientes de Realidade Virtual Imersivos**. 2012. 352p. Tese de Doutorado - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

A integração *on-line* de dados reais com ambientes de realidade virtual imersivos, criando sistemas híbridos, é um campo de pesquisa multidisciplinar, ainda pouco explorado. O alto nível de envolvimento do usuário proporcionado pela realidade virtual imersiva e a confiabilidade e precisão de informações provenientes da aquisição de dados reais, de maneira *on-line*, podem auxiliar o processo de tomada de decisão, em diversas áreas de aplicação. Este trabalho apresenta uma metodologia de integração *on-line* de dados reais de manufatura com ambientes de realidade virtual imersivos, mais especificamente, com sistemas CAVE. A metodologia foi desenvolvida voltada para a área de manufatura virtual, com foco em máquinas-ferramenta. Foram criados e interligados dois métodos de aquisição de dados *on-line*, um deles específico para máquinas-ferramenta, através do protocolo MTConnect e outro, genérico, via imagens de câmeras. O trabalho foi desenvolvido, em geral, com base em bibliotecas e *softwares* livres, bem como com dispositivos de baixo custo. Por fim, avaliou-se a metodologia através de estudos de caso, baseados em dados reais *on-line*, provenientes de máquinas-ferramenta reais e de simuladores de dados de máquinas reais. Os resultados deste trabalho podem auxiliar o desenvolvimento de ferramentas que aumentem do nível de confiabilidade das tomadas de decisões baseadas em simulações virtuais da área de manufatura. E, devido à sua implementação modular, ao formato hierárquico e unificado de sua estrutura e à padronização de seus tipos de dados, comunicações e formas de utilização, pode ser adequada a quaisquer áreas de aplicação que utilizem integração de dados reais *on-line* com ambientes virtuais imersivos.

Palavras-chave: Realidade Virtual Imersiva. Manufatura Virtual. Integração *On-Line* de Dados Reais com Ambientes Virtuais.

Abstract

VALE, H. M. C. **On-line Integration of Real Manufacturing Data with Immersive Virtual Reality Environments**. 2012. 352p. Thesis (Doctorate) – EESC-USP, University of São Paulo, São Carlos, 2012.

The on-line integration of real manufacturing data with immersive virtual reality environments, forming hybrid systems, is an underexplored multidisciplinary research field. The high level of users' involvement provided by immersive virtual reality and the reliability and accuracy of information acquired on-line from real data, can assist the decision making process, in many research areas. This research presents a way to on-line integrate manufacturing real data with immersive virtual reality environments, more specifically, with CAVEs. This work was developed toward the area of virtual manufacturing, with focus on machine tools. Two data acquisition methods were created and placed to work together, one specific for machine tools, through the MTConnect protocol and the other, generic, through camera images acquisition. This work was based on free libraries and software, as well as low cost devices. Finally, this work was evaluated through case studies, based on on-line real data, from real machine tools and real machine data simulators. The results achieved with this research can contribute to the development of new tools to increase the reliability level of manufacturing virtual simulations. Moreover, due to its modular implementation, the hierarquical and unified structure format and the standardization of its communication methods and data types, it can be arranged to almost any research areas which use integration of real on-line data with immersive virtual environments.

Keywords: Immersive Virtual Reality. Virtual Manufacturing. On-line Integration of Real Data and Virtual Environments.

LISTA DE FIGURAS

Figura 2-1- Representação de um dispositivo CAVE.....	49
Figura 2-2- Arquiteturas para a implementação do padrão MTConnect.	60
Figura 2-3- Relacionamento do adaptador MTConnect com um agente e um dispositivo.....	63
Figura 2-4- Inicialização de um agente.....	65
Figura 2-5- Exemplo de chão de fábrica com agentes e dispositivos.....	66
Figura 2-6- Montagem incorporada do agente	67
Figura 2-7- Trabalho sobre MTConnect e sistema de eventos discretos.	69
Figura 2-8- Monitoramento de fábrica via celular.....	70
Figura 2-9- Ambiente virtual de treinamento de manutenção para indústria.....	71
Figura 2-10- Cubos com transparência mostrando as camadas de controle de risco.	72
Figura 2-11- Diferentes níveis de temperatura mostrados em um objeto virtual através de cores.	73
Figura 3-1- Estrutura da sala do LMVI com paredes escuras e carpete no chão para maximização da qualidade visual e acústica	77
Figura 3-2- Disposição das telas de projeção do LMVI	78
Figura 3-3- Estruturas metálicas de sustentação dos projetores (a) superiores e (b) laterais...	80
Figura 3-4- (a) Duas imagens sobrepostas para utilização com óculos estereoscópicos e.....	81
Figura 3-5- Disposição e conexões do sistema de áudio.....	84
Figura 3-6- Visão frontal do sistema de sonorização do LMVI.....	85
Figura 3-7- Webcam e canetas laser com filtros do LMVI.....	87
Figura 3-8- Diagrama de Classes do <i>OpenSG</i> combinado com o <i>VR Juggler</i>	91
Figura 3-9- Opções de animação na exportação para VRML.....	103
Figura 3-10- Dispositivo com LEDs infravermelhos.....	113
Figura 3-11- Material utilizado para uniformizar e concentrar a luz do laser.....	115
Figura 3-12- Posicionamento da <i>webcam</i> no sistema de multiprojeção	116
Figura 3-13- Sequencia de imagens resultantes do pré-processamento do ponto luminoso..	117
Figura 3-14- Usuário afastando duas canetas <i>laser</i> para obtenção do <i>Zoom</i>	125
Figura 3-15- Dispositivo de interação por cor (azul, vermelho e verde).....	127
Figura 3-16- Atributos da seção <i>Header</i>	143
Figura 3-17- Atributos da seção <i>Header</i>	146

Figura 3-18- Comunicação típica entre uma aplicação e um agente	148
Figura 3-19- Exemplo de componentes e itens e suas relações.....	151
Figura 3-20- Marcadores coloridos, segmentação de imagem e obtenção de centróides.....	155
Figura 3-21- Passos para uso de dados reais de máquinas-ferramenta, adquiridos através de <i>webcams</i> , dentro de uma aplicação virtual.....	156
Figura 3-22- Matriz de pixels de uma imagem.....	157
Figura 3-23- Cores RGB (<i>Red, Green, Blue</i>).	160
Figura 3-24- Estrutura do padrão <i>Singleton</i>	162
Figura 3-25- Exemplo de estrutura do padrão <i>Factory</i>	164
Figura 3-26- Estrutura genérica do padrão <i>Factory</i>	164
Figura 4-1- Itens envolvidos na metodologia para a criação de novas ferramentas	168
Figura 4-2- Módulos principais do trabalho e seus relacionamentos	170
Figura 4-3- Macro visão do projeto	171
Figura 4-4- Aplicação virtual em comunicação com os módulos externos.	175
Figura 4-5- Tarefas paralelas (<i>Threads</i>).....	176
Figura 4-6- Configuração das unidades de medida do <i>3D Studio Max</i>	192
Figura 4-7- Eixos e direções adotadas pelo <i>software 3D Studio Max</i>	193
Figura 4-8- Eixos e direções do formato VRML e OpenSG (Y e Z invertidos e direção contrária de Z).....	193
Figura 4-9- Torno Mecânico modelado em <i>3D Studio Max</i>	196
Figura 4-10- Parte móvel “X” do torno mecânico.....	197
Figura 4-11- Exemplo de modelagem de centro de usinagem para o ROMI D600	198
Figura 4-12- Fresadora virtual.....	199
Figura 4-13- Acessório virtual para fresadora.....	199
Figura 4-14- Furadeira virtual	200
Figura 4-15- Acessório virtual para a furadeira.....	200
Figura 4-16- Esteira de transporte e operador	201
Figura 4-17- Braço robótico	202
Figura 4-18- Cenário genérico de chão de fábrica.....	203
Figura 4-19- Interface para definição dos parâmetros iniciais da aplicação virtual.....	206
Figura 4-20- Fluxograma de aquisição de dados com MTConnect.....	210
Figura 4-21- Casos de uso para um adaptador MTConnect.....	212
Figura 4-22- Diagrama de sequência para o adaptador.....	213
Figura 4-23- Diagrama de classes de um adaptador.....	215

Figura 4-24- Conversão de dados do agente MTConnect.....	226
Figura 4-25- Armazenamento de dados do agente MTConnect.....	229
Figura 4-26- Captação de dados com parâmetros <i>from</i> e <i>count</i>	230
Figura 4-27- Dados adquiridos de um agente.....	232
Figura 4-28- Janela de aquisição de dados de um agente	233
Figura 4-29- Marcadores coloridos circulares.....	236
Figura 4-30- Cores naturais RGB e suas primeiras intersecções.....	237
Figura 4-31- Arranjo com uma câmera para captação de 1 eixo para porta de dispositivo ...	238
Figura 4-32- Arranjo com uma câmera para captação de 1 eixo para esteira	238
Figura 4-33- Exemplo fictício de braço robótico com marcadores.	239
Figura 4-34- Exemplo de imagem com ruído.....	240
Figura 4-35- Passos do algoritmo de segmentação.....	241
Figura 4-36- Exemplo de imagem binária.....	243
Figura 4-37- Exemplo de figura binária fechada única.....	244
Figura 4-38- Centroide da figura binária única	245
Figura 4-39- Arquivo criado pelo módulo de aquisição de dados por câmera.....	246
Figura 4-40- Diagrama de classes do <i>Kernel</i> do VR Juggler	248
Figura 4-41- Vetor de armazenamento dos parâmetros.....	249
Figura 4-42- <i>VJKeyboard</i> no procedimento <i>init</i> da biblioteca OpenSG.....	251
Figura 4-43- <i>VJKeyboard</i> no arquivo de configuração do <i>VR Juggler</i>	251
Figura 4-44- Teclas com funcionalidades implementadas.....	252
Figura 4-45- Configuração da interação via teclado no <i>VR Juggler</i>	256
Figura 4-46- Grafo de cena somente com <i>root</i>	258
Figura 4-47- Criação e relação entre dispositivos na cena.....	259
Figura 4-48- Posição dos nós <i>Beacon</i> e <i>Point Light</i> no grafo de cena.	265
Figura 4-49- Luz posicionada no grafo de cena.	266
Figura 4-50- Aquisição de dados de dispositivo real via MTConnect.....	269
Figura 4-51- Aquisição de dados via câmera.	270
Figura 4-52- Aquisição de dados via câmera com intermediação de um agente.	271
Figura 4-53- Semáforo para leitura e escrita dos dois vetores do <i>buffer</i>	272
Figura 4-54- Atualização dos dados no procedimento <i>preFrame</i>	273
Figura 4-55- Métodos <i>translateObject</i> e <i>RotateObject</i>	274
Figura 4-56- Comunicação de mensagens via <i>socket</i>	277
Figura 4-57- Sincronia dos <i>frames</i> com dados de entrada de dispositivo externo.	280

Figura 4-58- <i>Flag</i> Cliente-Servidor.	283
Figura 4-59- Peça subdividida em unidades para modificação dinâmica	289
Figura 4-60- Colisão da ferramenta com as unidades de composição das peças.	290
Figura 5-1- Primeira parte da consulta <i>probe</i> para <i>Live Agent</i>	293
Figura 5-2- Segunda parte da consulta <i>probe</i> para <i>Live Agent</i>	294
Figura 5-3- Terceira parte da consulta <i>probe</i> para <i>Live Agent</i>	295
Figura 5-4- Quarta parte da consulta <i>probe</i> para <i>Live Agent</i>	296
Figura 5-5- Quinta parte da consulta <i>probe</i> para <i>Live Agent</i>	297
Figura 5-6- Sexta e última parte da consulta <i>probe</i> para <i>Live Agent</i>	298
Figura 5-7- Montagem interna da requisição <i>current</i> para o estudo de caso	299
Figura 5-8- Parâmetro <i>nextSequence</i> de <i>current</i>	299
Figura 5-9- Montagem interna da requisição <i>sample</i> para o estudo de caso.....	300
Figura 5-10- Arquivos XML obtidos do agente para dois dispositivos.....	301
Figura 5-11- Dados de X adquiridos pela requisição <i>sample</i> parametrizada por <i>Position</i>	302
Figura 5-12- Dados de Y adquiridos pela requisição <i>sample</i> parametrizada por <i>Position</i>	303
Figura 5-13- Dados de posicionamento dentro do arquivo XML.....	304
Figura 5-14- Dados de posicionamento no <i>buffer</i> da aplicação principal.....	305
Figura 5-15- Dados de posicionamento aplicados à mesa do dispositivo.....	305
Figura 5-16- Comunicação entre o módulo simulador de dados reais e a aplicação.....	307
Figura 5-17- Estrutura do arquivo XML do módulo simulador de dados reais	308
Figura 5-18- Arquivo criado pelo módulo simulador para este estudo de caso	309
Figura 5-19- Dados de posicionamento do eixo Z da furadeira no arquivo XML	310
Figura 5-20- Dados de posicionamento da furadeira no AV.....	310
Figura 5-21- Estudo de caso para ROMI D800 com CNC FANUC	312
Figura 5-22- Amostra de dados de posicionamento do adaptador	313
Figura 5-23- Câmera de baixo custo utilizada para aquisição de imagens	314
Figura 5-24- Marcadores de papel cartão, vermelho, verde e azul.....	315
Figura 5-25- Marcador vermelho acoplado à porta do dispositivo.....	316
Figura 5-26- Separação do canal vermelho da imagem via <i>Threshold</i>	316
Figura 5-27- Separação do canal vermelho com logomarca coberta manualmente	317
Figura 5-28- Imagem convertida para HSV	317
Figura 5-29- Segmentação com HSV	318
Figura 5-30- Identificação da figura fechada e seu centroide X, Y.....	319
Figura 5-31- Dispositivo com porta fechada.....	319

Figura 5-32- Dispositivo com porta entreaberta.....	320
Figura 5-33- Ambiente virtual híbrido.....	321
Figura 5-34- Relações envolvidas no ambiente híbrido testado.....	322

LISTA DE TABELAS

Tabela 3-1- Unidades utilizadas pelo agente MTConnect	144
Tabela 4-1- Solicitações aceitas pelo agente	227

LISTA DE QUADROS

Quadro 2-1- Exemplo de arquivo XML.....	59
Quadro 3-1- Procedimento para criação do nó raiz	104
Quadro 3-2- Adição de uma câmera virtual ao grafo de cena.....	105
Quadro 3-3- Adição de um objeto 3D ao grafo de cena	106
Quadro 3-4- Adição de múltiplos objetos 3D ao grafo de cena	107
Quadro 3-5- Adição do nó de iluminação ao grafo de cena.....	107
Quadro 3-6- Importação de objeto 3D a partir de um arquivo VRML.....	109
Quadro 3-7- Linha de comando com o parâmetro “vcamera”	109
Quadro 3-8- Implementação da função de busca de um nó no grafo de cena.....	110
Quadro 3-9- Trecho de código para rotação e translação.....	111
Quadro 3-10- Trecho de código condensado para o pré-processamento da imagem	118
Quadro 3-11- Detecção do ponto de luz na imagem	119
Quadro 3-12- Obtenção da posição e tamanho do ponto de luz.....	120
Quadro 3-13- Mapeamento da movimentação do laser no plano	121
Quadro 3-14- Criação e destruição da <i>Thread</i> de detecção de movimento do <i>laser</i>	122
Quadro 3-15- Função de envio de dados de movimentação para os clientes.....	123
Quadro 3-16- Código condensado do <i>Zoom</i> com dois pontos de luz	126
Quadro 3-17- Função para selecionar um objeto 3D a partir da posição de um ponteiro 2D	129
Quadro 3-18- Mapeamento das coordenadas do ponteiro para as dimensões da tela.....	130
Quadro 3-19- Projeção de um raio a partir da posição da câmera até o “infinito”	130
Quadro 3-20- Obtenção de um objeto interceptado pelo raio projetado.....	131
Quadro 3-21- Função para mover o objeto selecionado com o movimento do ponteiro.....	132
Quadro 3-22- Casal de itens <i>Key</i> e <i>Value</i>	134
Quadro 3-23- Exemplo de linha SHDR	135
Quadro 3-24- Mensagem sem data e hora.....	135
Quadro 3-25- Mensagem com dados de posição coletados ao mesmo tempo	136
Quadro 3-26- Mensagem do adaptador ao agente com vocabulário adequado.....	136
Quadro 3-27- Mensagem com <i>id</i> e <i>name</i>	137
Quadro 3-28- Mensagem somente com identificador <i>id</i>	137
Quadro 3-29- Valor de <i>Source</i> para nomes longos.....	137

Quadro 3-30- Dado referente à temperatura de óleo	139
Quadro 3-31- Item de condição de temperatura	139
Quadro 3-32- Dados internos do arquivo XML para a temperatura.....	139
Quadro 3-33- Exemplo de dados para uma situação normal.....	139
Quadro 3-34- Mensagem simples	140
Quadro 3-35- Mensagem simples em XML.....	140
Quadro 3-36- Primeira mensagem de comunicação do agente com o adaptador.....	141
Quadro 3-37- Resposta inicial do adaptador para o agente.....	141
Quadro 3-38- Estrutura do arquivo XML <i>MTConnectDevices</i>	145
Quadro 3-39- Estrutura do arquivo <i>MTConnectStreams</i>	145
Quadro 3-40- Estrutura do arquivo <i>MTConnectError</i>	146
Quadro 3-41- Exemplo de resposta a <i>probe</i>	150
Quadro 3-42- Declaração de uma classe <i>Singleton</i>	162
Quadro 3-43- Implementação de uma classe <i>Singleton</i>	162
Quadro 3-44- Implementação típica da classe <i>Creator</i>	165
Quadro 4-1- Exemplo de exportação VRML	194
Quadro 4-2- Linha de comando resultante do módulo de criação de parâmetros	204
Quadro 4-3- Estrutura mínima de um adaptador	216
Quadro 4-4- Implementações do adaptador mínimo	217
Quadro 4-5- Arquivo de cabeçalho para o controlador EMC	219
Quadro 4-6- Construtor do adaptador EMC.....	220
Quadro 4-7- Método para aquisição de dados do controlador EMC	221
Quadro 4-8- Método de conexão com o EMC	222
Quadro 4-9- Método de desconexão com o EMC	223
Quadro 4-10- Aquisição das posições dos eixos	223
Quadro 4-11- Verificação do estado do dispositivo	224
Quadro 4-12- Aquisição do estado da execução do dispositivo EMC.....	224
Quadro 4-13- Destrutor do adaptador para EMC	225
Quadro 4-14- Inicialização do agente	228
Quadro 4-15- Caminho completo para o arquivo de configuração	228
Quadro 4-16- Estrutura do arquivo de configuração do Adaptador	233
Quadro 4-17- <i>Thread</i> de <i>VR Juggler</i> e <i>Boost</i>	255
Quadro 4-18- Máscaras de OpenSG	257
Quadro 4-19- Linha de comando para criação de dispositivos	259

Quadro 4-20- Exemplo prático de criação de dispositivo em linha de comando	260
Quadro 4-21- Primeiro método utilizado para entrada de parâmetros da cena	261
Quadro B-1- Classe interna <i>DeviceDatum</i>	348
Quadro B-2- Subclasse interna <i>Event</i>	348
Quadro B-3- Subclasse interna <i>Sample</i>	349
Quadro B-4- Método de identificação de alteração nos valores	349
Quadro B-5- Método de inicialização de servidor de um adaptador	350
Quadro B-6- Formato padrão dos dados de um adaptador	350
Quadro B-7- Formato dos dados de <i>time_stamp</i>	351
Quadro B-8- Dados de um adaptador	351
Quadro B-9- Padrão de dados para o tipo <i>Alarm</i>	351
Quadro B-10- Exemplos de dados do tipo <i>Alarm</i>	352

LISTA DE ABREVIATURAS E SIGLAS

AV – Ambiente Virtual

CAD - *Computer-Aided Design*

CAM - *Computer-Aided Manufacturing*

CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CAVE - *Cave Automatic Virtual Environment*

CNC - *Computer Numerical Control*

EESC – Escola de Engenharia de São Carlos

FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo

HMD - *Head-Mounted Display*

HTTP – Hyper Text Transfer Protocol

HUD - Heads-Up Display

LMVI - Laboratório Multiusuário de Visualização 3D Imersiva

MP3 - *MPEG Layer-3*

MV - Manufatura Virtual

NEPAS – Núcleo de Ensino e Pesquisa em Automação e Simulação

PCM - *Pulse Code Modulation*

RGB – Red, Green, Blue

RV - Realidade Virtual

RVI - Realidade Virtual Imersiva

USP – Universidade de São Paulo

VRML – Virtual Reality Modeling Language

XML - *eXtensible Markup Language*

WMA - *Windows Media Audio*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

1	INTRODUÇÃO	33
1.1	OBJETIVOS	35
1.2	MOTIVAÇÃO	37
1.3	ORGANIZAÇÃO DO TRABALHO	38
2	REVISÃO DA LITERATURA	39
2.1	REALIDADE VIRTUAL	39
2.1.1	Sistemas de Multiprojeção Imersiva - CAVE	46
2.1.1.1	Sonorização Para Sistemas de Multiprojeção Imersivos	49
2.1.2	Manufatura Virtual	54
2.2	MONITORAMENTO DE DADOS REAIS EM AMBIENTES VIRTUAIS	57
2.2.1	Protocolo MTConnect	57
2.2.1.1	Adaptador	62
2.2.1.2	Agente	64
2.2.2	Outras pesquisas sobre monitoramento	68
3	MATERIAIS E MÉTODOS	75
3.1	ESPAÇO FÍSICO	76
3.2	ESTRUTURA DE PROJEÇÃO	77
3.2.1	Telas de Projeção	78
3.2.2	Projetores	79
3.2.3	Óculos Estereoscópicos Passivos	80
3.3	ESTRUTURA COMPUTACIONAL	81
3.3.1	Computadores Principais	82
3.3.2	Computadores Auxiliares	82
3.4	SISTEMA DE SOM 3D 5.1	83
3.4.1	Disposição dos Alto-Falantes	85
3.5	DISPOSITIVOS DE ENTRADA DE DADOS	86
3.5.1	Controle de Vídeo Game Wii	86
3.5.2	Câmeras e Canetas <i>Laser</i> para interação com o ambiente	87
3.5.3	Joystick	87
3.6	PLATAFORMA E BIBLIOTECAS DE PROGRAMAÇÃO	88
3.6.1	VR Juggler	89
3.6.2	OpenSG	90
3.6.3	<i>OpenCV</i>	91
3.6.4	OpenAL	92

3.7	UTILIZAÇÃO DO SISTEMA DE RVI	93
3.7.1	Posicionamento, foco e alinhamento dos projetores	94
3.7.1.1	Foco e Projeção com Auxílio de Espelho	95
3.7.2	Saídas de sinal DVI e RGB	96
3.7.3	Montagem do Cluster de computadores	97
3.7.3.1	Sincronização das telas de projeção	98
3.7.4	Sonorização	99
3.7.5	Modelagem Geométrica dos Objetos Tridimensionais	101
3.7.5.1	Extensão de arquivos 3D e VRML	102
3.7.6	O Grafo de Cena	103
3.7.7	Carregamento de objetos virtuais na cena	108
3.7.8	Busca de Objetos no Grafo	109
3.7.9	Método para Animação, Translação e Rotação de Objetos Virtuais Sob Controle das Plataformas <i>OpenSG</i> e <i>VR Juggler</i>	110
3.7.10	Navegação e interação no ambiente virtual de manufatura desenvolvido	112
3.7.10.1	Tentativas de navegação via <i>bluetooth</i> e <i>Leds</i>	112
3.7.10.2	Uso de canetas <i>laser</i> , câmeras e biblioteca <i>OpenCV</i> para navegação	113
3.7.10.3	Configuração e Posicionamento da <i>webcam</i>	115
3.7.10.4	Pré-processamento da Imagem para Filtragem do Laser	117
3.7.10.5	Deteccção do Ponto de Luz Na Imagem Pré-processada	118
3.7.10.6	Mapeamento da Movimentação do Ponto de Luz no Plano	120
3.7.10.7	Criação da <i>Thread</i> de Deteccção do <i>Laser</i>	121
3.7.10.8	Comunicação dos Dados de Movimentação	122
3.7.10.9	Implementação da Função <i>Zoom</i> com o <i>Laser</i>	123
3.7.10.10	Dispositivo de Interação por Cor	126
3.7.10.11	Ponteiro para Seleção de Objetos na Cena	127
3.7.11	Inicialização das Aplicações	132
3.8	AQUISIÇÃO DE DADOS DE MÁQUINAS-FERRAMENTA	133
3.8.1	Aquisição de dados via MTConnect	134
3.8.1.1	Visão geral do protocolo SHDR	134
3.8.1.1.1	Heartbeats	140
3.8.1.2	Estruturas dos documentos de um agente MTConnect	143
3.8.1.2.1	Requisições probe	149
3.8.1.2.2	Requisições sample	151
3.8.1.2.3	Requisições current	152
3.8.1.3	Vocabulário MTConnect	153
3.9	AQUISIÇÃO DE DADOS VIA WEBCAM	154
3.9.1	Segmentação de imagens	157
3.9.1.1	RGB e HSV	159
3.10	PADRÕES DE PROJETO DE SOFTWARE	161
3.10.1	<i>Singleton</i>	161
3.10.2	<i>Factory</i>	163
4	METODOLOGIA DE INTEGRAÇÃO ON-LINE DE DADOS REAIS DE MANUFATURA COM AMBIENTES DE REALIDADE VIRTUAL IMERSIVOS	167
4.1	CONSIDERAÇÕES INICIAIS	167
4.2	PROJETO	171
4.3	DEFINIÇÃO DOS TIPOS DE DADOS UTILIZADOS	176
4.4	DEFINIÇÃO DA HIERARQUIA DE CLASSES	183
4.5	CRIAÇÃO E PADRONIZAÇÃO DA MODELAGEM DE OBJETOS VIRTUAIS	188
4.5.1	Criação dos componentes móveis dos dispositivos	195
4.6	MÓDULO PARA CRIAÇÃO DOS PARÂMETROS E EXECUÇÃO DA APLICAÇÃO	203

4.7	MÓDULO PARA AQUISIÇÃO DE DADOS REAIS COM O PROTOCOLO MTCONNECT	209
4.7.1	Desenvolvimento do módulo adaptador	211
4.7.2	Desenvolvimento do módulo agente	225
4.8	MÓDULO PARA AQUISIÇÃO DE DADOS REAIS COM O USO DE CÂMERAS	234
4.9	ESTRUTURA DA APLICAÇÃO VIRTUAL	247
4.9.1	Procedimento principal	247
4.9.2	Funções do teclado	251
4.9.3	Inicialização do grafo de cena	256
4.9.4	Instanciação e relacionamento de dispositivos	258
4.9.5	Servidores e clientes	262
4.9.6	Iluminação	263
4.9.7	Sonorização	266
4.9.8	Aquisição dos dados dos módulos externos	267
4.9.9	Utilização e sincronização dos dados <i>on-line</i> adquiridos	271
4.9.10	Encerramento da aplicação	275
4.10	MÓDULO DE SINCRONIZAÇÃO DE OPERAÇÃO DE MÁQUINAS REAIS	275
4.11	SINCRONIZAÇÃO DOS DADOS ENTRE OS COMPUTADORES DO CLUSTER	278
4.11.1	Tarefas do computador principal do <i>cluster</i>	282
4.11.2	Tarefas dos demais computadores do <i>cluster</i>	286
4.12	ESTENDENDO A METODOLOGIA	287
4.12.1	Modificação dinâmica das peças	289
5	ESTUDO DE CASO E RESULTADOS	291
5.1	MTCONNECT <i>LIVE AGENT</i>	292
5.2	MÓDULO DE SIMULAÇÃO DE DADOS REAIS	306
5.3	AQUISIÇÃO DE DADOS REAIS DE CNC FANUC	311
5.4	AQUISIÇÃO DE DADOS REAIS ADICIONAIS ADQUIRIDOS POR CÂMERA	314
5.5	AMBIENTE VIRTUAL HÍBRIDO	320
6	CONCLUSÃO	323
6.1	CONTRIBUIÇÃO PRINCIPAL	324
6.2	CONSIDERAÇÕES FINAIS	325
6.3	TRABALHOS FUTUROS	327
REFERÊNCIAS		329
BIBLIOGRAFIA CONSULTADA		339
APÊNDICE A – VOCABULÁRIO MTCONNECT		341
APÊNDICE B – ESTRUTURA INTERNA GENÉRICA DE UM ADAPTADOR		347
A.1	DADOS DE UM ADAPTADOR	350

1 INTRODUÇÃO

A tecnologia de manufatura é uma área em contínuo desenvolvimento que, desde seu início em, aproximadamente, 4000AC, até os dias de hoje, sofre contínuo processo de inovação. No entanto, a maioria dos desenvolvimentos tecnológicos na área de manufatura ocorreu depois da revolução industrial, principalmente, nos últimos vinte e cinco anos, com a impactante evolução da tecnologia de computação e eletrônica (GARCÍA-PLAZA, 2011).

Recentemente, os processos de fabricação têm sido profundamente influenciados pela globalização e a elevada competição do mercado induz a um aperfeiçoamento contínuo dos processos de manufatura (HERNANDEZ-MATIAS ET AL. 2008), exigindo uma postura que minimize custos e maximize a produção, por parte dos gerentes de chão de fábrica.

Como agente viabilizador dessa nova e ágil forma de trabalho, a tecnologia de Realidade Virtual (RV) já foi vastamente aplicada a processos de manufatura, como, por exemplo, no aprimoramento de planejamento, modelagem, visualização e análise desses sistemas (LEE, 2011). Outras aplicações podem ser encontradas no trabalho de Palma (2001), Chai (2010), Marinov (2004), Sung (2003), Wong (2010), dentre muitos outros.

Devida à alta aplicabilidade da RV na área de manufatura, um novo termo foi criado, no meio científico, para simplificar as referências a esse tipo de aplicação: Manufatura Virtual (MV). O termo foi criado há mais de vinte anos atrás (CHAI, 2010) e possui várias definições na literatura. Zhang (2011) define MV como um mapeamento da usinagem prática em Ambiente Virtual (AV), tendo como resultado imediato uma produção totalmente digital, sem produção real e sem perda de material ou energia. Por sua vez, Souza (2006) afirma que a manufatura virtual representa uma abordagem emergente que as empresas podem utilizar para melhorar seus processos, introduzindo novos produtos no mercado, mais rapidamente e com melhor custo. Por fim, Marinov (2004) define Manufatura Virtual como um emergente

conceito, para integração de diferentes áreas da manufatura, através do uso de tecnologia computacional, na criação e execução de modelos virtuais.

Avanços da tecnologia de computação possibilitaram o desenvolvimento de dispositivos inovadores para a RV, permitindo, assim, o surgimento da Realidade Virtual Imersiva (RVI). Cruz-Neira (1992) criou em 1992 um dispositivo para possibilitar a imersão completa de um usuário em um AV. Esse dispositivo recebeu o nome de CAVE, um acrônimo recursivo para "*CAVE Automatic Virtual Environment*". Trata-se de um dispositivo composto por três ou mais telas de projeção que circundam um ou mais usuários e, com auxílio de óculos estereoscópicos, é capaz de isolar totalmente ou parcialmente o usuário, do mundo real. Banerjee (2001), afirma que a imersão completa de um usuário em um Ambiente Virtual (AV) é um grande avanço tecnológico se comparada aos pacotes CAD (*Computer-Aided Design*) e CAM (*Computer-Aided Manufacturing*) tradicionais, que limitam algumas interações intuitivas de usuários.

Além da utilização da RV imersiva para otimização da imersão para a análise e planejamento de processos, a utilização de dados reais *on-line*, juntamente a dados virtuais, em um sistema híbrido, cria novas possibilidades de avaliação para a área de manufatura, aumentando a confiabilidade e precisão dos dados e, conseqüentemente, das tomadas de decisão. Em geral, a maioria dos modelos de simulação tradicionais possuem poucas opções para a prática de alterações em tempo real (DESSOUKY et al. 2001). Além disso, embora muitos simuladores ofereçam a possibilidade de interação, eles não podem colocar o usuário em uma representação 3D imersiva do cenário simulado (PULUGURTHA et al. 2002).

O Laboratório Multi-usuário de Visualização 3D Imersiva (LMVI), da Escola de Engenharia de São Carlos - USP (EESC-USP), possui um dispositivo de RVI, do tipo CAVE, com três telas de projeção, que foi montado, instalado e configurado para pesquisas na área de RV e simulação para manufatura, dentre outras. Ferramentas de interação, de baixo custo,

foram desenvolvidas para que o usuário pudesse controlar o mundo virtual, durante a execução da aplicação. Módulos específicos para obtenção, comunicação e inserção de dados reais em AVs, tanto para máquinas-ferramenta, quanto para dispositivos genéricos, foram desenvolvidos neste trabalho. Além disso, diversos estudos de caso foram realizados com base nos dados reais adquiridos de diferentes fontes, como dispositivos reais e simuladores de dados. Todo o desenvolvimento utilizou, em geral, ferramentas, dispositivos, bibliotecas e *softwares* de baixo custo. Por fim, a metodologia foi desenvolvida de maneira modular e fundamentada por técnicas que visam sua flexibilidade e extensibilidade a outros dispositivos de manufatura, bem como a outras áreas de pesquisa. Os objetivos desta pesquisa são citados a seguir.

1.1 OBJETIVOS

O objetivo principal desta pesquisa foi o desenvolvimento de uma metodologia para viabilização e padronização do uso de dados reais, de maneira *on-line*, em ambientes virtuais imersivos, com foco em dados da área de manufatura. O uso de dados reais em um ambiente virtual é considerado um sistema híbrido e, no caso deste trabalho, a utilização da desse sistema pode ser considerada uma forma de simulação, pois as máquinas reais não estão todas em um mesmo local, mas os dados utilizados são reais. A utilização de dados reais de maneira *on-line* em ambientes virtuais, contribui para o aumento da confiabilidade das tomadas de decisão fundamentadas por simulações virtuais, uma vez que esse tipo de aquisição não compromete a fidedignidade dos dados simulados.

Todas as etapas do desenvolvimento foram alcançadas com base em bibliotecas e *softwares* livres e dispositivos de interação de baixo custo, além disso, alguns objetivos específicos nortearam o a busca pelo objetivo principal:

- Fundamentação do desenvolvimento por técnicas que acolhessem a flexibilidade e extensibilidade da metodologia a outros dispositivos de manufatura, diferentes dos utilizados neste desenvolvimento, bem como a outras áreas de aplicação.
- Documentação do estudo, montagem, configuração e instalação do *hardware* de um CAVE, uma vez que, durante o desenvolvimento deste trabalho, identificou-se esse tipo de documentação como escassa;
- Definição e instalação dos *softwares* livres, linguagens de programação e bibliotecas mais adequadas para o completo funcionamento de um dispositivo de RV imersiva do tipo CAVE;
- Desenvolvimento de ferramentas de baixo custo para operação de um dispositivo de RV imersiva do tipo CAVE;
- Desenvolvimento de um módulo especializado na aquisição e comunicação de dados de máquinas-ferramenta;
- Desenvolvimento de um módulo de aquisição de dados de posicionamento de partes móveis via câmeras de baixo custo;

1.2 MOTIVAÇÃO

O desenvolvimento deste trabalho foi motivado principalmente por pesquisas da área de manufatura que expõem a necessidade de aprimoramento e criação de novas ferramentas de planejamento para os profissionais desta área. Para Banerjee (2001) é evidente que ferramentas de planejamento e operação são importantes para administradores de fábricas que custam milhões, ou até mesmo bilhões de dólares. O autor acrescenta ainda que, geralmente, a própria fábrica é o campo de testes para novas operações, o que pode acarretar na interrupção de parte do processo. Outros autores que enfatizam a importância de novas ferramentas para a manufatura são Lim e Zhang (2003), que afirmam que o mercado mundial vem se tornando cada vez mais dinâmico requerendo dos sistemas de manufatura respostas rápidas e flexibilidade para mudanças de cenários por parte do cliente. A Realidade Virtual, principalmente a RV do tipo Imersiva, é uma ferramenta que pode ser utilizada para a reprodução virtual de ambientes reais com alto nível de detalhamento e imersão, podendo funcionar, assim, como uma ferramenta de planejamento para o embasamento adequado de tomadas de decisão na área de manufatura.

Existem, atualmente, várias ferramentas que utilizam a RV para auxílio no planejamento, modelagem, visualização e análise de problemas da área de manufatura. Algumas dessas ferramentas utilizam a RVI para possibilitar a imersão do usuário, maximizando a confiabilidade das tomadas de decisão baseadas no uso de um AV. Porém, não foi encontrado na literatura qualquer trabalho sobre a utilização de RVI juntamente a dados reais adquiridos de maneira *on-line*, em um AV de manufatura híbrido.

Um dos objetivos deste trabalho, descritos no tópico anterior, é a criação de dois módulos de aquisição de dados apropriados para a área de manufatura, mais especificamente para atividades de chão de fábrica, como, por exemplo, a captação de dados de máquinas-

ferramenta. A obtenção, padronização e junção do uso *on-line* desse tipo de dado com ambientes de RVI é uma tarefa com pouca documentação na área de manufatura. Portanto, a motivação deste trabalho foi a tanto a criação desses módulos, quanto sua padronização e documentação. Espera-se que esta pesquisa possa preencher parte desta lacuna e contribuir com o desenvolvimento de novas ferramentas para a área de manufatura.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis capítulos, incluindo este que apresentou o contexto no qual o trabalho se insere, seu objetivo e suas motivações.

No capítulo dois é apresentado o estado da arte e o embasamento teórico das áreas envolvidas na produção deste trabalho. No capítulo 3 é descrito o Laboratório Multiusuário de Visualização 3D Imersiva (LMVI), onde foram desenvolvidas todas as etapas práticas deste trabalho. O objetivo desse capítulo é o embasamento do leitor quanto às estruturas de *hardware* e *software* utilizadas, bem como a documentação dos materiais e métodos empregados na construção, configuração e utilização de um CAVE.

No capítulo quatro estão explicitados todos os módulos e etapas envolvidas no desenvolvimento da metodologia de integração *on-line* de dados reais de manufatura, em ambientes virtuais imersivos. No capítulo cinco são apresentados os estudos de caso para validação da metodologia, bem como os resultados obtidos.

Por fim, no capítulo final é apresentada a conclusão do trabalho e as últimas considerações desta pesquisa. Alguns trabalhos futuros são, também, sugeridos neste capítulo. Referências e apêndices completam o trabalho.

2 REVISÃO DA LITERATURA

2.1 REALIDADE VIRTUAL

Em 28 de agosto de 1962 foi registrada, nos Estados Unidos, por M. L. Heilig, a patente do simulador Sensorama, considerado por muitos como o primeiro dispositivo de Realidade Virtual. Desde então, inúmeras definições foram criadas para o termo Realidade Virtual (RV). Banerjee (2001) define RV como a capacidade de criação e interação no ciberespaço, um espaço que tenha grande similaridade com o mundo real ao nosso redor. Reihngold (1991), por sua vez, define RV como um ambiente imersivo, sintético, ou gerado por computador, que forneça a experiência de "estar lá".

Os primeiros trabalhos de RV na área de simulação foram os simuladores de vôo da força aérea dos Estados Unidos, desenvolvidos logo após a Segunda Guerra Mundial (JACOBSON, 2002). No entanto, o termo "Realidade Virtual" é creditado a Jaron Lanier, que nos anos 80 juntamente com Thomas Zimmerman desenvolveu a primeira luva de dados, chamada DataGlove, usada para manipular dados em aplicações de RV (ZIMMERMAN et al., 1987), além da introdução dos primeiros capacetes de visualização (HMD head-mounted displays) de forma comercial (BEIER, 1998). Outros termos relacionados são:

- Ciberespaço (GIBSON, 1984);
- Realidade Articial (KRUEGER; GIONFRIDDO; HINRICHSEN, 1985);
- Mundos Virtuais e Ambientes Virtuais (FISHER; TAZELAAR, 1990; ELLIS, 1994).

O termo "Realidade Virtual" é utilizado em uma variedade de formas e, algumas vezes, de uma maneira confusa e distorcida. Beier (1998) afirma que, originalmente, o termo se referia à "Realidade Virtual Imersiva", uma vez que, a utilização do termo, por Jaron Lanier, era uma forma de diferenciar as simulações computacionais tradicionais de simulações envolvendo usuários imersos em um ambiente artificial, gerado por computador (RHEINGOLD, 1991; ARAUJO, 1996).

Termos alternativos como "Mundos Virtuais" ou "Ambientes Virtuais" (AV), refletem mais adequadamente a vasta gama de aplicações possíveis e o fato de a tecnologia ainda estar distante de simular completamente a realidade. Ambientes Virtuais é um dos termos preferidos pela comunidade acadêmica (BEIER, 2000).

Também chamados de mundos virtuais ou cenas virtuais, esses ambientes sintéticos são caracterizados por três idéias básicas: interação, imersão e envolvimento (MORIE, 1994).

A idéia de imersão está ligada à sensação de estar dentro do ambiente, ou seja, o usuário está envolvido com o domínio da aplicação e tem sensação de presença dentro do mundo virtual.

A idéia de interação está ligada à capacidade do AV para detectar a entrada do usuário e modificar instantaneamente o mundo virtual. As pessoas são cativadas por uma boa simulação em que as cenas mudam em resposta aos seus comandos. Trata-se da característica mais marcante dos vídeo games.

Além disso, a idéia de envolvimento está ligada ao nível de motivação de um usuário para a realização de uma determinada atividade. O envolvimento pode ser passivo, como ler um livro ou assistir televisão, ou ativo, como participar de um jogo com algum

parceiro. A RVI tem potencial para os dois tipos de envolvimento, ao permitir a exploração de um ambiente virtual e propiciar a interação do usuário com o mundo virtual dinâmico.

No entanto, não é necessário que todas essas idéias coexistam simultaneamente, uma vez que nem todos os AVs são imersivos (Leston, 1996). A visualização de uma cena 3D em um monitor convencional, por exemplo, é considerada não-imersiva. Em AVs não-imersivos os usuários são parcialmente transportados para o mundo virtual, através de um projetor ou monitor, mas continuam sentindo predominantemente o mundo real.

Os sistemas imersivos são normalmente obtidos através do uso de:

- *Head-Mounted Display* (HMD) - dispositivo de reprodução de vídeo, usado na cabeça do usuário;
- CAVE - salas em que as paredes, teto e/ou piso servem como telas de projeção (CRUZ-NEIRA, 1995)

A percepção visual é o nosso sentido primário. Porém, para que seja possível atingir a imersão total do usuário é necessária a utilização da visão juntamente com outros sentidos, como audição e tato. O uso de sonorização estéreo ou multicanal possibilita a imersão através da audição e o uso de dispositivos *force feedback* ou táteis possibilita a imersão através do tato (BEGAULT, 1994).

Araújo (1996) mostra alguns artifícios para aumentar o realismo em ambientes de RV, como, por exemplo, a texturização dos objetos do ambiente e, assim como mostrou Begault (1994), a inserção de sons tanto ambientais quanto sons associados a objetos específicos.

Os sistemas de RV diferem entre si de acordo com os níveis de imersão e de interatividade proporcionado ao participante. Esses níveis são determinados pelos tipos de

dispositivos de entrada e saída de dados do sistema, além da velocidade e potência do computador que o hospeda. Eles podem ser classificados de acordo com o modelo de participação do usuário, ou seja, de acordo com o nível de interação que este sistema oferece.

Adams (1994) sugere que um aplicativo de RV pode proporcionar uma sessão sob três formas diferentes:

- Passiva: uma sessão de RV passiva não proporciona ao usuário a capacidade de interferir com a cena ou mundo virtual. A rota e os pontos de observação são explícitos e controlados exclusivamente pelo software. O usuário não tem controle algum, exceto, talvez, para sair da sessão;
- Exploratória: uma sessão de RV exploratória proporciona exploração do ambiente dirigida pelo usuário. O participante pode escolher a rota e os pontos de observação, mas não pode interagir de outra forma com entidades contidas na cena;
- Interativa: uma sessão de RV interativa proporciona exploração do ambiente dirigida pelo usuário e, além disso, as entidades virtuais do ambiente respondem e reagem às ações do participante. Ou seja, ela oferece todas as possibilidades que eram restritas nas participações passiva e exploratória. O usuário tem liberdade para escolha de rotas, pontos de vista e interação com os objetos do ambiente.

Quanto ao grau de imersão, os sistemas ou estilos de RV podem ser classificados como RV de Simulação, RV de Projeção, Projetores Visualmente Acoplados (Visually Coupled Displays), RV de Mesa, Realidade Aumentada, Virtualidade Aumentada e Hiper-Realidade (PIMENTEL; TEIXEIRA, 1994; MILGRAM et al., 1994; TIFFIN; TERASHIMA, 2001):

- Ambiente Virtual de Simulação: corresponde ao tipo mais antigo de RV, originado com os simuladores de vôo desenvolvidos pelos militares americanos após a Segunda Guerra Mundial (JACOBSON, 1994). Um sistema desse tipo basicamente imita o interior de um carro, avião ou jato, colocando o participante dentro de uma cabine com controles. Na cabine, telas de vídeo e monitores apresentam um mundo virtual que reage aos comandos do usuário;
- Ambiente Virtual de Projeção: também conhecido como Realidade Artificial. O usuário encontra-se fora do mundo virtual, mas pode se comunicar com personagens ou objetos nele contidos. O sistema VIDEOPLACE (KRUEGER; GIONFRIDDO; HINRICHSEN, 1985), capturava a imagem do usuário e projetava-a em uma grande tela que representava um mundo virtual, no qual os usuários podiam interagir uns com os outros ou com objetos. Krueger usou o termo Realidade Artificial para descrever o tipo de ambiente criado pelo seu sistema, que não exigia que o participante vestisse ou usasse dispositivos de entrada de dados (JACOBSON, 1994);
- Realidade Aumentada: é usualmente definida como a sobreposição de objetos virtuais tridimensionais, gerados por computador, no ambiente real, por meio

de algum dispositivo tecnológico (AZUMA, 1993; BOMAN, 1995). Onde, o ambiente é predominante real, porém enriquecido com objetos virtuais que sobrepõem o mesmo. Além de permitir que objetos virtuais possam ser introduzidos em ambientes reais, a Realidade Aumentada proporciona, também, ao usuário, o manuseio desses objetos com as próprias mãos, possibilitando uma interação atrativa e motivadora com o ambiente (BILLINGHURST; KATO; POUPYREV, 2001; ZHOU et al., 2004). No entanto, para que os objetos virtuais façam parte do ambiente real e sejam manuseados, é necessária a utilização de um software com capacidade de visão do AV e de posicionamento dos objetos virtuais, além de acionar dispositivos tecnológicos apropriados para Realidade Aumentada. Como exemplo, pode-se utilizar dispositivos visuais transparentes presos à cabeça (Heads-Up Display - HUD), nos quais os dados são projetados. Pelo fato desses displays ou visores serem transparentes, o usuário pode ver dados, diagramas, animações e gráficos 3D sem deixar de enxergar o mundo real;

- Virtualidade Aumentada: é a inserção de representações de elementos reais no mundo virtual, usando uma interface que permita ao usuário interagir com o AV (KIRNER, 2007). Além dos objetos estáticos, pode-se, também, levar objetos reais dinâmicos para o AV. Estes elementos reais podem ser pré-capturados ou capturados de forma on-line através de câmeras. Neste caso, o processo computacional faz a sua identificação e respectiva reconstrução no ambiente virtual. Milgram et al. (1994) acredita que a conceituação de

Realidade Aumentada e Virtualidade Aumentada é muito geral e só fica clara com sua inserção em um contexto mais amplo: o da Realidade Misturada;

- Realidade Misturada: combina o real com o virtual. A Realidade Aumentada, cujo ambiente predominante é o mundo real, pode ser considerada uma Realidade Misturada e a Virtualidade Aumentada, cujo ambiente predominante é o mundo virtual, também pode ser considerada como Realidade Misturada. Dessa forma, Milgram et al. (1994) descreve estes conceitos, em um contexto de Realidade/Virtualidade contínua, onde existe uma relação contínua entre o ambiente puramente virtual e o ambiente puramente real;
- Tiffin e Terashima (2001) descrevem uma outra classe de RV, chamada Hiper-Realidade, como um paradigma para o terceiro milênio. A Hiper-realidade é a capacidade tecnológica de combinar realidade virtual, realidade física, inteligência artificial e inteligência humana, integrando-as de forma natural para acesso do usuário. Na Hiper-Realidade o virtual é combinado com o real, incrementando-se novos elementos para potencializar a interação do usuário com os recursos necessários para realização de tarefas.

2.1.1 Sistemas de Multiprojeção Imersiva - CAVE

Desde o desenvolvimento dos primeiros capacetes de visualização (HMD's) no fim dos anos 80 até o início dos anos 90, vários dispositivos de exibição de imersão espacial, denominados SID's (*Spatially Immersive Displays*), foram desenvolvidos como uma proposta de interface para sistemas de RV.

No início de 1991, os pesquisadores Carolina Cruz-Neira, Thomas A. DeFanti e Daniel J. Sandin, do Laboratório de Visualização Eletrônica (*Electronic Visualization Laboratory*) da Universidade de Illinois em Chicago, projetaram um ambiente de multiprojeção estereoscópico voltado para aplicações imersivas de realidade virtual, baseado em grandes telas de projeção. Em 1992, a pesquisadora Carolina Cruz-Neira publicou o primeiro artigo sobre CAVE (CRUZ-NEIRA, 1992). Desde então, CAVETM é uma marca registrada.

Pelo projeto de Cruz-Neira, as imagens seriam projetadas em telas dispostas na forma de um cubo, para envolver o observador ou usuário do sistema e prover a sensação de imersão. Este sistema foi anunciado e demonstrado em uma conferência de computação gráfica e técnicas interativas, o SIGGRAPH'92, com o título: "The CAVE: Audio Visual Experience Automatic Virtual Environment" (CRUZ-NEIRA et al., 1992).

Cruz-Neira, Sandin e DeFanti (1993) afirmam que o nome "CAVE" é tanto um acrônimo recursivo (*CAVE Automatic Virtual Environment*), quanto uma referência ao "Mito da Caverna", narrado por Platão no livro VII de sua obra "A República", que retrata a exploração da percepção, ilusão e realidade através do uso de metáforas, como a projeção de sombras nas paredes da caverna.

Através de projetores posicionados atrás das telas que formam as paredes e acima do chão deste ambiente, são projetadas imagens estereoscópicas de alta resolução, que quando vistas através de óculos obturadores ou estereoscópicos (*shutter/stereoscopic glasses*), os usuários que encontram-se dentro do volume das telas obtêm uma sensação imediata e intensa de imersão em um ambiente virtual.

Além da projeção de imagens 3D, a CAVE também incorpora projeção acústica tridimensional, dispositivos de rastreamento de posição e de interação, permitindo que uma ou mais pessoas fiquem imersas no ambiente virtual e interajam com os objetos virtuais projetados neste ambiente.

Os sistemas de visualização do tipo CAVE apresentam ao usuário uma representação estereoscópica em escala natural de um ambiente tridimensional gerado por computador. Logo, o ambiente possui uma relação direta com as dimensões humanas que através de uma visão referenciada pelo *tracking* da cabeça e a interação realística com objetos virtuais, via dispositivos de interação, viabiliza a manipulação, operação e controle natural de mundos virtuais.

Por ser uma das mais completas tecnologias de visualização de dados (SHERMAN; CRAIG, 2002), a CAVE tornou-se uma importante ferramenta de pesquisa, cuja vasta gama de aplicações abrange as áreas de engenharia, arquitetura, medicina, arte, negócios e muitas outras. Algumas empresas de produção, por exemplo, utilizam os recursos de um dispositivo CAVE para exibirem seus produtos à clientes potenciais ou então corrigir eventuais problemas relacionados com seus produtos, antes que eles sejam submetidos ao processo de manufatura (CREAGH, 2003).

Uma grande vantagem da CAVE em relação a outros sistemas imersivos é a possibilidade de compartilhamento do mundo virtual por vários usuários simultaneamente. A

quantidade de pessoas que podem compartilhar do mesmo ambiente, está limitada diretamente às dimensões físicas da caverna utilizada.

A estrutura computacional envolvida no acionamento de CAVE's é bastante avançada e deve processar os pares estereoscópicos das imagens, além de gerenciar os dispositivos de interação, auditivos e projetores. A figura a seguir ilustra um sistema CAVE com o posicionamento dos projetores. Os espelhos ilustrados servem para ajustar a distância dos projetores com relação às telas.

Por ser um sistema de projeção cúbica, estes sistemas podem ser implementados a partir de diferentes configurações, que podem variar de 3 a 6 faces (ou paredes) de projeção.

A construção de um sistema de multiprojeção imersivo demanda uma série de conhecimentos interdisciplinares, como a implementação e controle do sistema de projeção, desenvolvimento de *softwares* de rede, controle de múltiplos equipamentos, renderização das imagens em tempo real, *softwares* de vídeo e áudio, projeção sincronizada, arquitetura e *design* etc. Além disso, por se tratar de uma tecnologia relativamente recente e de alto custo, muitos trabalhos envolvendo esta tecnologia ainda estão em fase de exploração e desenvolvimento.

Fonte: Symanzik (1996)

Figura 2-1- Representação de um dispositivo CAVE

2.1.1.1 Sonorização Para Sistemas de Multiprojeção Imersivos

DOEER (2007) mostra em seu trabalho que a habilidade para localização espacial do som, seja em ambientes reais ou virtuais tem sido material de pesquisa amplamente estudado. Por sua vez, FARIA (2005) alega que a percepção visual é amplamente aumentada pela percepção auditiva e vice-versa.

O sentido da visão proporciona a maior parte da informação do ambiente. Este sentido, no entanto, só permite percepção do espaço visual à sua frente, num ângulo de 180°. O sentido da audição, por sua vez, permite ao ser humano obter informação do ambiente sonoro envolvente em 360°, embora com uma resolução inferior. Assim, o nível de imersão

em um ambiente de RV é amplamente aumentado quando imagem e som são utilizados conjuntamente.

Existem diversos formatos de arquivos de som, com características diferentes entre si. Geralmente, quanto maior a qualidade do som armazenado, maior o tamanho do arquivo. Alguns dos formatos de arquivos de som utilizados neste trabalho são detalhados a seguir:

- WAV ou WAVE (*WAVEform audio format*) é um formato-padrão de arquivo de áudio da *Microsoft* e IBM para armazenamento de áudio em PCs. Apesar de um arquivo WAV poder conter áudio compactado, o formato mais comum de WAV contém áudio em formato de modulação de pulsos PCM (*Pulse-Code Modulation*). O PCM usa um método de armazenamento de áudio não-comprimido (sem perda). Mais detalhadamente, podemos dizer que os dados são transferidos a uma taxa de 1411,2 Kbps, ou seja, cada segundo de áudio ocupa 176,4 KB e cada minuto ocupa cerca de 10 MB. Uma música com 4 minutos ocupa cerca de 40 MB dentro de um CD ou HD. Usuários profissionais podem usar o formato WAV para qualidade máxima de áudio. O tamanho de arquivo WAV é considerado grande, com relação a outros tipos de arquivos que armazenam som, sendo cerca de dez vezes maior que um arquivo no formato MP3 padrão;
- MP3 (*MPEG Layer-3*) é o tipo mais popular de arquivo de som para armazenamento e Internet, devido à boa relação de qualidade e tamanho de arquivo. O usuário pode definir a taxa de compressão utilizada. Quanto maior

a compressão, menor a qualidade do som armazenado. O algoritmo usado pelo MP3 remove do som original, dados redundantes e irrelevantes, que normalmente não são percebidos pelo ouvido humano, com isso, os arquivos gerados possuem tamanhos menores. A taxa de compressão de um arquivo MP3 com 128 kbps é de 10:1. Taxas de compressão inferiores a 10:1 com 128 kbps reduzem a qualidade do som perceptivelmente à maioria dos ouvintes. Em contrapartida, arquivos de áudio no formato MP3, com taxa superior ou igual a 192 kbits, geralmente possuem qualidade muito semelhante ao seu formato original, não comprimido, ou seja, ouvintes leigos dificilmente reconhecem a diferença entre um arquivo de som no formato WAVE, sem compressão e um arquivo de som no formato MP3 com qualidade 192 kbps ou superior;

- WMA (*Windows Media Audio*) é uma tecnologia de compressão de dados de áudio, desenvolvida pela *Microsoft*. O nome WMA é usado para tipos de arquivos ou *codec* de áudio;
- WMA 9.1 (*Windows Media Audio Pro 6-Channel File*) é um tipo de arquivo especial para armazenamento de 6 canais de som em um mesmo arquivo. Esse tipo de arquivo pode ser utilizado para armazenamento de som tridimensional.

A síntese de som espacial em duas e três dimensões tem sido explorada por décadas e há uma variedade de técnicas e sistemas disponíveis, que atendem diversas aplicações e níveis de qualidade.

Os métodos de síntese de áudio espacial ou tridimensional (3D) podem ser classificados segundo o meio de reprodução final utilizado, sendo basicamente bi-aural ou multicanal.

- Bi-Aural: Os métodos bi-aurais de produção de campos sonoros 3D baseiam-se na geração de dois canais de som, um para cada ouvido, contendo todas as informações temporais e espaciais necessárias para que o sistema auditivo perceba ou possa reconstruir um espaço acústico tridimensional.
- Multicanal: As técnicas baseadas em reprodução multicanal se caracterizam pela reprodução em matrizes de alto-falantes, isto é, os sons tocados são mapeados para múltiplos canais. Estas técnicas são mais adequadas para emular ambientes acústicos reais para mais de uma pessoa ao mesmo tempo, sem a necessidade de utilização de fones de ouvido, projetando campos sonoros no ambiente tridimensional ao redor dos ouvintes, como ocorre na natureza (FARIA, 2005).

As matrizes de alto-falantes são controladas, geralmente, por um computador ou *Home Theater*. As matrizes mais comuns são compostas por seis canais de áudio, chamadas de *5.1 Surround Sound*. Essas matrizes possuem a seguinte configuração:

- Um alto-falante frontal esquerdo;
- Um alto-falante frontal direito;
- Um alto-falante frontal central;
- Um alto-falante traseiro esquerdo;

- Um alto-falante traseiro direito e
- Um *Subwoofer*.

O nome 5.1 se deve aos 5 canais de áudio mais 1 canal especial para o *Subwoofer*.

Um *Subwoofer* é um tipo de alto-falante usado para reproduzir frequências baixas (sons graves e sub-graves), geralmente abaixo de 45Hz.

Os sistemas *Dolby Digital* e *DTS*, muito utilizados em salas de cinema e *home theaters*, são exemplos de sistemas de áudio *5.1 Surround Sound*.

Existem dois tipos básicos de sonorização para ambientes virtuais (XIAOWEI, 2010):

- Reprodução através de disparo (*trigger*) – Variáveis associadas a eventos, dentro do ambiente virtual, disparam o som, já gravado e disponível no computador. O som é disparado junto ao evento e terminado logo após o final do evento. Alguns exemplos da utilização de som de disparo são: som de passos, conforme o usuário ou avatar se movimentam pelo ambiente virtual, ou o som de pressionamento de botões virtuais, ou ainda o som de abertura ou fechamento de portas virtuais etc.
- Som em *looping* – O som é repetido diversas vezes, sem interrupção durante a utilização do ambiente virtual. O som em *looping* pode ser utilizado como um som ambiente (*background*), disparado na inicialização do ambiente virtual, como por exemplo, o som de máquinas em uma fábrica, ou o som de crianças em um parque de diversões etc. O som em *looping* pode ser disparado, também, juntamente com algum evento e continuar sendo repetido até um

evento pré-definido, como, por exemplo, a ação de ligar ou desligar uma máquina. A maioria dos sistemas de sonorização para RV fornece ferramentas para manter um som em *looping*. Para a execução adequada de um som em *looping*, sem cortes de áudio no momento da repetição, esse som deve ser preparado corretamente em *softwares* de áudio especializados, antes de sua utilização.

2.1.2 Manufatura Virtual

A aplicação de ambientes virtuais (AV) no processo de manufatura está relacionada tanto com a manutenção e treinamento, como também a criação e simulação de protótipos virtuais antes de sua utilização no chão de fábrica (OLIVEIRA, 2008).

Manesh (2011) afirma que nos últimos anos a indústria manufatureira vem sofrendo intensamente o impacto da globalização, o que motivou um grande aumento na competição e levou as empresas a buscarem novas ferramentas para melhorar suas tarefas. Lee (2011) afirma que muitas formas de RV já foram exaustivamente aplicadas no aprimoramento da modelagem, planejamento, visualização e análise em sistemas de manufatura. O autor mostra ainda que em aplicações CAM, a tecnologia de RV foi utilizada para visualização dos resultados de simulações e monitoramento de processos de manufatura. Jin (2007) estende essas aplicações para planejamento e prototipagem do produto, visualização e planejamento do layout de fábrica, simulação e planejamento do processo de montagem, treinamento e assim por diante.

Wang, Shen e Lang (2004) listam trabalhos relacionados com manufatura virtual envolvendo a área automotiva; aeroespacial; indústria pesada e a indústria de ferramentas e máquinas, onde sistemas são projetados, entre outras funcionalidades, para monitoramento e controle de chão de fábrica e operação de máquinas CNC (*Computer Numerical Control*) remotas ou automação de fábrica avançada.

Seo (2006) publicou um trabalho sobre o monitoramento de diferentes tipos de máquinas de usinagem, como: fresadoras, tornos e furadeiras. Juntando-se a essas máquinas virtuais, esteiras, robôs industriais e operadores, tem-se um AV completo, capaz de representar fielmente um chão de fábrica.

Segundo Mousavi (2008), da mesma forma que o termo Realidade Virtual possui várias definições na literatura, o termo Manufatura Virtual (MV) possui diversas definições diferentes, dependentes do escopo de aplicação onde está inserido. O termo foi criado há mais de vinte anos atrás (CHAI, 2010) e possui várias definições na literatura. Zhang (2011) define MV como um mapeamento da usinagem prática em Ambiente Virtual (AV), tendo como resultado imediato uma produção totalmente digital, sem produção real e sem perda de material ou energia. Por sua vez, Souza (2006) afirma que a manufatura virtual representa uma abordagem emergente que as empresas podem utilizar para melhorar seus processos, introduzindo novos produtos no mercado, mais rapidamente e com melhor custo. Por fim, Marinov (2004) define Manufatura Virtual como um emergente conceito, para integração de diferentes áreas da manufatura, através do uso de tecnologia computacional, na criação e execução de modelos virtuais.

Conceitos de manufatura virtual são provenientes de operações de usinagem, mas logo começaram a evoluir na área de manufatura. Hoje é possível encontrar aplicações em diferentes campos, tais como fundição, forjamento e robótica. A idéia contida na grande

maioria das definições encontradas na literatura é "Manufatura Virtual é nada além de fabricação no computador".

De acordo com Banerjee e Zetu (2001), o termo Manufatura Virtual passou a ser utilizado em meado dos anos 90, em parte como resultado da iniciativa do Departamento de Defesa dos EUA. Até metade dessa década, os primeiros trabalhos nesse campo foram realizados por algumas organizações, principalmente as indústrias aeroespacial e automobilística.

Para Porto et al. (2002), a Manufatura Virtual pode ser usada em uma grande variedade de contextos de sistemas de manufatura, e pode ser definida como a modelagem desses sistemas e de componentes com o uso efetivo de computadores, de dispositivos audiovisuais e sensores para simular ou projetar alternativas para um ambiente de manufatura visando, principalmente, prever problemas potenciais e ineficiência na funcionalidade e manufaturabilidade do produto antes que a manufatura real ocorra. O autor define ainda três diferentes paradigmas, dentre os quais a Manufatura Virtual é definida: Manufatura Virtual orientada para o projeto, Manufatura Virtual orientada para a produção e Manufatura Virtual orientada para o controle.

Dentre os benefícios oferecidos pela virtualização de equipamentos de manufatura, Shridhar e Ravi (2002) citam a verificação e modificação rápida dos projetos; maior confiabilidade nas decisões; avaliação e testes de propriedades estruturais e funcionais de máquinas e produtos projetados; desenvolvimento de um produto ergonomicamente funcional e confiável, sem a necessidade de construir um modelo em escala real; projeção da estética do produto de acordo com as preferências do consumidor, além do desenvolvimento e avaliação de processos, assegurando a manufaturabilidade e a montabilidade, sem produzir de fato o produto em escala comercial. Porto et al. (2002) acrescentam a realização de

treinamento por intermédio de um ambiente virtual e distribuído para os operadores, técnicos e gerência no uso das instalações de manufatura, onde os custos de treinamento e produção podem, assim, ser reduzidos.

2.2 MONITORAMENTO DE DADOS REAIS EM AMBIENTES VIRTUAIS

Existem diversos trabalhos sobre o monitoramento *on-line* de operações via câmeras ou, mesmo, o monitoramento de operações com suporte de RV, porém, o segundo é mais escasso na literatura. No entanto, diferentemente das características principais da metodologia descrita nos próximos capítulos, não foram encontrados trabalhos específicos sobre a utilização conjunta de dados provenientes de câmeras e dados de máquinas-ferramenta, obtidos via protocolo MTConnect, dentro de um ambiente virtual. Essa escassez aumenta quando nos referimos especificamente a ambientes virtuais imersivos. Os próximos itens introduzem o protocolo MTConnect, bem como seus principais componentes.

2.2.1 Protocolo MTConnect

Atualmente, ainda existem muitas dificuldades para a integração de dados de dispositivos de manufatura com sistemas de informação. Diversas máquinas geram diferentes tipos e formatos de dados, dificultando ou até mesmo impossibilitando a integração adequada dos mesmos. A área de produção não possui ainda protocolos e interfaces padronizadas, como é comum, por exemplo, na área de informática, através das interfaces USB, Bluetooth etc.

O padrão oferecido pelo protocolo MTConnect provê um meio comum para comunicação entre dispositivos de chão de fábrica (VIJAYARAGHAVAN et al., 2008) e pode ser implantado visando a simplificação da captação e avaliação desses dados. Ferraz et al. (2011) lista alguns métodos de implantação de padrões em fábricas:

- Adoção incremental - não é utilizado como configuração de fábrica, mas é implementado através de adaptadores;
- Evolução - apenas funções básicas, evoluindo gradualmente sob rigorosos testes;
- Customização - O padrão pode ser estendido de acordo com a necessidade do usuário. *Softwares* e ferramentas podem ser desenvolvidas para máquinas específicas, sem interferir na compatibilidade do sistema como um todo;
- Não proprietário - Baseado em padrões abertos.

O protocolo MTConnect é baseado em XML (*eXtensible Markup Language*), uma linguagem de marcação genérica, que tem como principal propósito facilitar o compartilhamento de informações através da Internet. O XML utiliza método hierárquico para organização de dados, conforme mostra o exemplo a seguir:

Quadro 2-1- Exemplo de arquivo XML

```

<?xml version="1.0" encoding="UTF-8"?>
<curriculo>
  <InformacaoPessoal>
    <DataNascimento>23-07-68</DataNascimento>
    <NomeCompleto>...</NomeCompleto>
    <Contatos>
      <Morada>
        <Rua>R.Topazio</Rua>
        <Num>111</Num>
        <Cidade>Porto</Cidade>
        <Pais>Portugal</Pais>
      </Morada>
      <Telefone>9999-9999</Telefone>
    </Contatos>
    <CorreioEletronico>email@email.com</CorreioEletronico>
    <Nacionalidade>Portuguesa</Nacionalidade>
    <Sexo>M</Sexo>
  </InformacaoPessoal>
  <objetivo>Atuar na area de TI</objetivo>
  <Experiencia>
    <Cargo>Suporte técnico</Cargo>
    <Empregador>Empresa, Cidade - Estado</Empregador>
  </Experiencia>
  <Formacao>Superior Completo</Formacao>
</curriculo>

```

Além de utilizar o XML como base para o armazenamento e a transmissão de seus dados, o MTConnect foi desenvolvido em uma arquitetura modular, possibilitando a criação de ferramentas para a conexão de dispositivos. Dessa forma, quatro diferentes arquiteturas para a implementação do padrão podem ser utilizadas, conforme ilustra a figura seguinte.

Fonte: Ferraz (2011)

Figura 2-2- Arquiteturas para a implementação do padrão MTConnect.

No primeiro cenário, considerado ideal, o dispositivo é totalmente compatível com o padrão MTConnect, possuindo um Agente embarcado em seu controlador. Os módulos Agente e Adaptador MTConnect serão descritos nos próximos parágrafos deste tópico e explicados em detalhes ao longo deste trabalho. O segundo exemplo da figura mostra um dispositivo com um Adaptador embarcado. Nessa arquitetura é necessário o desenvolvimento de um Agente para coletar os dados do dispositivo e disponibilizá-los no padrão. Os cenários

da figura acima ilustram um dispositivo não compatível com o padrão. Nesse modelo é preciso implementar o Adaptador e o Agente, sendo que para cada modelo de CNC é necessário um Adaptador diferente, devido às diferentes APIs utilizadas por cada fabricante. O Adaptador pode ser implementado em uma aplicação independente (exemplo "c" da figura 2.2) ou na mesma do Agente (exemplo "d" da figura 2.2).

O Adaptador é o elemento responsável por controlar toda a comunicação com o dispositivo e com o Agente. Ele é um servidor TCP/IP e, enquanto não há Agente conectado, o Adaptador funciona em *Stand by*, sem se comunicar com o dispositivo. Quando um Agente é conectado, o Adaptador tenta se conectar ao dispositivo. Havendo erro na conexão o Adaptador considera que o dispositivo está desligado e envia essa informação ao Agente. Por outro lado, se a conexão é bem sucedida, o Adaptador inicia a coleta de dados, periodicamente, enviando os novos dados ao Agente, evitando sobrecarga na rede de comunicação.

O Agente é a aplicação que implementa a interface MTConnect, controlando o armazenamento dos dados coletados pelo Adaptador, além de implementar um servidor HTTP para manipulação das requisições de aplicativos clientes. Um Agente pode representar mais de um dispositivo dependendo da disponibilidade das máquinas na planta, facilitando a implantação da infraestrutura necessária. Todos os dados recebidos são salvos em um buffer de tamanho configurável. Um número sequencial é criado para identificar cada dado recebido. Dessa forma, é possível, através da interface HTTP, solicitar o histórico dos dados de determinados componentes ou do dispositivo inteiro. Se a capacidade do buffer for atingida os dados mais antigos são perdidos para poder guardar os dados novos. O usuário final também pode modificar o Agente (desde que não seja embarcado) para salvar os valores em um banco de dados, se necessário. Dessa forma todos os dados estarão disponíveis a qualquer momento.

2.2.1.1 Adaptador

Máquinas-ferramenta produzem diversos tipos de dados que podem ser úteis se adquiridos para análise e tomada de decisão. Porém, nem todas as máquinas-ferramenta possuem funcionalidades que permitam a exportação desses dados de maneira simplificada, principalmente as máquinas mais antigas.

O instituto MTConnect desenvolveu um protocolo de comunicação que visa padronizar os dados adquiridos de máquinas-ferramenta, bem como viabilizar sua aquisição para quaisquer máquinas. O processo inicia-se através de um módulo de *software* nomeado “adaptador”.

Diferentemente do módulo “agente”, que será descrito a seguir, a criação de um módulo adaptador MTConnect é algo particular para cada máquina-ferramenta e suas respectivas *APIs* (*Application Programming Interface* – Interface de Programação do Aplicativo). Trata-se de um programa de conversão de dados nativos de máquinas-ferramenta para dados equivalentes de MTConnect, que um determinado módulo agente possa compreender. O desenvolvimento de um adaptador está diretamente ligado, também, com a estrutura do *software* agente com o qual o adaptador se comunica.

Um módulo adaptador se comunica tanto com o dispositivo, quanto com o módulo agente. Essas comunicações podem ser configuradas de diversas formas. A figura a seguir mostra uma forma de relacionamento entre um adaptador MTConnect, uma máquina e um módulo agente, com base no documento MTConnect *Institute* (2012).

Figura 2-3- Relacionamento do adaptador MTConnect com um agente e um dispositivo

Geralmente o módulo adaptador é criado pelo próprio fabricante dos dispositivos, porém, devido à arquitetura aberta do protocolo MTConnect, qualquer programador pode criar seu próprio adaptador, desde que possua os conhecimentos necessários a respeito da máquina-ferramenta de seu interesse.

Apesar de ser específico para cada máquina-ferramenta, o módulo adaptador possui estrutura semelhante para quaisquer dispositivos. Há uma biblioteca de adaptadores abertos (*open-source*) disponível para *download* através do Instituto MTConnect, no repositório “github” (*MTConnect Institute github* – <http://github.com/MTConnect>). Esses adaptadores podem ser usados diretamente ou, mediante alterações no código fonte, podem ser preparados para o funcionamento em diferentes dispositivos. Porém, mesmo com a biblioteca de adaptadores em mãos, a complexidade de desenvolvimento de um adaptador depende em grande parte da documentação e *API* de cada máquina. Para dispositivos que forneçam dados originalmente com a terminologia MTConnect, não há necessidade de desenvolvimento de um tradutor, ou seja, de um módulo adaptador.

2.2.1.2 Agente

O agente é um *software* que se comunica com o adaptador e tem a responsabilidade de converter os dados para o formato MTConnect. Além disso, o agente atende requisições das aplicações clientes, enviando os arquivos XML adequados, com os dados requisitados. As partes conceituais envolvidas nos serviços do *software* agente são:

- *Header* (Cabeçalho): Informações relativas ao protocolo;
- *Components* (Componentes): Partes de um dispositivo;
- *DataItems* (Itens): Descrevem os dados disponíveis em um dispositivo;
- *Streams* (Conjunto): Conjunto de dados do tipo *Sample* (amostra) ou *Event* (evento) provenientes de componentes ou dispositivos;
- *Samples* (Amostras): Uma medida de um item em constante alteração, em um determinado tempo;
- *Events* (Eventos): Ocorrência discreta em um componente. Inclui mudanças de estado e alertas;
- *Alarms* (Alertas): Tipo de evento que indica um comportamento anormal.

A estrutura de inicialização genérica de um agente é mostrada na próxima figura, conforme o documento MTConnect Overview (2012) e será explicada em seguida.

Figura 2-4- Inicialização de um agente

1. Primeiramente o agente se conecta e se autentica com o serviço *Discovery* (*Name Service*). *Discovery* é um serviço que permite que uma aplicação encontre agentes para um determinado dispositivo. Porém, esse passo específico não foi usado neste trabalho. Utilizou-se conexão direta via endereço de rede;
2. Em seguida, um agente genérico registra seu URL para que possa ser localizado;
3. O agente se conecta ao dispositivo usando as *APIs* adequadas;
4. O dispositivo envia dados ao agente ou o agente solicita dados ao dispositivo.

A estrutura de inicialização descrita acima não levou em consideração a presença, não obrigatória, do módulo adaptador.

O diagrama a seguir, baseado no documento MTConnect *Overview* (2012), exibe um exemplo de chão de fábrica com três dispositivos, uma aplicação de controle e um *Discovery* (*Name Service*). Nesse exemplo há dois agentes MTConnect, sendo que um dos agentes está trabalhando com os tornos 1 e 2 ao mesmo tempo e o outro agente está em trabalho com a fresadora. A aplicação de controle está monitorando os três dispositivos.

Figura 2-5- Exemplo de chão de fábrica com agentes e dispositivos

Uma aplicação externa pode solicitar dados de qualquer um dos três dispositivos, usando o serviço *Discovery*. Para isso, a aplicação faria uma requisição para obter quais os dispositivos *on-line* e receberia como resultado: “Torno1, Torno2 e Fresadora”, que teriam, respectivamente, os endereços URL: “<http://10.1.10.32/Torno1>, <http://10.1.10.32/Torno2> e <http://10.1.10.33/Fresadora>”. Após a execução desse passo, a aplicação externa pode solicitar aos dispositivos quais tipos de dados eles estão aptos a fornecer.

O agente pode, também, ser implantado como um serviço incorporado, que não possua acesso à rede externa, não necessitando, assim, de um serviço *Discovery*. Um agente pode interconectar vários componentes de um único dispositivo. A figura a seguir mostra uma montagem desse tipo.

Figura 2-6- Montagem incorporada do agente

O dispositivo da figura acima possui um agente embutido, que se comunica apenas com o *Loader* e com o controlador. O coordenador é uma aplicação que recebe dados do agente e realiza algumas operações, baseado nessas informações.

Maiores detalhes das implementações e do funcionamento de agentes e adaptadores MTConnect podem ser encontrados no capítulo de materiais e métodos e nos anexos deste trabalho.

2.2.2 Outras pesquisas sobre monitoramento

Nos últimos anos, diversos trabalhos sobre monitoramento de dados foram publicados. Sakamaki (2009) utilizou o monitoramento remoto para análise de degradação das máquinas de manufatura. Wang (2009) e Yin (2008) criaram sistemas de monitoramento e manipulação para robôs industriais. Há, ainda, diversos trabalhos relacionados em outras áreas, como sistemas de monitoramento remoto para área médica, agricultura etc. (ROSATI, 2009) (DUVEILLER, 2010).

O trabalho mais antigo encontrado, durante as pesquisas realizadas nesta revisão bibliográfica, sobre o monitoramento de dados de manufatura em ambientes virtuais foi o de Miner e Stansfield (1994), que propuseram o treinamento e controle de robôs industriais através de AVs. Alguns anos depois, já no final da década de 90, os pesquisadores Javier et al. (1997) e Vermeulen et al. (1999) continuaram pesquisas sobre o monitoramento de operações reais através de ambientes virtuais. Vermeulen propôs a utilização de arquivos no formato VRML (*Virtual Reality Modelling Language*) para controle remoto de robôs industriais. Seu trabalho utilizou uma ferramenta proprietária, *RobComm Active X Controls*, para a aquisição

de dados de um robô específico. Por sua vez, Javier propôs o monitoramento de robôs de manufatura através de micro controladores específicos.

Já na década de 2000, mais especificamente em 2004, Navab et al. (2004) pesquisou a utilização de realidade virtual aumentada para monitoramento e manutenção de fábricas. Ainda no ano de 2004, Counsell começou a propor estudos sobre interação com objetos reais dentro de um espaço virtual, porém, o objetivo principal desses pesquisadores era contribuir com o início da realidade virtual aumentada e não a captação e padronização de dados reais em ambientes virtuais híbridos.

O trabalho de Skoogh et al. (2010) foi um dos poucos trabalhos encontrados que utilizou o protocolo MTConnect, visando a utilização de seus dados dentro de um ambiente de simulação de eventos discretos. A figura seguinte ilustra os principais pontos do trabalho de Skoogh.

Figura 2-7- Trabalho sobre MTConnect e sistema de eventos discretos.

Skoogh utilizou uma ferramenta desenvolvida pelo mesmo autor em trabalho anterior, chamada *GDM-Tool*, para analisar os dados coletados via MTConnect, a fim de proporcionar reutilização de dados em sistemas de eventos discretos. Porém, o trabalho de Skoogh difere em vários aspectos da metodologia descrita neste trabalho. Não foi desenvolvida uma forma de utilização dos dados reais *on-line* dentro de um mundo virtual. Além disso, o trabalho não abrange a utilização de dados contínuos e não mostra uma forma

de padronização de diferentes métodos de aquisição, como, por exemplo, informações provenientes de câmeras.

Os demais trabalhos a respeito de monitoração remota, estudados durante essa pesquisa, não utilizaram o protocolo MTConnect. Alguns fizeram uso de ambientes virtuais, como o trabalho de Qiu e Gao (2010) que utilizou um ambiente virtual para monitorar questões específicas de segurança de trânsito.

Ainda em 2010, Back et al. (2010) propôs a utilização de monitoramento via sistemas celulares e realidade mista, o que pode abrir, nos próximos anos, uma nova e vasta área de aplicações de monitoramento de dados via ambientes virtuais.

Figura 2-8- Monitoramento de fábrica via celular.

El-Chaar et al. (2011) construiu um complexo sistema de realidade virtual para manutenção e treinamento, porém não se utilizou da inserção de dados reais.

Figura 2-9- Ambiente virtual de treinamento de manutenção para indústria.

Abrate et al. (2011) desenvolveu um sistema para monitoramento seguro de um robô industrial através de um ambiente virtual. O pesquisador definiu áreas de risco em torno de objetos virtuais para minimização do atraso da comunicação entre objetos reais e virtuais. Dessa forma, os atrasos não permitiram que o robô real atingisse áreas proibidas. A figura a seguir mostra as camadas (*layers*) de controle de risco em torno de um objeto virtual.

Figura 2-10- Cubos com transparência mostrando as camadas de controle de risco.

Ainda em 2011, Torrisi desenvolveu um sistema de monitoramento de serviços para aplicações industriais, baseado em *CyberOPC* e MTConnect, bem como em XML e AJAX (*Asynchronous JavaScript and XML*). Torrisi utilizou a biblioteca FOCAS2 para obtenção de dados de um CNC Fanuc. A pesquisa descrita neste trabalho utilizou, também, a biblioteca FOCAS2 em conjunto com um adaptador MTConnect para obtenção e padronização de dados de um CNC Fanuc.

O trabalho de Ferraz et al. (2012) apresenta, também, a utilização do protocolo MTConnect para supervisão remota de processos de usinagem. O trabalho de Ferraz possui maior enfoque na produção de adaptadores e utilização de agentes MTConnect para a aquisição de dados de algumas máquinas de chão de fábrica, porém o módulo de observação e análise dos dados adquiridos, produzido por Ferraz, não faz uso da realidade virtual. Todos os trabalhos de Ferraz et al. (2012, 2011, 2007 e 2002) serviram como inspiração para o desenvolvimento desta metodologia e como base para a coleta de dados realizada nesta pesquisa.

Por fim, também no ano de 2012, Spišák et al. (2012) desenvolveu um sistema de monitoramento para tratamento térmico de materiais utilizando realidade virtual com VRML, e sensores específicos, como ilustra a figura seguinte.

Figura 2-11- Diferentes níveis de temperatura mostrados em um objeto virtual através de cores.

3 MATERIAIS E MÉTODOS

O desenvolvimento desta metodologia aconteceu concomitantemente com a criação dos materiais e métodos do Laboratório Multiusuário de Visualização 3D Imersiva (LMVI) da Escola de Engenharia de São Carlos (EESC) da Universidade de São Paulo (USP). Para o desenvolvimento desta metodologia foram criadas várias ferramentas e dispositivos de suporte a aplicações de realidade virtual imersiva. Nos primeiros anos do desenvolvimento foi montado, instalado e configurado um dispositivo do tipo CAVE com três telas de projeção e seis projetores. Foram testados diversos *softwares* e bibliotecas livres, para suporte à realidade virtual imersiva, até a escolha correta dos materiais e métodos mais adequados a este trabalho. Foram criadas, também, ferramentas de interação de baixo custo, específicas para utilização em dispositivos do tipo CAVE. Toda essa produção foi realizada em conjunto com outros pesquisadores, com trabalhos publicados na área de realidade virtual, como pode ser visto em Vale (2009), Vale (2010) e Oliveira (2009).

O Laboratório Multiusuário de Visualização Imersiva (LMVI), de São Carlos - um projeto financiado pela FAPESP, sob coordenação do professor Dr. Arthur José Vieira Porto – possui um dispositivo de realidade virtual do tipo CAVE, uma infraestrutura de visualização 3D imersiva, disponível aos pesquisadores São Carlos e região, para o desenvolvimento de pesquisas multidisciplinares.

Neste capítulo será apresentada uma descrição detalhada deste dispositivo, abordando desde sua estrutura física às bibliotecas utilizadas para implementação e controle do sistema.

3.1 ESPAÇO FÍSICO

O Laboratório Multiusuário de Visualização 3D Imersiva - LMVI - está localizado junto ao prédio do Núcleo de Ensino e Pesquisa em Automação e Simulação - NEPAS - do Departamento de Engenharia Mecânica da Escola de Engenharia de São Carlos da Universidade de São Paulo.

A estrutura física que abriga o LMVI é composta por um prédio construído exclusivamente para comportar o sistema de multiprojeção imersivo. Dentro deste prédio, foi montada uma sala com 10 metros de largura, 10 metros de comprimento e 5 metros de altura. As paredes da sala são compostas por divisórias em Eucatex e o teto por forro de PVC com luminárias fluorescentes. O chão é forrado com carpete para auxiliar na absorção do som 5.1, reduzindo o nível de reverberação no ambiente.

Como as telas de projeção oferecem melhores resultados em ambientes escuros, que maximizem seu brilho, tanto as paredes quanto o teto da sala que abriga o dispositivo foram pintadas de preto. Além disso, o carpete que forra o solo da sala tem a cor cinza escuro. Dessa forma, a sala do laboratório minimiza os "ruídos" de luz que podem atrapalhar a imersão do usuário na CAVE. O aspecto das paredes e chão da sala do LMVI pode ser visto na figura seguinte.

Figura 3-1- Estrutura da sala do LMVI com paredes escuras e carpete no chão para maximização da qualidade visual e acústica

3.2 ESTRUTURA DE PROJEÇÃO

Dentre as várias possibilidades de sistemas de multiprojeção imersivos, o NEPAS optou por montar um dispositivo do tipo CAVE, com três telas de projeção, costumeiramente chamado de CAVE de três faces ou CAVE de três lados. Nesse dispositivo, duas telas são perpendiculares ao chão e formam um ângulo de 90 graus entre si. A terceira tela corresponde à tela de projeção do chão.

As telas verticais são formadas por telas de projeção brancas flexíveis e posicionadas em forma de "L". A tela de projeção do chão é de madeira, para suportar o peso dos usuários, e também possui acabamento em branco. Para cada face existe um par de projetores responsáveis pela projeção das imagens estereoscópicas. A projeção das imagens nas paredes laterais é realizada pela parte de trás das telas, enquanto o chão recebe uma projeção pela parte superior, com auxílio de um espelho inclinado, para regular a distância entre o projetor e o solo.

3.2.1 Telas de Projeção

As telas de projeção frontal e lateral são flexíveis e sem emenda com dimensões: 3,2m x 2,4m para a tela frontal, 2,4m x 2,4m para a tela lateral. A tela de projeção do chão é rígida para que possa suportar o peso de um ou mais usuários do sistema e possui dimensões de 3,2m x 2,4m. A estrutura que sustenta as telas de projeção é fixa e feita em madeira de lei. A figura seguinte mostra o arranjo das telas de projeção do dispositivo.

Figura 3-2- Disposição das telas de projeção do LMVI

3.2.2 Projetores

O sistema é composto por seis projetores Christie modelo DS+60/DW30/Matrix 3000 de alta intensidade luminosa e alta resolução. Cada projetor é composto por um sistema de iluminação com duas lâmpadas de 300W que oferecem altos níveis de brilho e contraste. A resolução nativa de cada dispositivo é de 1400 x 1050 pixels.

Para a recepção dos dados enviados pelas saídas das placas de vídeo, as conexões entre os computadores e os projetores são estabelecidas através de cabos DVI (Digital Visual Interface).

A distância entre os projetores e as telas de projeção, bem como sua posição em relação às telas, devem ser ajustadas de forma a obter-se o melhor alinhamento possível das imagens divididas nas três telas que compõem o sistema.

Os projetores responsáveis pela tela do chão são mantidos em uma estrutura metálica fixada no topo da estrutura de madeira que sustenta as três telas, como pode ser visto no primeiro exemplo da próxima figura. Para ajustar a distância entre estes projetores e o solo, suas lentes são voltadas para um espelho fixado em um ângulo de aproximadamente 45 graus que reflete as imagens para a tela do chão.

Os projetores responsáveis pela tela frontal e tela lateral são mantidos em estantes metálicas e posicionados com suas lentes voltadas diretamente para as telas de projeção, como pode ser visto no segundo exemplo da figura a seguir.

Figura 3-3- Estruturas metálicas de sustentação dos projetores (a) superiores e (b) laterais

3.2.3 Óculos Estereoscópicos Passivos

Para cada tela de projeção, dois projetores com filtros polarizadores projetam um par estereoscópico de imagens, que ao olho nu o espectador consegue visualizar uma imagem com predominância da cor verde e a outra com predominância da cor roxa, sobrepostas e desfocadas, como pode ser visto no primeiro exemplo da figura seguinte. Somente através do uso de óculos estereoscópicos passivos com filtros polarizadores de mesma cor que as imagens projetadas, apresentado no segundo exemplo da figura seguinte, o espectador visualiza a imagem tridimensional.

Figura 3-4- (a) Duas imagens sobrepostas para utilização com óculos estereoscópicos e
(b) óculos estereoscópico

3.3 ESTRUTURA COMPUTACIONAL

O desenvolvimento e execução de aplicações para os sistema de multiprojeção depende de um conjunto amplo e sempre evolutivo de tecnologias, que inclui desde a infraestrutura de comunicação até a capacidade de geração e apresentação de imagens de maneira sincronizada pelos computadores que compõem o sistema.

A estrutura computacional do LMVI é composta por computadores principais responsáveis pela geração e apresentação das imagens para cada tela e computadores auxiliares de apoio ao desenvolvimento das aplicações.

3.3.1 Computadores Principais

Os computadores principais referem-se a um aglomerado (*cluster*) de três computadores responsáveis pela geração e apresentação das imagens em cada uma das três telas de projeção. Cada computador é equipado com um processador AMD Opteron 850 de 2.41 GHz, 4 GB de memória RAM e placa de vídeo Nvidia Quadro FX4500 com 512 MB de memória interna. Um dos computadores do cluster possui ainda uma placa de som Creative PCIe SB X-Fi Titanium, com saída de áudio digital 7.1.

3.3.2 Computadores Auxiliares

Os computadores auxiliares referem-se aos demais computadores do laboratório responsáveis pelo apoio na implementação de aplicações, modelagem de objetos 3D e interface com os computadores principais do sistema. Essas estações de trabalho possuem as mesmas configurações das máquinas do cluster principal e permitem que as aplicações sejam desenvolvidas e testadas localmente (no desktop) antes de serem enviadas ao cluster do sistema de multiprojeção.

O aglomerado de três computadores principais descrito anteriormente é o único responsável pela geração e apresentação das imagens no sistema de multiprojeção. Além de apoiar no desenvolvimento das aplicações, um dos computadores auxiliares é utilizado como interface/acesso aos computadores do aglomerado sem participação na projeção.

3.4 SISTEMA DE SOM 3D 5.1

Em sistemas de realidade virtual multiusuário, como os sistemas do tipo CAVE, é necessária a utilização de matrizes de alto-falantes, para que todos os usuários possam, ao mesmo tempo, ser imersos no ambiente virtual tanto visualmente, quanto com relação ao som tridimensional.

O LMVI possui um *Home Theater Samsung HT-C460* com 850W RMS que proporciona um sistema de som surround 5.1 composto por cinco auto falantes e um *Subwoofer*.

A conexão entre o PC e o home theater foi realizada por meio de cabo óptico, que liga a saída óptica da placa de som à entrada óptica digital do home theater, como pode ser visto na figura a seguir.

Figura 3-5- Disposição e conexões do sistema de áudio

Um sistema de sonorização como este possibilita a utilização de áudio padrão (estéreo ou mono) ou áudio 3D. Além disso, alguns algoritmos permitem ainda o uso de som surround 5.1 juntamente com rastreamento de movimento (*head tracking*), criando uma simulação sonora completa (SOARES, 2010).

Alguns cuidados com o ambiente real devem ser tomados para atingir a melhor qualidade de sonorização, como por exemplo, o uso de carpete no chão e/ou forro nas paredes, para a eliminação da reverberação inadequada. O excesso de reverberação pode confundir os usuários quanto ao posicionamento correto dos sons disparados.

3.4.1 Disposição dos Alto-Falantes

As telas de projeção do LMVI foram montadas entre duas salas, a sala de utilização e a sala de equipamentos. Os projetores, *Home Theater* e computadores controladores de cada uma das telas de projeção ficam dispostos na sala de equipamentos. A próxima figura mostra que ficam visíveis ao usuário, na sala de utilização, apenas os alto-falantes traseiros do *Home Theater*.

Figura 3-6- Visão frontal do sistema de sonorização do LMVI

Todos os auto-falantes do *Home Theater* são suspensos em suportes adequados, com exceção do *Subwoofer*, que se recomenda manter no chão, para melhor aproveitamento dos sons graves. Os auto-falantes frontais, situados na mesma sala dos projetores, são mantidos fora da área de projeção, posicionados próximos às junções das telas frontal e lateral. O auto-falante frontal central, que também se situa na sala de equipamentos, é posicionado abaixo da área de projeção.

3.5 DISPOSITIVOS DE ENTRADA DE DADOS

Como forma de interagir com o ambiente virtual projetado é necessário o uso de alguns dispositivos de entrada de dados, que possam ser lidos e interpretados como comandos do usuário para realizar alguma ação no ambiente 3D. Além dos dispositivos mais comuns como o mouse o teclado, o LMVI está equipado com outros dispositivos de interação que serão descritos nesta seção.

3.5.1 Controle de Vídeo Game Wii

O Controle *Wii*, também chamado de *Wiimote*, é o controle principal do vídeo game *Wii*, da Nintendo. O dispositivo possui diversas funcionalidades, como acelerômetro, botões direcionais, botões para funcionalidades genéricas e câmera de infravermelho. Dessa forma, através da utilização de bibliotecas especializadas, é possível implementar variadas formas de interação entre o usuário e o ambiente virtual.

O *Wiimote* pode ser conectado ao computador via *Bluetooth* e, associado a uma barra de LEDs infravermelhos, permite que seus movimentos sejam captados e transmitidos ao computador, funcionando como uma espécie de "mouse aéreo".

3.5.2 Câmeras e Canetas *Laser* para interação com o ambiente

Como forma de interagir de uma maneira eficiente, intuitiva e barata com o sistema de multiprojeção, o LMVI dispõe de uma *webcam* de baixo custo e três canetas *laser* (*Laser Pen*) que, através do uso de bibliotecas especializadas, possibilitam a navegação do usuário e a captação de diversos tipos de dados. A *webcam* e as canetas *laser* do LMVI podem ser vistas na figura seguinte.

Figura 3-7- Webcam e canetas laser com filtros do LMVI

3.5.3 Joystick

Apesar do uso da caneta *laser* proporcionar navegação de maneira intuitiva aos usuários, algumas aplicações, como, por exemplo, simuladores de voo, são mais bem aproveitadas com a utilização de *Joysticks*. O LMVI possui um *Joystick Clone Vibration* Águia com 12 botões de funções.

3.6 PLATAFORMA E BIBLIOTECAS DE PROGRAMAÇÃO

O desenvolvimento de aplicações gráficas tridimensionais para sistemas de multiprojeção envolve estudos e recursos ligados a *hardware*, *software*, percepção, interface do usuário, fatores humanos e aplicações. Geralmente, durante a implementação de um sistema desse tipo, o desenvolvedor trabalha com ferramentas com alto nível de abstração, para que este possa criar, manipular e visualizar objetos geométricos, deixando os detalhes de visualização de baixo nível para bibliotecas especializadas.

Existem algumas bibliotecas de programação para o desenvolvimento de aplicações tridimensionais imersivas, mas, até o momento, nenhuma oferece todos os recursos necessários para este domínio de aplicação. Dessa forma, o desenvolvedor necessita utilizar um conjunto de bibliotecas que, combinadas, oferecem soluções para o desenvolvimento de aplicações de Realidade Virtual Imersiva.

A plataforma de programação utilizada no LMVI é o *Visual C++ 2005 Express Edition*. Para obter o controle e execução das aplicações no sistema de multiprojeção sem se preocupar com a separação das imagens, sincronismos e gerenciamento de processamento das máquinas do cluster é utilizado o *framework VRJuggler 2.2.1*. Para a importação dos modelos 3D e a criação do grafo de cena é utilizada a biblioteca gráfica *OpenSG 1.6.0*. Para o desenvolvimento de soluções de interação por detecção de movimentos ou captura de dados de câmeras é utilizada a biblioteca *OpenCV 2.0*. Por fim, para manipulação de som 5.1 é utilizada a biblioteca *OpenAL 2.0*. Nesta seção são apresentados maiores detalhes sobre essas bibliotecas.

3.6.1 VR Juggler

O *VR Juggler* é um projeto de pesquisa do *Virtual Reality Applications Center*, da *Iowa State University*, que engloba uma coleção de tecnologias, as quais fornecem as ferramentas necessárias para o desenvolvimento de aplicações de RV. Constitui-se basicamente de uma plataforma para o desenvolvimento e execução de aplicações baseadas em RV, composta por um conjunto de bibliotecas de C++. Essas bibliotecas oferecem um conjunto de classes interligadas que facilitam a programação. A plataforma permite que a aplicação seja independente da arquitetura do *hardware* e do sistema operacional. Esta plataforma é quem gerencia esses aspectos e torna acessível a configuração do *hardware* dedicado, por meio da configuração visual de seus atributos sem que haja necessidade de modificar o código da aplicação. Isto disponibiliza um ambiente operacional unificado garantindo a portabilidade da aplicação entre diferentes recursos.

Toda ação que ocorre em um AV é controlada pelo *VR Juggler* mediante um dos seus componentes denominado *Kernel*. O *Kernel* atua como interface entre a aplicação e a plataforma, proporcionando abstração de *hardware*. A interface do *Kernel* disponibiliza para toda aplicação as funcionalidades acessíveis, exceto funções específicas de APIs gráficas.

Uma plataforma virtual simplifica o desenvolvimento de aplicações em sistemas de AVs complexos, proporcionando um ambiente de trabalho unificado. Além disso, garante a longevidade das aplicações e permite que desenvolvedores acompanhem os avanços tecnológicos sem investir tempo e recursos, modificando aplicações para suportar novas tecnologias.

3.6.2 OpenSG

As aplicações de AVs precisam ser preparadas para suportar cenas complexas tridimensionais, com centenas ou milhares de formas separadas. Faz-se necessária a utilização de métodos eficientes para o gerenciamento dessa grande quantidade de informação, de maneira efetiva, especialmente em aplicações com interatividade em tempo real.

OpenSG é um gerenciador de grafo de cena para desenvolvimento de aplicações gráficas de alto desempenho. Um grafo de cena é um arranjo de objetos 3D em uma estrutura de árvore que especifica o conteúdo de um universo virtual e indica como este será renderizado. Nele, é possível definir som, luz, orientação, geometria, localização e aparência visual dos objetos. Descrever uma cena usando um grafo de cena é uma tarefa mais simples do que construir a cena utilizando linhas de comando que especifiquem primitivas gráficas, tais como as do *OpenGL* (*Open Graphics Library*).

O grafo de cena permite ordenar dados em forma hierárquica onde "nós pai", itens com ramificações inferiores, afetem seus "nós filho", itens com ramificações superiores. Logo, ele permite estruturar os dados do AV em forma de uma árvore, onde o nó raiz agrupa todos os objetos do AV e os nós inferiores armazenam informações, agrupam objetos relacionados e armazenam a geometria dos objetos a serem representados no AV.

O *OpenSG* provê um *framework* orientado a objetos em que, em um nível de menor abstração, encontra-se a biblioteca *OpenGL*. Ou seja, o *OpenSG* é compatível com aplicações que suportam *OpenGL*. Sua função é livrar o desenvolvedor de implementar e otimizar chamadas gráficas de baixo nível, facilitando o desenvolvimento rápido de aplicações gráficas e poupando o desenvolvedor de programar rotinas complexas de

computação gráfica 3D e renderização. A figura seguinte apresenta um diagrama de classes que abrange a utilização do *OpenSG* em conjunto com o *framework VR Juggler*.

Figura 3-8- Diagrama de Classes do *OpenSG* combinado com o *VR Juggler*

3.6.3 OpenCV

A *Open Source Computer Vision - OpenCV* - é uma biblioteca de programação, com código aberto, desenvolvida inicialmente pela *Intel Corporation*. O *OpenCV* possui uma variedade de ferramentas de interpretação de imagens, desde operações simples, como um filtro de ruído, até operações complexas, como análise de movimentos, reconhecimento de padrões e reconstrução 3D.

A biblioteca *OpenCV* foi idealizada com o objetivo de tornar a visão computacional acessível a usuários e programadores em áreas como a interação humano-computador em tempo real e a robótica. A biblioteca está disponível tanto em código fonte, como em arquivos executáveis (binários), otimizados para os processadores *Intel*.

Um programa *OpenCV*, ao ser executado, invoca, automaticamente, uma DLL (*Dynamic Linked Library*), que detecta o tipo de processador e carrega a DLL apropriada.

A biblioteca IPL (*Image Processing Library*) é disponibilizada junto ao pacote *OpenCV*, pois o mesmo depende parcialmente da IPL. A biblioteca *OpenCV* está dividida em grupos de funções:

- Processamento de imagens;
- Análise estrutural;
- Análise de movimento e rastreamento de objetos;
- Reconhecimento de padrões;
- Calibração de câmera e
- Reconstrução 3D.

3.6.4 OpenAL

OpenAL (*Open Audio Library*) é uma interface de *software* para *hardwares* de áudio. Em geral, trata-se de uma maneira de gerar áudio para simulações tridimensionais. Essa interface consiste em diversas funções que afetam o CPU e o *hardware* de som,

permitindo que programadores criem saída multicanal para som tridimensional, em torno do usuário. Se o *hardware* de som consiste em apenas um buffer de saída, então a aplicação *OpenAL* deve ser totalmente direcionada ao CPU. O estado das informações *OpenAL* são explícitas ao usuário.

A biblioteca *OpenAL* foi criada no final dos anos 90, como um complemento para a biblioteca *OpenGL*, seguindo o mesmo estilo de codificação.

Inicialmente, a biblioteca havia sido criada apenas para a plataforma *Linux*, porém, a *Creative Labs* a padronizou, para que se tornasse uma biblioteca independente de plataforma.

O tópico seguinte apresenta os procedimentos e métodos envolvidos na utilização de todos os componentes desenvolvidos para o LMVI, para sonorização, iluminação, navegação, interação ou sincronização dos dados.

3.7 UTILIZAÇÃO DO SISTEMA DE RVI

Esta pesquisa foi iniciada anteriormente à montagem do dispositivo de RV imersiva do tipo CAVE do LMVI. Portanto, todas as tarefas e etapas envolvidas no processo de montagem e instalação foram acompanhadas desde o primeiro dia. Como este tópico não se trata de um objetivo exclusivo deste trabalho, as atividades serão descritas e comentadas de maneira concisa. Porém, mesmo assim, este tópico pode ser de grande valia para trabalhos futuros na área de RV imersiva, visto que não há trabalhos desse tipo na literatura e que o tempo despendido no aprendizado e implementação desses dispositivos é bastante grande.

Algumas tarefas são imprescindíveis para a montagem e instalação de um dispositivo de RV imersiva do tipo CAVE. A falta de precisão no alinhamento de projetores,

por exemplo, pode comprometer qualquer aplicação de RV imersiva. A escolha ou configuração incorreta das bibliotecas e *softwares* envolvidos pode, também, inviabilizar a utilização do equipamento ou diminuir sua eficiência.

Nos tópicos seguintes serão apresentadas as principais tarefas envolvidas nessa fase do trabalho.

3.7.1 Posicionamento, foco e alinhamento dos projetores

Cada projetor do LMVI possui diversos métodos disponíveis para teste e alinhamento das projeções, entre projetores de uma mesma tela. Como mostrado no capítulo anterior, cada projetor de uma mesma tela de projeção possui um filtro diferente, para que as imagens dos dois projetores sejam complementares, quando unidas por óculos estereoscópicos. Além disso, cada tela de projeção exibe imagens de dois projetores, ao mesmo tempo, desfocadas entre si, para que seja possível a emulação da sensação de profundidade.

Esse desfoque de imagens não é controlado manualmente durante a instalação do dispositivo. O desfoque é realizado pelo *software* que controlará a exibição e sincronismo das imagens. No momento da instalação, as imagens dos dois projetores de uma mesma tela precisam estar exatamente centradas uma na outra e focadas.

Para que seja possível alinhar as imagens com precisão, os dois projetores de uma mesma tela precisam ser posicionados próximos um do outro. No caso do LMVI, foram

construídos suportes especiais que seguram os projetores, um em cima do outro, à cerca de um metro do chão.

O alinhamento de projeções utilizado durante essa fase de instalação do dispositivo no LMVI foi baseado na aplicação de alinhamento fornecida com os próprios projetores. Essa aplicação consiste em quadricular as imagens sobrepostas dos projetores de uma mesma tela, para que seja possível saber, visualmente, se os dois projetores estão alinhados perfeitamente, em todas as regiões da tela.

Alinham-se as imagens movimentando-se cada projetor manualmente, descendo ou subindo as regulagens de seus pés e através das regulagens digitais fornecidas nos menus de controle do *software* interno de cada projetor. Geralmente, as regulagens finais são realizadas através do *software* do projetor e as regulagens iniciais, mais grosseiras, são realizadas manualmente.

3.7.1.1 Foco e Projeção com Auxílio de Espelho

A distância dos projetores com relação às telas de projeção é um fator importante para o foco das imagens projetadas. Há regulagens de foco nas lentes de cada projetor, porém, mesmo assim, estes precisam ser posicionados a uma distância adequada da tela, para que seja possível regular seu foco. No caso do LMVI, os projetores das paredes “Frontal” (*Front*) e “Lateral Direita” (*Right*) foram posicionados a cerca de três metros das telas de projeções respectivas.

Um dispositivo de visualização imersiva do tipo CAVE ocupa bastante espaço, devido ao tamanho das suas telas de projeção com cerca 3m² de altura e largura. Quando não há espaço suficiente para o correto posicionamento dos projetores, podem-se utilizar espelhos

para diminuir a necessidade de espaço. No caso do LMVI, como havia espaço suficiente na sala, não foi necessária a utilização de espelhos para as telas de projeção “Frontal” e Lateral Direita, porém, para as projeções da tela do chão, foi usado um espelho inclinado a cerca de 45 graus.

A regulagem dos projetores através do espelho é uma tarefa bem mais complicada que a regulagem dos demais projetores, pois o espelho tende a distorcer as imagens, esticando-as em determinadas regiões periféricas da tela.

3.7.2 Saídas de sinal DVI e RGB

Como mencionado anteriormente, há dois projetores para cada tela de projeção e, durante a execução de uma aplicação, suas imagens são sobrepostas nesta tela. Para o controle dos dois projetores de cada tela, há um computador, equipado com uma placa de vídeo especializada, que suporta duas saídas de sinal DVI (*Digital Visual Interface*) simultâneas - saídas de alta resolução.

Além das entradas DVI, os projetores possuem, também, entradas de sinal RGB (*Red, Green and Blue*). Como os cabos DVI possuem um custo bem maior que cabos RGB, primeiramente tentou-se utilizar cabos RGB para o dispositivo de RV imersiva do LMVI, porém, a baixa resolução e o alto nível de ruído, tornaram inviável sua utilização.

3.7.3 Montagem do Cluster de computadores

Existem, no total, quatro computadores que controlam todo o funcionamento do dispositivo:

- Um computador para controlar os dois projetores da tela frontal (*Front*);
- Um computador para controlar os dois projetores da tela lateral direita (*Right*);
- Um computador para controlar os dois projetores da tela do chão (*Floor*) e
- Um computador central, com acesso via rede aos outros três computadores, para disparo e controle remoto das aplicações.

O computador central não participa da renderização ou quaisquer tarefas durante a execução de um AV. Ele serve apenas para a realização de acesso remoto aos três computadores, referentes às três telas de projeção.

O acesso remoto é realizado através do *software UltraVNC* e o controle de sincronismo das projeções é realizado pela plataforma *VR Juggler*. Como o *VR Juggler* trabalha em *Cluster* com o modelo mestre-escravo, foi necessário selecionar um dos computadores para desempenhar a função de mestre e deixar os outros dois computadores como escravos. O computador selecionado para a função mestre foi a máquina nomeada como "*Front*", responsável pelas projeções da tela frontal.

Como o LMVI ainda não trabalha com a versão 3.0 do *VR Juggler*, que fornece uma interface para disparo das aplicações simultaneamente nas três máquinas, as aplicações precisam ser disparadas separadamente e sequencialmente em cada máquina.

Para que o *Cluster* dos computadores da CAVE funcione corretamente, para que haja sincronismo das telas de projeção e para que uma máquina saiba da existência de outras máquinas durante a execução de uma aplicação, os arquivos de configuração do *VR Juggler* necessitam ser preenchidos corretamente.

3.7.3.1 Sincronização das telas de projeção

As aplicações inicializadas sequencialmente nos três computadores do *Cluster* da CAVE necessitam conhecer exatamente a estrutura desse *Cluster* para que haja sincronismo e comunicação entre elas. Para isso, existem arquivos de configuração, com extensão "JCONF" (*Juggler Configuration*), que possuem funcionalidades disponíveis para suprir esta tarefa.

A configuração dos arquivos "JCONF" possui uma interface construída em JAVA, que visa facilitar o trabalho do programador. Porém, esses arquivos podem ser editados em seu modo texto, se forem abertos em um *software* comum para edição de texto.

Esses arquivos fornecem a configuração da aplicação, não somente com relação ao sincronismo de dados, mas também com relação aos dispositivos de entrada, tamanho das telas, posição do observador etc.

O item *Start Barrier Plug-in* garante que todas as máquinas comecem a execução da aplicação no mesmo *frame*¹. Isto é, o nó mestre do *Cluster* não permite que os demais nós executem seus *frame loopings* até que todos os nós identificados nos arquivos de configuração

¹ Um *frame* é um passo de uma sequencia de instruções programadas

estejam conectados e no estado "aguardando". Essa funcionalidade é crítica para aplicações com compartilhamento de dispositivos.

O sincronismo entre múltiplas telas é um dos grandes desafios em *Cluster* gráficos. Diferentes máquinas ou diferentes placas de vídeo podem demorar tempos diferentes para gerar uma mesma imagem. Equipamentos de *hardware* atuais utilizam uma técnica chamada *Double-Buffering*, que mostra o *frame* de um *Buffer* (*Front Buffer*), enquanto renderiza o próximo *frame* em outro *Buffer* (*Back Buffer*). Assim que o *Hardware* termina de renderizar o *frame* do *Back Buffer*, ele troca (*Swap*) o conteúdo dos dois *Buffers*. Nesse caso, a sincronização deve ser feita no momento em que as máquinas do *Cluster* trocam o conteúdo dos *Buffers*. Uma mensagem é enviada para todas as máquinas do *Cluster* informando se todas estão aptas a trocarem seus *Buffers*. Caso uma das máquinas não esteja pronta para a tarefa, uma mensagem de travamento é informada a todas as máquinas, para que todas possam prosseguir sincronizadas. Essa mensagem é conhecida como "*Swap-Locking*".

Para minimizar o tempo de latência desse sincronismo, o *Swap-Locking* deve ser feito pelo *hardware*, porém, quando isso não é possível, o *TCP Swap-lock Plug-in* é utilizado. Para a configuração adequada desse *plug-in*, os arquivos de configuração do *VR Juggler* devem informar o *Master Node* e o número da porta para comunicação do *Swap-Lock*.

3.7.4 Sonorização

Dispositivos do tipo CAVE geralmente funcionam junto à sonorização multicanal, para aumentar o nível de imersão do usuário. No entanto, a sincronização do mundo virtual gráfico com som espacial, não é uma tarefa simples.

Existem duas formas de utilização de sonorização multicanal dentro de um AV:

- A primeira delas consiste em criar-se um arquivo de som multicanal em um *software* de áudio especializado e, depois, dispará-lo dentro do ambiente virtual;
- A segunda consiste na utilização de bibliotecas apropriadas para o controle de intensidade, frequência e posição espacial do som, dentro do ambiente virtual, em tempo de execução.

Como o estudo principal desta pesquisa não é a sonorização de AV, a primeira opção de sonorização foi utilizada.

Através da primeira opção é possível criar diversos tipos de arquivos de áudio e dispará-los, posteriormente, dentro do AV. Porém, não é possível controlar dinamicamente a posição espacial do som, com relação ao posicionamento do usuário.

Como mostrado no capítulo anterior, o sistema de som utilizado neste trabalho foi um sistema de som multicanal 5.1, com um *Home Theater Samsung HT-C460* com 850W RMS, cinco auto falantes e um *Subwoofer*. A placa de som utilizada foi uma *Creative Sound Blaster X-fi Titanium 7.1*. Apenas o computador principal do *Cluster* necessita ter uma placa de som instalada, pois ele será o responsável pelos disparos dos arquivos de áudio. No caso do *Cluster* de computadores do LMVI, o computador principal é o computador responsável pela tela de projeção frontal (*Front Wall*).

Existem diversos *softwares* capazes de criar arquivos de som multicanal, porém nem todos os *softwares* reprodutores de som permitem a reprodução de arquivos de áudio multicanal.

O software usado para a criação de arquivos de sons com seis canais de áudio foi o *Adobe Cool Edit 2.1*. O formato de arquivo utilizado foi o *Windows PCM WAVE 16 bits 6-channel interleaved*, um arquivo com extensão WAV, com suporte a gravação de seis canais de áudio num mesmo arquivo.

A biblioteca de áudio *OpenAL*, compatível com o framework *VR Juggler*, foi utilizada para reprodução do arquivo multicanal, durante a execução dos ambientes virtuais.

Neste trabalho foi desenvolvido um sistema de disparo (*trigger*) para sons via teclado ou outro dispositivo de interação. Tanto o procedimento de leitura da entrada de dados, quanto o procedimento de disparo dos sons 5.1 foram colocados dentro de *Threads*, para que não houvesse interrupção ou atrasos da simulação, durante a utilização dos mesmos.

3.7.5 Modelagem Geométrica dos Objetos Tridimensionais

A modelagem geométrica de objetos tridimensionais consiste na utilização de *softwares* especializados para a construção de cada objeto 3D que constituirá o mundo virtual. Os objetos virtuais podem, ainda, ser criados diretamente no mundo virtual, através da utilização de bibliotecas gráficas, como, por exemplo, *OpenGL* e *DirectX*, dependendo da aplicação desenvolvida.

Como mostrado no capítulo 2, existem diversos *softwares* e tipos de extensões de arquivo que podem ser escolhidos para a modelagem geométrica de objetos 3D, cada qual com características próprias, vantagens e desvantagens. Neste trabalho, a geometria das máquinas virtuais que compõem a cena virtual, bem como de suas partes móveis, foi previamente modelada no *software 3D Studio Max*, para sua futura inserção no mundo virtual desenvolvido.

3.7.5.1 Extensão de arquivos 3D e VRML

O *software 3D Studio Max* permite que objetos sejam exportados para outros formatos, dentre eles o 3DS, seu antigo formato nativo, e o VRML, formato utilizado nas aplicações do LMVI.

O tempo das animações é baseado em *frames*, que são o número de imagens exibidas por segundo. Para utilização ideal das animações, geralmente usa-se taxa de 30 frames por segundo.

Após a modelagem e animação dos objetos, os dados de posição e orientação das animações são exportados para um arquivo VRML (com extensão WRL), juntamente com os objetos tridimensionais. Para que as animações criadas no *3D Studio Max* mantenham suas características quando usadas na LMVI, deve-se usar a opção “*Once per Animation frame*” no menu *Sample Rates*, durante a exportação do arquivo VRML, como pode ser visto na figura a seguir.

Figura 3-9- Opções de animação na exportação para VRML

As animações utilizadas no LMVI utilizam os dados de animação dos objetos importados de forma independente da leitura dos dados geométricos. Tentou-se usar um contador de frames dentro de uma *Thread*, porém a dificuldade de sincronização de *Threads* altera a velocidade de incremento da variável frame, inviabilizando sua utilização na animação. Dessa forma, optou-se por atualizar a variável frame em cada *looping* do programa principal.

3.7.6 O Grafo de Cena

De uma maneira simplificada, o grafo de cena nada mais é do que uma estrutura de dados em forma de árvore, onde as folhas dessa árvore (chamadas de nós do grafo) representam os objetos 3D, a textura e iluminação da cena virtual.

Portanto, após a importação dos modelos ou objetos 3D, é necessário inseri-los nesta árvore para posterior manipulação e renderização (apresentação dos objetos na tela) dos mesmos.

A criação do grafo de cena, ou árvore de objetos que farão parte do mundo virtual, tem início no procedimento *void OpenSG::initScene()*, onde são importados todos os arquivos VRML iniciais e onde é criada a raiz da árvore.

O procedimento *createRoot()*, mostrado na figura a seguir, cria o primeiro nó da árvore, onde serão adicionados todos os nós restantes.

Quadro 3-1- Procedimento para criação do nó raiz

```
NodePtr OpenSG::createRoot() {  
  
    //create Root Node  
    NodePtr root = Node::create();  
    OSG::setName(root,"ROOT");  
  
    //create Root Core (empty)  
    TransformPtr rootCore = Transform::create();  
  
    //add rootCore to root node  
    beginEditCP(root, Node::CoreFieldMask);  
        root->setCore(rootCore);  
    endEditCP(root, Node::CoreFieldMask);  
  
    return root;  
}
```

Para a criação de cada nó da árvore de objetos 3D, deve-se criar um nó adicional, chamado de “core”, responsável pelo gerenciamento do objeto. Um *core* muito utilizado é o *TransformCore*, responsável pela posição e orientação das geometrias associadas.

Em seguida, ainda durante a criação do grafo de cena, no procedimento *void OpenSG::initScene()*, o sistema verifica a existência do parâmetro “-vcamera” na linha de comando, para saber se o sistema animará a câmera, ou simplesmente adicionará os objetos 3D à cena, como pode ser visto no seguinte quadro.

Quadro 3-2- Adição de uma câmera virtual ao grafo de cena

```
if (vcamFile != "_BLANK_") {
    file = vcamFile.c_str();
    NodePtr wrlCamModel = SceneFileHandler::the().read(file);
    if (wrlCamModel == NullFC) {
        printf("\nLOAD FAILED!\n");
    }
    else {
        root = addNode(wrlCamModel, file, root);
        vcam = new VirtualCamera();
        vcam->nodeIndex=index;
        vcam->getData();
        index++;
    }
}
```

A função *NodePtr OpenSG::addNode(NodePtr wrlModel, const Char8* nodeName, NodePtr root* foi criada com a finalidade de inserir quaisquer novos nós na árvore de objetos, como pode ser visto no trecho de código da figura seguinte.

Quadro 3-3- Adição de um objeto 3D ao grafo de cena

```

NodePtr OpenSG::addNode(NodePtr wrlModel, const Char8* nodeName, NodePtr root){

    // create wrl transf node to be added to the root
    NodePtr wrlTransf = Node::create();
    OSG::setName(wrlTransf,nodeName);

    //create core for wrl transf node
    TransformPtr wrlCore = Transform::create();

    //add wrlTransf node to root
    beginEditCP(root, Node::ChildrenFieldMask);
        root->addChild(wrlTransf);
    endEditCP(root, Node::ChildrenFieldMask);

    //add wrlCore to wrlTransf
    beginEditCP(wrlTransf, Node::CoreFieldMask);
        wrlTransf->setCore(wrlCore);
    endEditCP(wrlTransf, Node::CoreFieldMask);

    //add wrlModel to wrlTransf
    beginEditCP(wrlTransf, Node::ChildrenFieldMask);
        wrlTransf->addChild(wrlModel);
    endEditCP(wrlTransf, Node::ChildrenFieldMask);

    return root;
}

```

Em seguida, logo após a criação do nó raiz, cada um dos objetos VRML definidos na linha de comando são adicionados à cena, como filhos do nó raiz e a um vetor de objetos 3D, como apresentado na figura seguinte.

Quadro 3-4- Adição de múltiplos objetos 3D ao grafo de cena

```

//Adds all the model files to the root
for (int i = 0; i < modelFiles.size(); i++) {
    file = modelFiles[i].c_str();
    NodePtr wrlModel = SceneFileHandler::the().read(file);
    if (wrlModel == NullFC){
        printf("\nLOAD FAILED!\n");
    }
    else{
        root = addNode(wrlModel, file, root);
        Model3D* temp = new Model3D();
        models.push_back(temp);
        models[i]->nodeIndex=index;
        models[i]->getData(i);
        index++;
    }
}

```

Finalmente a cena é iluminada, através do procedimento *void OpenSG::lightScene()*, como apresentado no trecho de código condensado do quadro a seguir.

Quadro 3-5- Adição do nó de iluminação ao grafo de cena

```

void OpenSG::lightScene(){
    mLightNode = Node::create();
    mLightBeacon = Node::create();
    PointLightPtr light_core = PointLight::create();
    TransformPtr light_beacon_core = Transform::create();
    ...
    // Setup light beacon
    Matrix light_pos;
    light_pos.setTransform(Vec3f(0.0f,0.0f,0.0f));
    ...
    // Setup light node
    beginEditCP(mLightNode, Node::CoreFieldMask);
        mLightNode->setCore(light_core);
    endEditCP(mLightNode, Node::CoreFieldMask);
    light_core->setSpecular(1.0,1.0,1.0,1);
    light_core->setPosition(25,2.5,-25);
    light_core->setAmbient(1.0,1.0,1.0,1);
    ...
}

```

3.7.7 Carregamento de objetos virtuais na cena

Há diversas maneiras de criar e exibir objetos tridimensionais nas telas de projeção de um sistema de multiprojeção. Usando-se a biblioteca *OpenGL* ou *DirectX*, pode-se criar primitivas 3D, como esferas, *torus*, cones, cubos etc., exibindo-as nas telas de projeção. Porém, o método mais comum de adição de objetos virtuais ao mundo virtual é a importação de objetos 3D.

Existem diversos formatos de arquivos de objetos 3D, porém a importação de objetos 3D especificamente para a árvore de objetos do *OpenSG* é geralmente feita através de arquivos VRML97 ou arquivos com extensão 3DS. Para o sistema de multiprojeção do LMVI optou-se pela utilização de arquivos VRML97, pois a disposição dos eixos exportados nesses arquivos é a mesma disposição encontrada no sistema *OpenSG*. A disposição dos eixos X, Y e Z dos arquivos exportados em 3DS é diferente da encontrada em *OpenSG*, o que obriga a execução de modificações no momento da importação de objetos.

O *OpenSG*, dentro do *VR Juggler*, é inicializado através dos procedimentos: *void OpenSG::initScene()* e *void OpenSG::init()*. Esses procedimentos são executados uma única vez durante a execução do sistema. Primeiro executa-se o *init* e, em seguida, o *initScene*.

A importação de objetos 3D, dentro do *OpenSG*, é feita, também, dentro do procedimento *void OpenSG::initScene()*, onde são inicializados, outros componentes da cena, como as câmeras importadas de VRML. Para a importação direta dos arquivos é utilizado o comando *SceneFileHandler*, que associa um arquivo VRML a um nó, que pode posteriormente ser adicionado ao grafo de cena, como pode ser visto no quadro seguinte.

Quadro 3-6- Importação de objeto 3D a partir de um arquivo VRML

```
NodePtr wrlCamModel = SceneFileHandler::the().read(file);  
NodePtr wrlModel = SceneFileHandler::the().read(file);
```

A animação da câmera também pode ser importada diretamente do arquivo VRML. Para informar ao sistema que ele deve animar a câmera principal da cena virtual, deve-se adicionar o parâmetro “-vcamera” à linha de comando, como o exemplo apresentado no quadro seguinte.

Quadro 3-7- Linha de comando com o parâmetro “vcamera”

```
VirtualWorld.exe -vcamera rollercoaster.wrl configuracao.jconf
```

3.7.8 Busca de Objetos no Grafo

Para transladar, rotacionar, editar, inserir ou remover nós de um grafo de cena foi criado uma rotina de buscas de objetos no grafo. Utilizando as funções nativas da biblioteca *OpenSG* foi possível desenvolver uma função que busca qualquer nó na árvore que seja filho de um nó utilizado como argumento da função (pode ser o nó raiz) e que possua um determinado nome como identificador único, como pode ser visto no código apresentado no quadro a seguir.

Quadro 3-8- Implementação da função de busca de um nó no grafo de cena

```

NodePtr OpenSG::findNode(NodePtr node, std::string nodeName) {

    UInt32 children = node->getNChildren();

    if (getName(node)) {
        if (getName(node) == nodeName) {
            return node;
        }
    }

    for (int i=0; i < children; i++) {
        NodePtr child = findNode(node->getChild(i), nodeName);
        if (child != NullFC) {
            return child;
        }
    }

    return NullFC;
}

```

3.7.9 Método para Animação, Translação e Rotação de Objetos Virtuais Sob Controle das Plataformas *OpenSG* e *VR Juggler*

A rotação ou translação dos objetos deve ser feita conjuntamente, através da multiplicação de uma matriz 4x4 contendo os dados da translação e outra matriz 4x4 contendo os dados da rotação. A ordem da multiplicação é importante para o resultado. Para a rotação de objetos deve-se usar a função “*mult*”, com a seguinte ordem de execução: “*translacao.mult(rotacao)*”. O resultado final é uma matriz 4x4 armazenada na matriz

“translação”. Para a rotação e translação das câmeras, deve-se fazer o oposto: “rotacao.mult(translacao)”, com o resultado armazenado na matriz “rotação”.

Um exemplo de multiplicação de matrizes de translação e rotação pode ser encontrado em um trecho de código exibido no quadro a seguir, criado para a movimentação dos objetos 3D que representam a Terra e a Lua.

Quadro 3-9- Trecho de código para rotação e translação

```
Real32 time = glutGet(GLUT_ELAPSED_TIME);

planetRot = Quaternion(Vec3f(0,1,0), time/float(velPlanet));
moonRot = Quaternion(Vec3f(0,1,0), time/float(-140));

rt = Quaternion(Vec3f(0,1,0), time/float(125));
rl = Quaternion(Vec3f(0,1,0), time/float(100));

rtt.setTransform(rt);
t.setTransform(Vec3f(20,0,0));
r.setTransform(planetRot);

t.mult(rtt);
r.mult(t);

pl.setValue(r);
rll.setTransform(rl);
t.setTransform(Vec3f(6,0,0));
r.setTransform(moonRot);

t.mult(rll);
r.mult(t);
```

3.7.10 Navegação e interação no ambiente virtual de manufatura desenvolvido

Neste tópico serão descritos os dispositivos e métodos para navegação e interação do usuário com o ambiente virtual.

3.7.10.1 Tentativas de navegação via *bluetooth* e *Leds*

Tentou-se utilizar, primeiramente, dispositivos *bluetooth* para viabilizar a navegação e interação do usuário na cena. Um dos dispositivos utilizados na pesquisa foi o controle do Nintendo Wii. Porém, foi constatado que o alcance de conexão *bluetooth* não atende às necessidades de um ambiente virtual imersivo.

Depois de diversos testes com outros dispositivos, dependentes de conexão via “*bluetooth*”, e de uma vasta busca, sem sucesso, por dispositivos “*bluetooth*” com maior alcance, optou-se por abandonar a utilização de tal tecnologia e testar a utilização de captura de luz infravermelha de vídeo para interação. Para isso, foram adquiridas *webcams* de baixo custo e dois *LEDs* (*Light-Emitting Diode*) infravermelhos de alta intensidade. Os *LEDs* foram fixados em uma caixa de plástico para fins de testes antes de sua utilização com o usuário, como pode ser visto em destaque na figura seguinte.

Figura 3-10- Dispositivo com LEDs infravermelhos

Uma *webcam* foi desmontada e um filtro de luz foi anexado à sua lente. O filtro de luz foi composto por folhas de filme de fotos, com o intuito de permitir apenas a passagem da luz infravermelha, bloqueando a luz natural. Logo se percebeu que a captura da luz infravermelha não era suficientemente boa para permitir a detecção correta dos movimentos dos *LEDs*.

Em seguida, como uma ideia derivada do uso de *LEDs* infravermelhos e *webcams*, optou-se por testar a captura de luz proveniente de canetas laser e lanternas de baixo custo.

3.7.10.2 Uso de canetas *laser*, câmeras e biblioteca *OpenCV* para navegação

Uma das formas de interação com o sistema do LMVI faz-se através da detecção de movimentos do usuário através do uso de uma ou mais canetas laser, dentro do campo de visão de uma *webcam*. Através do deslocamento do ponto luminoso do laser é possível movimentar objetos 3D no ambiente virtual de uma maneira simples e intuitiva.

No entanto, o maior desafio dessa solução consiste na identificação correta do ponto luminoso da caneta. As imagens de onde queremos extrair alguma informação em alguns casos precisam ser convertidas para um determinado formato ou tamanho e precisam ainda ser filtradas para remover ruídos provenientes do processo de aquisição da imagem.

Os ruídos podem aparecer de diversas fontes, como por exemplo, o tipo de sensor utilizado, a iluminação do ambiente, as condições climáticas no momento da aquisição da imagem, a posição relativa entre o objeto de interesse e a câmera. Note que ruído não é apenas interferência no sinal de captura da imagem, mas também interferências que possam atrapalhar a interpretação ou o reconhecimento de objetos na imagem.

A ideia deste tipo de interação é simples: a *webcam* capta as imagens do usuário movimentando a caneta para manipular os objetos 3D, utiliza funções de processamento de imagem para filtrar o ponto luminoso da caneta e calcula o deslocamento deste ponto no campo de visão da webcam. Com base no deslocamento da caneta no campo de visão da *webcam*, o desenvolvedor seleciona comandos de interação do usuário com o ambiente tridimensional.

A *webcam* foi posicionada no ponto de junção das três telas, com a lente voltada para os olhos do usuário. Dessa forma, foi possível obter um amplo campo de visão onde o usuário pode movimentar a caneta laser para interagir com o ambiente.

Foram adquiridas algumas canetas *laser* e lanternas e filtros de luz foram anexados às mesmas, para corrigir o ruído da luz captada pela câmera. Esses novos filtros eram formados por um material plástico branco translúcido, que é utilizado para embalar eletrônicos, como pode ser visto na figura seguinte.

Figura 3-11- Material utilizado para uniformizar e concentrar a luz do laser

Para o processamento da captura de imagens dessas canetas *laser* e lanternas foram testadas diversas bibliotecas especiais para processamento de imagens e vídeo. A biblioteca de C++ chamada *OpenCV* foi considerada a mais adequada para a tarefa.

3.7.10.3 Configuração e Posicionamento da *webcam*

Para que a *webcam* captasse corretamente apenas a luz transmitida pelo dispositivo e não toda a luz ambiente, foi necessário alterar a configuração da *webcam* da seguinte forma:

- Brilho = 0;
- Contraste = MAXIMO ;
- Saturacao = MAXIMO;
- Proporção de branco = MAXIMO;
- Comp. luz de fundo = 0;

- Exposicao à luz = desmarcada a opção “automático” e configurado o valor MÍNIMO.

Todas as alterações foram feitas para que a *webcam* exibisse uma imagem mais escura possível, captando somente a luz do laser ou lanterna. Apesar da biblioteca *OpenCV* permitir o uso de mais de uma câmera, foi utilizada somente uma *webcam* para uso do menor processamento possível.

Figura 3-12- Posicionamento da *webcam* no sistema de multiprojeção

Com o intuito de captar todos os movimentos do usuário com apenas uma *webcam*, diversas maneiras de posicionamento foram testadas. A melhor maneira encontrada foi o posicionamento do dispositivo sobre a tela do chão, exatamente na junção das três telas de projeção, apontada para os olhos do usuário, em um ângulo próximo a 60 graus, como pode ser visto na figura acima. As demais posições atrapalhavam a detecção do laser ou lanterna, por captar a luz dos projetores superiores que refletem no chão.

3.7.10.4 Pré-processamento da Imagem para Filtragem do Laser

Para que apenas o ponto luminoso do laser ou lanterna fosse identificado num primeiro momento, utilizando a biblioteca *OpenCV*, a imagem foi convertida de colorida para escala de cinza. Em seguida, um *threshold* (limiar) foi aplicado nessa imagem, gerando uma nova imagem, com valores preto ou branco (luz = branco, preto = restante). A figura seguinte apresenta uma sequencia com três imagens que ilustram o processam aqui descrito.

Figura 3-13- Sequencia de imagens resultantes do pré-processamento do ponto luminoso

Como pode ser visto na última imagem da figura, foi possível obter-se uma imagem sem ruídos, com fundo preto e somente o ponto de luz em destaque. O trecho de código responsável por este pré-processamento pode ser visto no quadro a seguir.

Quadro 3-10- Trecho de código condensado para o pré-processamento da imagem

```

void MoveObject::process(IplImage **_img)
{
    int threshold = 10, maxValue = 255;
    int thresholdType = CV_THRESH_BINARY;
    IplImage *img = *_img;
    CvSize size = cvGetSize(img);
    IplImage* gray = cvCreateImage(size, IPL_DEPTH_8U, 1);
    IplImage* grayThresh = cvCreateImage(size, IPL_DEPTH_8U, 1);

    // CV_RGB2GRAY: convert RGB image to grayscale

    cvCvtColor( img, gray, CV_RGB2GRAY );
    cvFlip(gray, NULL, 0); //vertical
    cvCopy(gray, grayThresh, NULL);
    cvThreshold(gray, grayThresh, threshold, maxValue, thresholdType);
}

```

3.7.10.5 Detecção do Ponto de Luz Na Imagem Pré-processada

Para a detecção do ponto de luz, varreu-se a imagem inteira (640x480 pixels), verificando pixel a pixel, por um valor maior que zero (que não fosse preto). Ao encontrá-lo, utilizou-se a função *cvfloodfill* para verificação da existência de outros pixels, de mesma cor, conectados ao pixel em questão. Caso existissem, seria possível descobrir a altura, largura, área, x e y do componente encontrado, através do parâmetro *comp*, da função *cvfloodfill* do *OpenCV*.

Caso exista mais de um componente na imagem, o componente atual é comparado com o componente anterior e armazenado apenas o maior. Através dos dados do componente foi possível obter sua posição (centro do objeto) e calcular seu raio (tamanho do objeto). Os

trechos de código condensado que implementam a detecção e obtenção da posição e tamanho do ponto de luz na imagem pode ser visto nos quadros seguintes.

Quadro 3-11- Detecção do ponto de luz na imagem

```
void MoveObject::detect(IplImage **_img)
{
    //scan the image / find the connected components / store the 2 biggest comp

    for (int i = 0; i < width*height; i++) {
        pixval = cvGet2D(grayThresh,i/width,i%width);
        if (pixval.val[0] > 240) {
            cvFloodFill(grayThresh,cvPoint(i%width, i/width),
                cvScalar (255), cvScalar(0), cvScalar(0), &comp,flags,mask);
            if (comp.area > 5) {
                if (comp.area != comp1.area) {
                    comp2.rect.x = comp1.rect.x;
                    comp2.rect.y = comp1.rect.y;
                    comp2.rect.width = comp1.rect.width;
                    comp2.rect.height = comp1.rect.height;
                    comp2.area = comp1.area;
                    comp1.rect.x = comp.rect.x;
                    comp1.rect.y = comp.rect.y;
                    comp1.rect.width = comp.rect.width;
                    comp1.rect.height = comp.rect.height;
                    comp1.area = comp.area;
                } //end if
            } //end if
        } //end if
    } //end for
}
```

Quadro 3-12- Obtenção da posição e tamanho do ponto de luz

```

void MoveObject::getCenter(comp1)
{
    //If flashlight is on get its center
    if (comp1.area > 0) {
        //get center
        x = comp1.rect.x + comp1.rect.width/2;
        y = comp1.rect.y + comp1.rect.height/2;
        //get radius
        if (comp1.rect.width > comp1.rect.height) {
            r = comp1.rect.width/2;
        } else {
            r = comp1.rect.height/2;
        }
    }
}

```

3.7.10.6 Mapeamento da Movimentação do Ponto de Luz no Plano

Com relação à movimentação, armazenou-se a posição X e Y do centro do componente e calculou-se a diferença da próxima posição X, Y, a cada frame. Em seguida, essa diferença foi multiplicada por uma variável chamada "velocidade" e o resultado foi utilizado como incremento da posição X, Y do objeto, no mundo 3D. A variável "velocidade" (*moveSpeed*) foi utilizada para aumentar ou diminuir a velocidade de movimentação dos objetos 3D. Essa variável é alterada pelas teclas "+" e "-" do teclado do operador. Um trecho condensado do código que implementa o mapeamento da movimentação do *laser* pode ser visto na quadro seguinte.

Quadro 3-13- Mapeamento da movimentação do laser no plano

```

void MoveObject::detect(IplImage **_img)
{
    if (comp2.area < 6) { //just one light is on
        int temp = (frameLightON - lastFrameLightON);
        if ((temp < 5) && (temp > 0)) {
            printf("piscou \n\n");
            laserClicked = true;
        }
        this->lastFrameLightON = this->frameLightON;
        if ((lastX==0)&&(lastY==0)) {
            lastX = x;
            lastY = y;
        }
        float deltax = x - lastX;
        float deltax = y - lastY;
        float tempx = deltax*moveSpeed, tempy = deltax*moveSpeed;
        application->distX += tempx/100;
        application->distY += (tempy/100)*-1;
        lastX = x;
        lastY = y;
    }
}

```

3.7.10.7 Criação da *Thread* de Detecção do *Laser*

Todo processamento da biblioteca *OpenCV* é realizado através de uma *Thread* do *VR Juggler*. Essa *Thread* é chamada dentro da biblioteca *OpenSG* para captura ininterrupta dos dados da câmera e detecção do movimento do *laser*. Um pequeno trecho de código, que demonstra como esta *Thread* é criada e destruída, pode ser visto no quadro seguinte.

Quadro 3-14- Criação e destruição da *Thread* de detecção de movimento do *laser*

```

void MoveObject::startThread() {
    opencvThread = new vpr::Thread(boost::bind(&MoveObject::start_cam,this));
}

void MoveObject::stopThread() {
    opencvThread->kill();
    printf("OpenCV stopped and Thread killed\n");
}

```

Ao final do processamento e cálculo dos valores necessários para a movimentação, descobriu-se que somente o computador com acesso a *webcam* atualizava as variáveis a cada *frame*. Portanto, a movimentação dos objetos 3D só podia ser visualizada na tela de projeção do computador *Front*, onde a *webcam* estava conectada.

As demais máquinas do cluster, que processam o *OpenSG* independentemente umas das outras, não eram capazes de identificar a entrada de dados da *webcam*, pois esse dispositivo não pode ser configurado diretamente nos arquivos de configuração do *VR Juggler*, onde são configurados, por exemplo, o teclado e outros dispositivos de entrada.

3.7.10.8 Comunicação dos Dados de Movimentação

Qualquer dispositivo conectado a um dos computadores do cluster, que não esteja configurado no arquivo “JCONF”, não pode ser compartilhado entre as máquinas, através do próprio *VR Juggler*. Em outras palavras, o *VR Juggler* não controla automaticamente a transmissão de dados da *webcam* para todas as máquinas do *cluster*. Logo, foi necessária uma

solução alternativa. A solução foi embasada em uma comunicação de rede por *Socket* (TCP/IP), onde o computador contendo a *webcam* foi designado como servidor e, os demais, como clientes. A cada *frame*, os valores obtidos pelo *Server*, com relação à imagem da *webcam*, são enviados para os clientes, para que os mesmos atualizem suas respectivas variáveis e possam movimentar os objetos 3D em suas telas de projeção, ao mesmo tempo que a tela da frente. Uma amostra do código de envio condensado pode ser visto no quadro a seguir.

Quadro 3-15- Função de envio de dados de movimentação para os clientes

```
void Server::sendData(){
    //DATA BUFFER
    printf(sendbuf, "%iF%4.2fX%4.2fY%4.2fZ%4.2fA%4.2fB%sH", flag, distX, distY, distZ);

    //SEND DATA TO CLIENT
    bytesSent = send(acceptSocket, sendbuf, len, 0);
    if (bytesSent == SOCKET_ERROR){
        printf("Server: send() error %ld.\n", WSAGetLastError());
    } else {
        printf("Server: send() to client OK.\n");
        printf("Server: Bytes Sent: %ld.\n", bytesSent);
        printf("Server: Data sent is: \"%s\"\n", sendbuf);
    }
}
```

3.7.10.9 Implementação da Função *Zoom* com o *Laser*

A movimentação com utilização de apenas uma caneta *laser* ou lanterna permite apenas a movimentação dos objetos em seus eixos X e Y. Para a implementação do *zoom* (eixo Z), seria necessário a utilização de outro ponto de luz, ou então, a utilização de menus,

ativados em alguma região da tela, para que o usuário pudesse alternar entre a movimentação X, Y e o *Zoom* (Z).

Optou-se pela utilização de um segundo *laser*, que seria responsável somente pela movimentação no plano Z, enquanto o primeiro continuaria com a movimentação no plano X, Y. Para que os lasers pudessem ser diferenciados, foram utilizados laser e lanternas com cores distintas. A partir da cor, escolhe-se o tipo de movimentação que o mesmo irá desempenhar.

Um grave problema foi encontrado. Dependendo da posição/ângulo do laser/lanterna em relação à *webcam*, pode haver alteração da cor do ponto luminoso na imagem colorida. Logo, torna-se difícil trabalhar com a detecção do laser por cor, em uma imagem que sofre alterações de cor o tempo todo. Para resolver esse problema, a cor do *laser* foi identificada dentro de certa tolerância e, em seguida, a imagem era convertida em escala de cinza, e depois, em preto e branco, eliminando ruídos de pixels fora de um limiar, como descrito anteriormente.

A movimentação em Z (*Zoom*) era obtida de acordo com a posição do *laser*: para cima de um determinado ponto da imagem, deslocava-se o objeto para frente e para baixo de um determinado ponto da imagem, deslocava-se o objeto para trás. A movimentação do objeto teve que ser feita utilizando somente um dos *lasers* por vez. Caso os dois estivessem ligados, somente um deles seria levado em consideração.

Com o tempo, percebeu-se que a identificação dos *lasers*/lanternas por cor não era suficientemente eficiente. Optou-se por trabalhar com dois *laser* ou lanternas, porém, ambos em escala de cinza. Obviamente, o sistema não conseguia mais detectar quando um ou outro *laser* estava aceso, portanto foi necessário mudar o sistema de *Zoom*.

Ao invés de trabalhar com um *laser* para *Zoom* e com outro laser para movimentação X, Y, optou-se por implementar o *Zoom* positivo quando os dois pontos

luminosos se distanciavam e o *Zoom* negativo quando os dois pontos luminosos se aproximavam. A figura seguinte apresenta um usuário segurando as canetas *laser* inicialmente próximas uma da outra e na sequência afastadas para obtenção do *Zoom*.

Figura 3-14- Usuário afastando duas canetas *laser* para obtenção do *Zoom*

O sistema identifica quando há apenas um *laser*/lanterna aceso, para a movimentação X, Y e quando há dois *lasers*/lanternas acesas para movimentação Z. Atualmente, este é o sistema de movimentação em funcionamento no LMVI. Um trecho condensado com a implementação do zoom com dois *lasers* pode ser visto no quadro seguinte.

Quadro 3-16- Código condensado do *Zoom* com dois pontos de luz

```

void MoveObject::detect(IplImage **_img)
{
    ...

    else { //two lights are on
        laserClicked = false;
        frameCount = 0;
        lastX = 0.0;
        lastY = 0.0;
        float distCenter = abs(x2-x);
        if (lastDistCenter > 0) {
            float deltaz = distCenter - lastDistCenter;
            float tempz = deltaz*moveSpeed*0.3;
            application->distZ += tempz/100;
        }
        lastDistCenter = distCenter;
    }

    ...
}

```

3.7.10.10 Dispositivo de Interação por Cor

Como alternativa às canetas *laser*, foi montado um dispositivo que permitisse a detecção de movimentação do usuário com base na cor do ponto luminoso. Assim como o uso da caneta laser descrito anteriormente, utiliza-se a *webcam* para captar as imagens do usuário movimentando o dispositivo para manipular os objetos 3D. A diferença é que ao invés de calcular o deslocamento de um ponto luminoso qualquer na imagem, identifica-se primeiro a cor desse ponto para posterior realização de uma ação.

O dispositivo dispõe de *LEDs* coloridos (azul, vermelho e verde) e três botões que permitem acender a cor desejada. Para facilitar a detecção do ponto luminoso através da

webcam, foi utilizado um filtro de material plástico que uniformiza e concentra o brilho dos *LEDs* em um mesmo ponto, como pode ser vista através da figura seguinte.

Figura 3-15- Dispositivo de interação por cor (azul, vermelho e verde)

3.7.10.11 Ponteiro para Seleção de Objetos na Cena

Após o sucesso no desenvolvimento da movimentação dos objetos, com o uso de uma caneta *laser*, foi necessário desenvolver um ponteiro que se comportasse como a “seta do mouse” para permitir a seleção de objetos específicos em uma cena para posterior manipulação.

Escrever na tela de projeção (2D), ao invés do mundo 3D, usando as bibliotecas *VR Juggler* e *OpenSG*, não é uma tarefa fácil. Portanto, foi modelado um ponteiro 2D no 3D

Studio Max, composto apenas por planos, capaz de selecionar objetos na tela de projeção, com o uso do dispositivo de interação (caneta *laser*, *joystick*, etc.).

Esse ponteiro 2D pode ser incorporado à quaisquer mundos virtuais, pois ele consiste em um arquivo VRML único que pode ser importado pela aplicação como qualquer outro objeto 3D. Para a movimentação correta do ponteiro, esse arquivo deve ser o primeiro a ser importado na linha de comando descrita na seção referente a execução de aplicações no LMVI.

O ponteiro criado é tratado pelo sistema como um nó comum da árvore de objetos, porém se movimenta somente no plano X, Y, para simular o trabalho de um *mouse* em um *Desktop*. Porém, como o ponteiro tem movimentação 2D e os objetos possuem posição 3D, foi necessário trabalhar com raios de intersecções para a correta localização dos objetos.

Para selecionar um objeto a partir da localização do ponteiro, foi necessário obter as coordenadas do centro do ponteiro e as coordenadas da cena a partir da visão da câmera, ou seja, as coordenadas *WorldToScreen* do mundo virtual, como pode ser visto no trecho de código condensado da próximo quadro. Após obter-se as coordenadas do centro do ponteiro no mundo virtual, essas coordenadas foram multiplicadas pelas coordenadas da cena, a partir da visão da câmera, obtendo-se as coordenadas do ponteiro no plano da tela.

Quadro 3-17- Função para selecionar um objeto 3D a partir da posição de um ponteiro 2D

```

NodePtr OpenSG::pickObject(NodePtr pointer) {

    //First, we get the pointer_center in world coords
    DynamicVolume pointer_vol;
    pointer->getWorldVolume(pointer_vol);
    Pnt3f pointer_center;
    pointer_vol.getCenter(pointer_center);

    //Map the camera view points from world coordinates to screen coords
    Matrix wts;
    mContextData->mWin->getPort(0)->getCamera()->
        getWorldToScreen(wts, *mContextData->mViewport);

    printf("Pointer_Center World Coords, x : %f, y :
        %f\n\n",pointer_center.x(),pointer_center.y());

    //Get the pointer_center in screen coords through wts x pointer_center
    Pnt3f result;
    wts.multFullMatrixPnt(pointer_center,result);
    Real32 x = result[0];
    Real32 y = result[1];

    printf("Pointer_Center Screen Coords, x : %f, y : %f\n\n",x,y);

}

```

As coordenadas do ponteiro no plano da tela inicialmente se encontravam no intervalo entre -1 e 1, onde o canto inferior esquerdo da tela consistia do valor “-1, -1” e o canto superior direito consistia do valor “1, 1”. No *OpenSG*, assim como no *OpenGL*, o ponto “0, 0” se encontra no canto inferior esquerdo, logo, as coordenadas X, Y variam em X, de 0 até a largura da tela e em Y de 0 até a altura da tela, de baixo para cima e da esquerda para a direita. Através do mapeamento deste intervalo para as dimensões da tela, variável do *Desktop* (640x480) para a LMVI (1024x768), obteve-se o ponto X, Y do ponteiro em pixels, como pode ser visto no trecho de código condensado do quadro a seguir.

Quadro 3-18- Mapeamento das coordenadas do ponteiro para as dimensões da tela

```

Int32 width = mContextData->mWin->getPort(0)->getPixelWidth();
Int32 height = mContextData->mWin->getPort(0)->getPixelHeight();

//The screen coords are between -1 and 1. Where: left bottom=(-1,-1) and right top=(1,1)
//Map (x,y) from -1..1 to a 0..1 interval

x = (x + 1)/2.0;
y = (y + 1)/2.0;
//Get the (x,y) in pixel coords
x = x * width;
y = (y*height)*-1;

```

Em seguida, calculou-se um raio/linha de intersecção, que parte da posição da câmera até o "infinito" da cena, passando pelo ponto X, Y do ponteiro, em coordenadas da tela. Esse raio é disparado paralelamente ao eixo Z da cena. A partir do raio/linha calculado, verificou-se se o mesmo interceptava algum objeto, como pode ser visto no trecho de código condensado do quadro seguinte.

Quadro 3-19- Projeção de um raio a partir da posição da câmera até o “infinito”

```

//Calculate a ray that starts at the camera position and goes through the pixel x, y in the
viewport port

Line ray;
mContextData->mWin->getPort(0)->getCamera()->
calcViewRay(ray,x,y,*mContextData->mWin->getPort(0));

```

Em caso positivo, retornava-se o primeiro objeto interceptado como resultado, bem como suas propriedades, como pode ser visto no trecho de código condensado do quadro a seguir.

Quadro 3-20- Obtenção de um objeto interceptado pelo raio projetado

```

if (iAct->didHit())    {
    // get the hit point
    fixedHitPoint = iAct->getHitPoint();
    //get the node that was hit (it will return a GeometryNode)
    NodePtr hitNode = iAct->getHitObject();

    //get the hitNode_center in world coords
    DynamicVolume hitNode_vol;
    hitNode->getWorldVolume(hitNode_vol);
    Pnt3f hitNode_center;
    hitNode_vol.getCenter(hitNode_center);

    this->initialHitNodeCenterX = hitNode_center.x();
    this->initialHitNodeCenterY = hitNode_center.y();

    distHitPoint2CenterX = fixedHitPoint.x() - hitNode_center.x();
    distHitPoint2CenterY = fixedHitPoint.y() - hitNode_center.y();

    this->hitNodeName = (char*)getName(hitNode);

    delete iAct;
    return hitNode;
} //end if didHit()
delete iAct;
return NullFC;

```

Com o objeto interceptado identificado, foi possível executar qualquer ação no mesmo, como arrastá-lo, ou disparar suas funcionalidades, como se estivesse sendo utilizado um clique do mouse.

Para arrastar o objeto, foi criada uma rotina para que o objeto seguisse o ponteiro 2D na tela após quando o usuário piscar rapidamente o caneta laser, como se fosse o clique duplo do *mouse* sobre um objeto na cena. Dessa forma, o usuário pode escolher qual objeto deseja arrastar no mundo virtual, como pode ser visto no trecho de código condensado do quadro seguinte.

Quadro 3-21- Função para mover o objeto selecionado com o movimento do ponteiro

```

void OpenSG::FollowPointer(NodePtr pointer, NodePtr hitNode) {
    if (hitNode != NullFC) {
        //First, we get the pointer_center in world coords
        DynamicVolume pointer_vol; pointer->getWorldVolume(pointer_vol);
        Pnt3f pointer_center; pointer_vol.getCenter(pointer_center);

        //Map the camera view points from world coordinates to screen coords
        Matrix wtsMatrix;
        mContextData->mWin->getPort(0)->getCamera()-
        >getWorldToScreen(wtsMatrix,*mContextData->mViewport);

        //Get the pointer_center in screen coords
        Pnt3f pointer_centerScreenCoords;
        wtsMatrix.multFullMatrixPnt(pointer_center,pointer_centerScreenCoords);
        //Get the hit point in screen coords
        Pnt3f hitPointScreenCoords;
        wtsMatrix.multFullMatrixPnt(fixedHitPoint,hitPointScreenCoords);

        //Update hitPointScreenCoords with the pointer_centerScreenCoords
        hitPointScreenCoords[0] = pointer_centerScreenCoords.x();
        hitPointScreenCoords[1] = pointer_centerScreenCoords.y();

        //Get the updated hitPointScreenCoords in world coords
        wtsMatrix.invert();
        wtsMatrix.multFullMatrixPnt(hitPointScreenCoords,fixedHitPoint);
        this->distNodeX = (fixedHitPoint.x() - distHitPoint2CenterX);
        this->distNodeY = (fixedHitPoint.y() - distHitPoint2CenterY);
    }
}

```

3.7.11 Inicialização das Aplicações

Após a compilação do código de programação das aplicações é gerado um arquivo executável que deverá ser executado em cada uma das máquinas do cluster. No entanto, a

execução das aplicações não depende exclusivamente deste arquivo “EXE”. É necessário executá-lo por linha de comando utilizando alguns parâmetros que definem os objetos 3D que serão importados e o arquivo de configuração do *VR Juggler* que define a configuração do dispositivo de multiprojeção, como tamanho das telas, distância entre as imagens sobrepostas de cada par de projetores, identificação de rede de cada um dos nós do cluster etc.

Uma maneira simples de executar a aplicação em cada uma das máquinas do cluster é através da criação de um arquivo de lotes do tipo “BAT”, com os argumentos necessários para a operação.

O arquivo “EXE” corresponde ao executável da aplicação, os arquivos “WRL” correspondem aos modelos 3D exportados para o formato VRML, enquanto o arquivo “JCONF” corresponde ao arquivo de configuração do *VR Juggler*. A ordem dos arquivos na linha de comando determina a maneira como a aplicação será executada. O arquivo executável deve vir sempre em primeiro lugar, depois os modelos 3D e por último o arquivo de configuração.

3.8 AQUISIÇÃO DE DADOS DE MÁQUINAS-FERRAMENTA

Este trabalho utilizou câmeras de baixo custo, webcams, e o protocolo MTConnect para a aquisição de dados reais de máquinas-ferramenta. Os detalhes mais importantes do protocolo MTConnect serão descritos a seguir.

3.8.1 Aquisição de dados via MTConnect

O protocolo MTConnect, seus adaptadores e seus agentes, apresentados no capítulo de revisão bibliográfica deste trabalho, utilizam formatos padrões para troca de mensagens de dados entre máquinas-ferramenta, aplicações, adaptadores e agentes. O formato SHDR utilizado pelos adaptadores MTConnect para aquisição e transmissão de dados de máquinas-ferramenta é descrito no próximo tópico e, em seguida, descreve-se a estrutura do padrão XML transmitidos dos agentes MTConnect para as aplicações.

3.8.1.1 Visão geral do protocolo SHDR

Como descrito em Sobel (2012), o *software* adaptador MTConnect utiliza o protocolo SHDR para a comunicação de dados. O protocolo SHDR utiliza a delimitação “|” e começa seu *stream* (corrente) de dados com um *timestamp* (definição de data e hora). A data e hora devem ser formatadas com o padrão UTC (*Coordinated Universal Time*), possuindo um Z à direita, informando que nenhum fuso horário está sendo usado.

Para praticamente qualquer tipo de dado, o formato utiliza a seguinte casal de itens:

Quadro 3-22- Casal de itens *Key* e *Value*

Um exemplo de linha SHDR é mostrado a seguir:

Quadro 3-23- Exemplo de linha SHDR

2010-09-29T23:59:33.460470Z Xact 1.4198908806

O exemplo acima mostra o valor 1.4198908806 para o dado de posicionamento “Xact”. Nesse caso, o valor de *Key* é “Xact”.

O *software* agente, por sua vez, é responsável por converter o valor para a unidade especificada pelo MTConnect, assumindo que as unidades nativas foram configuradas corretamente.

A data e a hora seguem a norma ISO 8601, no format: YYYY-MM-DDTHH:MM:SS.fffZ, sendo “YYYY” o valor do ano atual, “MM” o valor do mês atual, “DD” o valor do dia atual, “T” o marcador posicionado no começo do horário, “HH” o valor da hora, “MM” (do horário) o valor dos minutos, “SS” o valor dos segundos, “fff” as casas decimais opcionais a cada implementação e “Z” o marcador explicado acima. Se nenhuma data é fornecida pelo adaptador, o *software* agente irá providenciar o dado. O exemplo seguinte mostra um dado fornecido pelo adaptador sem a data e hora, começando com o símbolo “|”:

Quadro 3-24- Mensagem sem data e hora

Xact 1.4198908806

Se mais de um dado é coletado ao mesmo tempo, como, por exemplo, dados de posicionamento, eles podem ser fornecidos numa mesma linha. O exemplo seguinte mostra a

coleta de um dado de posicionamento do eixo X, sendo coletado conjuntamente com a coleta do dado de posicionamento do eixo Y:

Quadro 3-25- Mensagem com dados de posição coletados ao mesmo tempo

2010-09-29T23:59:33.460470Z Xact 1.4198908806 Yact -0.7842678428
--

Cada linha deve terminar com o marcador de final de linha “\n”, caractere “10” da tabela ASCII (ou 0xA).

O *software* agente é responsável pela eliminação de espaços em branco em excesso, porém recomenda-se a utilização correta do espaçamento dos dados.

O adaptador é responsável, também, pela escrita correta do vocabulário MTConnect. Por exemplo, as palavras “AVAILABLE” e “UNAVAILABLE” podem fazer parte de uma sequência de dados, porém, o agente não realizará quaisquer métodos para verificação de erros. A palavra deve ser enviada ao agente escrita corretamente, como:

Quadro 3-26- Mensagem do adaptador ao agente com vocabulário adequado

2010-09-29T23:59:29.458629Z avail AVAILABLE

No exemplo acima, o valor de *Key* é “*avail*”. Esse nome será testado com os diversos *Keys* existentes no padrão MTConnect e será combinado com o primeiro item encontrado que tiver esse valor. Os testes são realizados na seguinte ordem: *Source*, *name*, *id*. Alguns exemplos são mostrados a seguir:

Quadro 3-27- Mensagem com *id* e *name*

```
<DataItem category="EVENT" id="c1" name="mode" type="CONTROLLER_MODE"/>
```

No exemplo acima o valor de *id* é “c1” e o valor de *name* é “mode”. Portanto o valor de *Key* deve ser “c1” ou “mode”.

Quadro 3-28- Mensagem somente com identificador *id*

```
<DataItem category="EVENT" id="c1" type="CONTROLLER_MODE"/>
```

Por sua vez, o exemplo acima tem somente “id”, portanto o valor de *Key* deve ser “c1” obrigatoriamente. Se tanto “*name*”, quanto “*id*” forem fornecidos, o valor de “*name*” é testado primeiro.

O valor de *Source* é fornecido somente para nomes que sejam longos ou complexos demais para serem atributos, como, por exemplo:

Quadro 3-29- Valor de *Source* para nomes longos

```
<DataItem category="EVENT" id="c1" name="mode" type="CONTROLLER_MODE">
  <Source>channel.0.cnc@2-1</Source></DataItem>
```

No exemplo acima, o valor de *Key* pode ser “channel.0.cnc@2-1”, “mode”, ou “c1”, nessa ordem. A ordem de comparação é importante somente quando há conflitos. Os nomes e fontes (*names* e *sources*) são únicos somente dentro do escopo de cada dispositivo, enquanto que o atributo “*id*” é único entre todos os dispositivos de um arquivo XML. Como um adaptador, normalmente, trabalha com somente um dispositivo, o valor de *Key* pertence somente ao escopo desse dispositivo.

Os itens *Condition* (Estado) e *Messages* (Mensagens) possuem uma única chave (*Key*) com diversos valores. Eles devem ser enviados em uma linha separada devido à sua estrutura complexa.

O item *Condition* informa o estado de determinado componente através de seis informações conjuntas. A primeira informação é a chave, como nos exemplos acima. As outras cinco informações são listadas a seguir.

- 1) *Level* (nível) – normal, *warning* (alerta), *fault* (falha), ou *unavailable* (indisponível). Em caso de supressão do valor, *unavailable* é assumido;
- 2) *Native Code* (código nativo) – número ou nome do alarme ou aviso passado pelo dispositivo;
- 3) *Native Severity* (gravidade) – gravidade do alarme passado pelo dispositivo;
- 4) *Qualifier* (classificação) – modificador que indica a ordem da condição (*HIGH, LOW* etc.);
- 5) *Text* (texto) – texto obtido diretamente da fonte de dados, descrevendo a condição.

Um exemplo de temperatura alta de óleo (*high oil temperature*) é mostrado a seguir:

Quadro 3-30- Dado referente à temperatura de óleo

```
2010-09-29T23:59:33.460470Z|htemp|WARNING|HTEMP|1|HIGH|Oil Temperature High
```

Esse exemplo corresponde ao item de condição:

Quadro 3-31- Item de condição de temperatura

```
<DataItem category="CONDITION" id="htemp" type="TEMPERATURE"/>
```

E será entregue pelo agente dentro de uma *tag* XML, como:

Quadro 3-32- Dados internos do arquivo XML para a temperatura

```
<Warning dataItemId="htemp" timestamp="2010-09-29T23:59:33.460470Z"
sequence="399237840" type="TEMPERATURE" nativeSeverity="1" nativeCode="HTEMP"
qualifier="HIGH" >Oil Temperature High</Warning>
```

Caso não haja dados para serem enviados pelo adaptador, como é o caso de uma situação normal, sem alertas, ainda assim os delimitadores devem ser enviados para que a linha possa ser lida corretamente. O exemplo a seguir mostra um envio de dados típico de condições normais:

Quadro 3-33- Exemplo de dados para uma situação normal

```
2010-09-29T23:59:33.460470Z|htemp|NORMAL|||
```

Todos os campos são opcionais, exceto *Level*.

O atributo *Message* é uma mensagem simples que não implica diretamente na condição dos componentes do dispositivo. A mensagem carrega obrigatoriamente o *native code* que é passado logo após a chave.

Quadro 3-34- Mensagem simples

```
09-29T23:59:33.460470Z|message|CHG_INSRT|Change Inserts
```

O exemplo do quadro anterior informa o *native code* “CHG_INSRT” e o XML final teria o formato apresentado no quadro a seguir.

Quadro 3-35- Mensagem simples em XML

```
<Message dataItemId="message" timestamp="2010-09-29T23:59:33.460470Z"
sequence="399237840" nativeCode="CHG_INSRT">Change Inserts</Message>
```

Da mesma forma que o atributo *Condition*, o atributo *Message* precisa ser enviado em uma linha independente.

3.8.1.1.1 Heartbeats

O agente MTConnect utiliza *heartbeats* (pulsos) para detectar quando um adaptador foi desconectado. Essa funcionalidade é importante, pois um agente pode assumir que um adaptador foi desconectado, simplesmente porque esse adaptador parou de enviar

dados devido à ociosidade da máquina. O adaptador define a frequência dos pulsos e o agente adota esses valores. Tanto o adaptador, quanto o agente consideram que houve desconexão quando não há recebimento de pulsos durante $2 \cdot interval$ (intervalo definido pelo adaptador). Por exemplo, se um adaptador envia um pulso com o valor 10000ms, ou 10 segundos, o agente considerará que o adaptador foi desconectado, caso ele não envie pulsos durante 20 segundos. De maneira análoga, o adaptador considerará o agente desconectado, caso esse não envie pulsos durante 20 segundos.

A comunicação começa com o agente enviando uma solicitação ao adaptador em uma única linha:

Quadro 3-36- Primeira mensagem de comunicação do agente com o adaptador

* PING

Qualquer linha que comece com “*” é considerada um comando, portanto não é tratada como um par *Key/Value*, descrito acima. Quando o PING é recebido, o adaptador responde com um PONG:

Quadro 3-37- Resposta inicial do adaptador para o agente

* PONG 10000

O número após o PONG é o intervalo de pulso definido em milissegundos.

O adaptador deve checar o *socket* a cada tentativa de aquisição de dados do dispositivo. Caso ele tenha recebido um PING, ele deve retornar imediatamente um PONG.

Caso esse adaptador seja implementado com várias *Threads*, há necessidade de criar semáforos para prevenir a sobreposição dos dados.

Após a desconexão ou “*time-out*” do agente, o adaptador deve permanecer em espera por conexões.

Alguns dados de configuração dinâmica podem ser passados do adaptador para o agente:

- *uuid*
- *manufacturer*
- *station*
- *serialNumber*

Esses itens modificam os campos do XML e são passados da mesma forma que os comandos PING e PONG, com um caractere de comando “*”:

- * *uuid*: VMC1-1314421
- * *serialNumber*: 1314421
- * *manufacturer*: *Big Machine Tool Corp*

3.8.1.2 Estruturas dos documentos de um agente MTConnect

Um agente MTConnect recebe os dados de um adaptador MTConnect e os utiliza para a criação de diversos tipos de documentos antes de enviar dados a uma aplicação. Esses documentos utilizam o padrão XML para a organização de seus dados. Este tópico apresenta uma descrição detalhada da estrutura e utilização de documentos de um agente. A figura a seguir ilustra como os arquivos XML estão relacionados com a aquisição de dados reais via MTConnect e via câmeras.

Figura 3-16- Atributos da seção *Header*

Um agente MTConnect trabalha com diversas unidades de medida. A tabela a seguir mostra as principais unidades utilizadas por um agente.

Tabela 3-1- Unidades utilizadas pelo agente MTConnect

ATRIBUTO	SÍMBOLO	UNIDADE
Ângulo	°	Graus
Aceleração angular	° / s ²	Graus por segundo ao quadrado
Velocidade angular	° / s	Graus por segundo
Tempo	s	Segundos
Força	N	Newton
Tamanho	mm	Milímetros
Aceleração linear	mm / s ²	Milímetros por segundo ao quadrado
Velocidade linear	mm / s	Milímetros por segundo
Massa	kg	Quilogramas
Velocidade de rotação	rev / min	Revoluções por minuto
Temperatura	°C	Graus célsius

Qualquer documento XML de um agente MTConnect deve possuir, como raiz de documento, em seu cabeçalho, uma das três informações seguintes: *MTConnectDevices*, *MTConnectStreams*, ou *MTConnectError*. Toda resposta do agente será emitida através de um desses três tipos de arquivos XML.

Todos os documentos XML de MTConnect são divididos em duas seções principais. A primeira é chamada de *Header* (cabeçalho), que fornece informações

relacionadas ao protocolo, como os números de sequência e datas. A segunda seção refere-se ao conteúdo, seja para *Devices* (Dispositivos), *Streams* (Conjunto de dados) ou *Erros* (Erros).

Os arquivos *Devices* fornecem informações sobre cada dispositivo englobado pelo agente em questão e mostram todos os itens disponíveis para cada dispositivo. A estrutura de um arquivo XML *MTConnectDevices* é mostrada no quadro seguinte.

Quadro 3-38- Estrutura do arquivo XML *MTConnectDevices*

```
<mt:MTConnectDevices version="0.9">  
  <Header> ... </Header>  
  <Devices> ... </Devices>  
</mt:MTConnectDevices>
```

Os arquivos *MTConnectStreams* guardam diversas informações em série do tipo *Sample* (amostra) ou *Event* (evento) de dispositivos ou componentes. Sua estrutura é mostrada a seguir:

Quadro 3-39- Estrutura do arquivo *MTConnectStreams*

```
<mt:MTConnectStreams version="0.9">  
  <Header> ... </Header>  
  <Streams> ... </Streams>  
</mt:MTConnectStreams>
```

Por fim, os arquivos do tipo *MTConnectError* guardam informações sobre possíveis erros ocorridos durante uma requisição de dados. Sua estrutura pode ser vista a seguir:

Quadro 3-40- Estrutura do arquivo *MTConnectError*

```
<mt:MTConnectError version="0.9">  
  <Header> ... </Header>  
  <Error> ... </Error>  
</mt:MTConnectError>
```

A seção *Header* deve estar presente em todos os arquivos XML de MTConnect.

Todos os atributos desta seção são mostrados na figura apresentada a seguir.

Figura 3-17- Atributos da seção *Header*

Os atributos descrevem características do protocolo e da resposta, conforme os seguintes itens:

- *creationTime* refere-se ao momento em que a resposta foi criada;
- *nextSequence* refere-se ao número de sequência que deve ser usado para a próxima solicitação. É utilizado para requisições de *Sample* e *Current*. Não é usado para requisições do tipo *Probe*;
- *instanceId* refere-se ao número de instância do agente;
- *testIndicator*: *flag* opcional para indicar modo teste;
- *sender*: identificação do agente;
- *bufferSize*: quantidade de *Samples* ou *Events* que o agente armazenará.

Em geral, o tamanho do *buffer* deve ser suficiente para armazenar, pelo menos, cinco minutos de informações.

Um agente MTConnect recebe três tipos diferentes de requisições:

- *probe* – retorna os componentes e itens de determinado dispositivo;
- *current* – retorna uma “foto” dos valores mais recentes de cada item;
- *sample* – retorna dados sequenciais de *samples* e *events*.

Geralmente, a primeira requisição feita é *probe*, a fim de determinar todos os itens de cada dispositivo. Em seguida, faz-se uma requisição *current*, para obter os valores mais atuais de cada item de dados e, principalmente, para obter o próximo número de sequência, para que seja possível começar requisições *sample*. Geralmente armazena-se, também, o valor de *instanceId*, a fim de recuperar a aplicação em caso de erro do agente. A figura a seguir mostra uma comunicação típica de uma aplicação e um agente.

Figura 3-18- Comunicação típica entre uma aplicação e um agente

Primeiramente, a aplicação realiza uma requisição *probe*, recebendo como resposta um arquivo XML com os dispositivos englobados pelo agente em questão. Em seguida, a aplicação realiza uma requisição *current* a respeito do dispositivo de interesse, recebendo como resposta outro arquivo XML com os dados atuais de todos os itens de dado

do dispositivo, bem como o próximo número de sequência. Em posse do *nextSequence number*, a aplicação pode começar a realizar requisições do tipo *sample*, a fim de conseguir listas de valores de dados do dispositivo. A variável *count* define quantos dados de sequência devem ser enviados como resposta.

3.8.1.2.1 Requisições *probe*

Uma requisição *probe* retorna sempre, como resposta, um arquivo XML *MTConnectDevices*. Existem duas variações de requisições *probe*:

- Fornece informações de apenas um dispositivo, utilizando seu identificador, como por exemplo: <http://10.0.1.23/mil-1/probe>. Esse exemplo de requisição *probe* retorna como resposta apenas os componentes e dados do dispositivo “mil-1”;
- Fornece informações de todos os dispositivos englobados pelo agente: <http://10.0.1.23/probe>.

Um exemplo didático de *probe* para o dispositivo *mil-1* é mostrado a seguir:

Quadro 3-41- Exemplo de resposta a *probe*

```
<mt:MTConnectDevices>
  <Header>...</Header>
  <Devices>
    <Device name="mill-1" uuid="xxx" iso841Class="6">
      <Components>
        <Axes>
          <Linear name="X">
            <DataItems>
              <DataItem name="Xabs" type="POSITION" subType="ACTUAL" units="MILLIMETER" />
            </DataItems>
          </Linear>
          <Linear name="Y">
            <DataItems>
              <DataItem name="Yabs" type="POSITION" subType="ACTUAL" units="MILLIMETER" />
            </DataItems>
          </Linear>
        </Axes>
      </Components>
    </Device>
  </Devices>
</mt:MTConnectDevices>
```

No exemplo acima, a resposta informa que o dispositivo possui apenas dois eixos, X e Y. As informações desses eixos podem ser adquiridas via requisições *sample*.

3.8.1.2.2 Requisições sample

As requisições *sample* retornam sempre, como resposta, um arquivo XML *MTCConnectStreams*. O diagrama a seguir mostra um exemplo de componentes e itens e suas relações.

Figura 3-19- Exemplo de componentes e itens e suas relações

O dispositivo esquematizado acima possui um controlador, três eixos lineares, um eixo de rotação e dois itens para cada eixo. O controlador pode fornecer o estado de execução

do dispositivo, bem como o bloco de código. O dispositivo possui, também, um componente de energia que pode indicar se o dispositivo encontra-se ligado ou desligado.

A seguir são mostradas algumas formas de requisição *sample*, para o exemplo mostrado anteriormente:

- <http://10.0.1.23:3000/mill-1/sample>
Solicita dados de todos os componentes e itens do dispositivo;
- [http://10.0.1.23:3000/sample?path=//Device\[@name='mill-1'\]](http://10.0.1.23:3000/sample?path=//Device[@name='mill-1'])
Solicita somente dados com informação de *path*;
- <http://10.0.1.23:3000/mill-1/sample?path=//Axes>
Solicita somente informações de eixos;
- [http://10.0.1.23:3000/mill-1/sample?path=//Axes//DataItem\[@type='POSITION'\]](http://10.0.1.23:3000/mill-1/sample?path=//Axes//DataItem[@type='POSITION'])
Solicita apenas informações de posição dos eixos;

3.8.1.2.3 *Requisições current*

Uma requisição *current* responde um documento com os valores dos componentes e itens no exato momento em que a requisição foi recebida. Assemelha-se com a forma de

resposta de *sample*, porém as respostas *sample* fornecem uma sequência de dados para cada um dos itens. Um exemplo de requisição *current* que retorna todas as posições do tipo *ACTUAL* para os eixos do dispositivo, é mostrado a seguir:

- `http://10.0.1.23:3000/mill-1/current?path=//Axes//DataItem[@type="POSITION" and @subType="ACTUAL"]`

3.8.1.3 Vocabulário MTConnect

A lista completa de termos do vocabulário MTConnect pode ser encontrada no Apêndice A. A seguir são mostrados apenas alguns dos termos que foram utilizados nesta metodologia:

- *Position* (posição): Representa a localização em um eixo linear. Medida em milímetros;
- *RotaryVelocity* (velocidade de rotação): Taxa de rotação, medida em revoluções/minuto, de um eixo rotativo;

3.9 AQUISIÇÃO DE DADOS VIA WEBCAM

Além da aquisição de dados de máquinas-ferramenta via protocolo MTConnect, utilizou-se neste trabalho a aquisição de dados via câmeras de baixo custo, ou webcams. A metodologia envolvida na captação e interpretação de dados de vídeos é fundamentada pelos conceitos da área de segmentação de imagens. Porém, como o principal foco deste trabalho não é o estudo das ferramentas e tecnologias envolvidas com a otimização do processo de segmentação de imagens de vídeo, alguns conceitos serão apenas introduzidos brevemente no próximo tópico e, no próximo capítulo será mostrada uma forma de captação, transmissão e utilização de dados provenientes de webcams com estrutura de arquivos semelhante a utilizada pelo protocolo MTConnect.

Técnicas de segmentação de imagem foram utilizadas, também, para a navegação e interação do usuário com o ambiente virtual. O desenvolvimento prático descrito no tópico 3.7.10 foi utilizado em grande parte para o desenvolvimento da aquisição de dados de máquinas-ferramenta através de *webcams*. A figura a seguir ilustra a utilização de marcadores coloridos, segmentação de imagem e obtenção de centróides de figuras fechadas para a aquisição de dados de posicionamento de partes móveis de máquinas.

Figura 3-20- Marcadores coloridos, segmentação de imagem e obtenção de centróides

O fluxograma a seguir apresenta os passos necessários para a utilização de dados reais de máquinas-ferramenta, adquiridos através de *webcams*, dentro de uma aplicação virtual.

Figura 3-21- Passos para uso de dados reais de máquinas-ferramenta, adquiridos através de *webcams*, dentro de uma aplicação virtual.

O tópico 4.8 do quarto capítulo descreve com mais detalhes os passos gerais apresentados no fluxograma anterior e suas particularidades para este trabalho.

3.9.1 Segmentação de imagens

O processo de segmentação divide uma imagem em objetos ou regiões (KULKARNI, 2012). Segundo Egmont-Petersen et al. (2002), a segmentação de imagens é a etapa que une operações de processamento, com a redução de ruído e extração de bordas das regiões de interesse, dentro da imagem, e o reconhecimento de padrões para interpretação de cenas.

Grande parte dos algoritmos de segmentação verificam variações para detecção de linhas e bordas que possam dividir uma imagem em vários componentes. Algumas técnicas de segmentação são citadas por Lulio (2011).

Uma imagem pode ser representada por uma matriz, onde cada elemento corresponde a um pixel, como mostra a figura a seguir.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

Figura 3-22- Matriz de pixels de uma imagem.

As bordas demarcam o limite entre duas regiões distintas e são detectadas, geralmente, através de mudanças nos níveis de cinza de uma imagem. A existência de um ponto de borda é baseada num limiar. Ou seja, se a magnitude do gradiente é maior que o limiar, então este ponto é um ponto de borda. Portanto, o limiar deve ser escolhido de forma apropriada, a fim de se detectar verdadeiros pontos de borda.

Al-Kubati (2012) descreve o método de Canny para detecção de bordas. O operador de Canny detecta verdadeiros pontos de borda com erro mínimo e é considerado um dos melhores algoritmos para esta tarefa. Seus passos são descritos a seguir:

1. Aplicação de um filtro gaussiano para remoção de ruído. Essa tarefa embaça a imagem, removendo, assim, pontos isolados que caracterizem apenas ruído;
2. Marcação dos pixels onde o gradiente tem grande magnitude;
3. Minimização de falsas bordas na direção do gradiente;
4. Aplicar um *Threshold* duplo;
5. Eliminar bordas que não estejam conectadas às bordas principais.

Nesse método primeiramente é realizada a convolução da imagem com um filtro Gaussiano a fim de se reduzir o ruído na imagem. Essa técnica é bastante utilizada em processamento de imagens e é chamada de suavização (*Smoothing*). A magnitude do gradiente é determinada após o processo de suavização. A escolha de um limiar apropriado é de suma importância para o sucesso do processo de detecção de bordas. Se o valor do limiar for pequeno, falsos pontos de borda serão detectados como verdadeiros pontos de borda e, se o valor do limiar for grande, verdadeiros pontos serão perdidos. Para resolver este problema, dois limiares $T1$ e $T2$ são escolhidos para criar duas imagens diferentes $E1$ e $E2$, em que $T2 \approx 1,5T1$. A imagem $E1$ contém alguns falsos pontos de borda, enquanto que $E2$ além de possuir poucos falsos pontos de borda, perde poucos pontos verdadeiros. O algoritmo se inicia com os pontos de $E2$, vinculando os pontos de borda adjacentes formando um contorno. Na fronteira do contorno o algoritmo pesquisa pelos pontos de borda da imagem $E1$ com base nas

vizinhanças. Dessa forma as lacunas são preenchidas, resultando num contorno completo constituído pelos verdadeiros pontos de borda da imagem.

3.9.1.1 RGB e HSV

A cor é uma poderosa ferramenta que simplifica a identificação e extração de objetos em um cenário. Várias tonalidades e intensidades de cor podem ser discernidas. Basicamente, as cores que humanos e animais percebem em objetos são determinadas pela natureza da luz refletida pelo objeto. A luz visível é composta por uma faixa estreita de frequência do espectro eletromagnético.

Três aspectos são usados para descrever a qualidade de uma fonte de luz cromática:

- Radiância: refere-se à energia emitida por uma fonte luminosa;
- Luminância: é a quantidade de energia percebida por um observador a partir de uma fonte luminosa;
- Brilho: é um descritor subjetivo. Incorpora a noção de intensidade acromática. Pode ser entendido como a “quantidade de branco” ou intensidade da luz de uma cor.

Na retina são encontradas duas estruturas principais: os cones e os bastonetes. Os cones são responsáveis pela visão das cores e evidências experimentais estabelecem que entre seis e sete milhões de cones são sensíveis às cores vermelha, verde e azul. Aproximadamente 65% dos cones são sensíveis à luz vermelha, 33% a luz verde e 2% sensíveis a azul. Devido a

essas características de absorção do olho humano, as cores são vistas como combinações das chamadas cores primárias: vermelha, verde e azul. Somando-se as cores primárias são obtidas as cores secundárias de luminosidade: magenta, ciano e amarela.

Figura 3-23- Cores RGB (*Red, Green, Blue*).

As características usadas para distinguir as cores umas das outras são: brilho e tonalidade ou matiz e saturação. O brilho foi definido anteriormente. A tonalidade é um atributo que representa a cor dominante percebida por um observador, isto é, está associado ao comprimento de onda dominante numa mistura de ondas de luz. A saturação é o grau de pureza de uma cor. É inversamente proporcional à quantidade de luz branca adicionada.

Apesar de eficiente, o modelo RGB não é adequado quando se trata da descrição de cores realizada pela interpretação humana. Por exemplo, não se refere à cor de um automóvel em termos das porcentagens de cada componente primária. Seres humanos definem a cor de um objeto pela sua tonalidade ou matiz, saturação e brilho. O espaço HSI está estreitamente relacionado à forma com que humanos descrevem e interpretam as cores. Pode-se dizer que o modelo RGB é ideal para a geração de cores, porém seu uso para a descrição das cores torna-se limitado.

3.10 PADRÕES DE PROJETO DE *SOFTWARE*

Existem diversos obstáculos comuns, enfrentados por programadores, que já foram bem documentados, a fim de orientar e acelerar o desenvolvimento adequado de aplicações. Além disso, os padrões de projeto de *software* contribuem com a extensibilidade e flexibilidade das aplicações desenvolvidas. Com o intuito de tornar os módulos de *software* desenvolvidos neste trabalho extensíveis, de fácil compreensão e reutilizáveis, todas as implementações fundamentaram-se em alguns desses padrões. Os principais padrões utilizados estão brevemente descritos neste tópico.

O material descrito neste tópico é embasado nos livros de Freeman et al. (2007) e Gamma et al. (1994).

3.10.1 *Singleton*

O intuito do padrão de projeto chamado *Singleton* é garantir que uma classe possa ser instanciada uma única vez, durante a execução da aplicação. Esse padrão fornece, também, um ponto de acesso global a essa instância.

Algumas classes necessitam ter acesso único, como, por exemplo, uma classe de acesso a impressoras ou dispositivos compartilhados por muitos usuários. Para viabilizar essa característica, a melhor solução é fazer com que a própria classe seja responsável por manter o controle de sua única instância. A classe deve assegurar que nenhuma outra instância possa ser criada, interceptando pedidos para criar novos objetos, e pode fornecer uma maneira de acessar a instância.

A estrutura do padrão *Singleton* é mostrada a seguir:

Figura 3-24- Estrutura do padrão *Singleton*

Uma classe *Singleton* é declarada da seguinte maneira:

Quadro 3-42- Declaração de uma classe *Singleton*

```

class Singleton {
public:
    static Singleton* Instance();
protected:
    Singleton();
private:
    static Singleton* _instance;
};
  
```

Por fim, o quadro a seguir apresenta a implementação de uma classe desse tipo:

Quadro 3-43- Implementação de uma classe *Singleton*

```

Singleton* Singleton::_instance = 0;

Singleton* Singleton::Instance () {
    if (_instance == 0) {
        _instance = new Singleton;
    }
    return _instance;
}
  
```

Como seu construtor é protegido, clientes não podem instanciar a classe diretamente. Pode-se criar ou acessar a classe somente pela função membro (*Instance*). A própria classe define quando deve instanciar um objeto ou apenas retornar o objeto armazenado em *_instance*.

Como será visto nos próximos capítulos, neste trabalho o padrão *Singleton* foi utilizado nas classes *Shop_Floor*, *Conv_Start* e *Conv_End*, afim de preparar o projeto para uma extensão futura que o habilite ao acesso de múltiplas *threads*, ou seja, o acesso compartilhado por várias aplicações ao mesmo tempo.

3.10.2 Factory

O padrão *Factory* define uma interface para criação de objetos, deixando as subclasses decidirem qual classe deve ser instanciada, em outras palavras, esse padrão delega a instanciação para as subclasses.

O exemplo mais comum que sintetiza as funcionalidades do método *Factory* é ilustrado e descrito na próxima figura.

Figura 3-25- Exemplo de estrutura do padrão *Factory*.

Se tivermos uma classe abstrata chamada “Aplicação” e uma classe abstrata chamada “Documento”, um usuário pode, por exemplo, criar uma aplicação de desenho, que instancia novos documentos de desenho, através do menu “Novo”. A aplicação sabe apenas que um novo documento deve ser criado, porém ela desconhece qual o tipo de documento. Desse modo, o método *createDocument*, da classe *Application* é considerado uma “Fábrica de documentos” ou um método *Factory*.

A estrutura genérica desse padrão é mostrada na figura a seguir.

Figura 3-26- Estrutura genérica do padrão *Factory*.

Uma implementação típica da classe *Creator* é mostrada a seguir.

Quadro 3-44- Implementação típica da classe *Creator*

```
class Creator {  
public:  
    Product* GetProduct();  
protected:  
    virtual Product* CreateProduct();  
private:  
    Product* _product;  
};  
  
Product* Creator::GetProduct () {  
    if (_product == 0) {  
        _product = CreateProduct();  
    }  
    return _product;  
}
```

Neste trabalho, o padrão *Factory* foi utilizado para a criação de componentes de dispositivos e dos próprios dispositivos.

4 METODOLOGIA DE INTEGRAÇÃO *ON-LINE* DE DADOS REAIS DE MANUFATURA COM AMBIENTES DE REALIDADE VIRTUAL IMERSIVOS

Este capítulo descreve em detalhes a metodologia de integração *on-line* de dados reais de manufatura com ambientes virtuais imersivos do tipo CAVE, para criação de sistemas híbridos. A metodologia explicada neste capítulo utilizou a estrutura de *hardware* e *software* do LMVI, descrito no capítulo de materiais e métodos. O capítulo se inicia expondo alguns pensamentos iniciais e, em seguida, descreve passo a passo o projeto e implementações envolvidas na criação e utilização da metodologia proposta.

4.1 CONSIDERAÇÕES INICIAIS

Alguns passos descritos neste capítulo são específicos para a área de manufatura, porém, grande parte desta metodologia é flexível e pode ser estendida a diversas áreas de aplicação que necessitem de aquisição *on-line* de dados reais para aumentar a confiabilidade de suas simulações virtuais. Ainda, os tipos de dados de manufatura selecionados para este desenvolvimento podem ser ampliados, a fim de atingir a completeza da metodologia para esta área.

A adoção de dados reais *on-line* em simulações de RV visa, principalmente, o aumento da confiabilidade da fonte dos dados simulados, que passam a ser exibidos e processados paralelamente e quase instantaneamente com os dados do mundo real. Dessa forma, possibilita-se análise e tomada de decisão mais confiáveis por parte dos profissionais da área estudada, pois os dados exibidos comportam-se exatamente como sua fonte de dados

do mundo real. Os gerentes ou responsáveis por tomadas de decisão, de quaisquer áreas do conhecimento, não precisam, assim, se preocupar com a confiabilidade da fonte onde os dados simulados foram adquiridos, pois, com a adoção desta metodologia, a aquisição de dados é realizada *on-line* e diretamente de uma fonte real. No caso específico da manufatura, a fonte de dados reais seria um dispositivo ou máquina de chão de fábrica, em operação, durante a execução do AV. A captação de dados reais *on-line* tem como objetivo o fornecimento de informações integralmente reais, sem ignorar quaisquer dados ou circunstâncias enfrentadas pela máquina. Dados *off-line* armazenados em bancos de dados para posterior utilização em simulações, não garantem a mesma confiabilidade ou veracidade dos dados captados de forma *on-line*. A figura a seguir ilustra quais os itens envolvidos na criação de novas ferramentas para a área de manufatura através desta metodologia.

Figura 4-1- Itens envolvidos na metodologia para a criação de novas ferramentas

De maneira mais específica, a metodologia foi dividida em diversas etapas, devido à ampla abrangência de diferentes áreas envolvidas, como realidade virtual e manufatura. As principais partes que compõem essa metodologia e estruturam este capítulo são:

- 1) Projeto;
- 2) Definição dos tipos de dados utilizados;
- 3) Definição da hierarquia de classes;
- 4) Criação e padronização da modelagem de objetos virtuais;
- 5) Módulo para criação dos parâmetros e execução da aplicação;
- 6) Módulo para aquisição de dados reais com o protocolo MTConnect;
- 7) Módulo para aquisição de dados reais com o uso de câmeras;
- 8) Estrutura da aplicação virtual;
- 9) Módulo de sincronização de operações de máquinas reais;
- 10) Sincronização dos dados entre os computadores do *cluster: Front, Right e Floor*;
- 11) Estendendo a metodologia.

A figura a seguir apresenta os principais módulos deste trabalho, bem como os principais relacionamentos envolvidos entre cada um deles.

Figura 4-2- Módulos principais do trabalho e seus relacionamentos

Para que pesquisadores da área de manufatura possam adotar a metodologia desenvolvida neste trabalho para suas próprias simulações virtuais, o projeto e cada um dos passos descritos neste capítulo devem ser seguidos. Pesquisadores de outras áreas podem, também, adotar esta metodologia como base para suas simulações virtuais, através de pequenas adequações da mesma. As formas de captação de dados reais descritas neste capítulo foram construídas e embasadas com dados da área de manufatura.

Os nomes de variáveis, objetos, classes, atributos e métodos para esta metodologia foram criados e mantidos com base na língua inglesa, a fim de promover sua adoção entre o maior número de pesquisadores interessados em simulações virtuais. Além disso, como mencionado, o projeto não restringe seu uso a uma determinada área de pesquisa, podendo ser

reutilizado, em grande parte, a qualquer área de aplicação que vise a utilização de dados reais em ambientes virtuais imersivos híbridos.

4.2 PROJETO

O projeto para desenvolvimento desta metodologia visou uma forma de estudo e produção modular, para simplificar sua execução prática e o entendimento e reprodução posterior por outros pesquisadores. O fluxograma a seguir exibe uma macro visão do projeto, apenas com seus principais módulos.

Figura 4-3- Macro visão do projeto

Este desenvolvimento modular foi embasado por conceitos e técnicas fundamentais de programação, a fim de caracterizar a metodologia com:

1. Suscetibilidade à extensão, visando futura adição de novos componentes da área de manufatura e
2. Alto nível de flexibilidade para adaptação dos componentes a outras áreas de aplicação.

Primeiramente, estudou-se especificamente a área de manufatura para avaliar quais os principais dispositivos e dados disponíveis em chão de fábrica e como esses dados poderiam ser adquiridos. Em seguida, a partir da análise da área de manufatura, criou-se uma hierarquia de classes genérica visando a portabilidade da aplicação para outras áreas.

A definição do *software* mais adequado para desenvolvimento de objetos tridimensionais para esta metodologia ocorreu paralelamente aos estudos das características principais da área de manufatura, portanto a escolha do *software* adotado foi estritamente ligada à modelagem de objetos de manufatura. Com base nessa escolha, criou-se um padrão de construção a ser seguido para futuras extensões da metodologia e uma biblioteca de objetos virtuais composta de objetos de chão de fábrica.

A iluminação complexa de uma cena virtual pode ampliar em grande escala os níveis de imersão do usuário, porém, esse tipo de iluminação está fortemente ligado a diversas características de objetos virtuais complexos, não suportadas pelo formato VRML. Essa metodologia utilizou-se de *softwares* e bibliotecas ligadas à utilização do formato VRML, portanto o realismo da cena e o nível de imersão do usuário foram estritamente condicionados

à grande capacidade de imersão proporcionada do dispositivo CAVE e não pelas qualidades dos objetos importados à cena.

Devido ao grande número de áreas e conceitos envolvidos no desenvolvimento desta metodologia, optou-se pela divisão de toda programação em módulos independentes, para viabilizar, entre outras coisas, a execução remota de parte da aplicação e uma melhor documentação do projeto. Foram programados módulos para criação dos parâmetros e execução da aplicação, para desempenhar o papel inicial da aplicação, onde o usuário define o número e tipos de máquinas simuladas, bem como a relação e posicionamento dos dispositivos. Além do módulo de criação de parâmetros, foram criados dois módulos para funcionarem remotamente, como captadores de dados reais. O primeiro módulo desenvolvido foi o módulo para aquisição de dados reais de máquinas-ferramenta, através do protocolo MTConnect e o segundo módulo remoto desenvolvido foi o módulo para aquisição de dados reais com o uso de câmeras e segmentação de imagens de vídeo. O último e principal módulo desenvolvido foi a própria aplicação virtual imersiva, que se comunica com todos os demais módulos.

A instanciação de objetos não ocorre dinamicamente, isto é, durante a execução da aplicação, mas sim somente em sua inicialização. Porém, essa tarefa pode ser implementada em trabalhos futuros para possibilitar a geração de simulações dinâmicas, eliminando, assim, a necessidade de reinicialização da aplicação a cada nova instanciação de objetos.

Como o intuito desta metodologia é viabilizar e validar a utilização de dados reais *on-line* dentro de um CAVE e não a construção de uma aplicação completa para simulações de manufatura, o relacionamento entre cada instância de classe foi definido como responsabilidade dos próprios objetos de cada classe. Cada objeto guarda referências dos seus objetos vizinhos, para viabilizar acesso a seus métodos públicos. No entanto, trabalhos

futuros, com base nesta metodologia, podem criar, para simulações de grande porte, uma classe específica para controle da simulação.

Para a animação de objetos, baseada em dados reais, optou-se pela utilização de dois vetores de informação, com trabalho em série. Foi desenvolvida uma *Thread* de leitura de dados, para cada dispositivo, que atribui valores aos vetores, enquanto o *looping* principal da aplicação utiliza seus dados para animar todos os objetos. Utilizam-se apenas dois vetores para cada dispositivo, mesmo que este possua mais de um componente móvel. Diferentes dispositivos não compartilham vetores de informação.

Por fim, em uma aplicação *VR Juggler*, todos os dados de entrada que não podem ser configurados diretamente em seu arquivo externo de configuração, devem ser sincronizados entre os computadores do *cluster*, via métodos internos. Foram desenvolvidos métodos via *socket* para comunicação entre as máquinas da CAVE, a fim de sincronizar os dados reais recebidos dos módulos de captação remotos.

O fluxograma a seguir detalha o módulo da aplicação principal de RV conjuntamente com os métodos de captação de dados reais. Nos tópicos específicos de cada método, esse fluxograma será adequadamente estendido e particularizado.

Figura 4-4- Aplicação virtual em comunicação com os módulos externos.

A figura a seguir mostra as tarefas paralelas (*Threads*) que são executadas dentro da aplicação virtual:

Figura 4-5- Tarefas paralelas (*Threads*).

Cada um dos módulos e tarefas será descrito e explicado nos próximos tópicos deste capítulo, começando pela descrição detalhada dos tipos de dados utilizados para a área de manufatura, que foi a base para o desenvolvimento desta metodologia.

4.3 DEFINIÇÃO DOS TIPOS DE DADOS UTILIZADOS

Apesar de possuir características de flexibilidade e extensibilidade, esta metodologia foi desenvolvida com base em dados reais da área de manufatura. Em geral, máquinas-ferramenta produzem inúmeros dados diferentes durante a execução de suas operações. Além disso, normalmente, diferentes máquinas-ferramenta produzem diferentes tipos de dados. Portanto, a captação de dados provenientes dessas máquinas não é uma tarefa

trivial, ainda mais se levado em consideração que grande parte dessas máquinas não foi preparada para o fornecimento de dados a aplicações externas.

A definição dos tipos de dados utilizados no desenvolvimento desta metodologia foi estritamente ligada com as definições do protocolo MTConnect, descrito durante o capítulo de revisão da literatura, desta pesquisa.

O protocolo MTConnect foi desenvolvido como uma forma de padronização de dados de máquinas-ferramenta, visando facilitar a aquisição de diversos tipos de dados. Neste tópico serão apresentados os tipos de dados estipulados como essenciais ao escopo desta metodologia.

Os dados foram escolhidos com base na importância dos mesmos para simulações de realidade virtual imersiva, como, por exemplo, dados de início e término de execução de operações e dados visuais de movimentação de componentes. Como foi detalhado nos capítulos anteriores, a imersão e o realismo de objetos, movimentação, interação e animações é parte importante de aplicações e simulações com suporte de RV. Portanto, dados com ampla distinção visual, como o estado atual da porta (aberto, fechado ou destravado) e as sinalizações de alerta das máquinas-ferramenta, foram julgados como essenciais, tanto pela sua importância para a operação das máquinas, quanto pelo seu componente visual.

Com base nas considerações citadas acima, os dados selecionados para o desenvolvimento desta metodologia e suportados pelo protocolo MTConnect são listados e descritos a seguir:

- *Door_State* (*status* da porta): Estado atual da porta da máquina-ferramenta. Este pode variar dentre apenas três valores: aberto (*OPEN*), destravado (*UNLATCHED*) e fechado (*CLOSED*);

- *Emergency_Stop* (Parada de Emergência): Estado atual do atuador de parada de emergência. Este pode variar entre apenas dois valores: preparado (*ARMED*) e disparado (*TRIGGERED*);
- *Execution* (Operação): Estado de operação do controlador, que pode variar entre cinco valores: pronto (*READY*), ativo (*ACTIVE*), interrompido (*INTERRUPTED*), pausa da alimentação da ferramenta (*FEED_HOLD*) e parado (*STOPPED*);
- *Part_Count* (Contagem de peças): Número inteiro que representa a quantidade de peças produzidas até o momento
 - *All* (Todas): Contagem de todas as peças produzidas. Se o tipo de *Part_Count* for omitido, esse é o padrão;
 - *Good* (Boas): Contagem das peças produzidas corretamente;
 - *Bad* (Ruins): Contagem das peças produzidas com erros.
- *Part_Id* (Identificação da peça): Número de identificação da peça em produção;
- *Position* (Posição): Posição de um componente móvel de um dispositivo;
- *Rotary_Velocity* (Velocidade de Rotação): Velocidade de rotação de um componente, em um eixo.

A variável *Door_State* pode assumir três valores capturados pelo *software* Adaptador MTConnect e disponibilizados à aplicação principal através do arquivo XML do *software* Agente MTConnect. Esses valores informam, via uma variável do tipo texto, o estado atual da porta de determinado dispositivo. Os valores possíveis são: “porta aberta”, “porta fechada” ou “porta destravada”. Para informar esses estados, a variável, dentro do arquivo XML, assume respectivamente os seguintes valores: “*open*”, “*closed*” ou “*unlatched*”. A aplicação principal executa a leitura desses dados, localizados dentro da *tag* nomeada *Event*, atualiza os valores das variáveis internas à aplicação e dispara os métodos de animação correspondentes à animação de uma porta. Os métodos de animação são disparados pelo mesmo dispositivo (classe) que executou a leitura dos dados. Cada dispositivo ou classe instanciada possui seus próprios métodos, independentes, para leitura e processamento de dados. Métodos de diferentes dispositivos são alocados em *Threads* independentes para processamento paralelo e sem prioridades. Mais informações sobre a execução paralela de diversos dispositivos pode ser encontrada nos tópicos seguintes. A variável interna responsável pelo estado de *Door_State* controla, também, a possibilidade de inicialização de operações em dispositivos. Caso a porta de um dispositivo não possua em determinado momento o valor “porta fechada”, este não pode começar a operação de peças.

O mesmo conceito de leitura de processamento de dados descrito acima serve, também, para a variável *Emergency_Stop*. Essa variável pertence, também, à categoria *Event* dos *tags* de XML e pode assumir dois valores discretos: “preparado” ou “disparado” (“*Armed*” ou “*Triggered*”, respectivamente). Assim como a variável *Door_State*, a leitura deste dado é realizada através da *Thread* de leitura de XML independente para cada dispositivo. As mudanças de estado desta variável disparam a execução de métodos de animação para o botão de emergência e para a torre de sinais de alerta. Além de sua natureza

visual, a variável *Emergency_Stop* foi programada, também, como uma função sonora, que emite sons de alerta quando o estado da variável assume o valor “disparado” ou “*triggered*”.

Por sua vez, o estado de operação do controlador, ou a variável chamada “*Execution*”, é uma variável que pode assumir cinco valores diferentes: “pronto” ou “*ready*”, ativo ou “*active*”, interrompido ou “*interrupted*”, pausa da alimentação da ferramenta ou “*feed_hold*” e parado ou “*stopped*”. Assim como a variável descrita acima, essa variável relaciona-se com a torre de alerta, demonstrando ao usuário do dispositivo o estado de execução de uma máquina. Caso a máquina tenha sido corretamente inicializada e esteja pronta para começar uma nova operação, o estado dessa variável assume o valor “pronto” e a torre de alerta ilumina sua luz de cor verde. A animação da torre de alerta possui um procedimento específico de animação que, diferentemente de outros processos de animação desta metodologia, trabalha com escala, cores, texturas e nós de iluminação alocados especialmente para cada torre instanciada. De maneira análoga, quando um dispositivo encontra-se em operação, a variável *Execution* assume o valor “*active*”, a torre de alerta é informada e os parâmetros de seu método de animação são direcionados para a iluminação do *led* vermelho. Quando a execução é finalizada, a variável assume o valor “*stopped*” e a torre é novamente animada com a cor verde, pois se encontra disponível para entrada de dados. Igualmente, o *led* amarelo é iluminado quando a variável *Execution* assume o valor “*feed_hold*”, no momento de alimentação de dados. Como a torre de alerta normalmente não é um componente básico de um dispositivo CNC, as cores e valores de disparo relacionados foram estipulados dentro desta própria metodologia.

As variáveis *Part_Count(All)* e *Part_Id* são de extrema importância para o desenvolvimento de uma instância da aplicação que contenha diversos dispositivos interagindo entre si. Essas variáveis foram utilizadas, nesta metodologia, para a identificação

e sincronização de início e término de operação de uma peça em um dispositivo, disparando métodos que sincronizam a execução de todos os dispositivos e componentes adicionados a uma determinada instância da aplicação principal.

Dispositivos e transportadores são ligados entre si conforme a ordem de adição dos mesmos à cena virtual. A ordem de entrada é definida no módulo de criação de parâmetros, que permite que o usuário posicione cada componente da cena e defina sua relação, com base na numeração dos dispositivos. Por exemplo: o dispositivo identificado com o número “1” se relaciona apenas com o transportador numerado com o valor “2”. O dispositivo número “3” se relaciona com o transportador numerado com o valor “2” e com o transportador numerado com o valor “4” e assim por diante. Com base na relação definida pelo usuário, os itens da cena são enfileirados em uma *string* de parâmetros de execução da aplicação principal e, no momento inicial da execução, os dispositivos são instanciados e os dispositivos vizinhos são relacionados entre si, através de atributos de suas classes. Esses atributos específicos guardam o valor dos itens relacionados com o objeto, tanto quanto à posição anterior, quanto à posição posterior. Um objeto recebe dados do objeto guardado em sua variável “objeto anterior” e envia dados para o objeto guardado em sua variável “objeto posterior”.

Métodos de atualização, específicos para cada dispositivo, verificam constantemente o valor de *Part_Count (All)* no arquivo XML, criado pelo agente, e alteram as variáveis inteiras de controle quando há acréscimo ao valor de *Part_Count*. A variável *Part_Id* foi relacionada diretamente com *Part_Count Bad* e *Part_Count Good*, a fim de informar o usuário, após a execução da aplicação, sobre quais peças foram registradas como *Bad* (peça defeituosa) ou *Good* (peça produzida com sucesso).

Os demais dados selecionados como essenciais para esta metodologia se referem ao posicionamento dos componentes móveis dos dispositivos. A variável *Position* (Posição)

contém o valor da posição de um componente móvel de um dispositivo. Essa posição pode ser referente ao eixo X, Y ou Z. A variável *Position* guarda apenas valores de translação. Por sua vez, a variável *Rotary_Velocity* (Velocidade de Rotação) armazena valores de velocidade de rotação de um componente móvel de um dispositivo, com relação a ao eixo X, Y ou Z. Ambas as variáveis de posicionamento ou rotação possuem o tipo *double* e requerem alto nível de precisão dentro de uma simulação de realidade virtual.

Como citado acima, cada dispositivo possui uma *Thread* para leitura dos dados de seu arquivo XML particular, criado pelo agente MTConnect e disponibilizado na *Internet* para *download*. Os dados de posicionamento são lidos constantemente pela mesma *Thread* responsável por realizar a leitura dos demais dados descritos acima. Os dados de posicionamento e rotação são armazenados em dois vetores de informação, que funcionam como um *buffer* entre o *download* das informações e os procedimentos de animação para translação e rotação de objetos. Mais detalhes sobre esse procedimento podem ser encontrados nos próximos tópicos.

Como citado acima, além dos dados estipulados neste tópico, outros tipos de dados padronizados pelo MTConnect podem ser estendidos a esta metodologia. O próximo tópico apresenta uma estrutura de classes que foi criada com o objetivo de viabilizar a característica de extensão para a metodologia.

4.4 DEFINIÇÃO DA HIERARQUIA DE CLASSES

Após a definição dos tipos de dados utilizados, a tarefa subsequente estipulada para o desenvolvimento da metodologia foi a definição de uma estrutura de árvore hierárquica a ser adotada pelas classes e subclasses do sistema. Esse passo do desenvolvimento visou enriquecer esta metodologia com a capacidade de extensão com relação aos dispositivos e tipos de dados implementados, além de prover flexibilidade para sua adequação a outras áreas de aplicação. Todas as classes e a hierarquia adotada para os dispositivos e componentes são mostradas a seguir:

- Classe concreta *Shop_Floor* (Fábrica - classe principal).
- Classe base abstrata *Device* (Dispositivo)
 - Classe filha concreta *Device_MTC* (Dispositivo MTConnect);
 - Classe filha concreta *Device_Virtual* (Dispositivo Virtual);
 - Classe filha concreta *Device_Simulator* (Simulador de dispositivo MTConnect).
- Classe base abstrata *Conveyor* (Transportador)
 - Classe filha concreta *Conveyor_Robot*;
 - Classe filha concreta *Conveyor_Worker*;
 - Classe filha concreta *Conveyor_Start*;
 - Classe filha concreta *Conveyor_End*.
- Classe abstrata *Camera* (Câmera)

- Classe filha concreta *Cam_Component* (Componente via câmera);
- Classe filha concreta *Cam_Conveyor* (Transportador via câmera);
- Classe filha concreta *Cam_Generic* (Apenas animação).
- Classe base abstrata *Component* (Componente)
 - Classe filha concreta *Comp_Door* (Porta);
 - Classe filha abstrata *Comp_Axes* (Eixos)
 - Classe neta concreta *Comp_Axes_Linear* (Eixo de Translação);
 - Classe neta concreta *Comp_Axes_Rotary* (Eixo de Rotação).
- Classe fábrica de componentes *Component_Factory*.

Uma classe abstrata não pode ser implementada. Portanto, as classes *Device*, *Conveyor*, *Component* e *Camera* servem apenas como abstrações ou generalizações das suas respectivas classes concretas. Essas abstrações servem para tornar o projeto de *software* mais flexível, pois com a utilização de referências a classes abstratas, é possível tratar quaisquer tipos de classes concretas de uma mesma maneira. Assim, pode-se estender a quantidade de tipos classes concretas sem necessidade de modificação da parte principal do código fonte.

A classe *Shop_Floor* (chão de fábrica) é a classe principal da metodologia criada. Ela tem diversas responsabilidades, entre as quais, a instanciação das demais classes e a inicialização do grafo de cena (estrutura da biblioteca *OpenSG*, em forma de árvore, que armazena os objetos da cena virtual). Essa classe não é abstrata, portanto pode ser instanciada. Porém, como se trata de uma classe principal de controle, que deve ser instanciada uma única vez, decidiu-se implementá-la como uma classe *Singleton*. Assim, em uma futura extensão do

projeto, onde diversas aplicações pudessem ter acesso ao mesmo ambiente, todas compartilhariam adequadamente a classe *Shop_Floor*. Alguns detalhes específicos do compartilhamento de *Threads* (processamento paralelo) não serão tratados neste trabalho.

Afim de suprir as necessidades de um sistema híbrido, a classe *Device* é abstrata e possui três possíveis implementações, através das classes concretas *Device_MTC*, *Device_Virtual* e *Device_Simulator*. Essas classes concretas referem-se, respectivamente, à implementação de objetos alimentados por dados reais, através do protocolo MTConnect, à implementação de objetos puramente virtuais e à implementação de objetos que serão alimentados pelo módulo de simulação de dados reais. O último módulo foi desenvolvido para viabilizar testes da aplicação com determinados tipos de dados de máquinas reais que não puderam ser obtidos diretamente de dispositivos reais.

A classe *Conveyor* é abstrata e possui quatro possíveis implementações, através das classes concretas *Conveyor_Robot*, *Conveyor_Worker*, *Conveyor_Start* e *Conveyor_End*. As classes *Conveyor_Start* e *Conveyor_End* são classes de controle, que servem apenas para controlar, respectivamente, a entrada de dados para o primeiro dispositivo instanciado e a saída de dados do último dispositivo instanciado. Elas devem ser instanciadas uma única vez durante toda a execução da aplicação, portanto, assim como a classe *Shop_Floor*, essas duas classes foram implementadas como classes *Singleton*. Essas classes derivam da classe base *Conveyor*, pois, apesar de serem classes de controle, elas possuem comportamento muito semelhante às demais implementações da classe *Conveyor*, portanto devem herdar seus atributos e métodos.

As classes *Conveyor_Robot* e *Conveyor_Worker* referem-se, respectivamente às implementações de braços robóticos utilizados como transportadores e transportadores no formato de “esteira + operário”. Como explicado acima, outras formas de transportadores podem ser estendidas nesta metodologia, criando-se uma classe concreta que implemente os

métodos da classe base *Conveyor* e, derivando-a dessa mesma classe. Claramente, ainda há necessidade de modelagem dos novos objetos virtuais, seguindo-se os padrões descritos nos próximos tópicos desta metodologia.

A classe *Camera* é, também, uma classe abstrata e possui três formas diferentes de implementação: *Cam_Component*, *Cam_Conveyor* e *Cam_Generic*. Assim como todas as implementações concretas da classe *Device* se referem à aquisição de dados reais via MTConnect, todas as classes concretas de *Conveyor* se referem à captação de dados reais via câmera. Individualmente, elas se referem, respectivamente, à aquisição de dados de um componente móvel de um dispositivo, à aquisição de dados de partes móveis de transportadores e, por fim, à aquisição de dados móveis genéricos. Apesar de todas as três implementações concretas utilizarem dados de câmera, elas foram desenvolvidas a fim de processarem esses dados de maneira particular. Por exemplo, os objetos da classe *Cam_Generic* apenas movimentam os objetos móveis virtuais de acordo com os dados recebidos do objeto móvel real, em todos os eixos. Os objetos da classe *Cam_Conveyor* utilizam os dados de movimentação recebidos, porém, tratam de maneira especial o início e término de execução da movimentação. Para a classe *Cam_Generic* não há início ou término de execução. Por sua vez, a classe *Cam_Component* possui um tratamento especial dos dados, pois, apesar de receber dados de câmeras, foi criada para trabalhar em conjunto com os componentes móveis que recebam dados de MTConnect.

Dentro desta metodologia, todos os dispositivos são compostos por partes estáticas e partes móveis. Uma parte móvel de um dispositivo é considerada um componente e é instanciada através das implementações da classe *Component*. A classe *Component* é uma classe base abstrata e possui como classes filhas, tanto classes concretas, quanto classes abstratas. Desse modo, há um nível a mais de abstração para um dos galhos de ramificação da

árvore de filhos da classe *Component*. O galho mais simples dessa árvore é o galho que hospeda a classe *Comp_Door*. Essa classe refere-se à porta de um dispositivo, com movimentação em apenas um eixo de translação.

O outro galho de *Component*, a classe *Comp_Axes*, hospeda uma classe abstrata, referente aos outros componentes de movimentação de um dispositivo. Essa classe permite a implementação de múltiplos eixos para um componente, bem como a implementação tanto de eixos lineares, de translação, quanto de eixos rotacionais. As classes que derivam de *Comp_Axes* são *Comp_Linear* e *Comp_Rotary*. Ambas possuem as características de um eixo de movimentação, porém a primeira particulariza essas características com métodos de translação e a segunda particulariza essas características com métodos de rotação.

Por fim, a classe *Component_Factory* é uma classe concreta, baseada no padrão de projeto *Factory*, que facilita a extensão da quantidade de componentes da metodologia e sua criação. Essa classe permite a criação de componentes sem a necessidade de conhecimento das classes concretas. O usuário requisita a instanciación de um *Component* e a própria classe *Component_Factory* decide qual classe instanciar, baseada nos parâmetros passados pelo usuário.

Nos próximos tópicos deste capítulo serão descritos, em detalhes, os procedimentos, métodos, atributos e demais informações sobre todas as classes que compõem a metodologia. O próximo tópico mostra, principalmente, a padronização criada para a modelagem de objetos virtuais, para que seja possível sua utilização dentro desta metodologia.

4.5 CRIAÇÃO E PADRONIZAÇÃO DA MODELAGEM DE OBJETOS VIRTUAIS

Uma aplicação de realidade virtual é fundamentada na exibição de objetos virtuais, bem como na navegação e interação do usuário com o ambiente virtual. Portanto, a criação e adequação de objetos virtuais à aplicação de RV é uma parte importante de qualquer trabalho da área.

Este tópico apresenta detalhes importantes a respeito do *software* de modelagem *3D Studio Max*, que foi escolhido como *software* de modelagem de objetos virtuais mais adequado para esta metodologia. Além disso, são mostradas, também, em detalhes, as máquinas, seus componentes móveis e demais dispositivos modelados, bem como suas características essenciais e os passos envolvidos na criação dessas características.

O escopo deste trabalho é restringido à área de manufatura, portanto a biblioteca de objetos virtuais criada para esta pesquisa é composta apenas por dispositivos relacionados ao chão de fábrica. Porém, a metodologia proposta é suficientemente flexível e extensível para a adequação da mesma a outras áreas de aplicação, mesmo a áreas distantes da manufatura, ou até mesmo, da engenharia.

Vários *softwares* de modelagem foram testados e utilizados durante o desenvolvimento deste trabalho, porém o *software* definido como *software* mais adequado para a modelagem de objetos tridimensionais deste trabalho foi o *software 3D Studio Max* da *Autodesk*. A escolha foi tomada com base, principalmente, em sua capacidade de importação e exportação de arquivos VRML, com extensão “WRL”. Qualquer *software* de modelagem que exporte naturalmente, ou via *plug-in*, arquivos VRML, pode ser usado para a produção dos objetos a serem utilizados nas aplicações embasadas por esta metodologia. O arquivo

VRML é a base para a criação da cena virtual da aplicação desenvolvida para pesquisa, com exceção de sua iluminação.

Visando, entre outras coisas, a realização do estudo de caso, a fim de validar a metodologia proposta, foi criada uma biblioteca de objetos virtuais de chão de fábrica, com os seguintes componentes:

- Torno mecânico;
- Torno CNC;
- Furadeira;
- Fresadora;
- Esteira transportadora;
- Operador;
- Braço robótico;
- Cenário genérico para chão de fábrica.

Essa biblioteca pode ser estendida a qualquer momento por novos programadores ou usuários do sistema proposto que sigam os padrões descritos neste tópico.

Os objetos da biblioteca desta pesquisa não puderam ser modelados utilizando-se toda a capacidade de realismo do *software 3D Studio Max*, pois a exportação dos arquivos para o formato VRML acarreta em restrições de *features* (características) de modelagem, iluminação e texturização. Portanto, todos os objetos citados acima foram desenvolvidos com base nas características do formato VRML97.

Cada objeto foi modelado e nomeado de acordo com as necessidades de exibição, animação e interação da aplicação principal. Para isso, a principal ferramenta básica do *software 3D Studio Max* que foi utilizada durante esse desenvolvimento, foi a ferramenta de

agrupamento de objetos. Essa ferramenta permite a criação de grupos de objetos, nomeados de forma independente, em uma estrutura de árvore. Assim, grupos ou objetos filhos de outro grupo, ou em nível inferior com relação a outro grupo, movem-se de acordo com o posicionamento de seu grupo base, além de sua própria movimentação.

Estipulou-se como padrão para esta metodologia que todas as partes estáticas dos objetos virtuais fossem mantidas em um único grupo, com o nome de “*STATIC*”. Também, foi estipulado que cada componente móvel de cada um dos objetos deve ser agrupado com o nome de seu eixo de translação ou rotação, como, por exemplo, X, Y, Z ou C. Todos os objetos criados com movimentação no eixo X devem ser agrupados e esse grupo deve ser nomeado apenas pela letra “X”, em letras maiúsculas. Todos os objetos que possuem rotação devem ser agrupados e esse grupo deve ser nomeado apenas pela letra “C” e assim por diante.

Além dos eixos simples de translação ou rotação, há possibilidade de nomear os grupos de objetos contendo eixos de translação duplos ou triplos. Os objetos que possuem movimentação tanto no eixo X, quanto no eixo Y, devem ser agrupados e esse grupo deve ser nomeado apenas por duas letras maiúsculas “XY”. O mesmo procedimento deve ser usado para grupos de objetos com movimentação em XZ, YZ ou XYZ.

Dentro do *software* de modelagem, a criação de grupos em uma estrutura hierárquica acarreta na movimentação conjunta de partes agrupadas. Essa composição hierárquica e a nomenclatura correta dos grupos são importantes, também, para a animação adequada dos componentes na cena virtual da aplicação principal. Dentro da aplicação principal, quando as classes *Device* e *Component* são instanciadas, há um procedimento que busca no arquivo VRML exportado, os nomes de grupos X, Y, Z, C, XY, XZ, YZ e XYZ, a fim de cadastrar os componentes móveis de cada dispositivo e preparar sua animação. Não há tratamento especial para o grupo nomeado como *STATIC*. A classe *Component* é instanciada

somente para a criação de partes móveis de translação, rotação ou especificamente para a porta de um dispositivo.

Ao agrupar objetos ou subgrupos, deve-se respeitar a hierarquia de movimentação entre cada objeto. Por exemplo, todos os objetos ou subgrupos filhos do grupo nomeado “Z” se movimentarão no eixo Z, quando o grupo “Z” for animado. Essa regra deve ser válida mesmo para subgrupos que sejam objetos móveis, como, por exemplo, X ou Y ou C. Por fim, depois de agrupados e nomeados todos os componentes móveis e estáticos, deve-se agrupar todos os itens e subgrupos resultantes em um único grupo principal, com o nome do dispositivo modelado. Ao final do procedimento, deve-se restar apenas um grupo principal, que engloba todos os subgrupos e itens do objeto.

Duas outras formas de padronização dos objetos da biblioteca de objetos virtuais devem ser seguidas para a incorporação de novos objetos à biblioteca. A primeira delas se refere à unidade de medidas utilizada na criação dos objetos. Deve-se alterar a unidade de medida padrão do *3D Studio Max* para *Generic Units*. Dessa forma, as unidades de medida da aplicação principal e da biblioteca de objetos virtuais serão equivalentes. A figura seguinte mostra a localização da configuração de unidades do *3D Studio Max*.

Figura 4-6- Configuração das unidades de medida do *3D Studio Max*

A segunda forma de padronização se refere ao posicionamento, orientação e tamanho dos objetos virtuais modelados. A respeito da orientação e direção dos eixos, a aplicação OpenSG e o formato VRML adotam o mesmo padrão. Porém, o *software 3D Studio Max* adota um padrão um pouco diferente. Dentro da aplicação desenvolvida em OpenSG, o eixo Y é considerado como altura e o eixo Z como profundidade. No entanto, em um arquivo VRML, o eixo Y é considerado profundidade e o eixo Z, altura. E, ainda, a direção do eixo de profundidade é oposta entre as padronizações. As figuras seguintes esclarecem as diferenças entre as padronizações de eixos de OpenSG, VRML e *3D Studio Max*.

Figura 4-7- Eixos e direções adotadas pelo *software 3D Studio Max*

Figura 4-8- Eixos e direções do formato VRML e OpenSG (Y e Z invertidos e direção contrária de Z)

Tanto a modelagem quanto a codificação desta metodologia, em C++/OpenSG, foram baseadas nessas informações de padronização de eixos e quais extensões à biblioteca devem seguir o mesmo formato.

O quadro a seguir exemplifica um arquivo VRML, exportado pelo *3D Studio Max* 9.0. A figura exportada foi um paralelepípedo com as seguintes dimensões de *3D Studio Max*: largura (X) de 18 unidades, altura (Y) de 23 unidades e profundidade (Z) de 12 unidades. Os valores de posicionamento foram: 5 unidades para X, -6 unidades para Y e 7 unidades para Z.

Quadro 4-1- Exemplo de exportação VRML

```
#VRML V2.0 utf8
# Produced by 3D Studio MAX VRML97 exporter, Version 9, Revision 1
# Date: Thu Jul 26 08:50:18 2012
DEF Box01 Transform {
  translation 5 7 -6
  children [
    Transform {
      translation 0 11.5 0
      children [
        Shape {
          appearance Appearance {
            material Material {
              diffuseColor 0.3333 0.1098 0.6941
            }
          }
          geometry Box { size 18 23 12 }
        }
      ]
    }
  ]
}
```

Pode-se constatar, pelo quadro acima, que no campo *translation* (translação) os dois últimos valores estão invertidos, com relação ao *3D Studio Max*. Além disso, o sinal do terceiro valor está negativo, ou seja, com direção oposta à direção do *3D Studio Max*.

Durante a fase de modelagem, os objetos devem ser posicionados com sua base paralela ao plano XY. Além disso, os objetos virtuais devem ser mantidos paralelos ao plano XZ, de modo que sua frente seja mantida voltada para o lado negativo de Z e suas costas para o lado positivo do eixo Z, dentro do *3D Studio Max*.

Deve-se manter o centro dos objetos criados exatamente no centro do eixo de coordenadas do *software* de modelagem, ou seja, no ponto “0x0x0”, ou ponto de intersecção de X, Y e Z. Para executar essa tarefa, deve-se mover o grupo principal do objeto, logo após o término da fase de agrupamento e nomenclatura. A movimentação dos itens e subgrupos não garante o posicionamento correto do grupo final. Essa forma de padronização é importante para a aplicação principal, pois a operação de rotação é aplicada com base no centro dos eixos do objeto. Ou seja, para que um objeto possa girar em torno do seu próprio eixo, esse deve estar localizado exatamente ao centro do eixo de coordenadas. Os objetos que possuem seu centro localizado longe do centro do eixo de coordenadas, quando exportados para o formato VRML comportam-se como se houvesse um *bound box* (caixa de contorno) invisível circundando-o e forçando seu centro a permanecer na intersecção dos eixos.

Além de garantir a orientação e posicionamento correto do grupo final, deve-se garantir que o tamanho dos objetos criados seja condizente com a proporção estipulada para a aplicação principal. Para o estudo de caso desta metodologia foi estipulado que 200 unidades de medida genéricas equivaleriam a 2 metros.

O próximo tópico descreve os procedimentos de modelagem padronizados especificamente para a produção de componentes móveis.

4.5.1 Criação dos componentes móveis dos dispositivos

Este tópico apresenta os objetos virtuais modelados e suas partes móveis, para serem usados como referência na produção de novos objetos. No início da implementação deste trabalho, havia necessidade de exportação independente das partes estática e móveis dos objetos virtuais, porém, a versão final da metodologia foi estendida para se adequar à árvore

de objetos 3D criada pelo módulo de exportação de VRML do *3D Studio Max*. Portanto, a necessidade de exportação de partes independentes foi substituída pela nomenclatura adequada de grupos e demais formas de padronização.

O torno mecânico foi modelado contendo uma parte estática agrupada com o nome “*STATIC*” e com um eixo de rotação nomeado como “C” e um eixo de translação nomeado como “X”:

Figura 4-9- Torno Mecânico modelado em *3D Studio Max*

Figura 4-10- Parte móvel “X” do torno mecânico

O torno CNC foi modelado com um eixo de rotação C, filho de um eixo de translação Z, para que os movimentos do eixo árvore, em Z, movimentassem, também, a ferramenta com rotação, localizada na base do eixo árvore. Além de Z e C, foi criada uma mesa com translações em X e Y, portanto nomeada como “XY”. A figura a seguir mostra um exemplo de torno CNC para o ROMI D600, porém esse modelo pode ser usado genericamente ou adaptado para quaisquer tornos CNC.

Figura 4-11- Exemplo de modelagem de centro de usinagem para o ROMI D600

A fresadora foi modelada com um eixo de rotação C, filho de um eixo de translação Z, além de uma mesa com deslocamento em X e Y.

Figura 4-12- Fresadora virtual

Figura 4-13- Acessório virtual para fresadora

A furadeira foi modelada com um eixo de rotação C, filho de um eixo de translação Z.

Figura 4-14- Furadeira virtual

Figura 4-15- Acessório virtual para a furadeira

Além das máquinas típicas de chão de fábrica, foram modelados, também, transportadores, para simular o transporte de peças entre máquinas. Foram modelados dois tipos de transportadores: uma esteira transportadora e um braço robótico. Outros tipos de transportadores podem ser estendidos a esta metodologia, mediante adoção dos procedimentos corretos, descritos neste trabalho.

Para ser adequada a um sistema híbrido, a modelagem dos transportadores deve seguir, em geral, a mesma padronização usada para dispositivos, descrita acima, porém, além da possibilidade de trabalhar com dados reais de câmeras, os transportadores podem carregar em seu arquivo VRML dados referentes à sua animação *off-line*. Dessa forma, os transportadores podem ser animados com dados adquiridos diretamente de seu próprio arquivo VRML, o que pode ser importante para simulações entre máquinas reais independentes de transportadores reais. A figura seguinte mostra uma esteira modelada apenas com um eixo de translação X e um operador modelado com eixo de translação XZ.

Figura 4-16- Esteira de transporte e operador

Outra forma de transportador modelada neste trabalho foi o braço robótico. Não há cálculo de juntas para as animações dos braços robóticos, pois, novamente, essa tarefa encontra-se fora do escopo desta pesquisa. O arquivo VRML do braço robótico pode, assim como a esteira, carregar consigo os dados de animação, para que seja possível animá-lo *offline*, sem necessidade de alimentação por dados reais.

Figura 4-17- Braço robótico

Por fim, foi modelado um ambiente virtual genérico para chão de fábrica, que pode ser utilizado em quaisquer simulações da área. Outros cenários 3D de chão de fábrica podem ser modelados para estender a biblioteca de objetos virtuais criada neste trabalho, desde que a posição do solo seja coincidente com o plano formado pelos eixos X e Y do *3D Studio Max*. O tamanho do cenário virtual deve ser suficientemente grande para englobar

todas as máquinas utilizadas nas simulações virtuais, seguindo-se a padronização estipulada nesta metodologia (2m = 200 unidades genéricas).

Figura 4-18- Cenário genérico de chão de fábrica

4.6 MÓDULO PARA CRIAÇÃO DOS PARÂMETROS E EXECUÇÃO DA APLICAÇÃO

A certificação da criação correta de todos os parâmetros necessários é uma tarefa crucial para os interessados em implementar ou estender essa metodologia.

Para simplificar a criação dos parâmetros, desenvolveu-se um módulo específico para inicialização da aplicação, onde todos os parâmetros necessários são criados via interface visual. Este módulo é, na verdade, um *software* independente, que permite que o usuário da aplicação configure visualmente o tamanho do cenário, a posição e tipo das máquinas

utilizadas, bem como a relação entre cada uma das máquinas. O cenário tridimensional, nomes de arquivos de configuração externos e caminhos de objetos VRML devem, também, ser definidos dentro deste módulo. Ainda, pode-se disparar a aplicação principal através deste módulo, após a criação e validação de todos os parâmetros.

A primeira tarefa da aplicação principal é avaliar a lista de parâmetros que é passada pelo usuário junto ao disparo do arquivo executável. Isso é, geralmente, feito através de linha de comando, ou arquivos de lote com extensão “BAT”, que servem para automatizar tarefas. O quadro a seguir mostra um exemplo de linha de comando gravada em um arquivo com extensão “BAT”, que pode ser disparada pelo usuário, quando da execução do arquivo.

Quadro 4-2- Linha de comando resultante do módulo de criação de parâmetros

```
VirtualWorld.exe -scene fab3.wrl -dev_MTC VMC-3Axis.wrl -350 -150  
-conv_worker conveyor.wrl -150 -150 -dev_simulator furadeira2.wrl 50 -150  
-conv_worker conveyor.wrl 220 -150 -dev_virtual VMC-3Axis.wrl 500 -150 MTCStand.jconf
```

O módulo de criação de parâmetros tem como resultado principal linhas de comando idênticas àsquelas criadas manualmente para disparo da aplicação. Portanto, o módulo elimina a necessidade de criação manual dessas linhas, o que diminui consideravelmente a complexidade de inicialização da aplicação, pois linhas de comando possuem, geralmente, dezenas de parâmetros.

Além de simplificar a tarefa do usuário de definição dos parâmetros da aplicação, esse módulo possui, também, a função de validar os parâmetros definidos. Como um dos objetivos da metodologia é manter flexível sua área de utilização, é importante que haja uma ferramenta para validação de parâmetros, para que a execução da aplicação possa fluir da melhor maneira possível. Portanto, as principais tarefas deste módulo são:

- 1) Servir como uma interface visual para a definição dos parâmetros da aplicação
 - a. Visualização do tamanho do cenário utilizado para disposição correta das máquinas no *layout*;
 - b. Definição dos arquivos VRML de geometria dos objetos 3D;
 - c. Definição dos nomes, endereços IP e portas para requisições dos dados de cada dispositivo;
 - d. Definição do posicionamento das máquinas e outros objetos virtuais;
 - e. Definição das relações entre objetos e
 - f. Definição do cenário virtual.
- 2) Validar os parâmetros definidos pelo usuário e
- 3) Disparar o arquivo executável da aplicação principal.

A tela inicial da interface do módulo de criação de parâmetros pode ser vista na figura seguinte.

Figura 4-19- Interface para definição dos parâmetros iniciais da aplicação virtual

Além dessas tarefas, o módulo é responsável, também, pela configuração, ou calibragem, das cores definidas para os marcadores, no caso de aquisição de imagens via câmeras.

O primeiro nome da lista, contida no arquivo “BAT”, se refere ao nome do arquivo executável da aplicação. Dentro do módulo há um campo específico para a definição do nome da aplicação, que, por padrão, tem seu valor inicializado com o executável da aplicação desenvolvida neste trabalho: “*VirtualWorld.exe*”.

Os nomes precedidos por hífen são *tags* (rótulos), ou marcadores, que informam a aplicação que a *string* subsequente é um parâmetro do tipo informado pela *tag*. Os valores possíveis de *tags* são:

- *-scene*: define o arquivo WRL que será usado como cenário da aplicação;

- `-dev_mtc`: define que a aplicação deve instanciar um dispositivo do tipo *Device_MTC*, subclasse da classe abstrata *Device*, e o nome do arquivo WRL que contém a geometria do mesmo. Logo após o nome do arquivo WRL são passados os parâmetros *IP*, *PORT* e *X* e *Y* de posição do dispositivo. O nome do arquivo WRL sem a extensão, será usado, também, nas requisições de MTConnect;
- `-dev_cam`: define que a aplicação deve instanciar um dispositivo do tipo *Device_Cam*, subclasse da classe abstrata *Device*, e o nome do arquivo WRL que contém a geometria do mesmo.
- `-dev_virtual`: define um dispositivo que não utilizará dados reais de câmeras, MTConnect ou qualquer outra fonte de dados reais. Esse dispositivo se comportará baseado apenas em dados virtuais;
- `-conv_worker`: define que a aplicação deve instanciar um transportador do tipo *Conveyor_Worker*, subclasse da classe abstrata *Conveyor*, e o nome do arquivo WRL que contém a geometria do mesmo. Logo após o nome do arquivo WRL são passados os parâmetros *X* e *Y* de posição do transportador;
- `-conv_robot`: define que a aplicação deve instanciar um transportador do tipo *Conveyor_Robot*, subclasse da classe abstrata *Conveyor*, e o nome do arquivo WRL que contém a geometria do mesmo. Logo após o nome do arquivo WRL são passados os parâmetros *X* e *Y* de posição do transportador;

Todos os dispositivos, sejam baseados em dados exclusivamente virtuais ou reais, derivam da classe *Device*, portanto são tratados igualmente pela aplicação. Dessa forma, o nível de flexibilidade da metodologia é aumentado.

Além dos tipos de *tag* descritos acima, uma *tag* auxiliar foi criada exclusivamente para a validação da metodologia:

- `-dev_simulator`: define que a aplicação deve instanciar um dispositivo do tipo *Device_Simulator*, subclasse da classe abstrata *Device*, e o nome do arquivo WRL que contém a geometria do mesmo. Logo após o nome do arquivo WRL são passados os parâmetros IP, PORT e X e Y de posição do dispositivo. A diferença desse dispositivo para um *Device_MTC* é que, apesar de ele se comportar como um dispositivo real, lendo arquivos XML pela Internet, ele não executará requisições MTConnect. Esse dispositivo fará um simples *download* de arquivo XML, no padrão MTConnect, mas sem requisições padronizadas. Após o *download*, o dispositivo interpretará seus dados normalmente, como um *Device_MTC*. Portanto, a única diferença entre ambos é a forma de *download* de arquivos XML. Esse tipo de dispositivo foi construído para possibilitar a simulação de dados de MTConnect sem a necessidade de comunicação com uma máquina real em funcionamento. Ele deve ser usado conjuntamente com outro módulo de *software* desenvolvido neste trabalho, construído exclusivamente para validação da metodologia: o *MTC_XML_Writer*. Este módulo escreve dados em um arquivo XML, localizado em um computador remoto, enquanto o mesmo arquivo é baixado pela aplicação principal. Mais informações sobre este módulo podem ser encontradas no capítulo referente ao estudo de caso;

Após a passagem de todos os parâmetros definidos por *tags*, encerra-se a linha de comando com um último parâmetro. Trata-se do nome do arquivo de configuração do VR Juggler para essa aplicação. No arquivo de configuração estão dados referentes aos tamanhos das telas de projeção, dispositivos de interação, sincronismo do *cluster*, posição das imagens estereoscópicas, dentre outros.

No capítulo referente ao estudo de caso será mostrado um exemplo prático da utilização deste módulo.

4.7 MÓDULO PARA AQUISIÇÃO DE DADOS REAIS COM O PROTOCOLO MTCONNECT

O método para aquisição de dados reais baseado no protocolo MTConnect deve ser utilizado, principalmente, para máquinas-ferramenta e seus componentes. Para seu desenvolvimento, fez-se necessária a criação de outros módulos independentes de *software* e a definição dos tipos de dados de máquinas-ferramenta a serem adquiridos. Portanto, visando a simplificação de seu desenvolvimento, esse módulo foi dividido, e será explicado, em etapas:

- 1) Desenvolvimento do módulo adaptador de MTConnect;
- 2) Desenvolvimento do módulo agente MTConnect;
- 3) Desenvolvimento da aplicação cliente.

O fluxograma específico para o método de aquisição de dados reais via MTConnect é mostrado a seguir:

Figura 4-20- Fluxograma de aquisição de dados com MTConnect.

Como mostra o fluxograma acima, a aquisição de dados reais de máquinas-ferramenta necessita de dois módulos independentes, o adaptador e o agente. O desenvolvimento desses módulos é mostrado nos próximos tópicos.

4.7.1 Desenvolvimento do módulo adaptador

O MTConnect *Institute* mostra uma lista de passos iniciais a serem seguidos para o desenvolvimento de um *software* adaptador:

- Determinar quais dados devem ser adquiridos do dispositivo;
- Encontrar as *APIs* específicas do dispositivo;
- Implementar um adaptador ou modificar um *framework* de adaptador, específico para os dados estipulados como necessários, com base na *API* do dispositivo;
- Configurar um agente com os componentes do dispositivo e os itens de dado.

Cada passo descrito acima foi seguido para o desenvolvimento do estudo de caso desta tese.

Sobel (2012) detalha as principais responsabilidades de um módulo adaptador MTConnect:

1. Fornecer uma conexão *socket* onde uma aplicação possa se conectar para receber atualizações;
2. Enviar somente dados que foram atualizados;
3. Enviar somente dados em conformidade com o “vocabulário” estipulado;
4. Fornecer métodos de recuperação em caso de desconexão e suportar múltiplas conexões;
5. Enviar todos os valores iniciais para novos clientes, quando esses se conectarem;

6. Formatar os dados de acordo com a especificação SHDR descrita em Sobel (2012);
7. Realizar conexão com as interfaces de ferramentas e adquirir dados.

O *framework* fornecido pelo MTConnect *Institute* engloba todas as responsabilidades do adaptador, com exceção da última: realizar conexão com as interfaces de ferramentas e adquirir dados. Para implementar essa última responsabilidade são necessárias bibliotecas específicas de cada fabricante, para cada tipo de dispositivo: as *APIs*.

Com base nos adaptadores disponíveis no MTConnect *Institute* e nos trabalhos de Ferraz (2011), Ferraz (2007) e Ferraz (2002), estimou-se a estrutura genérica de um adaptador MTConnect, codificado em C++. O diagrama seguinte mostra os casos de uso para um adaptador MTConnect (Sobel, 2008).

Figura 4-21- Casos de uso para um adaptador MTConnect

O diagrama de casos de uso mostra que o adaptador (*Adapter*) tem duas responsabilidades principais, a normalização dos dados adquiridos (*Normalize Data*), para que um agente possa interpretá-los corretamente e a aquisição de dados (*Gather Data*) de uma fonte (*Data Source*), geralmente um controlador, sensor ou barramento de dados. Além disso, o diagrama de casos de uso mostra que um adaptador comunica-se, também, com um módulo agente (*Agent*). O diagrama a seguir mostra a relação temporal das atividades entre um adaptador e um agente (Sobel 2008).

Figura 4-22- Diagrama de sequência para o adaptador

O adaptador aguarda constantemente a conexão de clientes (agentes). Somente após a conexão de um agente, o adaptador executa o processo de tentativa de conexão com o dispositivo e, em caso de sucesso, começa a enviar dados para os clientes. Por fim, quando um agente se desconecta do adaptador, este se desconecta do dispositivo, para reduzir sua carga.

As responsabilidades de um adaptador implicam em alguns pré-requisitos de execução, como:

1. Aquisição exclusiva de dados novos, visando a economia de recursos;
2. Utilização exclusiva de *APIs* que correspondam aos elementos do padrão MTConnect;
3. Definição adequada da taxa de amostragem, a fim de aumentar a eficiência;
4. Criação de um servidor de dados para a conexão dos clientes (agentes);
5. Quando o dado em questão for um evento (*event*), como, por exemplo, um alarme, a normalização dos dados torna-se ainda mais importante, traduzindo-a de sua forma nativa para o vocabulário MTConnect;
6. Não é responsabilidade do adaptador a conversão de unidades. O agente converterá as unidades para as consultas da aplicação.

O diagrama de classes e relações detalha as responsabilidades de um módulo adaptador MTConnect, como mostrado, a seguir, por Sobel (2008).

Figura 4-23- Diagrama de classes de um adaptador

Um módulo adaptador realiza a leitura de dados de uma fonte, como, por exemplo, um controlador, e disponibiliza esses dados em uma comunicação *socket*. Os módulos agentes são clientes para um adaptador. Da perspectiva de um agente, o adaptador é uma aplicação *read-only* (somente leitura), pois um adaptador não pode receber requisições de aplicações externas. Um adaptador é responsável apenas por um único dispositivo, enquanto um agente pode receber dados de diversos adaptadores, ao mesmo tempo.

Internamente, um adaptador deve conter uma classe responsável pela detecção de alteração de dados, armazenamento dos novos dados e, por fim, alteração da *flag* `mChanged`. No diagrama de classes mostrado acima, a classe responsável por essas tarefas é chamada de *DeviceDatum*. Ainda, um adaptador deve conter, também, outra classe responsável pela monitoração de clientes, empacotamento de dados, estabelecimento da taxa de aquisição de

dados, criação do *socket*, monitoramento de mudanças, disponibilização de dados no *socket* e tratamento de erros de comunicação com clientes e fontes de dados. No diagrama de classes mostrado acima, a classe responsável por essas tarefas é chamada simplesmente de *Adapter*.

As classes *Sample* (Amostra), *Event* (Evento), *Power* (Força) e *Alarm* (Alerta) são subclasses de *DeviceDatum*, que é uma classe abstrata, portanto não pode ser instanciada diretamente. Portanto, os dados serão sempre considerados amostras, eventos, dados de energia ou alertas.

A estrutura mínima de um adaptador, ainda sem conexão com dispositivos, apenas alterna o valor de *Power*, mas é importante para a compreensão e orientação para a produção de adaptadores mais complexos. O quadro a seguir mostra o arquivo de cabeçalho do adaptador mínimo.

Quadro 4-3- Estrutura mínima de um adaptador

```
class FakeAdapter : public Adapter
{
    protected:
        Power mPower;

    public:
        FakeAdapter(int aPort);
        ~FakeAdapter();
        virtual void gatherDeviceData();
};
```

O método *gatherDeviceData* é obrigatório para quaisquer adaptadores.

O arquivo de implementações referente ao arquivo de declarações, mostrado acima, pode ser visto a seguir.

Quadro 4-4- Implementações do adaptador mínimo

```

#include "internal.hpp"
#include "fake_adapter.hpp"
#include "server.hpp"
#include "string_buffer.hpp"

FakeAdapter::FakeAdapter(int aPort): Adapter(aPort), mPower("power")
{
    addDatum(mPower);
}

FakeAdapter::~FakeAdapter()
{
}

void FakeAdapter::gatherDeviceData()
{
    if (!mPower.setValue(Power::eON))
        mPower.setValue(Power::eOFF);
    sleep(5);
}

int main(int aArgc, char *aArgv[])
{
    int port = 7878;
    if (aArgc > 1)
        port = atoi(aArgv[1]);
    FakeAdapter adapter(port);
    adapter.startServer();
    return 0;
}

```

Cada dado deve ser criado e nomeado, como mostra *mPower("power")* e, dentro do construtor, deve ser adicionado à lista de dados, através do método *addDatum*. Isso é necessário para todos os adaptadores e tipos de dados a serem coletados. Por sua vez, o método *gatherDeviceData* alterna o valor de *mPower* entre *ON* e *OFF* a cada cinco segundos.

A função principal, *main*, define o número da porta a ser utilizada para comunicação, seja a porta padrão, número 7878 ou definida por parâmetros da linha de comando. A função *startServer* inicializa um *looping* de execução infinito, criando um servidor e acessando o dispositivo com frequência pré-definida.

Após a compreensão da estrutura mínima de um adaptador, pode-se alterar o código para realizar a comunicação com um dispositivo real, no caso do exemplo, que possua

apenas dados de energia. Para isso, deve-se alterar o conteúdo de *gatherDeviceData* com os métodos adequados, fornecidos pela *API* do dispositivo em questão.

O exemplo, a seguir, mostra a forma mais completa de um adaptador MTConnect e foi utilizado no estudo de caso para produção de um adaptador específico para a marca Fanuc. Ele é baseado no EMC2, um controlador CNC com código fonte aberto, que foi adaptado para simular uma fresadora de três eixos. Os dados produzidos são os seguintes:

- *Alarms* (Alertas);
- *Power status* (Estado da energia);
- *Execution state* (Estado de execução);
- *Line number in the current program* (Número da linha no programa atual);
- *The name of the program being run* (Nome do programa atual);
- *The controller's mode* (*auto*, *MDI*, *manual* – Modo do controlador – Automático, MDI ou manual);
- *X, Y, e Z actual positions* (Posições reais dos eixos X, Y e Z);
- *X, Y, e Z commanded positions* (Posições ordenadas dos eixos X, Y e Z);
- *Spindle speed* (Velocidade de rotação) e
- *Path feederate* (Avanço da trajetória).

A lista acima é a lista de dados mais comumente captada de um dispositivo. O quadro mostrado a seguir exibe o arquivo de cabeçalho de um adaptador, específico para o controlador EMC.

Quadro 4-5- Arquivo de cabeçalho para o controlador EMC

```

class EmcAdapter : public Adapter
{
protected:
    Alarm mAlarm;
    Power mPower;
    Execution mExecution;
    IntEvent mLine;
    Event mProgram;
    ControllerMode mMode;
    Sample mXact;
    Sample mYact;
    Sample mZact;
    Sample mXcom;
    Sample mYcom;
    Sample mZcom;
    Sample mSpindleSpeed;
    Sample mPathFeedrate;

    EMC_STAT mEmcStatus;
    NML *mEmcErrorBuffer;
    char mErrorString[LINELLEN];
    RCS_STAT_CHANNEL *mEmcStatusBuffer;
    bool mConnected;
    char mNm1File[1024];

    bool connect();
    void disconnect();
    void actual();
    void commanded();
    void spindle();
    void feedrate();
    void program();
    void machine();
    void execution();
    void alarms();

public:
    EmcAdapter(int aPort, const char *aNm1File);
    ~EmcAdapter();
    virtual void gatherDeviceData();
};

```

O primeiro bloco de variáveis protegidas refere-se a objetos que armazenam os valores adquiridos. O segundo bloco refere-se às variáveis específicas da *API* do EMC, que são utilizadas para aquisição dos dados do dispositivo. Os métodos protegidos coletam informações específicas do dispositivo. A interface pública contém apenas uma diferença com relação ao adaptador mínimo. Faz-se necessário para o adaptador EMC que o usuário mostre o arquivo de configuração utilizado para inicializar a *API*.

Com relação à implementação, o construtor da classe contém as instruções mostradas no quadro seguinte.

Quadro 4-6- Construtor do adaptador EMC

```

EmcAdapter::EmcAdapter(int aPort, const char *aNmlFile): Adapter
(aPort),
mAlarm("alarm"), mPower("power"), mExecution("execution"),
mLine("line"), mXact("Xact"), mYact("Yact"), mZact("Zact"),
mXcom("Xcom"), mYcom("Ycom"), mZcom("Zcom"),
mSpindleSpeed("spindle_speed"), mPathFeedrate("path_feedrate"),
mProgram("program"), mMode("mode")
{
    addDatum(mAlarm);
    addDatum(mPower);
    addDatum(mExecution);
    addDatum(mLine);
    addDatum(mXact);
    addDatum(mYact);
    addDatum(mZact);
    addDatum(mXcom);
    addDatum(mYcom);
    addDatum(mZcom);
    addDatum(mSpindleSpeed);
    addDatum(mPathFeedrate);
    addDatum(mProgram);
    addDatum(mMode);

    mErrorString[0] = 0;
    mEmcErrorBuffer = mEmcStatusBuffer = 0;
    strcpy(mNmlFile, aNmlFile);
    mConnected = false;
}

```

Todos os tipos de dados são criados, nomeados e, em seguida, colocados na lista de dados do adaptador, que suporta até 128 tipos de dados. Por fim, as variáveis específicas da *API* são inicializadas e o arquivo de configuração é lido.

Assim como no adaptador mínimo, a aquisição de dados é realizada através do método *gatherDeviceData*, mostrado a seguir.

Quadro 4-7- Método para aquisição de dados do controlador EMC

```

void EmcAdapter::gatherDeviceData()
{
    if (!mConnected)
    {
        if (!connect())
            sleep(5);
    }
    else
    {
        if (!mEmcStatusBuffer->valid())
        {
            disconnect();
            return;
        }
        if (mEmcStatusBuffer->peek() == EMC_STAT_TYPE)
        {
            memcpy(&mEmcStatus, mEmcStatusBuffer->get_address(),
                sizeof(EMC_STAT));
            actual();
            commanded();
            spindle();
            program();
            machine();
            execution();
            alarms();
        }
    }
}

```

A primeira tarefa do método é checar se existe conexão ativa. Enquanto não houver conexão ativa, as únicas tarefas realizadas pelo método são as tentativas de conexão a cada cinco segundos. Assim que a conexão for estabelecida, o método testa se essa é uma conexão válida. E, em caso positivo, mantém-se conectado. Em seguida, o método realiza a cópia de informações da memória do dispositivo para a aplicação e adquire os dados específicos.

A conexão é realizada através do código descrito na próxima figura. O valor retornado é *true* para conexão com sucesso ou *false* para falha na conexão.

Quadro 4-8- Método de conexão com o EMC

```

bool EmcAdapter::connect()
{
    int retval = 0;
    if (mEmcStatusBuffer == 0)
    {
        mEmcStatusBuffer = new RCS_STAT_CHANNEL(emcFormat,
"emcStatus",
"adapter", mNm1File);
        if (! mEmcStatusBuffer->valid() || EMC_STAT_TYPE !=
mEmcStatusBuffer-
>peek())
        {
            rcs_print_error("emcStatus buffer not available\n");
            delete mEmcStatusBuffer;

            mEmcStatusBuffer = 0;
            mPower.setValue(Power::eOFF);
            return false;
        }
    }

    if (mEmcErrorBuffer == 0)
    {
        mEmcErrorBuffer = new NML(nmlErrorFormat, "emcError",
"adapter",
mNm1File);
        if (!mEmcErrorBuffer->valid())
        {
            rcs_print_error("emcError buffer not available\n");
            delete mEmcErrorBuffer;
            mEmcErrorBuffer = 0;
            mPower.setValue(Power::eOFF);
            return false;
        }
    }
    mConnected = true;
    return true;
}

```

Existe um método simples para desconexão que realiza a operação com segurança, liberando todos os *buffers*.

Quadro 4-9- Método de desconexão com o EMC

```

void EmcAdapter::disconnect()
{
    if (mConnected)
    {
        if (mEmcErrorBuffer)
            delete mEmcErrorBuffer;
        mEmcErrorBuffer = 0;

        if (mEmcStatusBuffer)
            delete mEmcStatusBuffer;
        mEmcStatusBuffer = 0;
        mConnected = false;
    }
}

```

O método que adquire as posições X, Y e Z dos eixos é mostrado pelo quadro a seguir.

Quadro 4-10- Aquisição das posições dos eixos

```

void EmcAdapter::actual()
{
    EmcPose &pose = mEmcStatus.motion.traj.actualPosition;
    mXact.setValue(pose.tran.x);
    mYact.setValue(pose.tran.y);
    mZact.setValue(pose.tran.z);
}

```

A classe *DeviceDatum* verifica as alterações, portanto essa verificação não precisa ser realizada nos métodos de aquisição específicos.

O método a seguir verifica se o dispositivo está *ON* (ligado) ou *OFF* (desligado), criando um alerta para viabilizar o envio da informação. Essa forma de envio de dados pode ser utilizada com qualquer informação, não apenas com o estado do dispositivo.

Quadro 4-11- Verificação do estado do dispositivo

```

void EmcAdapter::machine()
{
    if (mEmcStatus.task.state == EMC_TASK_STATE_ON)
    {
        if (mPower.setValue(Power::eON))
        {
            Alarm warn("alarm");
            warn.setValue(Alarm::eMESSAGE, "ON", Alarm::eINFO,
                Alarm::eINSTANT, "Power ON");
            sendDatum(&warn);
        }
    }
    else
    {
        if (mPower.setValue(Power::eOFF))
        {
            Alarm warn("alarm");
            warn.setValue(Alarm::eMESSAGE, "OFF", Alarm::eINFO,
                Alarm::eINSTANT, "Power OFF");
            sendDatum(&warn);
        }
    }
}

```

O estado da execução do dispositivo EMC pode ser adquirido através da seguinte função:

Quadro 4-12- Aquisição do estado da execução do dispositivo EMC

```

void EmcAdapter::execution()
{
    if (mEmcStatus.task.mode == EMC_TASK_MODE_MANUAL)
        mMode.setValue(ControllerMode::eMANUAL);
    else if (mEmcStatus.task.mode == EMC_TASK_MODE_AUTO)
        mMode.setValue(ControllerMode::eAUTOMATIC);
    else if (mEmcStatus.task.mode == EMC_TASK_MODE_MDI)
        mMode.setValue(ControllerMode::eMANUAL_DATA_INPUT);

    if (mEmcStatus.task.interpState == EMC_TASK_INTERP_PAUSED)
        mExecution.setValue(Execution::ePAUSED);
    else if (mEmcStatus.task.interpState == EMC_TASK_INTERP_IDLE)
        mExecution.setValue(Execution::eIDLE);
    else if (mEmcStatus.task.interpState == EMC_TASK_INTERP_WAITING)
        mExecution.setValue(Execution::eEXECUTING);
}

```

O restante dos métodos para aquisição de dados segue o mesmo padrão, descrito acima. Por fim, o destrutor do adaptador apenas garante desconexão adequada.

Quadro 4-13- Destrutor do adaptador para EMC

```
EmcAdapter::~~EmcAdapter()  
{  
    disconnect();  
}
```

Além da estrutura do adaptador específico para EMC, descrita acima, este trabalho fornece detalhes sobre toda a estrutura interna genérica de um adaptador, bem como exemplos de dados adquiridos. Essas informações podem ser encontradas no Apêndice B.

4.7.2 Desenvolvimento do módulo agente

O módulo adaptador tem a função de traduzir a linguagem específica de cada dispositivo, para uma linguagem comum. Após a aquisição, essas informações são enviadas para outro módulo, chamado agente, que compreende a linguagem comum do adaptador. Portanto, as tarefas desempenhadas pelo *software* agente MTConnect tomam como dados de entrada os dados fornecidos pelo adaptador MTConnect, como mostra a figura a seguir.

Figura 4-24- Conversão de dados do agente MTConnect.

Uma das tarefas principais do *software* agente é a formatação dos dados provenientes do adaptador em arquivos XML, para que quaisquer aplicações possam ler e interpretar corretamente essas informações. Além disso, o *software* agente deve manter um *buffer* dimensionável de informações para que seja possível alimentar a aplicação cliente, quando essa solicitar novos dados. O agente MTConnect é composto, também, por uma interface HTTP que recebe diversos tipos diferentes de requisições provenientes de aplicações clientes.

O propósito do agente é responder às solicitações descritas na seguinte tabela.

Tabela 4-1- Solicitações aceitas pelo agente

Comando	Argumentos	Respostas	Representação
<i>probe</i>	nenhum	Dados que descrevem os dados apresentáveis da máquina	<i>Devices</i> <i>Device</i> <i>Components</i> <i>DataItems</i>
<i>current</i>	<i>path</i>	Retorna somente itens do <i>probe</i> que façam parte de <i>path</i> . Caso o <i>path</i> seja suprimido, retorna todos os itens	<i>Streams</i> <i>DeviceStream</i> <i>ComponentStream</i> <i>Events</i> <i>Samples</i>
<i>sample</i>	<i>path</i>	Funciona como em <i>current</i>	
<i>sample</i>	<i>start</i>	Começa a sequência com o número do argumento	
<i>sample</i>	<i>count</i>	Define o número máximo de <i>samples</i> a retornar (se omitido, o <i>software</i> agente assume a responsabilidade)	

A utilização do agente faz-se através de linha de comando, da seguinte maneira:

Quadro 4-14- Inicialização do agente

```
agent [help|install|debug|run] [configuration_file]
```

Onde:

- “*help*” imprime a mensagem de ajuda;
- “*install*” instala o serviço;
- “*remove*” remove o service;
- “*debug*” executa o agente na linha de comando com log detalhado;
- “*run*” executa o agente na linha de comando;
- “*config_file*” nome do arquivo de configuração a ser lido pelo agente (arquivo padrão: “agent.cfg” no diretório corrente).

Quando o agente é executado sem passagem de parâmetros, é assumido que ele será executado como um serviço, começando assim a sequência de inicialização de serviço. O caminho completo para o arquivo de configuração gravado no registro é:

Quadro 4-15- Caminho completo para o arquivo de configuração

```
\\HKEY_LOCAL_MACHINE\\SOFTWARE\\MTConnect\\  
MTConnectAgent\\ConfigurationFile
```

Para esta pesquisa utilizou-se o agente MTConnect, adquirido do MTConnect Institute ([github](#)).

O agente MTConnect armazena uma quantidade fixa de dados. Esse armazenamento pode ser mais bem explicado através de uma analogia simples com um tubo, onde dados são colocados em uma extremidade, enquanto dados saem por outra. Isto é, quando o tubo encontra-se cheio, a colocação de um novo dado em uma extremidade implica na perda do dado mais antigo, que sai pela outra extremidade, como mostra a figura a seguir.

Figura 4-25- Armazenamento de dados do agente MTConnect

Para viabilizar a forma de armazenamento de dados descrita acima, cada dado colocado no tubo é numerado sequencialmente e variáveis de apoio, chamadas *firstSequence* e *lastSequence*, informam uma aplicação quais os dados armazenados em determinado momento. A capacidade máxima de armazenamento do agente é definida através de seu parâmetro *bufferSize*.

Quando um cliente realiza uma requisição ao agente, ele pode, ou não, especificar parâmetros auxiliares para restringir a abrangência de sua consulta. Os parâmetros *from* e

count são utilizados para a definição do número de sequência inicial, de interesse da aplicação, e a contagem de quantidade de dados de interesse, respectivamente. A próxima figura ilustra esse formato de requisição.

Figura 4-26- Captação de dados com parâmetros *from* e *count*

O exemplo acima mostra, inicialmente, uma solicitação de três dados a partir do número de sequência 15. Como consequência, o parâmetro *nextSequence* tem seu valor automaticamente alterado para o próximo número de sequência, ou seja, 18.

Caso uma solicitação abranja itens que não estejam dentro do tubo, como mostrado na segunda parte do exemplo acima, o parâmetro *nextSequence* tem seu valor modificado para *lastSequence+1*.

Ainda sobre o exemplo anterior, uma solicitação *sample* não pode retornar o valor do dado com número de sequência 11, que foi removido do tubo. Porém, uma solicitação *current* ainda pode retornar o valor desse dado.

A figura a seguir exemplifica uma aquisição de dados de um agente.

Seq	Time	Data Item	Value
101	2007-12-13T09:44:00.0221	Availability	UNAVAILABLE
102	2007-12-13T09:54:00.4412	Availability	AVAILABLE
103	2007-12-13T10:00:00.0002	Position Y	25
104	2007-12-13T10:00:00.0002	Position Z	1
105	2007-12-13T10:00:00.0002	Spindle Speed	0
106	2007-12-13T10:01:02.0012	Position X	11
107	2007-12-13T10:01:02.0012	Position Y	24
108	2007-12-13T10:01:02.0012	Position Z	1.1
109	2007-12-13T10:01:04.0012	Spindle Speed	1000
110	2007-12-13T10:01:04.5012	Position X	12
111	2007-12-13T10:01:04.5012	Position Y	23
112	2007-12-13T10:01:04.5012	Position Z	1.2
113	2007-12-13T10:01:05.5012	Position X	13
114	2007-12-13T10:01:05.5012	Position Y	22
115	2007-12-13T10:01:06.5012	Position X	14
116	2007-12-13T10:01:06.9012	Position Y	22
117	2007-12-13T10:01:07.0001	Position X	14
118	2007-12-13T10:01:07.0001	Position Z	1.3
119	2007-12-13T10:01:07.5001	Position X	15
120	2007-12-13T10:01:07.5001	Position Y	21
121	2007-12-13T10:01:07.5001	Position Z	1.4
122	2007-12-13T10:01:08.9012	Spindle Speed	0
123	2007-12-13T10:01:09.9012	Position X	10
124	2007-12-13T10:01:09.9012	Position Y	15
125	2007-12-13T10:01:09.9012	Position Z	0
126	2007-12-13T10:01:12.9012	Availability	UNAVAILABLE

Figura 4-27- Dados adquiridos de um agente

A figura a seguir mostra a janela de aquisição de dados de um agente.

Figura 4-28- Janela de aquisição de dados de um agente

O Agente precisa ser executado em modo texto, com parâmetro "-c <nome do arquivo>", que indica o nome do arquivo de configuração do Adaptador. O arquivo de configuração tem a estrutura mostrada no quadro a seguir:

Quadro 4-16- Estrutura do arquivo de configuração do Adaptador

<nome do dispositivo>:<endereço IP>:<porta de comunicação>
--

4.8 MÓDULO PARA AQUISIÇÃO DE DADOS REAIS COM O USO DE CÂMERAS

O módulo proposto e descrito no tópico anterior viabiliza a simulação virtual de dados reais *on-line* para determinados tipos de dados de máquinas-ferramenta. Porém, alguns dados de chão de fábrica, sejam eles provenientes de máquinas-ferramenta ou não, não podem ser adquiridos via MTConnect, como é pré-requisito do módulo anterior. A viabilização da aquisição *on-line* desses dados contribuiria para a veracidade de simulações virtuais de manufatura. Alguns desses dados são listados a seguir:

- Dados internos de máquinas-ferramenta:
 - Posição contínua de portas de dispositivos;
 - O cálculo automático do ponto zero da ferramenta em caso de câmeras com alta resolução;
- Dados de outros itens de chão de fábrica:
 - Posicionamento de um braço robótico;
 - Posicionamento de transportadores;
 - Posicionamento de operadores.

Este tópico apresenta um método para aquisição desses tipos de dados através da utilização de câmeras de baixo custo, como *webcams*.

O estabelecimento do ponto zero da ferramenta deve ser feito antes, ou mesmo durante o processo de fabricação de uma peça, quando a ferramenta se desgasta. Trata-se de um processo frequente e demorado, que, se captado externamente, pode otimizar a produção.

Porém, a precisão desse processo é extremamente importante e pode tornar inviável sua aquisição via câmeras de baixo custo, como é a proposta deste trabalho.

As máquinas-ferramenta, seus CNCs e o protocolo MTConnect não estão preparados, também, para a captação de dados contínuos das portas dos dispositivos. As informações relativas às portas são discretas: abertas, fechadas ou destravadas. A utilização de câmeras de baixo custo para captação dos dados das portas poderia aprimorar o posicionamento de uma porta, dentre de aplicações de realidade virtual. Essa tarefa utilizaria apenas uma câmera para captura de dados apenas no eixo linear X.

O posicionamento tridimensional de um braço robótico em uma aplicação virtual é uma tarefa mais complexa, porém poderia valer-se de câmeras para viabilização da aquisição de dados de posicionamento, para dispositivos que não forneçam esses dados originalmente.

O posicionamento de esteiras e operadores pode ser importante para simulações de realidade virtual que utilizem dados *on-line*. Esse tipo de dado pode ser adquirido através de câmeras de baixo custo e pode contribuir para a veracidade das simulações.

Este tópico descreve um módulo adicional para obtenção de informações baseado em imagens de vídeo, que pode auxiliar qualquer uma das situações descritas acima, bem como diversas outras situações de trabalho da área de manufatura e outras áreas de aplicação com interesse em simulações virtuais.

O algoritmo de segmentação utilizado baseia-se em cores, portanto o módulo de aquisição de imagens por câmera é baseado na utilização de marcadores RGB (*Red*, *Green*, *Blue*) acoplados a pontos estratégicos da ferramenta, dispositivo ou demais itens móveis de chão de fábrica. A segmentação de imagens possibilita a transformação desses marcadores em dados de posicionamento. A utilização de marcadores visa a simplificação da segmentação de imagens de vídeo. A figura a seguir mostra um exemplo com marcadores coloridos circulares:

Figura 4-29- Marcadores coloridos circulares

Para a implementação deste módulo verificou-se que câmeras de baixo custo captam com mais fidelidade determinadas cores, principalmente cores próximas às cores naturais RGB, vermelho, verde e azul. A segmentação de imagens fornece, consequentemente, resultados melhores quando usam como dados de entrada imagens compostas por essas cores. Portanto recomenda-se a utilização de marcadores com cor única e mais próxima possível das cores naturais de RGB ou suas primeiras intersecções, como mostra a figura a seguir.

Figura 4-30- Cores naturais RGB e suas primeiras intersecções

Os marcadores devem ser acoplados aos dispositivos de maneira que sua fixação não altere o funcionamento do dispositivo, nem a superfície colorida voltada para a câmera. Preferencialmente, os marcadores e câmeras devem ser posicionados ortogonalmente, para facilitar a segmentação das imagens.

O material de composição dos marcadores pode ser plástico, papel ou mesmo luminoso. Quanto mais "pura" a cor RGB do marcador, melhor será seu reconhecimento e processamento no computador.

O formato dos marcadores pode ser utilizado no algoritmo de segmentação para aprimorar o reconhecimento das imagens, porém essa tarefa não é imprescindível. Nesta metodologia, o formato dos marcadores não foi levado em consideração na construção do algoritmo de segmentação.

Câmeras de baixo custo, como *webcams*, podem ser utilizadas como ferramentas desta parte da metodologia. Para a viabilização de captura de posicionamento de apenas duas dimensões, a utilização de uma câmera é suficiente. Porém, para captura de dados em três dimensões, são necessárias duas câmeras, uma para os dados do eixo X e Y (largura e altura) e outra específica para o dado de profundidade ou, uma câmera para os eixos X e Z e outra

para o eixo Y. As figuras a seguir ilustram arranjos de câmeras para a captação de dados de chão de fábrica, começando pelos arranjos simples de uma câmera para captação de um ou dois eixos de movimentação.

Figura 4-31- Arranjo com uma câmera para captação de 1 eixo para porta de dispositivo

Figura 4-32- Arranjo com uma câmera para captação de 1 eixo para esteira

Figura 4-33- Exemplo fictício de braço robótico com marcadores.

A biblioteca aberta *OpenCV* foi utilizada tanto para captura, quanto para segmentação das imagens, que são processadas no computador local, a fim de minimizar o envio de dados pela Internet, para a CAVE.

Como a segmentação das imagens é realizada com relação à cor, as cores dos marcadores devem contrastar ao máximo com as demais cores do equipamento onde serão acoplados. A iluminação do ambiente deve ser suficientemente grande para a correta captação das imagens pela câmera de baixo custo. Iluminações ruins tornam as imagens granuladas e difíceis de serem segmentadas.

Figura 4-34- Exemplo de imagem com ruído

A figura a seguir mostra os principais componentes do algoritmo utilizado e sintetiza sua forma de execução.

Figura 4-35- Passos do algoritmo de segmentação.

Apesar da utilização de marcadores RGB, o algoritmo utilizado baseia-se no espaço de cor HSV (*Hue Saturation Value*), onde *Hue* define uma cor, *Saturation* define a quantidade dessa cor e *Value* ou *Intensity* define o brilho dessa cor. Através desses valores, pode-se calibrar o valor exato da cor do marcador, capturada pela *webcam*, para que o algoritmo de segmentação possa prover melhores resultados. Essa calibragem pode ser feita manualmente e os valores obtidos podem ser colocados na aplicação através do módulo de criação de parâmetros.

Primeiramente, adquire-se a imagem disponível através do método *cvCaptureFromCAM*, em seguida, cria-se uma janela com o método *cvNamedWindow* para a disponibilização dos *frames* capturados. Para a aquisição dos *frames* utiliza-se o método *cvQueryFrame* e para a exibição dos *frames* na janela criada, utiliza-se o método

cvShowImage. Foram definidas variáveis inteiras de *threshold* (limiar), utilizadas no procedimento de segmentação de cor. Para isso, utilizou-se os métodos *cvCvtColor* e *cvInRangeS*, respectivamente, para conversão da imagem em HSV e aquisição de imagem binária apenas com marcador segmentado em branco e *background*, em preto. Nesse método, todos os pixels da imagem são percorridos, da esquerda para a direita, com base nos valores de *height* e *width* da imagem, partindo de seu canto superior esquerdo. Cada um desses pixels possui três campos de dados, HSV, que possuem seus valores comparados com o *threshold* definido, a fim da montagem de uma imagem binária.

Uma imagem binária é composta apenas por pixels pretos e brancos, onde, no caso deste algoritmo, os pixels brancos informam a posição do marcador e os pixels pretos informam pixels descartáveis, de *background*. A criação dessa imagem binária é um passo intermediário do resultado principal deste algoritmo e, como parte deste processo, o algoritmo compara os três campos HSV de cada pixel com um limiar pré-definido, em busca de pixels que contenham informação de marcadores. O limiar definido para os testes deste trabalho foi o valor 50 para o *Threshold*. Todos os pixels que estiverem dentro do limiar serão considerados como parte do marcador e, conseqüentemente, terão o valor de seus três campos de cor alterados para 255, de forma que sejam representados pela cor branca. De maneira análoga, os pixels que não estiverem dentro do limiar, terão seus valores de cor alterados para 0, de forma a serem representados pela cor preta.

Após a finalização da busca, o algoritmo possui como resultado intermediário, uma imagem binária da forma ilustrada pela figura seguinte.

Figura 4-36- Exemplo de imagem binária

Em seguida, o algoritmo realiza a interligação de pontos vizinhos visando a construção de figuras fechadas. Como o algoritmo considera que haja apenas um marcador para cada câmera, se houver mais de uma região que resulte em uma figura fechada, o algoritmo ignora as regiões menores e assume que a maior figura fechada seja o marcador RGB acoplado a uma parte móvel.

Figura 4-37- Exemplo de figura binária fechada única

Em seguida, encontram-se as bordas da imagem através do detector de bordas de Canny e, por fim, o algoritmo encontra o centroide da figura em posições de pixel X e Y e armazena-o a fim de enviar suas coordenadas pela Internet.

Figura 4-38- Centroide da figura binária única

No início do desenvolvimento deste módulo, os dados eram enviados pela rede e recebidos no nó mestre (*Front Wall*) da CAVE, através de um *Socket* de comunicação Cliente-Servidor, desenvolvido em C++. As informações eram recebidas em um *Buffer*, onde as posições X, Y e Z eram armazenadas. No entanto, como esta metodologia utiliza arquivos XML MTConnect para armazenamento das demais informações, optou-se por armazenar, também, dados de posicionamento provenientes de câmera, em arquivos XML MTConnect. Essa padronização de armazenamento proporciona maior flexibilidade e simplicidade da implementação da aplicação principal, que pode reutilizar procedimentos de aquisição de dados, *download* de arquivos etc.

Os dados de posicionamento X e Y adquiridos pela câmera são armazenados em arquivos XML com estrutura *MTConnectStreams* para dois eixos. A figura a seguir exemplifica esse armazenamento.

```

▼<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.2"
xmlns="urn:mtconnect.org:MTConnectStreams:1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.2
http://www.mtconnect.org/schemas/MTConnectStreams_1.2.xsd">
  <Header creationTime="2012-07-17T02:45:10Z" sender="mtconnect" instanceId="1340212647"
  version="1.2.0.10" bufferSize="131072" nextSequence="446686424" firstSequence="446555352"
  lastSequence="446686423"/>
  ▼<Streams>
    ▼<DeviceStream name="Camera" uuid="000">
      ▼<ComponentStream component="Linear" name="X" componentId="x1">
        ▼<Samples>
          <Position dataItemId="x2" timestamp="2012-07-17T02:45:09.611574" name="Xact"
          sequence="446686118" subType="ACTUAL">-0.7211295366</Position>
          <Position dataItemId="x2" timestamp="2012-07-17T02:45:09.623574" name="Xact"
          sequence="446686122" subType="ACTUAL">-0.7187634110</Position>
          <Position dataItemId="x2" timestamp="2012-07-17T02:45:09.639574" name="Xact"
          sequence="446686126" subType="ACTUAL">-0.7163973451</Position>
        </Samples>
      </ComponentStream>
      ▼<ComponentStream component="Linear" name="Y" componentId="y1">
        ▼<Samples>
          <Position dataItemId="y2" timestamp="2012-07-17T02:45:09.611574" name="Yact"
          sequence="446686119" subType="ACTUAL">0.3617376685</Position>
          <Position dataItemId="y2" timestamp="2012-07-17T02:45:09.623574" name="Yact"
          sequence="446686123" subType="ACTUAL">0.3659139872</Position>
          <Position dataItemId="y2" timestamp="2012-07-17T02:45:09.639574" name="Yact"
          sequence="446686127" subType="ACTUAL">0.3700902760</Position>
        </Samples>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>

```

Figura 4-39- Arquivo criado pelo módulo de aquisição de dados por câmera

Dessa forma, o procedimento de aquisição e utilização dos dados de componentes móveis adquiridos via câmeras é o mesmo procedimento criado para aquisição e utilização de dados MTConnect. No momento da criação de um item da aplicação virtual que utilizará os dados reais *on-line* de câmeras, deve-se apenas informar que este item é do tipo *dev_cam*, da classe *Device_Cam*, para que a aplicação instancie um dispositivo que utilize apenas dados de um, dois ou três eixos. A quantidade de eixos é definida dinamicamente com base nas informações contidas no arquivo XML. Se o arquivo possuir apenas informações do eixo X, a aplicação utilizará apenas os dados de posicionamento do eixo X do componente. Caso o arquivo XML contenha informações de dois eixos, X e Y, a aplicação atualizará o

posicionamento do objeto nos eixos X e Y. No caso de três eixos, são utilizados dois arquivos XML provenientes de computadores e câmeras diferentes.

4.9 ESTRUTURA DA APLICAÇÃO VIRTUAL

Este tópico descreve os pontos mais importantes da estrutura de programação e funcionamento da aplicação virtual. Todos os procedimentos utilizados, referentes às bibliotecas principais, *VR Juggler* e *OpenSG*, são abordados, bem como as funções criadas para comunicação de dados, instanciação de classes e utilização dos dados adquiridos pelos módulos externos.

4.9.1 Procedimento principal

Diversas classes compõem a aplicação, porém a classe *Shop_Floor* é a que possui as principais responsabilidades. Ela é instanciada uma única vez, logo no início do procedimento principal da aplicação. O resultante objeto de *Shop_Floor* é, na verdade, um objeto da classe de aplicação *OpenSG*, responsável por todos os objetos virtuais alocados no grafo de cena. Um objeto *OpenSG* é armazenado na memória, em uma variável de mesmo tipo, para que todos os objetos virtuais da cena possam ser acessados a qualquer momento, durante todo o processo de execução da aplicação.

Ainda dentro da função principal, o núcleo da aplicação *VR Juggler* é instanciado através da classe *vrj::Kernel* (Núcleo). Esse núcleo é responsável, entre outras tarefas, pelo controle do *looping* interno e procedimentos de inicialização da aplicação *VR Juggler*. O

Kernel é executado através de uma *Thread*, uma função que paraleliza a execução de outra função. A figura a seguir apresenta o diagrama de classes do *Kernel* do *software* VR Juggler.

Figura 4-40- Diagrama de classes do *Kernel* do VR Juggler

O arquivo de configuração é lido para dentro do *Kernel* e todos os parâmetros de execução são armazenados dentro de um vetor do tipo *string*, para serem processados no momento adequado. Nesse momento, faz-se importante a criação adequada da linha de comando, com parâmetros válidos, em uma ordem válida. Para garantir a inicialização correta da aplicação, foi criado um módulo específico de *software*, que auxilia o usuário a montar

corretamente os parâmetros da aplicação. Esse módulo é descrito em outro tópico deste capítulo. A estrutura de armazenamento dos parâmetros está exemplificada na figura a seguir:

Figura 4-41- Vetor de armazenamento dos parâmetros.

A primeira e a última informação da linha de comando (as únicas que não se encontram em vermelho na figura acima) são as únicas que não são armazenadas no vetor de parâmetros da aplicação, pois se referem ao arquivo executável e arquivo de configuração da aplicação, respectivamente. Todos os demais dados são armazenados, inicialmente, nesse vetor, para serem processados, em seguida, durante a execução do procedimento de inicialização da cena. Enquanto não são processados, os dados permanecem armazenados como *strings* de informação simples, mesmo os dados numéricos.

Toda a aplicação de realidade virtual, desenvolvida nesta metodologia, tomou como base a biblioteca *OpenSG*. Essa biblioteca fornece uma estrutura para criação e controle de grafos de cena, para objetos virtuais.

A fim de tornar intuitiva a nomenclatura das classes desta metodologia, esse trabalho se refere à classe *OpenSG* como *Shop_Floor*, pois essa parte da aplicação virtual se refere especificamente a dados de chão de fábrica. A classe *Shop_Floor* é a maior e mais

importante classe da aplicação e, também, a que acumula maior número de responsabilidades a desempenhar. Ela exerce funções nativas do *OpenSG*, como criação e controle do grafo de cena, além de todas as funções da metodologia que armazenam e controlam a relação entre os dispositivos e seus dados.

Estruturalmente, essa classe é composta por dois arquivos: um arquivo de cabeçalho chamado “*OpenSG.h*” e um arquivo de código fonte chamado “*OpenSG.cpp*”. A classe *Shop_Floor* é uma subclasse de *OpenSGApp* (aplicação de *OpenSG*), portanto deve implementar todos os métodos abstratos de sua classe base. A lista de métodos nativos de *OpenSGApp* que foram implementados nesta metodologia é mostrada a seguir:

- *void InitScene*
- *NodePtr getScene*
- *void preFrame* e
- *void exit*

O método *initScene* é o primeiro método da classe a ser executado. Ele é executado uma única vez e é responsável pela inicialização de todas as variáveis e instanciação de todos os objetos que devam estar na memória antes da cena virtual ser inicializada. Pode-se utilizar, também, o método de inicialização *init* utilizado pelas próprias funções de *OpenSG* e convocado antes mesmo de *initScene*, para inicializações não ligadas ao grafo de cena.

4.9.2 Funções do teclado

Para esta metodologia codifica-se como conteúdo destes métodos, entre outras coisas, a inicialização da interação via teclado, através da *string VJKeyboard*, configurada previamente no arquivo de configuração do VR Juggler.

```
void OpenSG::init() {
    // Call base class init
    vrj::OpenSGApp::init();
    mKeyboard.init("VJKeyboard");
}
```

Figura 4-42- *VJKeyboard* no procedimento *init* da biblioteca OpenSG.

Figura 4-43- *VJKeyboard* no arquivo de configuração do VR Juggler.

Uma das funcionalidades importantes da interação via teclado é a configuração dinâmica da velocidade de movimentação da cena proporcionada pelo dispositivo de interação por luz, desenvolvido no LMVI. Além disso, a interação via teclado proporciona uma forma

Como citado acima, a tecla “X” encerra a aplicação de maneira adequada. A leitura do teclado é realizada constantemente pela *Thread* responsável pela função *readKey*. Quando o usuário da aplicação aciona a tecla “X”, o procedimento *readKey* direciona a execução para a função de finalização do *Kernel* do VR Juggler: “*Kernel->stop()*”. Esse procedimento, por sua vez, executa o método *exit* de *OpenSG* e os demais métodos de encerramento do próprio VR Juggler. Dessa forma, todas as variáveis e objetos, criados pela aplicação, são liberados da memória adequadamente.

As teclas “R” e “T” são utilizadas como suporte à ferramenta de movimentação por luz, desenvolvida durante esta pesquisa. Essa ferramenta tem como funcionalidade principal proporcionar ao usuário uma forma de navegação pelo ambiente virtual. Inicialmente essa ferramenta foi construída e codificada de forma a proporcionar somente translação, porém, logo se sentiu a necessidade de aplicar rotação em cada um dos três eixos, para aprimorar a navegação na cena. Assim, foram implementadas, temporariamente, as funcionalidades das teclas “T” (*translation* ou translação) e “R” (*rotation* ou rotação), para proporcionar a troca de funcionalidade da ferramenta de interação por luz, de translação para rotação e vice-versa.

As teclas “R” e “T” controlam o valor de uma variável booleana, *rotationOption*, que define para o restante da aplicação, mais especificamente para o procedimento *moveObject*, qual a forma de interação que deve ser adotada em determinado momento. O valor da variável começa como *false*, ou seja, com a opção de translação ativa. Quando o usuário aciona a tecla “R” o valor desta variável se torna *true* e o procedimento de navegação entende que a partir deste momento a ferramenta de navegação deve aplicar rotações aos eixos. Quando o usuário aciona a tecla “T”, o valor dessa variável volta a ser *false* e o procedimento de navegação volta a aplicar matrizes de translações aos eixos.

Como mencionado acima, as teclas “+” e “-” são responsáveis pela quantidade de movimento que a ferramenta de interação por luz aplica na cena. Em outras palavras, essas teclas alteram a velocidade de navegação do usuário dentro do ambiente virtual. Elas são responsáveis pelo controle da variável do tipo *double* com nome *moveSpeed*. Essa variável controla a velocidade de navegação, tanto para translação, quanto para rotação da cena. Porém, os valores de *moveSpeed* são diferentes para cada uma dessas funcionalidades. Quando a funcionalidade *rotationOption* é desativada, *moveSpeed* é imediatamente inicializado com o valor *double* “16.0” e cada acionamento das teclas “+” e “-” dobra seu valor ou divide seu valor por dois. Quando *rotationOption* é ativada, *moveSpeed* é imediatamente inicializado com o valor *double* “1.0” e cada acionamento das teclas “+” e “-” multiplica seu valor por “1.5” ou divide seu valor por “1.5”, respectivamente. Essas constantes foram estipuladas como os valores mais adequados para as formas de interação da CAVE do LMVI. Em outros dispositivos, com outros computadores e ferramentas, pode ser que essas constantes devam ser alteradas.

As teclas numéricas “4” e “6”, “2” e “8” e “7” e “9” controlam variáveis que movimentam a cena, proporcionando navegação via teclado. Elas definem, respectivamente, o valor das variáveis do tipo *double* *distX*, *distY* e *distZ*. Essas variáveis, por sua vez, são utilizadas dentro do procedimento *moveObject* para translação ou rotação da cena.

Para que uma tecla possa ser lida a qualquer momento da execução da aplicação, uma *Thread* teve que ser criada especialmente para controlar o acionamento do teclado. Portanto o arquivo de cabeçalho de *Shop_Floor* guarda um objeto do tipo *Thread*, chamado *keyThread*. O valor desse objeto é inicializado dentro do procedimento *initScene*, disparando uma *Thread* que começa o processamento paralelo do método *readKey* com conteúdo alocado

dentro de um *looping* infinito a fim de proporcionar a execução paralela da *Thread* durante toda a execução da aplicação principal.

A *Thread* utilizada dentro de um programa VR Juggler deve pertencer à própria biblioteca do *software*, chamada “VR Juggler Portable Runtime” ou VPR. Essa biblioteca trabalha conjuntamente com a biblioteca *boost*, como demonstrado no quadro a seguir:

Quadro 4-17- *Thread* de VR Juggler e Boost

```
keyThread = new vpr::Thread(boost::bind(&OpenSG::readKey, this)) ;
```

Para o correto funcionamento das *Threads* internas do VR Juggler, a inclusão de *boost::bind* deve ser realizada dentro do arquivo de cabeçalho da classe principal: OpenSG.h.

O método *readKey* baseia-se no arquivo de cabeçalho *KeyEvent.h* e na classe *gadget::KeyboardMouse*, configurada dentro do arquivo de configuração externo do VR Juggler e em bibliotecas incluídas dentro do arquivo de cabeçalho da classe principal OpenSG.h.

A figura a seguir mostra a forma de configuração da interação via teclado em um arquivo externo do VR Juggler.

Figura 4-45- Configuração da interação via teclado no *VR Juggler*.

4.9.3 Inicialização do grafo de cena

A primeira tarefa do método *initScene*, relacionada com a criação do grafo de cena, é a criação da raiz do grafo de cena da aplicação. Para isso, o método executa uma função criada especificamente para a tarefa, chamada *createRoot*.

A função *createRoot* é uma função do tipo *NodePtr*, que retorna como resultado o endereço de um nó do grafo de cena e, seu conteúdo inclui:

- Criação de um novo nó, através de *Node::create*;
- Utilização de *setName* para nomenclatura do novo nó;
- Criação de um novo nó de transformação através de *Transform::create*;
- Interligação dos dois novos nós através de *setCore*;
- Por fim, retorno do resultado através de *return*.

Em OpenSG, toda operação de interligação de nós deve ser realizada com segurança, através de máscaras, como por exemplo a operação *setCore*, citada acima, deve ser executada da seguinte maneira:

Quadro 4-18- Máscaras de OpenSG

```
beginEditCP (root, Node::CoreFieldMask) ;
    root->setCore (rootCore) ;
endEditCP (root, Node::CoreFieldMask) ;
```

Os comandos *beginEditCP* e *endEditCP* informam ao processador o início e término de uma operação do tipo *Node::CoreFieldMask* (máscara de transformação de núcleo). Dessa forma, a operação é realizada mais rapidamente e erros específicos são tratados adequadamente. Nesse momento, o grafo de cena virtual da aplicação OpenSG encontra-se com seu primeiro nó, a raiz *root*, como mostrado na figura a seguir:

Figura 4-46- Grafo de cena somente com *root*.

4.9.4 Instanciação e relacionamento de dispositivos

Ainda dentro do procedimento *initScene*, logo após a criação da raiz do grafo de cena, o vetor de parâmetros, inicializado no procedimento *main*, é lido e processado. Os parâmetros não são processados diretamente no procedimento inicial, *main*, pois naquele momento a cena virtual da aplicação ainda não continha o grafo de cena, necessário para colocação dos nós de objetos virtuais. Os parâmetros alocados nesse vetor definem a quantidade, tipos e posicionamento de cada dispositivo dentro da cena virtual. Por sua vez, o relacionamento entre cada dispositivo não é definido por valor de parâmetros, mas sim pela ordem de entrada de cada parâmetro. Por exemplo, uma máquina instanciada recebe peças do transportador instanciada imediatamente antes de sua instancição e envia peças para o transportador instanciada imediatamente após sua criação. A figura a seguir ilustra a criação dos dispositivos e seus relacionamentos.

Figura 4-47- Criação e relação entre dispositivos na cena

A criação adequada da ordem de relacionamentos entre dispositivos deve ser realizada antes da inicialização da aplicação virtual, através do módulo de criação de parâmetros, criado nesta metodologia e descrito, também, neste capítulo.

Um dispositivo (*device*) virtual é criado através de uma linha de comando como mostra de maneira genérica o quadro a seguir:

Quadro 4-19- Linha de comando para criação de dispositivos

<code>-dev_NNN</code>	<code>nome_do_objeto.WRL</code>	<code>XXX</code>	<code>YYY</code>
-----------------------	---------------------------------	------------------	------------------

Onde NNN se refere ao tipo de dispositivo, XXX à posição no eixo X e YYY à posição no eixo Y, ambas referentes ao plano do chão (largura e profundidade). Não há inicialização da posição no eixo Z, pois essa metodologia considera que todas as máquinas e

demais objetos serão inicializados com sua base tocando o chão. Um exemplo prático de linha de comando para criação de dispositivo é mostrado no próximo quadro:

Quadro 4-20- Exemplo prático de criação de dispositivo em linha de comando

<code>-dev_MTC</code>	<code>VMC-3Axis.wrl</code>	<code>-350</code>	<code>-150</code>
-----------------------	----------------------------	-------------------	-------------------

De maneira análoga à linha de comando para dispositivos, pode-se criar, também, transportadores (*conveyors*) e a cena ou cenário virtual pode ser criado utilizando-se a tag “–*scene*”, seguida pelo nome do objeto VRML a ser usado como cenário da aplicação.

Além de transportadores e dispositivos, no início do desenvolvimento desta metodologia, fazia-se necessária a adição de componentes móveis de dispositivos, através, também, de linha de comando. Em outras palavras, era necessário que o usuário criasse uma linha de comando para a parte estática do dispositivo e outras linhas de comando, ou partes de linhas de comando associadas ao dispositivo, para cada uma das partes móveis desse dispositivo. Por exemplo, era necessário que o usuário criasse uma linha de comando para a parte estática de um torno, passando como parâmetros seu arquivo WRL e suas posições X e Y e, em seguida, criasse uma extensão para essa linha de comando, onde ele informasse os arquivos WRL referentes às partes móveis desse dispositivo, como a mesa ou eixo árvore. Essas extensões necessitavam de parâmetros adicionais que informassem o tipo de componente móvel que estava sendo adicionado ao dispositivo, como, por exemplo, “1X” ou “1Y” ou “1DOOR” (porta), a fim de possibilitar a correta animação dos componentes. O quadro a seguir apresenta um exemplo prático da antiga forma de entrada de parâmetros.

Quadro 4-21- Primeiro método utilizado para entrada de parâmetros da cena

```

VirtualWorld.exe -scene fabrica.wrl
-dev_mtc  torno.wrl  100  200  2
ferramenta.wrl  0X  0Y  1Z  0C  0DOOR
mesa.wrl  1X  1Y  0Z  0C  0DOOR
-dev_mtc  furadeira.wrl  400  200  1
eixo.wrl  0X  0Y  1Z  1C  0DOOR
config_file.jconf

```

A linha de comando foi dividida em várias linhas para simplificar o entendimento, porém sua alocação dentro de um arquivo de lote é realizada em uma única linha. No exemplo acima, os números destacados em vermelho, ao final de cada entrada de dispositivo, informam à aplicação qual o número de componentes móveis do dispositivo. Assim, no caso do primeiro dispositivo do exemplo acima, a aplicação alocaria dois componentes móveis como parte, ou filhos, do dispositivo “torno”. Dentro da aplicação o torno se comportaria com uma parte estática, referenciada pelo próprio arquivo “torno.WRL” e duas partes móveis, referenciadas por “ferramenta.WRL” e “mesa.WRL”, sendo que a ferramenta possuiria movimentação no eixo Z e a mesa possuiria movimentação nos eixos X e Y.

Com o aprimoramento da metodologia ao longo do tempo, alterou-se essa forma de adição de dispositivos e componentes móveis, com base em uma funcionalidade fornecida pelos *softwares* de modelagem e em outra funcionalidade fornecida pela biblioteca *OpenSG*. O propósito da alteração foi proporcionar ao usuário maior eficiência, controle e simplicidade no momento da entrada de parâmetros e criação de objetos da aplicação.

Os arquivos tridimensionais exportados como VRML, com extensão WRL, carregam consigo informações geométricas de objetos tridimensionais, criados em *softwares* de modelagem e exportados para esse tipo de arquivo. No entanto, há várias formas de criar um objeto composto das mesmas geometrias. Essa diferença acontece devido às diferentes

formas de agrupamento de objetos proporcionadas pelos *softwares* de modelagem 3D. Esta característica de modelagem foi utilizada nesta metodologia como fator de extrema importância na criação de dispositivos e componentes, pois cada dispositivo deve, obrigatoriamente, conter implicitamente seus próprios componentes móveis em um único arquivo VRML. Cada componente seja estático ou móvel deve ser agrupado distintamente, a fim de funcionar como um objeto separado dentro da aplicação virtual.

Para viabilizar o acesso a diferentes grupos de objetos tridimensionais dentro de um mesmo arquivo VRML, o *OpenSG* fornece a função *getChild(index)*. Há, ainda, outra função disponível para acessar o nome de cada nó: *getName(node)*. Os nomes dos nós dentro dos arquivos VRML são os mesmos nós dos grupos criados dentro dos *softwares* de modelagem. Utilizando-se essas duas funções conjuntamente é possível identificar os níveis nos quais os nós se encontram dentro do arquivo VRML, registrá-los como identificadores numéricos dos componentes e utilizá-los posteriormente para acesso e animação de cada componente móvel.

4.9.5 Servidores e clientes

Além da inicialização de todas as variáveis e instanciação de todos os objetos da aplicação principal, o procedimento *initScene* é responsável, também, pelo controle de fluxo da aplicação, com relação a aplicações cliente e servidor. Em resumo, uma aplicação servidor envia dados para as aplicações clientes e uma aplicação cliente apenas recebe dados de um servidor. Os dados comunicados se referem ao sincronismo de dados de movimentação capturados por dispositivos de interação que não puderam ser configurados em arquivos

externos do VR Juggler, como é o caso do dispositivo de interação por luz. O procedimento *initScene* utiliza um sistema de *flags* externas para o reconhecimento de um computador servidor ou cliente. Esse processo é detalhado em outro tópico deste capítulo.

4.9.6 Iluminação

Apesar de ser possível a exportação de nós de iluminação do 3D Studio Max para arquivos VRML, esse tipo de nó não é exibido automaticamente pelo *OpenSG*. Portanto, seria necessário que o programador percorresse a árvore de objetos de cada VRML para identificação das luzes exportadas. Mesmo assim, os tipos de luzes criados no 3DStudio Max não se comportam exatamente como no *OpenSG*, portanto a tradução da iluminação de um ambiente ao outro seria dispendiosa e aumentaria a complexidade da metodologia. Optou-se, então, por iluminar a cena virtual através do código fonte, em *OpenSG*, para simplificar e acelerar o processo de desenvolvimento. Assim, a iluminação básica da cena virtual está pronta no procedimento *lightScene* e pode ser alterada a qualquer momento pelo programador que siga os passos da metodologia.

Existem diversos tipos de iluminação que podem ser aplicados a uma cena virtual. Alguns são direcionados a todos os objetos da cena, com intensidade idêntica. Outros são pontuais e iluminam os objetos da cena de maneira diferente uns dos outros. Os tipos de iluminação permitidos pelo *OpenSG* são:

- *Point Light*: É o tipo mais comum de luz em uma aplicação de *OpenSG*. Ele emite raios de luz igualmente em todas as direções. O programador deve definir uma posição para o ponto de luz e alguns parâmetros de atenuação que

afetarão a intensidade com que os raios atingem cada objeto. Além disso, a cor da iluminação deve ser fornecida;

- *Directional Light*: é um tipo de iluminação que não possui um lugar definido dentro da cena virtual. Todos os raios de luz são paralelos, necessitando apenas da definição de cor e direção. Geralmente fornece menor realismo do que o realismo conseguido com a utilização de um *Point Light*, porém, computacionalmente são mais rápidas;
- *Spot Light*: É, computacionalmente, a luz que exige maior processamento. Assim como o nó *Point Light*, possui cor, atenuação e posição dentro da cena, porém, possui também direção e ângulo, que definem a região iluminada.

Além dos três tipos de nó de iluminação fornecidos pelo OpenSG, é importante citar um quarto nó, que desempenha uma função importante para a iluminação de uma cena virtual. O nó *Beacon* é responsável pela posição e orientação dos nós de iluminação. A posição de um nó de iluminação no grafo de cena define quais objetos serão afetados. Todos os filhos de um nó de iluminação serão atingidos pela luz emitida por esse nó. Porém, a posição e orientação da luz proveniente desse nó é definida pelo nó de transformação relacionado ao *Beacon*, que, por sua vez, está relacionado com o nó de iluminação. A figura a seguir demonstra com clareza essa situação:

Figura 4-48- Posição dos nós *Beacon* e *Point Light* no grafo de cena.

A iluminação base criada para as cenas desta metodologia, consiste de um nó *PointLight* e um *Beacon*. A iluminação criada funcionou adequadamente para o estudo de caso e demais testes envolvidos neste trabalho, porém, para cenas virtuais maiores, há necessidade de ampliação do número de nós de iluminação existentes, dentro do procedimento *lightScene*.

A figura seguinte mostra um nó de iluminação básico, iluminando toda a cena virtual instanciada, ao mesmo tempo, como é o caso utilizado nesta metodologia.

Figura 4-49- Luz posicionada no grafo de cena.

4.9.7 Sonorização

Vários aspectos devem ser levados em consideração visando a ampliação da imersão do usuário em um ambiente virtual imersivo. Um desses aspectos é a sonorização do ambiente.

Há várias formas de sonorizar um ambiente virtual e algumas delas foram descritas nos capítulos anteriores deste trabalho. Nesta metodologia optou-se por sonorizar o ambiente virtual com um sistema de *Home-Theater*, ao invés de fones de ouvido, para simplificar a utilização multiusuário do dispositivo. A distribuição tridimensional do som

através das seis caixas de som do sistema 5.1 do *Home-Theater* foi criada, a princípio, sem levar em consideração o posicionamento dos objetos virtuais. Dessa forma, a navegação do usuário dentro da cena não altera a intensidade e posição do som. Esse método simples de adição de som funciona para testes individuais de máquinas, porém não é eficiente para uma simulação virtual que contenha mais máquinas. Faz-se necessário o tratamento do posicionamento tridimensional do som, ou seja, a alteração da intensidade do som em cada uma das seis caixas do sistema, com base na movimentação do usuário. Por exemplo, conforme o usuário se aproxima de uma máquina, ele deve escutar seu som com maior intensidade do que os sons das demais máquinas. Ou ainda, conforme o usuário se afaste das máquinas sonorizadas do ambiente, a intensidade dos sons deve ser conjuntamente reduzida.

A tarefa de tratamento tridimensional de sons em tempo real dentro de uma aplicação virtual não é trivial, ainda mais se levadas em conta as restrições impostas pela utilização de diversas bibliotecas de *software* trabalhando conjuntamente, como é o caso deste trabalho. Este trabalho utilizou a biblioteca *OpenAL*, configurada adequadamente dentro do arquivo de configuração do VR Juggler, para o disparo de sons em tempo real dentro do ambiente virtual. A criação de sons 5.1 foi realizada de maneira *off-line* através do *software Cool Edit 2.1*, conforme descrito nos capítulos anteriores. Porém, o tratamento *on-line* do posicionamento do som tridimensional ainda não foi finalizado.

4.9.8 Aquisição dos dados dos módulos externos

Para padronizar as formas de aquisição de dados nesta metodologia, foi definido que todos os dados entrariam na aplicação principal através de arquivos XML no formato

MTConnectStreams. Porém, o método de requisição de dados varia entre as formas de aquisição.

Para dados de máquinas-ferramenta originalmente adquiridos via MTConnect, há necessidade de comunicação com o módulo agente, tanto para verificação das capacidades do dispositivo, quanto para a requisição de amostras de dados. A verificação das capacidades é realizada através de um *download* inicial, chamado “*current*”. Essa verificação retorna um arquivo XML do tipo *MTConnectStreams*, onde podem ser observados todos os componentes e itens de um dispositivo. No entanto, o dado mais importante de *current* para essa metodologia é o atributo *nextSequence*, utilizado como dado de entrada para a primeira requisição de amostras de dados.

Após a obtenção de *nextSequence* via *current*, o objeto dispositivo instanciado começa a realizar requisições *sample*, para aquisição de dados de seu interesse. O atributo *nextSequence* é atualizado a cada arquivo XML recebido. O fluxograma seguinte ilustra esse procedimento.

Figura 4-50- Aquisição de dados de dispositivo real via MTConnect.

Apesar da padronização do recebimento de dados, para aquisição de dados via câmera não há requisições *current* ou *sample*. O arquivo *XML MTConnectStreams* é obtido através de um *download* simples.

Figura 4-51- Aquisição de dados via câmera.

A aquisição de dados via câmera pode, ainda, ser intermediada por um agente MTConnect, modificado, e números de sequência, para sincronização dos dados obtidos. A figura seguinte ilustra essa forma de obtenção de dados.

Figura 4-52- Aquisição de dados via câmera com intermediação de um agente.

4.9.9 Utilização e sincronização dos dados *on-line* adquiridos

A utilização dos dados adquiridos é idêntica para ambas as formas de aquisição de dados reais *on-line*, o que torna esta metodologia flexível a quaisquer fontes de dados, desde que sua aquisição siga os procedimentos descritos neste trabalho.

Após a aquisição de dados via arquivos *XML MTConnectStreams*, os mesmos são, inicialmente, armazenados em *buffers* para utilização quase imediata, ou *on-line*, na cena virtual. O *buffer* de recepção é composto por dois vetores de *string*, que são utilizados conjuntamente e serialmente para evitar acesso concorrente e possibilitar leitura contínua dos dados.

A sincronização é arranjada através de semáforos de *Threads* entre o *looping* da aplicação principal e os *downloads* de arquivos e escrita e leitura dos vetores de dados. O fluxograma a seguir mostra a visão geral do método de sincronização desenvolvido.

Figura 4-53- Semáforo para leitura e escrita dos dois vetores do *buffer*.

Os dados de comportamento são exibidos na cena virtual através dos procedimentos *translateObject* e *rotateObject* criados para esta aplicação e convocados no método *preFrame*, nativo de OpenSG. Esse método é utilizado por aplicações de OpenSG para o *looping* principal da aplicação, que só será encerrado em seu término. Os procedimentos *translateObject* e *rotateObject* requerem, como dados de entrada, os incrementos em todos os eixos X, Y e Z. Esses valores são obtidos diretamente dos vetores de *buffer* de cada dispositivo. Cada passagem pelo *looping* principal do *preFrame* atualiza os valores de todos os dispositivos instanciados.

Figura 4-54- Atualização dos dados no procedimento *preFrame*

Por sua vez, a atualização dos dados na cena é detalhada a seguir.

Figura 4-55- Métodos *translateObject* e *RotateObject*

Os vetores de *buffer* são, na verdade, vetores de estruturas, que armazenam quatro tipos de dados ao mesmo tempo, em cada um de seus índices:

1. Primeira posição armazena: dado;
2. Segunda posição armazena: eixo;
3. Terceira posição armazena: número do componente;
4. Quarta posição armazena: número de sequência do dado.

No momento da atualização dos valores dos *buffers* na cena virtual, dentro do *looping* do *preFrame*, todos esses dados são avaliados para que a aplicação possa determinar qual componente deve ter os dados atualizados e em quais de seus eixos de movimentação.

O procedimento *preFrame* realiza a leitura dos dados de cada dispositivo de maneira sequencial, baseado no número de sequência armazenado junto a cada dado. Como, atualmente, os vetores são ordenados antes de chegarem ao procedimento *preFrame*, essa

responsabilidade deixou de ser de *preFrame* e passou a pertencer ao procedimento *readXML*, que coloca os dados do XML nos vetores de *buffer*.

Além do uso de semáforos para o controle de acesso de *download* e leitura de arquivos XML, o sincronismo de leitura dos vetores de *buffer* é realizado através de uma variável *booleana*, chamada *firstVector*, que informa aos procedimentos *readXML* e *preFrame*, qual vetor deve ser lido ou escrito no momento atual da execução.

4.9.10 Encerramento da aplicação

O método *exit* é o último método executado pela aplicação VR Juggler, no momento de seu encerramento. Ele é utilizado para liberar as variáveis, ponteiros, *Threads* e objetos alocados na memória, tanto para itens nativos do VR Juggler, quanto para objetos criados pelo programador. Outra responsabilidade atribuída a este método, nesta metodologia, foi a finalização requerida pelo sistema de som da biblioteca *OpenAL*, através do método *changeAPI*.

4.10 MÓDULO DE SINCRONIZAÇÃO DE OPERAÇÃO DE MÁQUINAS REAIS

A metodologia descrita neste capítulo viabiliza a simulação virtual de diversos dispositivos alimentados por dados reais *on-line*, dentro de um mesmo ambiente virtual híbrido. Cada dispositivo é responsável pela aquisição de seus dados de comportamento e execução dos mesmos dentro do AV. No caso de máquinas-ferramenta e células de

manufatura, faz-se necessária a troca de mensagens de sincronização, para que os dispositivos virtuais e reais possam interagir entre si.

A sincronização de máquinas puramente virtuais dentro de um ambiente virtual é obtida de maneira simples, através de atributos de controle dos objetos instanciados ou, ainda, de uma classe controladora, desenvolvida especificamente para essa tarefa. No entanto, a sincronização de máquinas reais, dentro de um ambiente virtual, não é uma tarefa trivial.

Na área de manufatura, existem dispositivos com controladores *CNCs* com estrutura aberta, que possibilitariam o envio de dados bilateral e, até mesmo, a tele operação de máquinas via ambientes virtuais. No entanto, a maioria das máquinas-ferramenta não possui *CNC* com estrutura aberta, mas sim proprietária. Desse modo, não se faz possível o envio de dados à maioria das máquinas-ferramenta. A comunicação com esses controladores é feita de maneira unilateral.

Essa metodologia engloba um sistema de troca de mensagens entre computadores, via *socket*, para viabilizar a sincronização de operação das máquinas reais, sendo simuladas dentro de um ambiente virtual. Devido a estrutura fechada da maioria dos *CNCs*, o método adotado neste trabalho não pode ser automatizado, devendo ser intermediado por operadores.

A forma de comunicação adotada inclui a conexão de máquina através de seu número de *IP* e sua porta de comunicação, além de mensagens de confirmação de conexão. O envio de dados é realizado por *strings* que carregam a mensagem através da rede e são recepcionadas e processadas no computador destino. A comunicação é bilateral, sendo que tanto o computador da aplicação, quanto o módulo externo de captação de dados podem enviar e receber mensagens. A figura a seguir mostra os componentes principais da troca de mensagens de sincronização.

Figura 4-56- Comunicação de mensagens via *socket*

Cada dispositivo instanciado possui atributos onde são guardadas as relações deste objeto com seus objetos vizinhos. Portanto, um dispositivo sempre conhece qual objeto necessita receber suas informações de atualização e qual objeto o atualiza. Além disso, todo dispositivo possui uma variável de controle de peças processadas, onde é armazenado o número de peças concluídas. Assim que uma máquina termina a operação de uma peça, seja ela virtual ou real, o dispositivo atualiza o valor da variável *part_Count* (contagem de peças) e emite uma mensagem informando a conclusão da operação. O dispositivo subsequente, dependente desta informação, recebe a atualização e começa sua operação sobre a peça recebida.

No caso de máquinas-reais, esse processo necessita do intermédio de um operador. Esta metodologia pressupõe que uma máquina-ferramenta com dados capturados

via MTConnect, possua um computador de apoio localiza próximo à mesma. Esse computador, geralmente, hospedaria o módulo adaptador e, no caso de sincronização da máquina com as máquinas virtuais, hospedaria, também, um módulo de recebimento de mensagens de sincronização. A mensagem de uma máquina virtual informando a conclusão de uma operação chegaria à tela do computador de apoio, informando o operador que a máquina real pode começar sua operação sobre a peça. Como mencionado acima, esse processo pode ser automatizado para tele operação de máquinas com *CNC* de estrutura aberta, porém, esta metodologia se restringiu ao desenvolvimento da comunicação de mensagens de sincronização.

4.11 SINCRONIZAÇÃO DOS DADOS ENTRE OS COMPUTADORES DO *CLUSTER*

O *cluster* ou aglomerado de computadores da CAVE do LMVI foi introduzido nos capítulos anteriores e será detalhado neste tópico.

O sistema de computadores que controla a CAVE do LMVI é composto por três computadores principais que trabalham como um *cluster*. Cada uma das máquinas possui uma placa de vídeo com duas saídas DVI e é responsável pelo controle de dois projetores ao mesmo tempo. Os modelos das placas de vídeo e som dos computadores da CAVE foram descritos anteriormente, no capítulo que descreve o LMVI.

Cada um dos dois projetores, controlados por uma mesma máquina, exibem imagens sobrepostas em uma mesma tela de projeção, criando um efeito de imagem desfocada, para observadores desprovidos de óculos estereoscópicos. Há um arquivo de

configuração do *software VR Juggler*, onde é possível configurar a distância entre as imagens. Geralmente, inicializa-se o valor dessa distância com o valor da distância dos olhos de um usuário comum.

Todos os computadores do *cluster* da CAVE são idênticos em configuração e dispositivos e executam a mesma aplicação, ao mesmo tempo. No entanto, há necessidade de sincronização do fluxo das três aplicações inicializadas, para que todos os computadores consigam exibir o mesmo *frame* da cena virtual em um determinado momento. Isso se faz necessário, principalmente, devido ao uso de dispositivos de interação ou navegação que são plugados apenas no computador principal do *cluster*. Não há maneira adequada, ou não é viável, a utilização de um mesmo dispositivo, ao mesmo tempo, em três computadores diferentes, portanto, normalmente, em um *cluster* de realidade virtual imersiva, instala-se um dispositivo na máquina principal (*master*) e enviam-se os dados do mesmo para os demais computadores, através da rede interna que conecta as máquinas do *cluster*. A figura a seguir mostra um exemplo de utilização de dispositivo de interação em um ambiente virtual imersivo, bem como seu funcionamento interno.

Figura 4-57- Sincronia dos *frames* com dados de entrada de dispositivo externo.

Esses dispositivos alteram as posições da câmera e dos objetos virtuais da cena no computador onde foram plugados. Dessa forma, se não houvesse sincronização dos dados de movimentação, apenas o computador principal exibiria corretamente as novas posições dos objetos virtuais. Para controlar esse tipo de problema, o *software* VR Juggler possui um módulo de controle, onde é realizada a sincronização de exibição de *frames* de cada uma das máquinas do *cluster*. O VR Juggler deixa a cargo do programador apenas a devida configuração de seus arquivos com extensão “JCONF”, responsáveis, entre outras coisas, pela configuração do sincronismo do *cluster* e instalação de dispositivos de interação.

No caso do LMVI, para viabilizar a comunicação entre as máquinas, todos os computadores estão conectados a uma rede interna e cada máquina possui um nome para referência. Além das três máquinas principais, existe uma máquina de apoio, também

pertencente à rede interna, que é capaz de se comunicar com as outras três máquinas, com a finalidade de envio e disparo de arquivos e suporte em geral. Os computadores do *cluster* da CAVE do LMVI são listados a seguir:

- 1) Computador “*FRONT*” (Frontal): nó principal (*master*) do *cluster* de computadores da CAVE. Responsável pelas imagens dos projetores da tela frontal da CAVE e conexão de dispositivos;
- 2) Computador “*RIGHT*” (Direita): nó secundário (*slave*) do *cluster* de computadores da CAVE.
- 3) Computador “*FLOOR*” (Chão): outro nó secundário (*slave*) do *cluster* de computadores da CAVE;
- 4) Computador “*CAVE01*”: faz parte da rede interna do LMVI, porém não faz parte das execuções de aplicações da CAVE.

O *software* VR Juggler coordena a exibição correta dos *frames* da cena virtual de cada aplicação, quando os dispositivos de interação e navegação podem ser configurados nos seus arquivos de configuração. Porém, o VR Juggler não está preparado para sincronizar automaticamente dados específicos de cada aplicação que não possam ser configurados diretamente nos arquivos de configuração. Neste trabalho, determinados tipos de dados de posicionamento, adquiridos tanto pelo primeiro, quanto pelo segundo método de aquisição de dados, necessitaram de tratamento especial para sincronização.

A seguir serão detalhadas as tarefas executadas pelo nó principal, “*FRONT*”, do *cluster* de computadores da CAVE, que recebe e envia dados de posicionamento para as demais máquinas do *cluster*, a fim de sincronizar a exibição das imagens. E, em seguida, serão descritos, também, todos os passos executados pelas máquinas secundárias da CAVE.

4.11.1 Tarefas do computador principal do *cluster*

O computador denominado “*FRONT*” é o computador, ou nó, principal do cluster de computadores da CAVE e é o responsável pela recepção, processamento, envio e sincronização dos dados entre as três máquinas do cluster. Tanto o primeiro, quanto o segundo método de aquisição de dados desenvolvidos, comunicam-se exclusivamente com o nó “*FRONT*”. Este, por sua vez, possui duas responsabilidades exclusivas:

- 1) Recebimento dos dados provenientes da aplicação remota e
- 2) Envio dos dados para os demais computadores do *cluster*.

Embora seja o nó principal do cluster, o “*FRONT*” executa a mesma aplicação executada pelos computadores secundários. Dessa forma, uma única aplicação virtual de sistemas imersivos foi desenvolvida, visando simplificar o método de compilação e execução das aplicações.

Dentro da aplicação foi programado um módulo de controle baseado em *flags* externas, ou arquivos de sinalização externos à aplicação, responsável por alterar o fluxo de execução das instâncias da aplicação. A *flag* externa de controle é um arquivo de sinalização, sem conteúdo, que deve ser colocado na pasta da aplicação, junto ao arquivo executável. No caso deste trabalho, essa *flag* foi colocada apenas no computador principal, para sinalizar à aplicação que, nesta máquina, o fluxo de execução seria de servidor e não de cliente. A *flag* não foi colocada na pasta de aplicação das demais máquinas, que, portanto, executam a

aplicação como clientes. A tarefa de verificação de presença da *flag* “servidor/cliente” é executada logo após a leitura dos parâmetros de inicialização da aplicação e inicialização de valores das principais variáveis do programa. Há um procedimento específico da estrutura do VR Juggler adequado para tarefas iniciais de mudança de fluxo de execução, pois é executado uma única vez, antes do carregamento da cena virtual. Esse procedimento é denominado *InitScene*.

Figura 4-58- *Flag* Cliente-Servidor.

A finalidade principal do sistema de *flags* é tornar viável o desenvolvimento de uma única aplicação, que pode ser disparada nos três computadores do cluster da CAVE. Caso esse sistema não fosse desenvolvido, haveria necessidade de implementação de alterações no código fonte de cada aplicação e cada uma delas teria que ser compilada de

forma independente e exclusiva para o computador principal e computadores escravos. Uma alternativa ao sistema de *flags* seria a criação e inicialização de parâmetros auxiliares, específicos para o tratamento de nós do cluster.

A aplicação executada pelo computador principal encontra a *flag* de servidor, com o nome “*ServerFlag*”, na mesma pasta onde encontram-se os demais arquivos da aplicação, e instancia a classe *Server*, com o nome “*serverSocket*”. Em seguida, uma *Thread* de execução, instanciada através do método *startThread*, inicializa a execução paralela do método *sendData*, da classe *Server*. O método *sendData* envia dados para os outros dois computadores do cluster sempre que há novos dados disponíveis. Isso garante a sincronia dos três computadores e, ao mesmo tempo, evita o congestionamento da rede, pois deixa de enviar dados, caso sejam antigos. A função *sendData* possui um controle para enviar apenas dados novos, porém não controla o recebimento correto dos dados nos computadores secundários. O controle de recebimento de dados não foi implementado, pois atrasaria a execução e o dinamismo da aplicação, causando certamente travamentos da movimentação do usuário. A interação e navegação intuitiva e sem atrasos foi considerada mais importante do que a garantia de recebimento de cada frame por parte dos computadores secundários. A comunicação foi implementada de uma forma onde os computadores secundários se adequam rapidamente ao processamento e exibição dos frames do computador principal, mesmo quando há frames perdidos na transmissão de dados pela rede.

Para que o servidor possa enviar dados durante todo o tempo de execução da aplicação, todo o código do método *sendData* está alocado internamente a um *looping* infinito, que é executado até o término da aplicação. Quando o usuário encerra a aplicação virtual, um método chamado “*stopThread*” é executado e encerra o processamento paralelo da *Thread* através do comando *kill*.

Antes de começar a transmitir dados pela rede, o servidor precisa garantir a conexão com os computadores remotos (clientes). Todo esse procedimento é realizado antes da inicialização da cena virtual, durante a execução do construtor da classe *Server*. Para a realização desta tarefa, foi implementado um método chamado “*startServer*”, que cria um socket e inicializa o número de porta e o número de IP que serão utilizados pelo socket. Após a inicialização, o socket mantém-se em estado de espera, aguardando a conexão das outras duas máquinas do cluster. Assim que ambas concluírem a conexão, a Thread de envio de dados do servidor é disparada.

O procedimento *sendData* trabalha com um buffer de dados com tamanho estipulado com base nos tipos das variáveis a serem enviadas. Todos os dados são enviados em uma única *string* com marcadores que informam a posição de cada dado, para que seja possível a separação adequada dos dados nos computadores clientes.

Além do envio de dados para os computadores secundários, o computador principal desempenha outra tarefa exclusiva. Ele precisa receber os dados da aplicação remota, que está em execução dentro de uma fábrica real ou, mais genericamente, qualquer ambiente real de interesse da aplicação, e envia dados constantemente através da Internet ou rede Ethernet. No caso deste trabalho, a aplicação remota envia dados para o computador principal da CAVE através da Internet.

O computador principal da CAVE tem a função de servidor e cliente ao mesmo tempo. Ele desempenha a função de servidor com relação aos computadores secundários e de cliente com relação à aplicação remota. Portanto, diferentemente dos demais computadores envolvidos na aplicação, o computador principal implementa Threads de recebimento e de envio de dados, centralizando todas as informações, seja recebendo-as de outros computadores ou enviando-as para outras máquinas. Os demais computadores do cluster da

CAVE desempenham somente a função de receptor de dados, como será detalhado no próximo tópico.

4.11.2 Tarefas dos demais computadores do *cluster*

Os computadores denominados “*RIGHT*” e “*FLOOR*” são computadores secundários (*slave*) dentro do *cluster* de computadores da CAVE do LMVI. Como visto anteriormente, ambos executam a mesma aplicação destinada ao computador principal do *cluster*, o nó “*FRONT*”, porém, essa aplicação possui um sistema de *flags* (bandeiras) interno, que altera os rumos tomados por cada uma das três aplicações disparadas, no caso do LMVI.

Quando as aplicações são inicializadas, o *software* VR Juggler inicializa todas as suas variáveis internas, que estão fora do escopo do programador, e, em seguida, direciona a aplicação para o primeiro procedimento a ser executado, o procedimento *Init*. Nesse procedimento, foi implementado um algoritmo de verificação de *flags* externas, que permite a cada aplicação seguir um fluxo diferente de execução. Se uma aplicação possui a *flag* de servidor, ela inicializa as *Threads* e procedimentos específicos de um servidor. Se uma aplicação não possui a *Thread* de servidor, ela executa apenas os procedimentos e *Threads* específicas de um cliente. No caso dos computadores *Right* e *Floor*, a *flag* de sinalização não deve ser colocada na pasta da aplicação, para que ambos sejam instanciados como computadores clientes.

Após o procedimento *Init*, o VR Juggler destina a aplicação ao método *InitScene* que inicializa a cena virtual. Porém, nesse momento, todas as *Threads* de comunicação já

foram disparadas, tanto pelo servidor, quanto pelos clientes. Da mesma forma que o computador servidor instancia um objeto *Server* na memória, os computadores clientes instanciam objetos *Client* na memória da máquina e buscam na rede uma conexão com um computador servidor. Após efetivar a conexão através de seu método *connectClient*, um computador cliente dispara sua *Thread receiveData* (Recebimento de Dados) e passa a checar constantemente a porta configurada, para receber os dados transmitidos pelo servidor. Os clientes trabalham com um *buffer* exatamente igual ao *buffer* definido para o servidor, para que seja possível a recepção de todos os dados transmitidos. Não há controle de recepção correta dos dados, nem mensagem de confirmação. Os dados são recebidos, processados e as novas posições dos objetos ou câmeras virtuais são exibidas nas telas *Right* ou *Floor* dos clientes. Nesse momento da aplicação, quando os dados são recebidos, o computador servidor considera que os dados foram enviados e recebidos corretamente, mesmo que este não seja o caso, e também processa e exibe o novo posicionamento dos objetos na tela *Front*. Ao final da execução da aplicação, tanto as *Threads Server*, quanto as *Threads Client* são desalocadas, para garantir a correta finalização da aplicação.

4.12 ESTENDENDO A METODOLOGIA

Como citado anteriormente, a metodologia descrita neste capítulo tomou como foco principal a área de manufatura, porém pode ser adaptada para outras áreas de aplicação. Este tópico visa traçar algumas considerações iniciais sobre a extensão da metodologia a outras áreas de aplicação, bem como discutir algumas ideias fundamentais a respeito dos trabalhos futuros sugeridos ao final deste documento.

A estrutura de classes desenvolvida e os objetos tridimensionais modelados tomaram como base a área de manufatura. Porém, toda a padronização utilizada para a produção dos objetos virtuais de manufatura ou mesmo para a estruturação das classes da metodologia pode ser seguida para sua adequação a outras áreas. O algoritmo de segmentação foi modularizado e pode ser readaptado a quaisquer tipos de segmentação considerados mais adequados para a aquisição de dados de posicionamento ou movimentação de outras áreas de aplicação. A transmissão de dados pode utilizar o mesmo formato utilizado nesta metodologia tanto para aquisição de dados via MTConnect, quanto para aquisição de dados via câmeras. Neste trabalho, seguiu-se o protocolo MTConnect para a transmissão de todos os tipos de dados adquiridos, pois, apesar de ter sido criado especificamente para máquinas-ferramenta, trata-se de um protocolo bem documentado, simples e robusto, que funciona adequadamente para quaisquer tipos de dados de posicionamento.

Outras áreas de aplicação podem, inclusive, herdar as classes abstratas da hierarquia definida, a fim de simplificar seu acoplamento à aplicação principal. Por exemplo, a classe *Device* é abstrata e pode funcionar como classe base para quaisquer objetos que escrevam XML no padrão proposto, através da Internet, mesmo que esses objetos não sejam máquinas-ferramenta. Analogamente, a classe abstrata *Component* pode ser herdada para a viabilização dos componentes móveis internos de qualquer item, em qualquer área de aplicação. Além disso, o agente MTConnect pode ser reutilizado ou adaptado ou, ainda, um novo escritor de XML pode ser desenvolvido apenas para os dados de interesse de cada aplicação. Esses dados devem ser registrados no padrão XML *MTConnectStreams*, como foi feito neste trabalho para o módulo simulador de dados reais.

4.12.1 Modificação dinâmica das peças

A produção dinâmica de peças dentro do ambiente virtual é uma tarefa complexa, sugerida como trabalho futuro. Porém, paralelamente ao desenvolvimento da metodologia, iniciou-se um trabalho referente à modelagem dinâmica de objetos virtuais, dentro dos AVs.

Como ilustrado na figura a seguir, sugere-se a criação de uma peça única, porém composta por uma quantidade pré-definida de blocos menores, que constituiriam a unidade formadora de quaisquer tipos de peças.

Figura 4-59- Peça subdividida em unidades para modificação dinâmica

Desse modo, a verificação da colisão da ferramenta do dispositivo, individualmente, com cada unidade de composição da peça, pode ser utilizada para a modelagem de seu formato, dinamicamente, dentro do ambiente virtual. Para isso, no momento da colisão da ferramenta com uma unidade de composição, a unidade deve ter seu modelo geométrico escondido (*hidden*) pela aplicação, caso a ferramenta esteja em operação. A figura a seguir exemplifica essa operação.

Figura 4-60- Colisão da ferramenta com as unidades de composição das peças.

Como mencionado anteriormente, essa funcionalidade não foi concluída dentro dessa metodologia, porém sua adição pode ser uma característica importante, que deve abrir o escopo da metodologia a outros tipos de análise visual, como, por exemplo, a fabricação virtual de uma peça, antes de sua produção real.

O próximo capítulo apresenta alguns estudos de caso dos módulos desenvolvidos, tanto independentemente, quanto em utilização conjunta. São apresentados, também, alguns resultados, através da disponibilização dos dados obtidos e situações no mundo virtual ilustradas.

5 ESTUDO DE CASO E RESULTADOS

Este capítulo apresenta estudos de caso da metodologia proposta, englobando a maioria dos passos descritos no capítulo anterior. Concomitantemente serão apresentados, como resultados, os dados *on-line* obtidos e o respectivo comportamento dos objetos virtuais dentro da aplicação virtual desenvolvida.

Para a realização do estudo de caso, foram utilizados ambos os métodos para aquisição de dados reais *on-line* descritos nesta metodologia:

- Aquisição de dados via MTConnect;
- Aquisição de dados via câmeras de baixo custo.

O primeiro método foi utilizado em três situações diferentes:

- Aquisição de dados de um simulador de dados reais, MTConnect *Live Agent*, fornecido pelo instituto MTConnect;
- Aquisição de dados de um simulador de dados reais desenvolvido neste trabalho para proporcionar a aquisição de tipos de dados não fornecidos pelo MTConnect *Live Agent*;
- Aquisição de dados de uma máquina-ferramenta, mais especificamente, de um centro de usinagem.

O método de aquisição de dados reais *on-line* via câmeras foi utilizado para dados reais adicionais, provenientes de objetos ou dados não padronizados pelo protocolo

MTConnect. Neste caso, validou-se o método com os dados contínuos de uma porta de centro de usinagem.

5.1 MTCONNECT *LIVE AGENT*

O instituto MTConnect disponibiliza um simulador de dados reais *on-line* para ser utilizado por aplicações que não disponham de máquinas-ferramenta reais para seus testes. Esta metodologia foi testada e validada, entre outros meios, através deste simulador *on-line*.

O simulador encontra-se hospedado no servidor do instituto e pode ser acessado através do endereço: “agent.mtconnect.org”. O detalhamento das formas de requisição a agentes MTConnect pode ser encontrado, anteriormente, em outro capítulo deste trabalho.

O simulador MTConnect *Live Agent* não simula todos os tipos de dados padronizados por MTConnect, porém este estudo de caso mostra a definição, aquisição e utilização de alguns tipos de dados disponibilizados.

Como não foi encontrada documentação disponível para o MTConnect *Live Agent*, toda a análise dos dados disponíveis foi baseada na consulta *probe*. O endereço para realização de consulta *probe* para o agente simulador é: <http://agent.mtconnect.org/probe>. O resultado retornado por essa consulta é um arquivo XML do tipo *MTConnectDevices*. Esse arquivo é mostrado e explicado a seguir, todavia optou-se por dividi-lo em seções para a simplificação da explicação.


```

<MTConnectDevices xmlns:m="urn:mtconnect.org:MTConnectDevices:1.2"
xmlns="urn:mtconnect.org:MTConnectDevices:1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mtconnect.org:MTConnectDevices:1.2
http://www.mtconnect.org/schemas/MTConnectDevices_1.2.xsd">
  <Header creationTime="2012-07-28T20:40:20Z" sender="mtconnect" instanceId="1340212647"
  version="1.2.0.10" assetBufferSize="1024" assetCount="0" bufferSize="131072"/>
  ▼<Devices>
    ▼<Device id="dev" iso841Class="6" name="VMC-3Axis" sampleRate="10" uuid="000">
      <Description manufacturer="SystemInsights"/>
      ▼<DataItems>
        <DataItem category="EVENT" id="avail" type="AVAILABILITY"/>
        <DataItem category="EVENT" id="dev_asset_chg" type="ASSET_CHANGED"/>
      </DataItems>
      ▼<Components>
        ▼<Axes id="ax" name="Axes">
          ▼<Components>
            ▼<Rotary id="c1" name="C">
              ▼<DataItems>
                ▼<DataItem category="SAMPLE" id="c2" name="Sspeed" nativeUnits="REVOLUTION/MINUTE"
                subType="ACTUAL" type="SPINDLE_SPEED" units="REVOLUTION/MINUTE">
                  <Source>spindle_speed</Source>
                </DataItem>
              </DataItems>
            </Rotary>
          </Components>
        </Axes>
      </Components>
    </Device>
  </Devices>

```

Figura 5-1- Primeira parte da consulta *probe* para *Live Agent*

A parte superior da primeira parte do arquivo mostra o tipo de arquivo XML e suas informações de protocolo, bem como o cabeçalho (*Header*) e os dispositivos englobados por esse agente, no caso, o dispositivo VMC-3Axis. Os itens particulares deste dispositivo são mostrados em seguida: “*avail*” e “*dev_asset_chg*”, porém estes itens de dispositivo não foram utilizados para este estudo de caso.

A parte inferior da figura acima mostra o começo da lista de componentes do dispositivo em questão e é parte importante deste estudo de caso, pois os dados utilizados foram consultados diretamente de componentes do dispositivo. O primeiro componente de VMC-3Axis é *Axes*, um componente com eixos de movimentação. Os itens internos desse componente mostram que esse dispositivo possui movimentação rotacional no componente nomeado como “C”.

```

▼<DataItem category="SAMPLE" id="c3" name="Sovr" nativeUnits="PERCENT"
  subType="OVERRIDE" type="SPINDLE_SPEED" units="PERCENT">
  <Source>SspeedOvr</Source>
</DataItem>
▼<DataItem category="EVENT" id="cm" name="Cmode" type="ROTARY_MODE">
  ▼<Constraints>
    <Value>SPINDLE</Value>
  </Constraints>
</DataItem>
<DataItem category="CONDITION" id="Cloadc" type="LOAD"/>
</DataItems>
</Rotary>
▼<Linear id="x1" name="X">
  ▼<DataItems>
    <DataItem category="SAMPLE" id="x2" name="Xact" nativeUnits="MILLIMETER"
      subType="ACTUAL" type="POSITION" units="MILLIMETER"/>
    <DataItem category="SAMPLE" id="x3" name="Xcom" nativeUnits="MILLIMETER"
      subType="COMMANDED" type="POSITION" units="MILLIMETER"/>
    <DataItem category="SAMPLE" id="n3" name="Xload" nativeUnits="PERCENT"
      type="LOAD" units="PERCENT"/>
    <DataItem category="CONDITION" id="Xloadc" type="LOAD"/>
  </DataItems>
</Linear>

```

Figura 5-2- Segunda parte da consulta *probe* para *Live Agent*

A segunda parte do arquivo *probe* começa com um item do tipo *sample*, um item do tipo *Event* e um item do tipo *Condition* ainda referentes ao componente de rotação. Esses itens não serão detalhados, pois não foram utilizados neste estudo de caso.

Após toda definição do componente de rotação, mostram-se os dados de um novo componente: o componente linear nomeado como “X”. Esse componente foi utilizado neste estudo de caso, em conjunto com o componente “Y”. O componente Z não foi utilizado neste estudo de caso, pois constatou-se que o simulador *on-line*, apesar de fornecer o componente em *probe*, não aplica movimentação ao mesmo, para viabilizar a obtenção de seus dados via consultas do tipo *sample*.

Os dados dos componentes lineares são: “*Xact*”, “*Xcom*”, “*Xload*” e “*Xloadc*”. Apenas o item “*Xact*” foi utilizado neste estudo de caso. Este dado informa a posição atual do

componente com relação ao eixo X. Os demais dados informam, respectivamente, sua posição de entrada, a carga e o estado do componente.

```
▼<Linear id="y1" name="Y">
  ▼<DataItems>
    <DataItem category="SAMPLE" id="y2" name="Yact" nativeUnits="MILLIMETER"
      subType="ACTUAL" type="POSITION" units="MILLIMETER"/>
    <DataItem category="SAMPLE" id="y3" name="Ycom" nativeUnits="MILLIMETER"
      subType="COMMANDED" type="POSITION" units="MILLIMETER"/>
    <DataItem category="SAMPLE" id="y4" name="Yload" nativeUnits="PERCENT"
      type="LOAD" units="PERCENT"/>
    <DataItem category="CONDITION" id="Yloadc" type="LOAD"/>
  </DataItems>
</Linear>
▼<Linear id="z1" name="Z">
  ▼<DataItems>
    <DataItem category="SAMPLE" id="z2" name="Zact" nativeUnits="MILLIMETER"
      subType="ACTUAL" type="POSITION" units="MILLIMETER"/>
    <DataItem category="SAMPLE" id="z3" name="Zcom" nativeUnits="MILLIMETER"
      subType="COMMANDED" type="POSITION" units="MILLIMETER"/>
    <DataItem category="SAMPLE" id="z4" name="Zload" nativeUnits="PERCENT"
      type="LOAD" units="PERCENT"/>
    <DataItem category="CONDITION" id="Zloadc" type="LOAD"/>
  </DataItems>
</Linear>
</Components>
</Axes>
```

Figura 5-3- Terceira parte da consulta *probe* para *Live Agent*

A figura acima mostra dois componentes lineares Y e Z, com funcionamento idêntico ao descrito para o componente linear X.

```

▼<Controller id="cn1" name="controller">
  ▼<DataItems>
    <DataItem category="EVENT" id="msg" type="MESSAGE"/>
    <DataItem category="EVENT" id="estop" type="EMERGENCY_STOP"/>
    <DataItem category="CONDITION" id="clp" type="LOGIC_PROGRAM"/>
  </DataItems>
  ▼<Components>
    ▼<Path id="pth" name="path">
      ▼<DataItems>
        <DataItem category="EVENT" id="cn2" name="block" type="BLOCK"/>
        <DataItem category="EVENT" id="cn3" name="mode" type="CONTROLLER_MODE"/>
        <DataItem category="EVENT" id="cn4" name="line" type="LINE"/>
        <DataItem category="EVENT" id="cn5" name="program" type="PROGRAM"/>
        <DataItem category="EVENT" id="cn6" name="execution" type="EXECUTION"/>
        <DataItem category="SAMPLE" id="Ppos" nativeUnits="MILLIMETER_3D"
          subType="ACTUAL" type="PATH_POSITION" units="MILLIMETER_3D"/>
        ▼<DataItem category="SAMPLE" id="Frt" nativeUnits="MILLIMETER/SECOND"
          type="PATH_FEEDRATE" units="MILLIMETER/SECOND">
          <Source>path_feedrate</Source>
        </DataItem>
        ▼<DataItem category="SAMPLE" id="Fovr" nativeUnits="PERCENT" type="PATH_FEEDRATE"
          units="PERCENT">
          <Source>feed_ovr</Source>
        </DataItem>
      </DataItems>
    </Path>
  </Components>
</Controller>

```

Figura 5-4- Quarta parte da consulta *probe* para *Live Agent*

O componente controlador (*Controller*), seus itens e subcomponentes, mostrados na figura acima, não foram utilizados neste estudo de caso, portanto não serão detalhados. O componente *Systems*, mostrado a seguir, também não teve seus dados utilizados.

```

▼<Systems id="systems" name="systems">
  ▼<Components>
    ▼<Electric id="el" name="electric">
      ▼<DataItems>
        <DataItem category="EVENT" id="p2" name="power" type="POWER_STATE"/>
        <DataItem category="SAMPLE" id="freq" name="frequency" nativeUnits="HERTZ"
        type="FREQUENCY" units="HERTZ"/>
        <DataItem category="SAMPLE" id="power_sum" nativeUnits="WATT" type="WATTAGE"
        units="WATT"/>
        <DataItem category="SAMPLE" id="power_a" nativeUnits="WATT" type="WATTAGE"
        units="WATT"/>
        <DataItem category="SAMPLE" id="power_b" nativeUnits="WATT" type="WATTAGE"
        units="WATT"/>
        <DataItem category="SAMPLE" id="power_c" nativeUnits="WATT" type="WATTAGE"
        units="WATT"/>
        <DataItem category="SAMPLE" id="volt_a" nativeUnits="VOLT" type="VOLTAGE"
        units="VOLT"/>
        <DataItem category="SAMPLE" id="volt_b" nativeUnits="VOLT" type="VOLTAGE"
        units="VOLT"/>
        <DataItem category="SAMPLE" id="volt_c" name="volt_c" nativeUnits="VOLT"
        type="VOLTAGE" units="VOLT"/>
        <DataItem category="SAMPLE" id="amp_a" nativeUnits="AMPERE" type="AMPERAGE"
        units="AMPERE"/>
        <DataItem category="SAMPLE" id="amp_b" nativeUnits="AMPERE" type="AMPERAGE"
        units="AMPERE"/>
        <DataItem category="SAMPLE" id="amp_c" nativeUnits="AMPERE" type="AMPERAGE"
        units="AMPERE"/>
      </DataItems>
    </Electric>
  </Components>
</Systems>

```

Figura 5-5- Quinta parte da consulta *probe* para *Live Agent*

Todos os componentes não utilizados neste estudo de caso podem ser facilmente adicionados à metodologia e incorporados em outros estudos de caso.

A figura a seguir mostra os últimos itens do componente de sistema e encerra as *tags* de XML.


```

▼<Coolant id="cool" name="coolant">
  ▼<DataItems>
    <DataItem category="CONDITION" id="clow" type="LEVEL"/>
  </DataItems>
</Coolant>
▼<Hydraulic id="hsys" name="hydrolic">
  ▼<DataItems>
    <DataItem category="CONDITION" id="hlow" type="LEVEL"/>
    <DataItem category="CONDITION" id="hpres" type="PRESSURE"/>
    <DataItem category="CONDITION" id="htemp" type="TEMPERATURE"/>
  </DataItems>
</Hydraulic>
</Components>
</Systems>
</Components>
</Device>
</Devices>
</MTConnectDevices>

```

Figura 5-6- Sexta e última parte da consulta *probe* para *Live Agent*

Como visto acima, uma consulta *probe* retorna um documento com todas as informações de todos os dispositivos englobados por um agente. Através desse documento é possível estipular quais dados estão disponíveis e quais dados serão utilizados em uma aplicação. No caso da aplicação virtual deste estudo de caso, optou-se por utilizar apenas os dados de eixo: *Xact* e *Yact*.

A requisição *probe* foi utilizada apenas uma vez durante o desenvolvimento da aplicação, visando listar as capacidades do *MTConnect Live Agent*. Conhecidas as capacidades do dispositivo simulado, pôde-se implementar a parte da aplicação responsável pelas requisições de dados, através apenas de requisições *sample* e *current*.

Uma requisição *current* retorna sempre um arquivo XML do tipo *MTConnectStreams*. Na aplicação virtual implementada, as requisições *current* são utilizadas apenas na inicialização de cada dispositivo. O parâmetro *nextSequence* é obtido e armazenado em uma variável de mesmo nome, dentro do objeto *Device*. Após a inicialização de um

dispositivo, apenas requisições do tipo *sample* são realizadas, não havendo mais requisições *probe* ou *current*.

Estipulou-se, nesta metodologia, que requisições *sample* ou *current* sempre estariam associadas ao nome do dispositivo passado como parâmetro para a aplicação principal. Em outras palavras, o usuário deve conhecer previamente o nome do dispositivo englobado pelo agente e utilizar esse mesmo nome para a criação do arquivo VRML que será passado como parâmetro. Neste estudo de caso o nome do dispositivo simulado pelo *Live Agent* é “VMC-3Axis” (dado obtido pela consulta *probe*). Portanto, este nome foi usado para nomear o arquivo VRML exportado do *3D Studio Max* e, internamente, foi utilizado, também, para a requisição *current*, resultando: <http://agent.mtconnect.org/VMC-3Axis/current>.

Figura 5-7- Montagem interna da requisição *current* para o estudo de caso

Com finalidade de sintetização, os valores da resposta de *current* mostrados a seguir referem-se somente aos dados selecionados para esta parte do estudo de caso: *Xact* e *Yact*. Os outros dados fornecidos por *current* foram ignorados.

```
<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.2"
xmlns="urn:mtconnect.org:MTConnectStreams:1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.2
http://www.mtconnect.org/schemas/MTConnectStreams_1.2.xsd">
  <Header creationTime="2012-07-29T02:12:30Z" sender="mtconnect" instanceId="1340212647"
version="1.2.0.10" bufferSize="131072" nextSequence="649372632" firstSequence="649241560"
lastSequence="649372631"/>
```

Figura 5-8- Parâmetro *nextSequence* de *current*

O valor de *nextSequence* é usado para as requisições *sample* com parâmetro *from*.

A montagem interna das requisições *sample* para o VMC-3Axis é mostrada a seguir.

Figura 5-9- Montagem interna da requisição *sample* para o estudo de caso

Na figura acima, os valores das *strings* referem-se a:

- *string1* é o endereço do dispositivo simulado por *Live Agent*. Esse dado é constante para este estudo de caso;
- *string2* é obtido diretamente do nome do dispositivo instanciado, no momento de sua inicialização. É um valor constante para cada dispositivo;
- *string3* é constante e definido por *probe*. Para este estudo de caso definiu-se, baseado nas capacidades de *Live Agent*, utilizar somente os dados dos eixos lineares X e Y;
- *string4* é o valor de *nextSequence*. Esse valor varia a cada nova requisição de *sample*;
- *string5* define a quantidade de dados que será fornecida por cada requisição *sample*. Esse valor foi estipulado como constante para toda a

metodologia, mas pode variar conforme a necessidade da aplicação. Esse valor pode ser alterado pelo módulo responsável pela criação dos parâmetros.

Os dados obtidos por uma requisição *sample* são armazenados em arquivos XML locais, no mesmo diretório do arquivo executável da aplicação. Os nomes dos arquivos são criados sequencialmente, conforme o número de dispositivos, sendo iniciados por “*dev*”, seguido pelo número de sequência. A figura a seguir mostra dois arquivos criados, para uma aplicação virtual que simule dois dispositivos.

Figura 5-10- Arquivos XML obtidos do agente para dois dispositivos

Dentro desses arquivos encontram-se os dados das requisições *sample* parametrizada pelos valores das *strings* descritas acima. As figuras seguintes mostram, resumidamente, esses valores.

```

▼<ComponentStream component="Linear" name="X" componentId="x1">
  ▼<Samples>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.619708" name="Xact"
sequence="649813943" subType="ACTUAL">-1.5563164949</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.635708" name="Xact"
sequence="649813947" subType="ACTUAL">-1.5570951700</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.651707" name="Xact"
sequence="649813951" subType="ACTUAL">-1.5577180386</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.667707" name="Xact"
sequence="649813955" subType="ACTUAL">-1.5581852198</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.679707" name="Xact"
sequence="649813959" subType="ACTUAL">-1.5586522818</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.695706" name="Xact"
sequence="649813963" subType="ACTUAL">-1.5591194630</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.711706" name="Xact"
sequence="649813967" subType="ACTUAL">-1.5594309568</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.723706" name="Xact"
sequence="649813971" subType="ACTUAL">-1.5598981380</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.739705" name="Xact"
sequence="649813975" subType="ACTUAL">-1.5603652000</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.755705" name="Xact"
sequence="649813979" subType="ACTUAL">-1.5608323812</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.771704" name="Xact"
sequence="649813983" subType="ACTUAL">-1.5612995625</Position>
    <Position dataItemId="x2" timestamp="2012-07-29T02:39:10.799704" name="Xact"
sequence="649813987" subType="ACTUAL">-1.5623896122</Position>

    .
    .
    .

```

Figura 5-11- Dados de X adquiridos pela requisição *sample* parametrizada por *Position*

```

▼<ComponentStream component="Linear" name="Y" componentId="y1">
  ▼<Samples>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.619708" name="Yact"
sequence="649813944" subType="ACTUAL">-0.2258290350</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.635708" name="Yact"
sequence="649813948" subType="ACTUAL">-0.2178670168</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.651707" name="Yact"
sequence="649813952" subType="ACTUAL">-0.2114973962</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.667707" name="Yact"
sequence="649813956" subType="ACTUAL">-0.2067201883</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.679707" name="Yact"
sequence="649813960" subType="ACTUAL">-0.2019429803</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.695706" name="Yact"
sequence="649813964" subType="ACTUAL">-0.1971657574</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.711706" name="Yact"
sequence="649813968" subType="ACTUAL">-0.1939809620</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.723706" name="Yact"
sequence="649813972" subType="ACTUAL">-0.1892037392</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.739705" name="Yact"
sequence="649813976" subType="ACTUAL">-0.1844265312</Position>
    <Position dataItemId="y2" timestamp="2012-07-29T02:39:10.755705" name="Yact"
sequence="649813980" subType="ACTUAL">-0.1796493232</Position>
    .
    .
    .

```

Figura 5-12- Dados de Y adquiridos pela requisição *sample* parametrizada por *Position*

Utilizando-se o parâmetro *count* com valor “1000”, o arquivo XML de resposta a *sample* conterá mil valores de sequência, entre todos os tipos de dados. Após a finalização do *download* do arquivo, todos os valores de X e Y são armazenados sequencialmente nos vetores *buffer* de cada dispositivo instanciado. Esses vetores são ordenados com base no número de sequência de cada um dos dados *Position*, para simplificação da utilização dos dados, no *looping* principal da aplicação.

As figuras a seguir mostram, respectivamente, os dados no arquivo XML, no *buffer* e o resultado do posicionamento de X e Y dentro da aplicação virtual.

```

▼<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.2"
  xmlns="urn:mtconnect.org:MTConnectStreams:1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.2
http://www.mtconnect.org/schemas/MTConnectStreams_1.2.xsd">
  <Header creationTime="2012-07-31T18:03:10Z" sender="mtconnect"
    instanceId="1340212647" version="1.2.0.10" bufferSize="131072"
    nextSequence="694576407" firstSequence="694461887"
    lastSequence="694592958"/>
▼<Streams>
  ▼<DeviceStream name="VMC-3Axis" uuid="000">
    ▼<ComponentStream component="Linear" name="X" componentId="x1">
      ▼<Samples>
        <Position dataItemId="x2" timestamp="2012-07-31T18:01:10.292582"
          name="Xact" sequence="694575409"
          subType="ACTUAL">1.6287847757</Position>
        <Position dataItemId="x2" timestamp="2012-07-31T18:01:10.308581"
          name="Xact" sequence="694575413"
          subType="ACTUAL">1.6305388212</Position>
        <Position dataItemId="x2" timestamp="2012-07-31T18:01:10.324581"
          name="Xact" sequence="694575417"
          subType="ACTUAL">1.6346316338</Position>
        <Position dataItemId="x2" timestamp="2012-07-31T18:01:10.352580"
          name="Xact" sequence="694575421"
          subType="ACTUAL">1.6375551224</Position>
      
```

Figura 5-13- Dados de posicionamento dentro do arquivo XML


```

SEQ: 695032105, AXIS: 0, COMP: 0, DATA: 0.839937
SEQ: 695032106, AXIS: 1, COMP: 0, DATA: -1.37498
SEQ: 695032107, AXIS: 0, COMP: 0, DATA: 0.835788
SEQ: 695032108, AXIS: 1, COMP: 0, DATA: -1.37739
SEQ: 695032111, AXIS: 0, COMP: 0, DATA: 0.830948
SEQ: 695032112, AXIS: 1, COMP: 0, DATA: -1.38021
SEQ: 695032115, AXIS: 0, COMP: 0, DATA: 0.827145
SEQ: 695032116, AXIS: 1, COMP: 0, DATA: -1.38242
SEQ: 695032119, AXIS: 0, COMP: 0, DATA: 0.81781
SEQ: 695032120, AXIS: 1, COMP: 0, DATA: -1.38785
SEQ: 695032123, AXIS: 0, COMP: 0, DATA: 0.815044
SEQ: 695032124, AXIS: 1, COMP: 0, DATA: -1.38946
SEQ: 695032127, AXIS: 0, COMP: 0, DATA: 0.810896
SEQ: 695032128, AXIS: 1, COMP: 0, DATA: -1.39188
SEQ: 695032131, AXIS: 0, COMP: 0, DATA: 0.805364
SEQ: 695032132, AXIS: 1, COMP: 0, DATA: -1.39509
SEQ: 695032135, AXIS: 0, COMP: 0, DATA: 0.802598
SEQ: 695032136, AXIS: 1, COMP: 0, DATA: -1.3967
SEQ: 695032139, AXIS: 0, COMP: 0, DATA: 0.798449
SEQ: 695032140, AXIS: 1, COMP: 0, DATA: -1.39912
SEQ: 695032143, AXIS: 0, COMP: 0, DATA: 0.792918
SEQ: 695032144, AXIS: 1, COMP: 0, DATA: -1.40234
SEQ: 695032147, AXIS: 0, COMP: 0, DATA: 0.790152
SEQ: 695032148, AXIS: 1, COMP: 0, DATA: -1.40395
SEQ: 695032151, AXIS: 0, COMP: 0, DATA: 0.781855
SEQ: 695032152, AXIS: 1, COMP: 0, DATA: -1.40877
SEQ: 695032155, AXIS: 0, COMP: 0, DATA: 0.77736
SEQ: 695032156, AXIS: 1, COMP: 0, DATA: -1.41139
SEQ: 695032159, AXIS: 0, COMP: 0, DATA: 0.773557

```

Figura 5-14- Dados de posicionamento no *buffer* da aplicação principal.

Figura 5-15- Dados de posicionamento aplicados à mesa do dispositivo

5.2 MÓDULO DE SIMULAÇÃO DE DADOS REAIS

No tópico anterior foram mostrados os resultados obtidos com a utilização do simulador *Live Agent*, para os dados dos eixos lineares X e Y. Porém, esse simulador não fornece uma variação de tipos de dados suficientemente grande para as necessidades desta pesquisa. Assim, desenvolveu-se um módulo externo específico para a simulação dos demais dados e consequente validação da metodologia.

Os dados simulados pelo módulo de simulação de dados reais, para este estudo de caso, são:

- *Part_Count*: utilizado para validação da sincronização entre vários dispositivos numa mesma instância da aplicação virtual;
- *Rotary_Velocity*: utilizado para simular o comportamento de componentes com eixos de rotação, como ferramentas de máquinas;
- *Zact*: utilizado para simular o comportamento de dispositivos com eixos-árvore verticais, neste caso, uma furadeira;
- *Execution*: utilizado para simular o estado de execução do dispositivo;
- *Emergency_Stop*: utilizado para simular o estado da parada de emergência;
- *Door_State*: utilizado para simular o estado da porta do dispositivo.

O módulo foi hospedado em um computador externo à rede de computadores da aplicação principal, para que fosse possível simular o comportamento de uma máquina-ferramenta, com dados obtidos à distância, pela Internet. Inicialmente criou-se uma conexão

socket implementada para viabilizar a troca de dados entre o computador principal da CAVE e o computador responsável pelo módulo de simulação de dados reais. Porém, visando a interligação e unificação dos métodos de aquisição dentro da aplicação virtual, decidiu-se por implementar um método de transmissão de dados via arquivos XML MTConnect, mesmo para os dados do simulador desenvolvido. A figura a seguir mostra os componentes deste módulo, onde o computador principal da CAVE faz o *download* do arquivo XML diretamente do servidor que hospeda o módulo.

Figura 5-16- Comunicação entre o módulo simulador de dados reais e a aplicação

O módulo simulador de dados reais cria um arquivo XML com estrutura idêntica ao padrão fornecido pelo MTConnect, porém, contendo apenas os dados listados acima.

Assim, a aplicação principal pode receber e utilizar os dados desse módulo exatamente como utiliza os dados de um dispositivo real, ou do simulador *Live Agent*.

A estrutura resumida do arquivo XML criado pelo simulador de dados reais, para este estudo de caso é mostrada a seguir.

```

▼<Samples>
  <RotaryVelocity dataItemId="c2" timestamp="2012-07-12T22:56:28.820305" name="Sspeed"
    sequence="231" subType="ACTUAL">1000</RotaryVelocity>
</Samples>
</ComponentStream>
▼<ComponentStream component="Linear" name="Z" componentId="z1">
  ▼<Samples>
    <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.820305" name="Zact"
      sequence="0" subType="ACTUAL">0.11</Position>
    <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.836305" name="Zact"
      sequence="1" subType="ACTUAL">0.12</Position>
    <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.852305" name="Zact"
      sequence="2" subType="ACTUAL">0.13</Position>
    <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.868304" name="Zact"
      sequence="3" subType="ACTUAL">0.14</Position>
    <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.884304" name="Zact"
      sequence="4" subType="ACTUAL">0.15</Position>
    <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.912303" name="Zact"
      sequence="5" subType="ACTUAL">0.16</Position>
  </Samples>
</ComponentStream>
▼<ComponentStream component="Controller" name="Controller" componentId="cont">
  ▼<Events>
    <PartCount dataItemId="all" timestamp="2010-11-22T17:55:38.00932" name="all"
      sequence="921">4</PartCount>
    <PartCount dataItemId="bad" timestamp="2010-11-22T13:33:00.03282" name="bad"
      sequence="108">0</PartCount>
    <PartCount dataItemId="good" timestamp="2010-11-22T13:33:00.03282" name="good"
      sequence="107">0</PartCount>
    <EmergencyStop dataItemId="estop" timestamp="2010-11-22T13:33:00.03282" name="estop"
      sequence="89">TRIGGERED</EmergencyStop>
    <Execution dataItemId="exec" timestamp="2010-11-22T13:33:00.03282" name="execution"
      sequence="91">STOPPED</Execution>
    <DoorState dataItemId="doorstate" timestamp="2010-11-22T13:33:00.03282" name="door"
      sequence="107">CLOSED</DoorState>
  </Events>

```

Figura 5-17- Estrutura do arquivo XML do módulo simulador de dados reais

O arquivo criado possui o nome “furadeira.XML”, pois, para este estudo de caso, o módulo simula dados de uma furadeira de bancada.

Figura 5-18- Arquivo criado pelo módulo simulador para este estudo de caso

De maneira semelhante à aquisição de dados feita no tópico anterior, para o simulador *Live Agent*, os dados do módulo simulador de dados reais são armazenados dentro de vetores de *buffer*, que ficam disponíveis para os procedimentos de utilização de dados, dentro do *looping* principal da aplicação. A figura a seguir ilustra os dados recebidos e sua utilização dentro da aplicação virtual.

```

▼<MTConnectStreams xmlns:m="urn:mtconnect.org:MTConnectStreams:1.2"
  xmlns="urn:mtconnect.org:MTConnectStreams:1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mtconnect.org:MTConnectStreams:1.2
    http://www.mtconnect.org/schemas/MTConnectStreams_1.2.xsd">
  <Header creationTime="2012-07-12T22:57:08Z" sender="mtconnect"
    instanceId="1340212647" version="1.2.0.10" bufferSize="131072"
    nextSequence="376434262" firstSequence="376312120"
    lastSequence="376443191"/>
  ▼<Streams>
    ▼<DeviceStream name="Furadeira" uuid="000">
      ▼<ComponentStream component="Rotary" name="C" componentId="c1">
        ▼<Samples>
          <RotaryVelocity dataItemId="c2" timestamp="2012-07-
            12T22:56:28.820305" name="Sspeed" sequence="231"
            subType="ACTUAL">1000</RotaryVelocity>
        </Samples>
      </ComponentStream>
      ▼<ComponentStream component="Linear" name="Z" componentId="z1">
        ▼<Samples>
          <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.820305"
            name="Zact" sequence="0" subType="ACTUAL">0.11</Position>
          <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.836305"
            name="Zact" sequence="1" subType="ACTUAL">0.12</Position>
          <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.852305"
            name="Zact" sequence="2" subType="ACTUAL">0.13</Position>
          <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.868304"
            name="Zact" sequence="3" subType="ACTUAL">0.14</Position>
          <Position dataItemId="z2" timestamp="2012-07-12T22:56:28.884304"
            name="Zact" sequence="4" subType="ACTUAL">0.15</Position>
        </Samples>
      </ComponentStream>
    </DeviceStream>
  </Streams>
</MTConnectStreams>

```

Figura 5-19- Dados de posicionamento do eixo Z da furadeira no arquivo XML

Figura 5-20- Dados de posicionamento da furadeira no AV

5.3 AQUISIÇÃO DE DADOS REAIS DE CNC FANUC

Além dos estudos de caso com base na aquisição de dados *on-line* via simuladores de dados, descrita nos tópicos anteriores, foi realizado, nesta pesquisa, um estudo de caso com base no centro de usinagem ROMI Discovery 800, com CNC GE-FANUC Série 21i. A máquina-ferramenta foi escolhida devido à possibilidade, mesmo que restrita, de acesso à mesma, no prédio do NUMA (Núcleo de Manufatura Avançada), da Escola de Engenharia de São Carlos (EESC-USP).

Para este estudo de caso foi desenvolvido um *módulo* Adaptador específico para o CNC GE-FANUC com base na estrutura fornecida pelo instituto MTConnect e nos trabalhos de Ferraz (2011). O Adaptador desenvolvido adquire dados do CNC e os converte para um formato compreendido pelo agente, através da utilização da biblioteca FOCAS2, específica para CNCs Fanuc.

O sistema completo definido para este estudo de caso é composto pela máquina-ferramenta e seu computador de apoio, pelo computador servidor de dados (Agente), e pelo nó mestre do *Cluster* de computadores da CAVE, conforme ilustrado na figura a seguir.

Baseado em: Ferraz (2011)

Figura 5-21- Estudo de caso para ROMI D800 com CNC FANUC

O centro de usinagem ROMI D800 não é compatível com o padrão MTConnect, portanto há necessidade de instalação de um *módulo* Adaptador no computador de apoio da máquina. Esse adaptador pode, também, ser embarcado diretamente no PC de IHM (Interface Homem Máquina) da máquina. Pacotes de dados são enviados do Adaptador ao Agente, localizado no servidor. A função do Agente é adquirir os dados do adaptador e transformá-los para o padrão MTConnect, para que possam ser enviados em arquivos XML, conforme as requisições realizadas pelo computador principal do *cluster* da CAVE (nó mestre).

O *software* Agente utilizado neste desenvolvimento foi fornecido pelo MTConnectTM Institute, através de seu *website*. O código-fonte em C++ também é fornecido

e pode ser livremente modificado, porém esse *software* será utilizado sem modificações, pois já contém todas as funções necessárias para o funcionamento do sistema:

- Conectar-se aos adaptadores configurados externamente;
- Receber os dados dos adaptadores no padrão MTConnect definido para um adaptador;
- Converter os dados recebidos em dados do padrão MTConnect para agentes;
- Disponibilizar os dados convertidos em formato MTConnect, em um servidor HTTP.

A figura a seguir exibe uma amostra de dados de um adaptador MTConnect.

```
2012-04-20-T19:05:12.456325!Xact!1.6287847757
2012-04-20-T19:05:12.602214!Xact!1.6305388212
2012-04-20-T19:05:12.754102!Xact!1.6346316338
2012-04-20-T19:05:12.890045!Xact!1.6375551224
```

Figura 5-22- Amostra de dados de posicionamento do adaptador

Os dados do agente, via XML *MTConnectStreams*, bem como o posicionamento dos componentes do torno são análogos aos mostrados no estudo de caso do *Live Agent*, portanto serão omitidos.

O estudo de caso com dados reais foi limitado às restrições de acesso às máquinas-ferramenta reais. Porém, após a escrita dos dados em XML o procedimento restante para inclusão dos dados no AV, para qualquer forma de aquisição de dados, é exatamente o mesmo. Portanto, a aquisição de dados diretamente do adaptador, em conjunto com os testes realizados com simuladores de dados reais, foram considerados suficientes para a validação

da aquisição de dados *on-line* via MTConnect, bem como da sua utilização dentro de ambientes virtuais imersivos.

5.4 AQUISIÇÃO DE DADOS REAIS ADICIONAIS ADQUIRIDOS POR CÂMERA

O estudo de caso para avaliação do tópico referente à metodologia que viabiliza a aquisição de dados via câmeras de baixo custo foi realizado com base na movimentação da porta do dispositivo ROMI D600 situado no laboratório NEPAS da Escola de Engenharia de São Carlos (EESC-USP). Todavia, validado este método, pode-se utilizar o mesmo para diversos tipos de dados diferentes, como descrito no capítulo anterior.

A câmera utilizada neste estudo de caso foi uma *webcam* simples, da marca *Godlship Nightvision*:

Figura 5-23- Câmera de baixo custo utilizada para aquisição de imagens

Primeiramente, construíram-se marcadores quadrados, de papel cartão, nas cores RGB, vermelho, verde e azul, com aproximadamente 20 centímetros de lado. A figura a seguir mostra uma foto dos marcadores sobre uma mesa.

Figura 5-24- Marcadores de papel cartão, vermelho, verde e azul.

Após a construção dos marcadores, testou-se o algoritmo de segmentação para todos os marcadores e para algumas posições diferentes de câmera. Os resultados são mostrados a seguir. Para o marcador vermelho teve-se que cobrir a logomarca do dispositivo, devido à sua cor ser semelhante à cor do marcador. Os primeiros *frames* adquiridos pela câmera se referem à imagem sem processamento ou segmentação. Essas imagens são mostradas nas figuras seguintes.

Figura 5-25- Marcador vermelho acoplado à porta do dispositivo

Figura 5-26- Separação do canal vermelho da imagem via *Threshold*

Figura 5-27- Separação do canal vermelho com logomarca coberta manualmente

Os resultados mostrados acima foram baseados em *Threshold* e separação de canais RGB da imagem. Para situações que exijam melhor definição de segmentação, pode-se utilizar o tratamento da imagem em HSV. O tratamento da imagem em HSV é mostrado a seguir.

Figura 5-28- Imagem convertida para HSV

Figura 5-29- Segmentação com HSV

A última parte do algoritmo foi a identificação da maior figura fechada, para aquisição dos dados X e Y de seu centroide. Os valores do centroide da figura foram armazenado em um arquivo XML do tipo *MTConnectStreams*, para acesso da aplicação principal.

Figura 5-30- Identificação da figura fechada e seu centroide X, Y

Figura 5-31- Dispositivo com porta fechada

Figura 5-32- Dispositivo com porta entreaberta

A segmentação com base em cores, realizada por métodos da biblioteca *OpenCV*, proporcionou o reconhecimento adequado do marcador dentro da imagem capturada pela *webcam*. Esse procedimento foi utilizado aqui apenas para um eixo de uma porta de dispositivo, mas pode ser facilmente adequado para funcionamento com objetos que possuam movimentação em mais de um eixo.

5.5 AMBIENTE VIRTUAL HÍBRIDO

O último estudo de caso utilizou todos os testes independentes, descritos acima, em um único ambiente virtual, a fim de testar a sincronização entre as máquinas. Para isso,

foram instanciados no ambiente, um dispositivo virtual baseado em dados do simulador de dados reais desenvolvido nesta metodologia e um dispositivo baseado em dados adquiridos pelo MTConnect *Live Agent*. O primeiro simulou uma furadeira de bancada e o segundo simulou um centro de usinagem. A relação entre os dispositivos foi definida no módulo de criação de parâmetros, instanciando os dispositivos na seguinte ordem: furadeira, transportador e torno. A figura a seguir mostra a disposição das máquinas dentro da cena virtual.

Figura 5-33- Ambiente virtual híbrido

A furadeira foi controlada em outro computador, através da Internet. Seus movimentos foram restritos à translação no eixo Z e rotação da ferramenta. Além disso, o usuário controlador da furadeira podia informar o término da operação de uma peça, transmitindo o dado no mesmo arquivo XML *MTConnectStream* das informações de Z e C.

Como a furadeira foi configurada para se relacionar diretamente com o transportador, este aguardava dados de entrada provenientes da furadeira, mais especificamente, o dado *part_Count*, responsável por informar a conclusão de operações em peças. Assim que o transportador recebia essa informação, ativava seu comportamento de movimentação e, em seguida, informava o dispositivo subsequente, no caso, o torno, que havia peça disponível para operação. O torno verificava constantemente seus dados de entrada, no caso, os dados de disponibilidade de peça, fornecidos pelo dispositivo imediatamente anterior, o transportador, e, imediatamente após a verificação de disponibilidade de peça, o torno ativava seu comportamento de movimentação. Os fluxogramas mostrados na figura a seguir apresentam as principais relações entre os componentes do sistema híbrido testado neste estudo de caso.

Figura 5-34- Relações envolvidas no ambiente híbrido testado

6 CONCLUSÃO

Este trabalho apresentou uma metodologia para integração, *on-line*, de dados reais com ambientes de realidade virtual imersivos, como, por exemplo, dispositivos CAVE, direcionada, principalmente, à manufatura virtual. A utilização de dados virtuais e reais, conjuntamente, em um ambiente virtual, classifica-o como um sistema virtual híbrido, que agrega inúmeras capacidades e vantagens com relação a um sistema puramente virtual, como mostrado no próximo tópico, contribuição principal. Vários temas e áreas diferentes foram estudados e colaboraram com o resultado final desta pesquisa, bem como com a análise final de suas contribuições e limitações. Durante o desenvolvimento foram utilizados métodos de aquisição de dados reais, específicos para ambientes de manufatura, como a aquisição de dados via protocolo MTConnect e métodos genéricos, como a aquisição de dados via câmeras de baixo custo. As generalizações adotadas na aquisição de dados, na padronização da comunicação e na criação da estrutura e hierarquia da metodologia, possibilitam grande capacidade de extensão e flexibilidade. A metodologia, foi, também, testada e validada através de estudos de caso e implementada, em geral, com base em bibliotecas e *softwares* livres e dispositivos de baixo custo.

Este capítulo apresenta a contribuição principal desta pesquisa e as demais contribuições, originadas das áreas estudadas, ferramentas desenvolvidas e ideias compartilhadas. Além disso, o capítulo descreve as considerações finais e limitações observadas durante o desenvolvimento. Por fim, os trabalhos futuros, derivados do desenvolvimento desta metodologia, são indicados.

6.1 CONTRIBUIÇÃO PRINCIPAL

A contribuição principal deste trabalho foi a viabilização e padronização do uso de dados reais, de maneira *on-line*, dentro de ambientes virtuais imersivos híbridos, voltadas, principalmente, para a manufatura virtual. Este desenvolvimento visou aumentar a confiabilidade de tomadas de decisão baseadas em experiências virtuais imersivas, através da maximização do envolvimento de usuários na cena virtual e, também, da confiabilidade originada de dados reais simulados de maneira *on-line*. Além de contribuir com a área de manufatura virtual, os resultados deste trabalho podem, também, ser utilizados como base para quaisquer outras áreas interessadas na utilização de dados reais de forma *on-line*, dentro de AVs. O enriquecimento do nível de confiabilidade das simulações virtuais, obtido com esta pesquisa, deve-se ao fato de que dados utilizados de maneira *on-line* não são previamente armazenados, processados ou modificados. Dessa forma, as simulações virtuais híbridas são abastecidas com dados precisamente iguais aos dados reais, em tempo quase real, limitado apenas pelos tempos de transmissão de mensagens via rede entre adaptadores, agentes e a aplicação principal e o tempo de conversão dos dados nos formatos dos protocolos SHDR e MTConnect (XML). Dados *off-line*, armazenados para utilização posterior, podem distorcer informações de valor ou informações temporais, diminuindo, assim, a qualidade dos resultados de tomadas de decisão, com base em simulações virtuais.

A RVI atrelada a altos padrões de modelagem e renderização tridimensionais possibilita que usuários sintam-se presentes em um ambiente muito semelhante ao ambiente real de interesse de diferentes tipos de projetos. Dessa forma, situações presenciais que possam colocar em risco a vida de participantes podem ser evitadas e simuladas

adequadamente em um ambiente fechado, eliminando, assim, todo o risco envolvido em situações presenciais. Ainda, situações financeiramente ou tecnicamente inviáveis a projetos comerciais ou de pesquisa, como, por exemplo, o deslocamento de um grande número de pessoas ou de grandes máquinas a longas distâncias podem ser, também, evitadas e simuladas adequadamente, eliminando gastos possivelmente exorbitantes.

A metodologia desenvolvida permite não somente a utilização conjunta de dados reais e virtuais, mas, também, a criação de um ambiente virtual inteiramente fundamentado em dados reais. Nesse ambiente, apesar de sua construção virtual, o comportamento de todos os objetos virtuais seria abastecido por dados reais, criando, assim, um novo conceito de simulação virtual híbrida. Diferentes máquinas e objetos reais, de diferentes regiões do planeta, podem ser simulados de maneira *on-line*, conjuntamente, dentro de um mesmo ambiente virtual. Ainda, suas operações podem ser sincronizadas para viabilizar tomadas de decisão a respeito do trabalho conjunto de diferentes tipos de máquinas.

A padronização do método de comunicação entre os módulos externos de aquisição de dados e a aplicação principal de realidade virtual contribui com o nível de flexibilidade da metodologia desenvolvida, possibilitando, dessa forma, sua extensão dentro da área de manufatura e sua adequação a outras áreas.

6.2 CONSIDERAÇÕES FINAIS

O desenvolvimento modular desta metodologia viabiliza a utilização independente de seus principais componentes a outros trabalhos e aplicações. O módulo adaptador de MTConnect, utilizado para a aquisição de dados reais de máquinas-ferramenta, pode ser utilizado, em seu estado original, para diversos modelos de máquinas-ferramenta com mesmo

CNC. Além disso, esse módulo pode ser adaptado rapidamente a máquinas compostas por outras *APIs* de programação, pois a estrutura de código mantém-se a mesma.

Todo o estudo e trabalho desenvolvidos para embasamento desta pesquisa a respeito da montagem, utilização e configuração de sistemas virtuais imersivos, podem ser utilizados por pesquisadores de quaisquer áreas, que estejam interessados em adornar suas pesquisas com aplicações de realidade virtual imersiva.

A metodologia descrita neste trabalho utilizou como base para a comunicação entre seus módulos, a Internet, portanto os dados reais de alimentação da cena virtual podem ser adquiridos de qualquer parte do mundo, o que amplia enormemente suas possibilidades de aplicação.

Outro fator importante que fundamentou este desenvolvimento foi a utilização de arquivos XML para a troca de informações entre os módulos da aplicação. Os arquivos XML foram utilizados para a padronização de informações provenientes de qualquer fonte de dados, possibilitando assim, um formato mais flexível para a aplicação. Desse modo, a aplicação desconhece a fonte de dados, o que possibilita a extensão simplificada de novas fontes, desde que essas utilizem o protocolo de comunicação descrito neste documento.

A biblioteca de objetos virtuais criada, bem como a padronização utilizada para o agrupamento e nomenclatura de seus componentes, pode ser utilizada por quaisquer pesquisadores e pode, também, ser estendida facilmente, através da reutilização de seus componentes e grupos.

O desenvolvimento da classe genérica para comunicação de dados via rede em sistemas de RV imersiva, com funcionamento em *VR Juggler* e *OpenSG*, pode ser utilizada não somente para a construção desta metodologia, como para quaisquer fins dentro da área de RVI.

Por fim, o desenvolvimento do módulo genérico para aquisição, segmentação e processamento de dados de movimentação de objetos, via *câmeras de baixo custo*, pode ser, também, reutilizado para diversos tipos diferentes de aplicação.

6.3 TRABALHOS FUTUROS

A realidade virtual imersiva é fundamentada na qualidade de imersão proporcionada aos usuários. Nesta pesquisa, este fator foi considerado secundário, pois os estudos e resultados descritos tiveram como foco principal a viabilização da utilização de dados reais, de maneira *on-line*, dentro de ambientes virtuais imersivos. Algumas ferramentas e métodos utilizados no desenvolvimento da metodologia afunilaram a capacidade de realismo que poderia ser obtida através de estudos específicos desta área. Portanto, propõe-se como trabalho futuro, entre outros, o estudo aprofundado do realismo de ambientes virtuais imersivos, a fim de melhorar a imersão dos usuários na cena virtual.

Além da extensão do realismo, muitos outros trabalhos futuros podem derivar da metodologia de utilização de dados reais, *on-line*, descrita neste trabalho. A abrangência de tipos de dados do protocolo MTConnect pode ser melhor explorada em trabalhos futuros, mediante o desenvolvimento de novas classes de objetos que herdem a estrutura descrita neste trabalho. Determinados tipos de dados, como o estabelecimento do ponto zero da ferramenta, a modelagem de suportes de fixação de peças à mesa, a captação do momento da troca de ferramenta, entre outros, podem acrescentar enfoque adicional à metodologia. Para isso, a metodologia foi criada com embasamento de técnicas de programação que permitem sua extensão de maneira simplificada.

O arranjo de *layout* da cena virtual pode, também, ser mais bem explorado no módulo de criação de parâmetros que inicializa a aplicação. Diferentes orientações dos componentes da aplicação podem fornecer a esta metodologia uma abordagem ideal, também, para tomadas de decisões na área de arranjos físicos. Além disso, sugere-se como trabalho futuro a modelagem dinâmica de peças, como proposto no tópico 4.12.1, do quarto capítulo.

Por fim, o trabalho desenvolvido pode, até mesmo, conduzir um estudo para tele operação de linhas de produção reais, através de ambientes virtuais imersivos. Porém, no momento, nem todas as máquinas-ferramenta possuem sistemas abertos, que permitam entrada de informações remotas.

REFERÊNCIAS

- ABRATE, F., INDRI, M., LAZZERO, I. e BOTTERO, A. Efficient solutions for programming and safe monitoring of an industrial robot via a virtual cell. 2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM2011), Budapest, Hungary, July 3-7, 2011.
- ADAMS, L. Visualização e Realidade Virtual. São Paulo, SP: Makron Books, 1994.
- AL-KUBATI, A. A. M., SAIF, J. A. M., TAHER, M. A. A. Evaluation of Canny and Otsu Image Segmentation. International Conference on Emerging Trends in Computer and Electronics Engineering (ICETCEE'2012) March 24-25, Dubai, 2012.
- ARAUJO, R. B. Especificação e Análise de um Sistema Distribuído de Realidade Virtual. Tese (Doutorado). Escola Politécnica, Universidade de São Paulo, São Paulo, 1996.
- AZUMA, R. Tracking requirements for augmented reality. Communications of the ACM, v. 36, n. 7, p. 50-51, 1993.
- BACK, M., CHILDS, T., DUNNIGAN, A., FOOTE, J., GATTERPALLY, S., LIEW, B., SHINGU, J., VAUGHAN, J. The Virtual Factory: Exploring 3D worlds as industrial collaboration and control environments. IEEE Virtual Reality 2010, 20-24 March, Waltham, Massachusetts, USA, 2010.
- BANERJEE, P., ZETU, D., "Virtual Manufacturing", John Wiley & Sons, Inc., 2001.
- BEGAULT, D. R. 3-D Sound for Virtual Reality and Multimedia. San Diego, CA, USA: Academic Press Professional, Inc., 1994.
- BEIER, K.-P. A virtual reality check for manufacturers - a short introduction to virtual reality. ASTD - American Society for Training and Development, v. 9, n. 1, p. 14, February 1998.
- BEIER, K.-P. Web-based virtual reality in design and manufacturing applications. Proceedings of COMPIT'2000 - 1st International EuroConference on Computer Applications and Information Technology in the Maritime Industry, Potsdam, Germany, p. 45-55, March 2000.

BILLINGHURST, M.; KATO, H.; POUPYREV, I. The magicbook - moving seamlessly between reality and virtuality. *IEEE Computer Graphics and Applications*, v. 21, n. 3, p. 6-8, 2001.

BOMAN, D. K. International survey: Virtual-environment research. *IEEE Computer*, v. 28, n. 6, p. 57-65, 1995.

CHAI, Y.; ZHOU, Y.; WANG, K. E-Blocks of Computer Disk Drives based on Virtual Manufacturing. *Third International Workshop on Advanced Computational Intelligence*, 2010.

COUNSELL, J. Interacting with real objects in virtual space. *Proceedings of the Eighth International Conference on Information Visualisation (IV'04)*, 2004.

CREAGH, H. Cave automatic virtual environment. *Proceedings of Electrical Insulation Conference and Electrical Manufacturing & Coil Winding Technology Conference*, p. 499-504, 2003.

CRUZ-NEIRA, C. Virtual Reality Based on Multiple Projection Screens: The Cave and its Applications to Computational Science and Engineering. *Tese (Doutorado)*. University of Illinois, Chicago, IL, USA, 1995.

CRUZ-NEIRA, C.; SANDIN, D. J.; DEFANTI, T. A. Surround-screen projection-based virtual reality: The design and implementation of the cave. *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, p. 135-142, 1993.

CRUZ-NEIRA, C. et al. Cave. Audio visual experience automatic virtual environment. *Communications of the ACM*, v. 35, n. 6, p. 64-72, 1992.

DESSOUKY, M., VERMA S., BAILEY, D. E, RICKEL, J., 2001, "A methodology for developing a web-based factory simulator for manufacturing education". *IIE Trans* 33(3):167-180, 2001.

DUVEILLER, G. A conceptual framework to define the spatial resolution requirements for agricultural monitoring using remote sensing. *Remote Sensing of Environment*, v 114, n 11, p 2637-2650, November 2010.

EGMONT-PETERSEN, M., RIDDER, H. e HANDELS, H. Image Processing With Neural Networks—A Review. *Pattern Recognition* 35 (2002) 2279–2301, 2002.

EL-CHAAR, J., BOER, C. R., PEDRAZZOLI, P. Interactive 3D Virtual Environments for Industrial Operation Training and Maintenance. *IEEE*, 2011.

ELLIS, S. R. What are virtual environments? *IEEE Computer Graphics and Applications*, v. 14, n. 1, p. 17-22, 1994.

FARIA, R. R. A., Improving spatial perception through sound field simulation in VR, *VECIMS 2005 – IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems*, Giardini Naxos, Italy, 18-20 July, 2005.

FERRAZ, F. Jr., SILVA, E. J., RONDON L. C., GIUFFRIDA, A. B., ABUJAMRA, R. C., VALENTE, M. O. Supervisão remota da usinagem com o padrão MTConnect. *Máquinas e metais*, editora Aranda, n.556, 2012.

FERRAZ, F. Jr. Desenvolvimento de um sistema de monitoramento e supervisão para o processo de torneamento. São Carlos, 94 p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2002.

FERRAZ, F. Jr. Arquiteturas para monitoramento e supervisão integrados de processos de usinagem em máquinas com controle numérico aberto. Tese (Doutorado) — Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

FERRAZ, F. Jr.; SILVA, E.J.; RONDON, L.C.;GIUFFRIDA, A.D.B; ABUJAMRA, L.C.; VALENTE, C.M.O. Supervisão remota do processo de usinagem utilizando o padrão MTConnect. Não publicado. Consultado em 2011.

FISHER, S. S.; TAZELAAR, J. M. Living in a virtual world. *Byte*, v. 15, n. 7, p. 215-221, 1990.

FREEMAN, E. E FREEMAN E. Use a Cabeça – Padrões de Projeto (*Design Patterns*). Alta Books. 2ª. Edição. ISBN: 978-85-7608-174-6. 496p. 2007.

GAMMA, E., HELM R., JOHNSON R., VLISSIDES J. Design Patterns: Elements of Reusable Object-Oriented Software. ISBN 0-201-63361-2, 395p, 1994.

GARCÍA-PLAZA, E.; NUÑES, P.J.; MARTIN, A.R.; BEAMUD, E. Virtual Machining applied to the teaching of Manufacturing Technology. Materials Science Forum Vol. 692 pp 120-127, 2011.

GIBSON, W. Neuromancer. New York, USA: ACE Books, 1984.

HERNANDEZ-MATIAS, J.C., VIZAN, A., PEREZ-GARCIA, J. e RIOS, J. An integrated modeling framework to support manufacturing system diagnosis for continuous improvement, Robotics and Computer-integrated manufacturing, 24, 2, 187-199, 2008.

JACOBSON, J. Configuring multiscreen displays with existing computer equipment. Proceedings of Conference on Human Factors HFES'2002, p. 760-765, 2002.

JACOBSON, L. Realidade Virtual em Casa. Rio de Janeiro, RJ: Berkeley, 1994.

JAVIER, L. C. e SANTIAGO, A. R. Virtual Reality Environment for Real-Time Simulation, Control and Monitoring of Robotic Manufacturing Flexible Systems. ICAR '97 Monterey, CA, July 7-9, 1997.

JIN, L.; WEN, Z.; ORAIFIGE, I.A. Distributed VR for Collaborative Design and Manufacturing. 11th International Conference Information Visualization, 2007.

KIRNER, C.; KIRNER, T. G. Virtual reality and augmented reality applied to simulation visualization. In: Simulation and Modeling: Current Technologies and Applications. Hershey, NY, USA: IGI Publishing, p. 391-419, 2007.

KRUEGER, M. W.; GIONFRIDDO, T.; HINRICHSEN, K. Videoplace - an artificial reality. CHI '85: Proceedings of The Sigchi Conference On Human Factors In Computing Systems, p. 35-40, 1985.

LEE, H.; BANERJEE, A. A self-configurable large-scale virtual manufacturing environment for collaborative designers, Manufacturing And Construction Virtual Reality 15:21–40, 2011

LESTON, J. Virtual reality: The IT perspective. The Computer Bulletin, Vol.38, No.3, pp. 12-13, 1996.

LIM, M. K.; ZHANG, Z. A mult-agent based manufacturing control strategy for responsive manufacturing. Journal of Materials Processing Technology, 139, pp. 379-384, 2003.

LULIO, L.,C., Técnicas de visão computacional aplicadas ao reconhecimento de cenas naturais e locomoção autônoma em robôs agrícolas móveis. Dissertação (Mestrado) | Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2011.

MANESH, H.F; SCHAEFER, D.; HASHEMIPOUR, M. Information requirements analysis for holonic manufacturing systems in a virtual environment. Int J Adv Manuf Technol 53:385–398, 2011.

MARINOV, V.P.; SEETHARAMU, S. Virtual machining operation: a concept and an example. The International Society for Optical Engineering, 2004.

MILGRAM, P. et al. Augmented reality: A class of displays on the reality-virtuality continuum. SPIE - Telemanipulator and Telepresence Technologies, v. 2351, p. 282-292, 1994.

MINER, N., STANSFIELD, S. A. An Interactive Virtual Reality Simulation System for Robot Control and Operator Training. Sandia National Laboratories Albuquerque, NM, 1994.

MORIE, J. F. Inspiring the future: Merging mass communication, art, entertainment and virtual environments. ACM SIGGRAPH Computer Graphics, v. 28, n. 2, p. 135-138, 1994.

MOUSAVI, M.; AZIS, F. A. State of the art of haptic feedback in virtual reality in manufacturing. International Symposium on Information Technology, v. 4, p. 1-7, 2008.

MTCONNECT Institute. Understanding and Developing MTConnect® Agents and Adapters - Getting started developing your first Agent and Adapter. Link: http://www.mtconnect.org/media/13086/215pm_frampton-_understanding_and_developing_mtconnect__agents_and_adapters.pdf. Accessed in May 2012.

NAVAB, N. Developing Killer Apps for Industrial Augmented Reality. Projects in VR. Editors: Lawrence Rosenblum and Michael Macedonia. Technical University of Munich, May/June 2004.

OLIVEIRA, C. A Real Object Control System For Use In Manufacturing Telepresence Environments, ABCM Symposium Series in Mechatronics - Vol. 3 - pp.653-658, 2008.

OLIVEIRA, C.; SENA, D. C.; VALE, H. M. C.; CARVALHO, H. J. R.; PORTO, A., J., V. Low-cost gesture detection as a form of interaction in a virtual reality multi-projection system.. In: Innovative Developments in Design and Manufacturing : Advanced Research in Virtual and Rapid Prototyping.. London: Taylor & Francis Ltd, v., p. 615-618, 2009.

PALMA, J. G. Metamodelo para a Modelagem e Simulação de Sistemas a Eventos discretos, baseado em Redes de Petri e Realidade Virtual. Uma Aplicação em Sistema de Manufatura. Tese (Doutorado) | Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2001.

PIMENTEL, K.; TEIXEIRA, K. Virtual Reality: Through the New Looking Glass. New York, NY, USA: McGraw-Hill, Inc., 1994.

PORTO, A. J. V.; SOUZA, M.C.F.; RAVELLI, C.A.; BATOCCHIO, A. Manufatura Virtual: Conceituação e Desafios, Gestão & produção v.9, n.3, p.297-312, dez. 2002.

PULUGURTHA S., NAMBISAN S., DANGETI M., KASEKO M. Simulating and analyzing incidents using CORSIM and VISSIM traffic simulation software. In: 7th International conference on: applications of advanced technology in transportation, 2002.

QIU, D. e GAO, L. Application of Virtual Reality Technology in Bridge Structure Safety Monitoring. International Conference on Computer and Information Application (ICCIA 2010), 2010.

RHEINGOLD, H. Virtual Reality. New York: Summit Books, 1991.

ROSATI, R.J. Evaluation of remote monitoring in home health care. Proceedings - International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED 2009, p 151-153, 2009

SAKAMAKI, K.; NAKAMURA, H. Analysis of the degradation tendency of industrial manufacturing devices and development of a general-purpose remote monitoring system. Proceedings of the IEEE International Conference on Industrial Technology, 2009 IEEE ICIT 2009.

SEO, Y.; KIM, D-Y.; SUH, S-H. Development of Web-based CAM system. Int J Adv Manuf Technol 28: 101–108, 2006

SHERMAN, W. R.; CRAIG, A. Understanding Virtual Reality: Interface, Application, and Design. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

SHRIDHAR, J. M.; RAVI, S. Virtual manufacturing: An important aspect of collaborative product commerce. Journal of Advanced Manufacturing Systems, Great Britain, v. 1, p.113{119, 2002.

SKOOGH, A., BENGTSSON, N., MICHALOSKI, J., Towards Continuously Updated Simulation Models: Combining Automated Raw Data Collection And Automated Data Processing. Proceedings of the 2010 Winter Simulation Conference B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds, 2010.

SOARES L. P. et al, Designing a Highly Immersive Interactive Environment: The Virtual Mine, COMPUTER GRAPHICS forum, Volume 29, number 6 pp. 1756–1769, 2010.

SOBEL W., MTConnect® Standard SHDR Protocol Companion Specification Version 1.1.0 – Draft, MTConnect Institute, 2012.

SOUZA, M.; SACCO, M.; PORTO, A.J.V. Virtual manufacturing as a way for the factory of the future. J Intell Manuf 17:725–735, 2006.

SPIŠÁK, J., MIKULA, J., NAŠČÁK, D., ZELKO, M. Advanced monitoring system for granular material thermal treatment. 2012 13th International Carpathian Control Conference (ICCC), 2012.

SUNG, W-T.; OU., S-C. Using Virtual Reality technologies for manufacturing applications. International Journal of Computer Applications in Technology (IJCAT), Vol. 17, No. 4, 2003.

SYMANZIK, J.; COOK, D.; KOHLMAYER, B.D.; CRUZ-NEIRA, C. Dynamic Statistical Graphics in the CAVE Virtual Reality Environment. Working Paper for the Dynamic Statistical Graphics Workshop, Sydney, July 7. Department of Statistics, Iowa State University, Ames, IA, Preprint 96-10, 1996. Available at: http://www.math.usu.edu/~symanzik/papers/1996_dsg.pdf.

TIFFIN, J.; TERASHIMA, N. HyperReality: Paradigm for the Third Millennium. London, UK: Routledge, 2001.

TORRISI, N. M. Monitoring Services for Industrial Applications. Ieee Industrial Electronics Magazine, 2011.

VALE, H. M. C.; OLIVEIRA, C. ; SENA, D. C.; PORTO, A., J., V. Construção de Ambientes de Multiprojeção Imersivos para Tratamento de Fobias no Ambiente de Trabalho. Innovare, v. 9, p. 84-100, 2010.

VALE, H. M. C.; OLIVEIRA, C.; SENA, D. C.; PORTO, A., J., V. Construction of immersive multi-projection environments for treatment of phobia of heights.. In: Innovative Developments in Design and Manufacturing : Advanced Research in Virtual and Rapid Prototyping.. London: Taylor & Francis Ltd, v., p. 619-623, 2009.

VERMEULEN, H., NIEKERK, T. I., HUANG, J., HATTING, D. VRML to monitor and control an industrial robot via the internet. IEEE, 1999.

WANG, L.; SHEN, W.; LANG, S. Wise-shopfloor: A web-based and sensor-driven e-shopfloor. Journal of Computing and Information Science in Engineering, v. 4, p. 56-60, March 2004.

WANG, Y.M.; YIN, H.L.; XIAO, N.F.; JIANG, Y.R. Internet-based remote manipulation and monitoring of an industry robot in advanced manufacturing systems. International Journal of Advanced Manufacturing Technology, v 43, n 9-10, p 907-913, August 2009

WONG, S.F.; YANG, Z.X.; CAO, N.; HO, W.I. Applied RFID and Virtual Reality Technology in professional training system for manufacturing. Proceedings of the 2010 IEEE IEEM, 2010.

XIAOWEI S., et al. A 3D Sound Simulation System for Flight Simulator, Second International Conference on Computer Modeling and Simulation, 2010.

ZHANG, X. et al. The development of virtual manufacturing technology, Applied Mechanics and Materials Vols. 58-60 pp 854-858, 2011.

ZHOU, Z. et al. Interactive entertainment systems using tangible cubes. Proceedings of Australian Workshop on Interactive Entertainment, p. 19-24, 2004.

ZIMMERMAN, T. G. et al. A hand gesture interface device. Chi '87: Proceedings Of The SIGCHI/GI - Conference on Human Factors in Computing Systems and Graphics Interface, p. 189-192, 1987.

BIBLIOGRAFIA CONSULTADA

BLAUERT, J.; BUTLER, R.A. "Spatial Hearing: The Psychophysics of Human Sound Localization by Jens Blauert," The J. Acoustical Soc. of Am., vol. 77, no. 1, pp. 334-335, 1985.

ÇAPIN, T. K. et al. Avatars in Networked Virtual Environments. New York, NY, USA: John Wiley & Sons, 1999.

CARLILE, S. Virtual Auditory Space: Generation and Application. R.G. Landes Company, 1996.

CHURCHILL, E. F.; SNOWDON, D. Collaborative virtual environments: An introductory review of issues and systems. Virtual Reality: Research, Development and Applications, v. 3, n. 1, p. 3-15, 1998.

COMANICIU, D.; MEER, P. (1987) Robust analysis of feature spaces: color image segmentation. In: IEEE Computer Society, 1997. Conference on Computer Vision and Pattern Recognition.

DENG, Y. et al. (1999). Peer group filtering and perceptual color image quantization. In: International Symposium on Circuits and Systems. 1999. IEEE, vol 4, p; 21-25, 1999.

DOERR, K.; RADEMACHER, H.; HUESGEN, S.; KUBBAT, W. Evaluation of a Low-Cost 3D Sound System for Immersive Virtual Reality Training Systems, Ieee Transactions On Visualization And Computer Graphics, Vol. 13, No. 2, 2007.

DRAPER, J. V.; KABER, D. B.; USHER, J. M. Speculations on the value of telepresence. CyberPsychology & Behavior, v. 2, n. 4, p. 349-362, 1999.

FOX, J.; ARENA, D.; BAIENSON, J. N. Virtual Reality A Survival Guide for the Social Scientist, Stanford University, Stanford, CA, USA Journal of Media Psychology 2009; Vol. 21(3):95–113, 2009.

GONZALEZ, R. C.; WINTZ, P. Digital Image Processing, 2 ed. Nova Iorque, EUA: Assison-Wesley Publishing Company, 1987.

GORGULHO, J., H., C., Jr. *Análise Do Desempenho Dos Arranjos Físicos Distribuídos Em Ambiente De Roteamento De Tarefas Com Flexibilidade De Seqüência De Fabricação*, 2007. 427f. Tese (Doutorado em Engenharia Mecânica). Depto. de Eng. Mecânica - Escola de Engenharia de São Carlos (EESC) - Universidade de São Paulo (USP), 2007.

HANSEN, G., A. "Simulating Software Development Processes", IEEE Computer, January 1996.

HARRISON, S.; DOURISH, P. Re-place-ing space: The roles of place and space in collaborative systems. CSCW '96: Proceedings of the 1996 ACM conference on Computer Supported Cooperative Work, New York, NY, USA, p. 67-76, 1996.

LOOMIS, J.M.; HEBERT, C.; CICINELLI, J.G. "Active Localization of Virtual Sounds," The J. Acoustical Soc. of Am., vol. 88, no. 4, pp. 1757-1764, 1990.

MOREIRA, J.; COSTA, L. D. F. Neural-based color image segmentation and classification using self-organizing maps. In: SIBGRAPI, IX, 1996. Anais. P. 47-54, 1996.

NORMANN, R. *Administração de serviços: estratégia e liderança na empresa de serviços*. Trad. de Ailton B. Brandão. São Paulo, Atlas, 1993.

OLIVEIRA, C. *Associação de Redes de Petri com Objetos Virtuais e Reais para Controle de Ambientes Virtuais Imersivos e Telepresença*, 2007.

TIAN, L. F.; SLAUGHTER, D. C. Environment adaptive segmentation algorithm for outdoor image segmentation. Computers and Electronics in Agriculture, v. 21, p. 153-168, 1998.

YUCEER, C.; OFLAZER, K. A rotation, scaling and translation invariant pattern classification system. Pattern Recognition, v. 26, n. 5, 1993.

APÊNDICE A – VOCABULÁRIO MTCONNECT

A seguir, é apresentada a lista completa de termos do vocabulário MTConnect:

- *Acceleration* (aceleração): A aceleração é sempre tratada como um valor numérico em milímetro / segundo 2 ;
- *AccumulatedTime* (tempo acumulado): Referente ao tempo acumulado, em segundos, referente a um componente;
- *Amperage* (Amperagem): Valor medido em Ampère, referente à corrente em determinado circuito;
- *Angle* (Ângulo): Valor em graus de determinado componente;
- *AngularAcceleration* (Aceleração angular): Valor medido em graus/segundo 2 ;
- *AngularVelocity* (Velocidade angular): Taxa de mudança de um ângulo medida em graus/segundo;
- *AxisFeedrate* (Avanço do Eixo): Taxa de movimentação medida em milímetros/segundo, do eixo linear da ferramenta;
- *ClockTime* (Tempo): Leitura de um dispositivo de tempo em um determinado momento. Informado sob o padrão W3C ISO 8601 com format YYYY-MM-

DDThh:mm:ss.ffff, ano, mês, dia, horas, minutos, segundos e casas decimais, respectivamente;

- *Concentration* (concentração): Porcentagem de um componente em meio a outros componentes;
- *Conductivity* (condutividade): Capacidade de um material de conduzir eletricidade. Informada em siemens/metro;
- *Displacement* (deslocamento): Deslocamento medido, em milímetros, quando um objeto altera sua posição;
- *ElectricalEnergy* (energia elétrica): Quantidade de energia elétrica consumida por um componente. Medida em watt/second;
- *Flow* (fluxo): Taxa de fluxo de um fluido em litros/segundo;
- *Frequency* (frequência): Quantidade de repetições de um evento por unidade de tempo. Medida em hertz;
- *FillLevel* (nível): Porcentagem da quantidade restante de substância, comparada com a quantidade máxima estipulada;
- *LinearForce* (força linear): Força de *pull* ou *push* exercida por um atuador sobre um objeto. Medida em newton;

- *Load* (carga): Porcentagem de carga máxima;
- *Mass* (Massa): Quantidade de quilos de um objeto;
- *PathFeedrate* (avanço da trajetória): Taxa de movimentação da trajetória da ferramenta em milímetros/segundo;
- *PathPosition* (posição da trajetória): Posição do programa em três dimensões para X, Y e Z. Exemplo: <PathPosition ...>10.123 55.232 100.981</PathPosition> Onde X = 10.123, Y = 55.232, e Z=100.981;
- *PH*: Alcalinidade ou acidez medida em PH;
- *Position* (posição): Representa a localização em um eixo linear. Medida em milímetros;
- *PowerFactor* (fator de potência): Porcentagem entre a potência real e a potência aparente;
- *Pressure* (pressão): Força exercida por um gás ou líquido, medida em pascal, por unidade de área;

- *Resistance* (resistência): Mede o quanto um objeto se opõe a uma corrente elétrica. Medida em ohm;
- *RotaryVelocity* (velocidade de rotação): Taxa de rotação, medida em revoluções/minuto, de um eixo rotativo;
- *SoundLevel* (nível do som): Volume do som medido em decibéis;
- *Strain* (deformação): Deformação por unidade de comprimento de um objeto. Medida em porcentagem;
- *Temperature* (temperatura): Medida em Celsius;
- *Tilt* (inclinação): Deslocamento angular de um objeto, medido em micro_radiano;
- *Torque*: Força de rotação medida em newton_metro;
- *Velocity* (velocidade): Taxa de mudança de posição em um ou mais eixos, medida em milímetros/segundo;
- *Viscosity* (viscosidade): Medida da resistência do fluido em pascal_segundo;
- *Voltage* (tensão): Potencial elétrico medido em volts;

- *Wattage* (potência): Potencial elétrico dissipado ou consumido por um circuito elétrico ou dispositivo, medido em watt.

APÊNDICE B – ESTRUTURA INTERNA GENÉRICA DE UM ADAPTADOR

A estrutura interna genérica de um adaptador é estritamente dependente da classe abstrata *DeviceDatum*, que força a estrutura básica de suas subclasses concretas. Essas subclasses possuem a responsabilidade de identificação de alterações nos dados e, como consequência, alteram o valor da *flag* de informação de alteração. Essa estrutura é mostrada pelos códigos dos quadros apresentados a seguir.

Quadro B-1- Classe interna *DeviceDatum*

```
class DeviceDatum {
protected:
    char mName[NAME_LEN];
    bool mChanged;
    bool mHasValue;

public:
    DeviceDatum(const char *aName);
    virtual ~DeviceDatum();

    bool changed() { return mChanged; }
    void reset() { mChanged = false; }

    char *getName() { return mName; }
    virtual char *toString(char *aBuffer, int aMaxLen) = 0;
    virtual bool append(StringBuffer &aBuffer);
    virtual bool hasInitialValue();
    virtual bool requiresFlush();
};
```

Quadro B-2- Subclasse interna *Event*

```
class Event : public DeviceDatum
{
protected:
    char mValue[EVENT_VALUE_LEN];

public:
    Event(const char *aName);
    bool setValue(const char *aValue);
    virtual char *toString(char *aBuffer, int aMaxLen);
};
```


Quadro B-3- Subclasse interna *Sample*

```
class Sample : public DeviceDatum
{
    protected:
        double mValue;

    public:
        Sample(const char *aName);
        bool setValue(double aValue);
        virtual char *toString(char *aBuffer, int aMaxLen);
};
```

Toda vez que um dado é adquirido, ele é testado com o valor anterior. Se a informação adquirida for considerada um novo valor, o valor é modificado e a flag *mChanged* é alterada para true. Essa flag informa o adaptador que os dados devem ser enviados novamente ao socket. O quadro apresentado a seguir contém o método da classe *Sample* que realiza essa tarefa.

Quadro B-4- Método de identificação de alteração nos valores

```
bool Sample::setValue(double aValue)
{
    if (aValue != mValue)
    {
        mChanged = true;
        mValue = aValue;
    }
    mHasValue = true;
    return mChanged;
}
```

Os clientes realizarão conexões com o adaptador, através de um número de porta pré-definido, com padrão 7878. O adaptador permite a conexão de diversos clientes ao mesmo tempo. Ele não tenta conexão com o dispositivo até que pelo menos um cliente tenha realizado conexão, o que garante baixo *overhead* do dispositivo.

Quadro B-5- Método de inicialização de servidor de um adaptador

```

void Adapter::startServer()
{
    mServer = new Server(mPort);
    while (1)
    {
        Client **clients = mServer->connectToClients();
        if (clients != 0)
        {
            for (int i = 0; clients[i] != 0; i++)
            {
                sendInitialData(clients[i]);
            }
        }
        mServer->readFromClients();
        if (mServer->numClients() > 0)
        {
            mBuffer.timestamp();
            gatherDeviceData();
            sendChangedData();
            mBuffer.reset();
        }
        sleepMs(mScanDelay);
    }
}

```

O método é executado em *looping* infinito, sempre em busca de novos clientes. Quando um cliente realiza conexão, os dados iniciais são enviados e o adaptador realiza a aquisição de dados do dispositivo. Os métodos de aquisição de dados são específicos para cada *API*. O método *sendChangedData* é responsável por enviar somente dados atualizados.

A.1 DADOS DE UM ADAPTADOR

O formato dos dados de um adaptador seguem o seguinte padrão:

Quadro B-6- Formato padrão dos dados de um adaptador

```
<time_stamp>|<data_item_1>|<value_1>|<data_item_2>|<value_2>|...
```

O *time_stamp* (dados de data e hora) deve seguir o seguinte padrão, onde “YYYY” refere-se ao ano, “MM” ao mês, “DD” ao dia, “hh” à hora, “mm” aos minutos, “ss” aos segundos e “ffff” às casas decimais opcionais:

Quadro B-7- Formato dos dados de *time_stamp*

```
YYYY-MM-DDThh:mm:ss.ffff
```

Exemplos de dados do tipo *Sample* e *Event* são encontrados, portanto, no formato apresentado no próximo quadro.

Quadro B-8- Dados de um adaptador

```
2008-04-20T18:28:18.687603|power|ON|execution|EXECUTING|Xact|1.3549591303|
Yact|0.2879030704|Zact|-0.1000000015|Xcom|1.3561842845|Ycom|0.2824387292|
Zcom|-0.1000000000|spindle_speed|3400.0000000000|program|/home/mtconnect/
emc2/nc_files/examples/spiral.ngc|mode|AUTOMATIC
2008-04-20 18:28:18.797576|Xact|1.3640016317|Yact|0.2413364202|Xcom|
1.3647230322|Ycom|0.2353807124
2008-04-20 18:28:18.907572|Xact|1.3692923784|Yact|0.1976564378|Xcom|
1.3700138636|Ycom|0.1916564378
2008-04-20T18:30:20.927574|Zact|0.7356040478|Zcom|0.7536040307
2008-04-20T18:30:21.037569|Zact|0.8676040173|Zcom|0.8856040307
2008-04-20T18:30:21.147571|Zact|0.9815723896|Zcom|0.9896790496
2008-04-20T18:30:21.307639|execution|IDLE|Zact|1.0000000000|Zcom|
1.0000000000|spindle_speed|0.0000000000
```

Exemplos de dados de *Alarm* são separados por linha e seguem o seguinte padrão:

Quadro B-9- Padrão de dados para o tipo *Alarm*

```
<time_stamp>|<data_item>|<code>|<nativeCode>|<severity>|<state>|<description>
```

Exemplos de dados do tipo *Alarm* são encontrados, na prática, da seguinte forma:

Quadro B-10- Exemplos de dados do tipo *Alarm*

2008-04-20T18:34:51.780917	alarm	MESSAGE	OFF	INFO	INSTANT	Power OFF
2008-04-20T18:34:51.780917	alarm	ESTOP	ESTOP	CRITICAL	ACTIVE	ESTOP Pressed
2008-04-20T18:34:53.314707	alarm	ESTOP	ESTOP	CRITICAL	CLEARED	ESTOP Reset
2008-04-20T18:34:54.437511	alarm	MESSAGE	ON	INFO	INSTANT	Power ON