HERÁCLITO LOPES JAGUARIBE PONTES

**Desenvolvimento de um Ambiente para Simulação da
Manufatura baseado em Features e Realidade Virtual**

Tese apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo para obtenção do título de Doutor em Engenharia Mecânica.

Área de Concentração: Manufatura

Orientador: Prof. Tit. Arthur José Vieira Porto

São Carlos
2012

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

P813d Pontes, Heráclito Lopes Jaguaribe
Desenvolvimento de um Ambiente para Simulação da
Manufatura baseado em Features e Realidade Virtual. /
Heráclito Lopes Jaguaribe Pontes; 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. Simulação. 2. Manufatura. 3. Features. 4.
Realidade Virtual. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **HERÁCLITO LOPES JAGUARIBE PONTES**.

Título da tese: "Desenvolvimento de um ambiente para simulação da manufatura baseado em features e realidade virtual".

Data da defesa: 10/10/2012

Comissão Julgadora:

Resultado:

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

Aprovado.

Prof. Dr. **Roberto Hideaki Tsunaki**
(Escola de Engenharia de São Carlos/EESC)

Aprovado

Dr. **Alexandre José Tiberti**
(Escola de Engenharia de São Carlos/EESC)

Aprovado

Dr. **José Luiz Miranda Junior**
(DCDN Energia)

Aprovado

Prof. Associado **José Reinaldo Silva**
(Escola Politécnica/USP)

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,
pelo apoio e dedicação,
sem os quais, esta conquista
não seria possível.*

AGRADECIMENTOS

Ao meu orientador Professor Titular Arthur José Vieira Porto, pelo apoio e orientação durante a elaboração deste trabalho.

Aos professores Miguel Antônio Bueno da Costa, Mario Luiz Tronco e José Alexandre Tiberti pela atenção disponibilizada no exame de qualificação da tese.

À minha esposa Melina, pelo carinho, amor, compreensão e paciência.

Ao meu filho Pedro, pela sua existência, minha maior felicidade.

À minha mãe Regina, pelo apoio constante, pela compreensão e pela ótima educação que me proporcionou. Ao meu padrasto, André, pelo incentivo, amizade e companheirismo.

Aos amigos pela grande amizade, pelo incentivo e pelas valiosas contribuições.

A todos os colegas, professores e funcionários, em especial a Beth, a Ana Paula e a Michele, do Laboratório de Simulação e Controle e do Departamento de Engenharia Mecânica da EESC/USP, pela colaboração.

RESUMO

PONTES, H. L. J. **Desenvolvimento de um Ambiente para Simulação da Manufatura baseado em *Features* e Realidade Virtual**. 2012. 301f. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

Atualmente, com a diversificação dos produtos demandados, os sistemas de manufatura têm se tornado cada vez mais complexos e de difícil operacionalização. Dificilmente o responsável pela manufatura tem toda a informação necessária para antever os detalhes necessários à sua execução. Por outro lado, as empresas não dispõem de tempo nem de recursos para testar alternativas de manufatura no sistema real. Com isso, a simulação baseada nas *features*, ou seja, na geometria da peça, se torna importante, pois retrata fielmente as características do produto a ser manufaturado, facilitando a gestão da manufatura. Esse trabalho foi estruturado a partir da revisão da literatura sobre os conceitos de sistemas de manufatura, tecnologia de *features*, simulação, realidade virtual e modelos de processos de *software*. O objetivo do trabalho foi desenvolver um ambiente a partir de uma nova abordagem para simulação da manufatura baseado nas *features* das peças com uma interface tridimensional baseada em realidade virtual. A partir dessas simulações, o usuário poderá tomar decisões baseadas em indicadores como: tempo total de manufatura, custo e tempo por recurso de fabricação e movimentação, tempo de espera em fila e taxa de utilização de cada recurso num sistema de manufatura específico sem alterar o cotidiano da empresa. O desenvolvimento do sistema foi executado dentro do paradigma da orientação a objetos e foi dividido em etapas: análise de requisitos, projeto, implementação, verificação e validação e documentação. Para mostrar que as funcionalidades do ambiente de simulação decorrentes da abordagem proposta no trabalho são aplicáveis em sistemas de manufatura reais foram propostas duas aplicações com suas soluções.

Palavras-chave: Simulação. Manufatura. *Features*. Realidade Virtual.

ABSTRACT

PONTES, H. L. J. **Development of Environment for Simulation of Manufacturing based on Features and Virtual Reality**. 2012. 301f. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

Nowadays with the diversification of products demanded, manufacturing systems have become increasingly complex and with difficult operation. Hardly the person responsible for the manufacturing area has all the information necessary to anticipate the details for their implementation. On the other hand, companies don't have the time or resources to test alternatives in the real manufacturing system. With this simulation based on the features, i.e., the part geometry becomes important because it represents faithfully the characteristics of the product being manufactured, making it easy the management of manufacturing. This work was structured from the literature review on the concepts of manufacturing systems, features technology simulation, virtual reality and software process models. The objective of this study was to develop an environment from a new approach for simulation of manufacturing based on features of the parts with a three-dimensional interface based on virtual reality. From these simulations, the user can make decisions based on indicators such as: total time manufacturing, costs and time for manufacturing and resource handling, time of queuing and rate of utilization resource in a manufacturing system without changing the daily life of the company. The development of the system was implemented within the paradigm of object orientation and was divided into steps: requirements analysis, project, implementation, verification and validation and documentation. To show that the functions of the simulation environment due the approach proposed in this work are applicable to real manufacture systems have been proposed two applications with their solutions.

Keywords: Simulation. Manufacturing. Features. Virtual Reality.

LISTA DE FIGURAS

Figura 1.1 - Formas de classificação das pesquisas científicas	29
Figura 2.1 - Sistema de manufatura	34
Figura 2.2 - Principais processos de fabricação	39
Figura 2.3 - Equação da movimentação física.....	55
Figura 2.4 - Exemplo de <i>features</i> em peças	60
Figura 2.5 - <i>Features</i> padrão ao projeto e à manufatura.....	63
Figura 2.6 - Uma peça com diferentes modelos de <i>features</i>	64
Figura 2.7 - <i>Features</i> rotacional e prismática.....	66
Figura 2.8 - Tipos de <i>features</i>	67
Figura 3.1 - Formas de estudo de um sistema	79
Figura 3.2 - Metodologia de estudo de simulação	87
Figura 3.3 - A evolução das ferramentas de simulação	90
Figura 3.4 - Seis graus de liberdade dos elementos de ambiente virtual	105
Figura 4.1 - Chegada de peças no sistema de manufatura	121
Figura 4.2 - Execução da simulação	122
Figura 4.3 - Processamento das filas.....	123
Figura 5.1 - Etapas do desenvolvimento do ambiente de simulação	130
Figura 5.2 - Tipos de features do ambiente de simulação	133
Figura 5.3 - Arquitetura do ambiente de simulação.....	134
Figura 5.4 - Exemplo da interface peças do banco de dados	136
Figura 5.5 - Modelagem do banco de dados	136
Figura 5.6 - Interface features do banco de dados	138
Figura 5.7 - Interface recursos do banco de dados	140
Figura 5.8 - Interface de modelagem com recurso transporte.....	141
Figura 5.9 - Interface de modelagem com recurso fabricação	142
Figura 5.10 - Módulo simulador do ambiente de simulação	144
Figura 5.11 - Interface principal do ambiente de simulação	146
Figura 5.12 - Menus secundários do ambiente de simulação.....	146
Figura 5.13 - Peça modelada com features no ambiente de simulação	148
Figura 5.14 - Exemplo de uma cena 3D do ambiente de simulação.....	148
Figura 5.15 - Tabela de resultados da simulação.....	149
Figura 5.16 - Interface de resultados da simulação	150
Figura 5.17 - Funcionamento do ambiente de simulação	152
Figura 5.18 - Interface de acesso ao ambiente de simulação	153
Figura 5.19 - Interface de gerenciamento de acesso	153
Figura 5.20 - Interface de alteração de senha	154
Figura 5.21 - Interface de saída do ambiente de simulação	154
Figura 5.22 - Fluxograma da entrada de dados no ambiente de simulação	155
Figura 5.23 - Funcionamento da modelagem da simulação.....	157
Figura 5.24 - Desenvolvimento da cena 3D	159
Figura 5.25 - Gráfico de barra do ambiente de simulação	160
Figura 5.26 - Gráfico de pizza do ambiente de simulação	161
Figura 5.27 - Gráfico de linha do ambiente de simulação	161
Figura 5.28 - Exemplo de diagrama de caso de uso para cadastrar operador do sistema	163

Figura 5.29 - Diagrama de pacotes do sistema	165
Figura 5.30 - Diagrama de classe do pacote Core	166
Figura 5.31 - Diagrama de classe do pacote GUI.....	166
Figura 5.32 - Diagrama de classe do pacote Scene.....	167
Figura 5.33 - Diagrama de sequência geral do ambiente de simulação	168
Figura 5.34 - Código para criação de um novo banco de dados.....	171
Figura 5.35 - Código para conectar o sistema num banco de dados existente	172
Figura 5.36 - Código para salvar e carregar da base de dados os atributos da peça.....	172
Figura 5.37 - Exemplos de widgets do Qt.....	174
Figura 5.38 - Exemplo da interface resultados do ambiente de simulação com seus widgets.....	174
Figura 5.39 - Exemplo código da interface principal.....	175
Figura 5.40 - Exemplo da utilização do signal/slot	176
Figura 5.41 - Código do planejamento da simulação	177
Figura 5.42 - Código da chegada peças no ambiente de simulação	178
Figura 5.43 - Código da execução da simulação	179
Figura 5.44 - Código do processamento das filas.....	180
Figura 5.45 - Código do cálculo do tempo e custo de fabricação.....	181
Figura 5.46 - Exemplo de recurso de fabricação no formato “obj”	182
Figura 5.47 - Exemplo de recurso de transporte no formato “obj”	183
Figura 5.48 - Código para importar objetos no formato “obj”	184
Figura 5.49 - Exemplo de diferentes tipos de features inseridas numa peça.....	185
Figura 5.50 - Opções de visualização da cena.....	187
Figura 5.51 - Exemplo da codificação da geometria de uma peça	188
Figura 5.52 - Exemplo da codificação da visualização de uma cena ou da peça com as features	189
Figura 6.1 - Sequência de produção do caso de teste 2.....	196
Figura 6.2 - Sequência de produção do caso de teste 3.....	198
Figura 6.3 - Modelo do primeiro cenário caso de teste 3.....	199
Figura 6.4 - Erro na modelagem da lista de simulação	202
Figura 6.5 - Correção na modelagem da lista de simulação.....	203
Figura 6.6 - Arquivo XML resultados com erro.....	204
Figura 6.7 - Arquivo XML com resultados corretos.....	204
Figura 6.8 - Peça com as features do caso de teste 2	206
Figura 6.9 - Cena 3D simulada no caso de teste 2	206
Figura 7.1 - Passos da aplicação do ambiente de simulação em sistemas de manufatura reais	211
Figura 7.2 - Amostra do corrimão da aplicação 1.....	214
Figura 7.3 - Base de fixação soldada no montante	214
Figura 7.4 - Sistema de manufatura da peça da aplicação 1.....	215
Figura 7.5 - A peça da aplicação 1.....	216
Figura 7.6 - Modelo da peça da aplicação 1	217
Figura 7.7 - Comparativo do tempo de fabricação por furo entre cenários da aplicação 1....	219
Figura 7.8 - Comparativo do custo de fabricação por furo entre cenários da aplicação 1	219
Figura 7.9 - Peça da aplicação 2	221
Figura 7.10 - Projeto da peça da aplicação 2.....	221
Figura 7.11 - Sistema de manufatura da aplicação 2	222
Figura 7.12 - Modelo da peça da aplicação 2	224
Figura 7.13 - Modelo do sistema de manufatura da aplicação 2	225

LISTA DE TABELAS

Tabela 2.1 - Cálculo do volume das principais <i>features</i>	70
Tabela 2.2 - Cálculo da área das principais <i>features</i>	71
Tabela 5.1 - Codificação das <i>features</i> no banco de dados	137
Tabela 5.2 - Codificação dos recursos no banco de dados.....	139
Tabela 5.3 - Diagramas de caso de uso do ambiente de simulação	163
Tabela 5.4 - Exemplo de diagrama de caso de uso para cadastrar operador do sistema.....	163
Tabela 5.5 - Código da face para inserção das <i>features</i>	186
Tabela 6.1 - Objetos do caso de teste 2.....	195
Tabela 6.2 - Parâmetros de entrada do caso de teste 2.....	196
Tabela 6.3 - Parâmetros da peça do caso de teste 2.....	196
Tabela 6.4 - Parâmetros das <i>features</i> do caso de teste 2.....	196
Tabela 6.5 - Parâmetros dos recursos do caso de teste 2	197
Tabela 6.6 - Objetos do caso de teste 3.....	199
Tabela 6.7 - Parâmetros de entrada do caso de teste 3.....	199
Tabela 6.8 - Parâmetros da peça do caso de teste 3.....	199
Tabela 6.9 - Parâmetros das <i>features</i> do caso de teste 3	200
Tabela 6.10 - Parâmetros dos recursos do caso de teste 3.....	200
Tabela 6.11 - Resultados da simulação por recurso do caso de teste 2	205
Tabela 6.12 - Resultados da simulação gerais do caso de teste 2.....	205
Tabela 6.13 - Comparativo dos resultados da simulação por recurso dos cenários do caso de teste 3.....	207
Tabela 6.14 - Comparativo dos resultados da simulação gerais dos cenários do caso de teste 3	208
Tabela 6.15 - Resultados da simulação por recurso do cenário 4 do caso de teste 3.....	209
Tabela 6.16 - Resultados da simulação gerais do cenário 4 do caso de teste 3.....	209
Tabela 7.1 - Parâmetros de entrada aplicação 1	216
Tabela 7.2 - Parâmetros da peça da aplicação 1	216
Tabela 7.3 - Parâmetros das <i>features</i> da aplicação 1	216
Tabela 7.4 - Parâmetros do recurso atual da aplicação 1	217
Tabela 7.5 - Parâmetros do novo recurso da aplicação 1	218
Tabela 7.6 - Comparativo dos resultados da simulação dos cenários da aplicação 1.....	218
Tabela 7.7 - Parâmetros de entrada aplicação 2	223
Tabela 7.8 - Parâmetros da peça da aplicação 2	223
Tabela 7.9 - Parâmetros das <i>features</i> da aplicação 2	223
Tabela 7.10 - Parâmetros dos recursos atuais da aplicação 2.....	223
Tabela 7.11 - Resultados da simulação por recurso da aplicação 2.....	226
Tabela 7.12 - Resultados da simulação gerais da aplicação 2.....	226
Tabela 7.13 - Resultados por <i>features</i> da aplicação 2.....	226

LISTA DE ABREVIATURAS E SIGLAS

3D	Tridimensional
API	<i>Application Program Interface</i>
AV	Ambiente Virtual
AGV	<i>Automated Guided Vehicle</i>
CAD	<i>Computer-Aided Design</i>
CAM	<i>Computer Aided Manufacturing</i>
CAPP	<i>Computer Aided Process Planning</i>
CIM	<i>Computer Integrated Manufacturing</i>
CNC	Comando Numérico Computadorizado
CSV	<i>Comma Separated Value</i>
DOF	<i>Degree of Freedom</i>
GUI	<i>Graphical User Interface</i>
HMD	<i>Head Mounted Display</i>
IDE	<i>Integrated Development Environment</i>
NS	Núcleo de Simulação
OO	Orientação a Objetos
OpenGL	<i>Open Graphics Library</i>
RV	Realidade Virtual
SGBD	Sistema Gerenciador de Banco de Dados
SQL	<i>Structured Query Language</i>
UML	<i>Unified Modeling Language</i>
VRML	<i>Virtual Reality Modeling Language</i>
XML	<i>Extensible Markup Language</i>

LISTA DE SIMBOLOS

d	Diâmetro da ferramenta (mm)
E_m	Espaço ocupado pela máquina-ferramenta (m^2)
f	Avanço (mm/min)
H	Número de horas de trabalho previstas por ano (horas)
l_f	Movimento do avanço (mm)
i_m	Idade da máquina-ferramenta (anos)
j	Taxa anual de juros (% a.a)
K_e	Custo do m^2 ocupado pela máquina-ferramenta (R\$/ m^2 -ano)
K_{ft}	Custo de cada aresta de corte do inserto (R\$)
K_m	Custo de manutenção (R\$/ano)
K_p	Custo de produção por peça (R\$/peça)
K_{pi}	Custo de aquisição do inserto (R\$)
K_{us}	Custo de mão-de-obra envolvida na usinagem (R\$/peça)
K_{um}	Custo da máquina-ferramenta (R\$/peça)
K_{uf}	Custo da(s) ferramentas (R\$/peça)
M	Vida prevista para a máquina-ferramenta (anos)
N_{fp}	Vida média para o porta ferramentas em número de trocas (min)
N_s	Número de arestas de corte disponíveis em cada inserto
N_t	Número de trocas da ferramenta na usinagem do lote Z
S_h	Salário do operador (R\$/hora)
S_m	Salário máquina (R\$/hora)
T_a	Tempo de aproximação e afastamento da ferramenta (min)
T_c	Tempo de corte (min)
T_{ft}	Tempo de troca da ferramenta (min)
T_p	Tempo de preparo da máquina (min)
T_s	Tempo secundário (min)
T_t	Tempo total de confecção da peça (min)
V_c	Velocidade de corte (m/min)
V_{mi}	Valor inicial de aquisição da máquina-ferramenta (R\$)
V_{si}	Custo de aquisição do porta-ferramenta (R\$)
Z	Lote de peça a ser usinado (und)
T_m	Tempo de manufatura (min)

T_{ot}	Tempo total de operação (min)
T_{no}	Tempo total de não-operação (min)
C_m	Custo de manufatura (R\$/peça)
R_o	Salário do operador (R\$/hora)
R_m	Salário da máquina (R\$/hora)
T_{su}	Tempo de <i>setup</i> (min)
Q	Tamanho do lote de produção (unid)
C_{ma}	Custo do material (R\$/peça)
ta_i	Tempo de <i>setup</i> da máquina (min)
tb_{ij}	Tempo de <i>setup</i> das ferramentas (min)
T_{ot}	Tempo total de operação (min)
Tr_i	Tempo de corte (desbaste) da <i>feature</i> (min)
Tf_i	Tempo de acabamento da <i>feature</i> (min)
Ta_j	Tempo de aproximação da ferramenta (min)
T_{ot}	Tempo total de operação (min)
Tr_i	Tempo de corte (desbaste) da <i>feature</i> (min)
Tf_i	Tempo de acabamento da <i>feature</i> (min)
Ta_j	Tempo de aproximação da ferramenta (min)
V_r	Volume do material removido (mm ³)
MRR_r	Taxa de remoção do material de <i>features</i> rotacionais (mm ³ /min)
MRR_p	Taxa de remoção do material de <i>features</i> prismáticos (mm ³ /min)
MRR_{Ca}	Taxa de remoção do material de <i>features</i> cavidades (mm ³ /min)
MRR_{ci}	Taxa de remoção do material de <i>features</i> circulares (mm ³ /min)
A_f	Área da <i>feature</i> (mm ²)
R_t	Taxa de remoção da área da <i>feature</i> (mm ² /min)

SUMÁRIO

1. INTRODUÇÃO	23
1.1 CONTEXTUALIZAÇÃO.....	23
1.2 JUSTIFICATIVA DO TRABALHO	25
1.3 OBJETIVO DO TRABALHO.....	28
1.4 METODOLOGIA DE PESQUISA	28
1.5 ESTRUTURA DO TRABALHO	31
2. TECNOLOGIA DE FEATURES APLICADA A SISTEMAS DE MANUFATURA..	33
2.1 SISTEMA DE MANUFATURA.....	33
2.1.1 Definição de Sistema de Manufatura.....	33
2.1.2 Importância do Sistema de Manufatura.....	35
2.1.3 Principais Tipos de Sistema de Manufatura.....	36
2.1.4 Processos de Fabricação	38
2.1.4.1 Torneamento.....	45
2.1.4.2 Fresamento.....	46
2.1.4.3 Furação.....	48
2.1.4.4 Retificação.....	49
2.1.4.5 Mandrilamento	50
2.1.4.6 Brochamento.....	51
2.1.4.7 Aplainamento.....	52
2.1.4.8 Alargamento	53
2.1.5 Processos de Movimentação Interna	54
2.2. TECNOLOGIA DE FEATURES.....	59
2.2.1 Evolução da Utilização de Features na Manufatura	59
2.2.2 Definição de Features	60
2.2.3 Vantagens e Limitações da Utilização de Features	62
2.2.4 Principais Tipos de Features.....	63
2.2.4.1 Features de Forma.....	64
2.2.4.2 Features de Manufatura.....	66
2.2.5 Tempo e Custo de Manufatura das Features.....	68
2.2.6 Reconhecimento, Projeto e Mapeamento de Features	72
2.2.7 Aplicações de Features na Manufatura	74
3. TÉCNICAS UTILIZADAS NO DESENVOLVIMENTO DO AMBIENTE DE	
SIMULAÇÃO.....	77
3.1 SIMULAÇÃO.....	77
3.1.1 Conceituação de Modelos e Sistemas	77
3.1.2 Conceituação de Simulação	80
3.1.3 Quando Usar Simulação.....	81
3.1.4 Vantagens e Desvantagens da Simulação	82
3.1.5 Principais Componentes e Resultados da Simulação.....	84
3.1.6 Metodologia de Estudos de Simulação	86
3.1.7 Ferramentas de Simulação.....	89
3.1.7.1 Linguagem de Propósito Geral.....	90
3.1.7.2 Linguagem de Simulação	91
3.1.7.3 Programas de Simulação	92
3.1.8 Simulação de Sistemas de Manufatura	92
3.2. REALIDADE VIRTUAL.....	95
3.2.1 Definição de Realidade Virtual	95
3.2.2 Tipos de Sistemas de Realidade Virtual	96
3.2.3 Ferramentas da Realidade Virtual	98
3.2.3.1 Dispositivos de Entrada de Dados.....	99
3.2.3.2 Dispositivos de Saída de Dados.....	100
3.2.3.3 Programas para a Realidade Virtual.....	100
3.2.4 Utilização da Realidade Virtual na Manufatura	103
3.2.5 Ambientes Virtuais Tridimensionais	104
3.3. MODELOS DE PROCESSO DE SOFTWARE	106

3.3.1 Principais Modelos de Processo de Software	106
3.3.2 Especificação	109
3.3.3 Projeto	109
3.3.3.1 Linguagem Unificada de Modelagem (UML).....	110
3.3.4 Implementação	111
3.3.5 Validação	112
3.3.6 Manutenção.....	115
4. UMA ABORDAGEM PARA SIMULAÇÃO DE SISTEMAS DE MANUFATURA A PARTIR DE FEATURES DE UMA PEÇA	117
4.1 ABORDAGENS PARA SIMULAÇÃO DE SISTEMA DE MANUFATURA	117
4.2 UTILIZAÇÃO DA TECNOLOGIA DAS FEATURES NOS SISTEMAS DE MANUFATURA.....	118
4.3 ABORDAGEM PROPOSTA.....	119
4.3 FUNCIONAMENTO DA ABORDAGEM PROPOSTA	120
4.3.1 Chegada de Peça.....	121
4.3.2 Execução da Simulação	121
4.3.3 Processamento das Filas	123
4.3.4 Cálculo dos Tempos e dos Custos dos Recursos	124
5. O AMBIENTE DE SIMULAÇÃO	129
5.1 INTRODUÇÃO	129
5.2 ETAPAS DO DESENVOLVIMENTO	130
5.3 ANÁLISE DE REQUISITOS	131
5.4 PROJETO DO AMBIENTE DE SIMULAÇÃO	134
5.4.1 Arquitetura do Ambiente de Simulação	134
5.4.1.1 Módulo Banco de Dados.....	135
5.4.1.2 Módulo Modelador	140
5.4.1.3 Módulo Simulador	142
5.4.1.4 Módulo Interface Gráfica com Usuário.....	145
5.4.1.5 Módulo Sistema de Visualização Tridimensional.....	147
5.4.1.6 Módulo Resultados	149
5.4.2 Funcionamento do Ambiente de Simulação	151
5.4.2.1 Acesso e Saída do Sistema	153
5.4.2.2 Entrada de Dados.....	155
5.4.2.3 Modelagem da Simulação	156
5.4.2.4 Simulação.....	158
5.4.2.5 Saída de Dados.....	159
5.4.3 Modelagem do Ambiente de Simulação	162
5.4.3.1 Diagrama de Caso de Uso	163
5.4.3.2 Diagrama de Classes	164
5.4.3.3 Diagrama de Sequência	167
5.5 IMPLEMENTAÇÃO DO AMBIENTE DE SIMULAÇÃO.....	168
5.5.1 Banco de Dados	170
5.5.2 Interfaces	173
5.5.3 Simulação	176
5.5.4 Desenvolvimento da Visualização 3D.....	182
5.6 DOCUMENTAÇÃO.....	189
6. VERIFICAÇÃO E VALIDAÇÃO	191
6.1 INTRODUÇÃO	191
6.2 DESENVOLVIMENTO DE TESTE	192
6.3 PLANEJAMENTO E EXECUÇÃO DE CASOS DE TESTE	193
6.3.1 Caso de Teste 1	193
6.3.2 Caso de Teste 2	195
6.3.3 Caso de Teste 3	198
6.4 AVALIAÇÃO DOS RESULTADOS.....	200
6.4.1 Caso de Teste 1	201
6.4.2 Caso de Teste 2	205
6.4.3 Caso de Teste 3	207
7. APLICAÇÃO DO AMBIENTE DE SIMULAÇÃO EM SISTEMAS DE MANUFATURA REAIS	211
7.1 INTRODUÇÃO	211

7.2 DESENVOLVIMENTO DAS APLICAÇÕES.....	213
7.2.1 Aplicação 1.....	213
7.2.2 Aplicação 2.....	220
8. CONCLUSÃO	229
8.1 CONCLUSÕES E COMENTÁRIOS.....	229
8.2 SUGESTÕES PARA TRABALHOS FUTUROS	231
REFERENCIAS.....	233
APÊNDICE A	247
APÊNDICE B	248
APÊNDICE C	249
APÊNDICE D	260
APÊNDICE E	264
ANEXO A	282
ANEXO B	294

1. INTRODUÇÃO

Este capítulo inicia com uma contextualização do trabalho. Após a contextualização, são detalhadas as justificativas da existência do trabalho. Logo em seguida é apresentado o objetivo do trabalho, assim como as principais etapas para o seu desenvolvimento. Após, é tratada a metodologia do trabalho. A última parte do capítulo é a estrutura da tese.

1.1 CONTEXTUALIZAÇÃO

Atualmente as organizações estão passando por muitas mudanças em seus sistemas de manufatura devido ao crescimento dos mercados, assim como das exigências dos clientes. Alguns fatores ampliam essa necessidade de mudança, como: o aumento do número e variedade de produtos; a queda do tamanho do lote de produção; minimização de estoques; menores tolerâncias a erros; aumento na quantidade de materiais complexos; aumento dos custos de materiais; e diminuição entre o tempo de concepção do projeto e o produto fabricado.

As empresas têm que se adequar ao novo perfil do mercado, no qual são palavras de ordem o repensar constante da organização. A manufatura é uma área fundamental nas empresas. Há então a necessidade de uma manufatura com capacidade para mudança que permita produzir ora produto A, ora produto B, mantendo preços e prazos compatíveis com os conseguidos no passado.

A manufatura é o processo de transformação de matérias-primas e informações em produtos acabados com valor agregado satisfazendo assim as necessidades dos clientes. Ela tem fornecido uma grande contribuição para o crescimento econômico das nações nos últimos séculos e continuará sendo vital no futuro. No ambiente de negócio atual, dinâmico e competitivo, a manufatura enfrenta novos desafios que exigem uma perspectiva holística sobre quatro atributos principais que são: tempo, custo, qualidade e flexibilidade (CHRYSSOLOURIS, 2006).

Para ter uma vantagem competitiva na manufatura, as empresas precisam conhecer muito bem seus sistemas. Para isso algumas ferramentas podem ser utilizadas em conjunto como: Tecnologia de *Features*, Simulação Computacional e Realidade Virtual.

A representação de um produto é um dos itens mais importantes para sua manufatura. A tecnologia de *features* relaciona um conjunto de informações referentes aos aspectos do projeto de um produto com a manufatura. Por exemplo, na usinagem, as *features* são volumes a serem removidos ou volumes remanescentes da peça.

As *features* são muito utilizadas na integração do projeto com a manufatura para geração de um planejamento mais detalhado e conciso, com isso consegue-se ganhos de tempo e economia de recursos.

De acordo com Ryan e Heavey (2006), os sistemas de manufatura são complexos e difíceis, no que tange ao seu entendimento e à sua operacionalização de forma eficiente.

A simulação devido à sua grande versatilidade, flexibilidade e poder de análise é uma das técnicas mais amplamente utilizadas em sistemas de manufatura. Através da simulação é possível ter uma visão geral do sistema, assim como obter respostas para questões do tipo “o que aconteceria se...?”. Banks *et al.* (2009) afirmam que a simulação é muito usada para realizar projeções sem alterar o cotidiano da empresa.

Uma técnica que pode ser utilizada em conjunto com a simulação para modelagem e análise de sistemas de manufatura é a realidade virtual (RV). A RV basicamente coloca o usuário no mundo virtual que imita o interior de uma indústria com máquinas, operadores, equipamentos que reagem aos comandos do usuário.

A RV é uma técnica avançada de interface através da qual o usuário pode realizar imersão, navegação e interação em ambientes sintéticos tridimensionais gerados por computador, utilizando canais multisensoriais. A imersão é a sensação vivenciada pelo usuário de estar “mergulhado” dentro do ambiente (JACOB, 2002). Complementa Dorozhkin *et al.* (2012) dizendo que a interligação da simulação com a realidade virtual permite ao usuário uma maior interação com modelo, objetos e com as operações que estão sendo simuladas.

A realidade virtual traduz a realidade dos sistemas de manufatura para mundos virtuais tridimensionais que é um meio eficaz de comunicação e que permite

que os erros de lógica ou planejamento da simulação possam ser facilmente identificados e corrigidos.

O ambiente proposto neste trabalho busca fazer estudos de simulação em sistemas de manufatura tridimensionais modelados a partir de *features* contidas nas peças para se obter como resultados indicadores baseados em custo e tempo como: tempo total de manufatura; custo e tempo por recurso de fabricação e transporte; tempo de espera nas filas e taxa de utilização de recursos.

1.2 JUSTIFICATIVA DO TRABALHO

Com a globalização dos mercados e a diversificação de produtos demandados, os sistemas de manufatura têm se tornado cada vez mais complexos e de difícil operacionalização de forma eficiente. Dificilmente o responsável pela produção tem toda a informação necessária para antever todos os detalhes necessários à sua execução. Por outro lado, as empresas não dispõem de tempo nem recursos para testar alternativas de produção no sistema real, pois acarreta em paradas de máquinas e operários, gerando assim um alto custo.

Porém, para atender à demanda de produtos, as organizações precisam identificar e analisar qual é a melhor forma (roteiros de produção, recursos e processos) de fabricar tais produtos. A partir dessas análises, os gestores podem tomar decisões mais acuradas, acarretando menores erros e possíveis melhorias na manufatura.

Uma das formas de se realizar análises e projeções é através da simulação. Os ambientes existentes aplicados à simulação de sistemas de manufatura ou são desenvolvidos para casos específicos (um estudo específico) ou são de uso geral e neste caso são escritos em linguagens de simulação ou programação. Geralmente, os sistemas para casos específicos são restritos ao estudo de uma empresa ou situação específica e os sistemas de caso geral são complexos quanto à modelagem da simulação, gerando a necessidade de conhecimento da linguagem utilizada pelo desenvolvedor do sistema.

Além disso, os ambientes de simulação para sistemas de manufatura encontrados no mercado possuem em sua maioria interface 2D, o que dificulta o

acompanhamento e controle por parte do usuário no momento da simulação. Essa qualidade é melhorada quando se utiliza interfaces tridimensionais baseadas em realidade virtual que possibilitam à simulação um maior envolvimento do usuário com o sistema. Dessa forma, o usuário do ambiente se torna um agente ativo envolvido com o sistema.

A motivação inicial para a realização deste trabalho surgiu da constatação da possibilidade de utilização da tecnologia de *features* para simulação de sistemas de manufatura.

Essa constatação gerou a primeira contribuição do trabalho que é o desenvolvimento de uma nova abordagem para a realização da simulação de sistemas de manufatura. Essa abordagem é baseada na geometria da peça. Essa geometria é composta pelas *features* que são inseridas de acordo com o projeto da peça. Ela permite que a simulação seja feita com flexibilidade porém com confiança, pois os dados de entrada da simulação são as características da peça a ser simulada. Essa forma de simular está alinhada com a abordagem de vários sistemas de projetos (CAD) atuais que utilizam a tecnologia de *features* para a modelagem de seus produtos.

Após uma pesquisa por programas que pudessem auxiliar na demonstração e/ou validação da abordagem proposta, não foi encontrado nenhum que contemplasse todas as características necessárias.

Assim, propõe-se um ambiente de simulação que é a segunda contribuição do trabalho. A proposta é de uma ferramenta flexível que possa ser aplicada ao maior número possível de sistemas de manufatura, de fácil manuseio e com interface simples que possibilite ao responsável por um sistema de manufatura, sem conhecimento de linguagens específicas de simulação ou programação, realizar análises, avaliações e modificações de seu sistema com um alto grau de detalhamento.

Essa ferramenta permite ao gestor do sistema de manufatura analisar desde a inserção das *features* que devem ser processadas na peça no banco de dados até a projeção de cenários hipotéticos com a interação entre as peças e os recursos de fabricação e movimentação que compõem o sistema.

Com o ambiente de simulação se tem uma ferramenta que é um laboratório voltado para simulação de sistemas de manufatura com facilidade de uso, custo e tempo menores do que a experimentação no sistema real.

Com o ambiente de simulação é possível:

- Pesquisar e desenvolver novas tecnologias, metodologias e modelos de referência para sistemas de manufatura, visando contribuir para o aumento de potencial competitivo através da investigação de novas formas e oportunidades;
- Ensinar e treinar fora do sistema de manufatura, quando se tem tarefas consideradas críticas;
- Permitir identificar e armazenar conhecimentos envolvidos na gestão do sistema de manufatura para projetar e/ou reprojetar operações susceptíveis de aumentar a capacidade de adaptação de uma empresa com seu ambiente;
- Criar e testar cenários, permitindo vivenciar situações que possibilitem a aplicação de conhecimentos e técnicas relacionadas à manufatura;
- Facilitar a visualização e manipulação do objeto em estudo, reproduzindo os dados complexos sob a forma de objetos tridimensionais, aumentando a capacidade de percepção do usuário, que é estimulado pela possibilidade de interação com a interface.

A utilização do ambiente de simulação nas organizações permite expandir o espaço dos experimentos, simulando diversas alternativas de manufatura e criando protótipos de sistemas de manufatura no computador. Ele pode ser aplicado nas seguintes situações:

- Análise da manufaturabilidade de um produto;
- Avaliação e validação das possibilidades dos planos de produção;
- Melhoria do desempenho do sistema de manufatura como um todo;
- Análise dos projetos do produto e dos processos de sistemas reais;
- Realização de treinamento para os operadores, técnicos e gerência no uso do sistema de manufatura;
- Estudo do comportamento estático e dinâmico do sistema de manufatura permitindo, dessa forma, projetar e prever a resposta do sistema sob investigação nas condições de trabalho que irão ocorrer em situações reais.

1.3 OBJETIVO DO TRABALHO

O objetivo deste trabalho foi propor um ambiente para a simulação da manufatura, a partir de uma nova abordagem baseada nas *features* das peças a serem simuladas e na técnica de realidade virtual.

Para atingir o objetivo proposto foi necessário executar as seguintes etapas:

- Desenvolver uma nova abordagem para simulação de sistemas de manufatura baseado na tecnologia de *features*;
- Pesquisar e delimitar os principais componentes de sistemas de manufatura em geral;
- Especificar, modelar, implementar e testar um ambiente de simulação baseado em *features* da peça e análise orientada a objetos;
- Identificar e selecionar os recursos de *software* necessários à implementação do ambiente de simulação de acordo com as especificações;
- Desenvolver um banco de dados, uma interface gráfica interativa com o usuário e um sistema de visualização tridimensional baseado em técnicas de realidade virtual;
- Aplicar o ambiente de simulação em sistemas de manufatura reais.

1.4 METODOLOGIA DE PESQUISA

Metodologia é o conjunto de técnicas e processos utilizados pela ciência para formular e resolver problemas de aquisição objetiva do conhecimento de maneira sistemática (JUNG, 2003).

Segundo Gil (2008), a pesquisa tem um caráter pragmático, é um processo formal e sistemático de desenvolvimento do método científico. O objetivo fundamental da pesquisa é descobrir respostas para problemas mediante o emprego de procedimentos científicos. Complementa Jung (2003) dizendo que a pesquisa é o processo através do qual as pessoas adquirem um novo conhecimento sobre si mesmas ou sobre o mundo em que vivem, com as finalidades de: responder a um questionamento, resolver um problema ou satisfazer uma necessidade.

Silva e Menezes (2005) unem os termos metodologia e pesquisa resultando na metodologia de pesquisa que tem a função de mostrar como andar no “caminho das pedras” na elaboração de um projeto de pesquisa, baseado em planejamento cuidadoso, reflexões conceituais sólidas e alicerçados em conhecimentos já existentes.

Figura 1.1 - Formas de classificação das pesquisas científicas

Fonte: Adaptado de Silva e Menezes (2005)

A partir da pesquisa realizada nesse trabalho foi desenvolvido um ambiente de simulação para sistemas de manufatura que auxilia na tomada de decisões. De acordo com a Figura 1.1, classifica-se essa pesquisa da seguinte forma:

- É de natureza aplicada, pois tem como objetivo gerar conhecimentos para aplicação prática em sistemas de manufatura;
- A forma de abordagem é a quantitativa devido utilizar equações matemáticas para descrever o funcionamento de sistemas de manufatura e os resultados da simulação são obtidos a partir do cálculo de indicadores baseados em custo e tempo, além do percentual de utilização de cada recurso;
- É uma pesquisa descritiva, pois propõe uma nova abordagem para realização da simulação de sistemas de manufatura. Para o desenvolvimento da pesquisa foram levantados os principais componentes (recursos e processos) de diferentes tipos de sistemas de manufatura;
- Os procedimentos técnicos utilizados foram a pesquisa bibliográfica e o estudo de caso. Inicialmente realizou-se uma pesquisa bibliográfica sobre os temas envolvidos no trabalho (sistemas de manufatura, tecnologia de *features*, simulação, realidade virtual e modelos de processos de *software*). Além da pesquisa bibliográfica foram identificados e estudados sistemas de manufatura reais buscando projetar e analisar cenários baseados na abordagem proposta.

O processo de desenvolvimento da abordagem proposta e do ambiente de simulação dividiu-se em etapas que são:

- Análise de requisitos: foram definidos os requisitos funcionais e não funcionais do ambiente de simulação, além das suas limitações;
- Projeto: foi desenvolvida a arquitetura, descrito o funcionamento do sistema e a desenvolvida a modelagem do sistema. A técnica utilizada foi a modelagem orientada a objetos com a utilização de diagramas de caso de uso, de classes e de sequência;
- Implementação: foi realizada a programação do ambiente; a programação visual; uma modelagem 3D baseada em realidade virtual; o desenvolvimento das interfaces e; a integração dos conteúdos;

- Verificação e Validação: foram realizados os testes do sistema a partir do planejamento e execução de casos de teste com o intuito de eliminar possíveis erros e falhas do sistema;
- Documentação: foram documentados todas as etapas desenvolvidas.

Por fim, o ambiente de simulação foi aplicado em duas situações práticas distintas: dois sistemas de manufaturas reais de setores industriais diferentes, buscando testar/melhorar os tempos e custos de produção a partir dos cenários propostos.

1.5 ESTRUTURA DO TRABALHO

Esta tese está estruturada em oito capítulos, descritos a seguir:

Capítulo 1 – Introdução: busca prover informações para a compreensão do trabalho realizado. Define-se, neste capítulo, o contexto em que está inserido o trabalho, a justificativa da sua realização, sendo, ainda, apresentados os objetivos, a metodologia de pesquisa e a estrutura da tese.

Capítulo 2 – Tecnologia de *Features* aplicada a Sistemas de Manufatura: este capítulo apresenta o aporte teórico necessário à discussão do tema em estudo. São abordados os temas: Sistemas de Manufatura e Tecnologia de *Features*.

Capítulo 3 – Técnicas Utilizadas no Desenvolvimento do Ambiente de Simulação: este capítulo apresenta as técnicas utilizadas para desenvolver o ambiente proposto. Essas técnicas são simulação, realidade virtual e modelos de processos de *software*.

Capítulo 4 – Uma Abordagem para Simulação de Sistemas de Manufatura a partir das *Features* de uma peça: este capítulo detalha a abordagem proposta no trabalho. Inicialmente é realizada uma descrição das abordagens de simulação de sistema de manufatura, após é mostrado a utilização das *features* em sistemas de manufatura e em seguida é detalhada a abordagem e seu funcionamento.

Capítulo 5 – O Ambiente de Simulação: este capítulo apresenta uma caracterização do ambiente de simulação, etapas do seu desenvolvimento, seus

requisitos, sua arquitetura e seu funcionamento. Além disso, são também detalhadas a modelagem, a implementação e a documentação do sistema

Capítulo 6 – Verificação e Validação: este capítulo apresenta a verificação e validação a partir de testes realizados no ambiente de simulação. Também são detalhados os resultados obtidos nos testes.

Capítulo 7 – Aplicação do Ambiente de Simulação em Sistemas Manufatura Reais: este capítulo apresenta aplicação do ambiente de simulação em dois sistemas de manufatura diferentes a partir do processamento de duas peças com geometrias diferentes.

Capítulo 8 – Conclusões do Estudo: são apresentadas as conclusões obtidas com o estudo, buscando responder aos objetivos propostos. Apresenta-se, por fim, sugestões para pesquisas futuras.

2. TECNOLOGIA DE *FEATURES* APLICADA A SISTEMAS DE MANUFATURA

Este capítulo inicialmente define sistema de manufatura e sua importância. Após são detalhados os principais processos de fabricação e movimentação de materiais. A segunda parte do capítulo aborda a evolução e definições das *features*. Após, são tratadas as vantagens e limitações da utilização das *features*. Em seguida, são apresentados os principais tipos de *features* e como calcular o tempo e o custo de fabricação baseado nas *features*. A última parte do capítulo trata do reconhecimento, projeto e mapeamento das *features* e das aplicações de *features* na manufatura.

2.1 SISTEMA DE MANUFATURA

O objetivo deste tópico é apresentar uma revisão bibliográfica de sistema de manufatura (SM). Inicialmente define-se o conceito de SM, sua importância e características, os principais tipos e, em seguida, abordam-se as partes que o compõem.

2.1.1 Definição de Sistema de Manufatura

De acordo com Stair e Reynolds (2008), sistema é um conjunto de elementos ou componentes que interagem para atingir objetivos. Os elementos em si e as relações entre eles determinam como funciona o sistema. Complementa Mattos (2005) dizendo que um sistema é constituído de dois elementos: uma coleção de objetos e uma relação lógica entre eles. Esses elementos físicos e lógicos fazem com que o sistema se comporte como um organismo.

De acordo com Guerrini e Coelho (2005), manufatura vem do latim (*manus + factus*) que significa “feito à mão”, mas passou, ao longo da história, a designar todo processo intensivo em mão-de-obra para geração de produtos.

A junção de sistema com manufatura é, segundo Passos (1986), definida como um conjunto de máquinas e/ou equipamentos interligados por um sistema de transporte e manuseio de materiais, com auxílio ou não de trabalhadores.

Sistemas de Manufatura consistem em máquinas e centros de produção, cujas operações ocorrem em itens ou peças formando produtos destinados aos clientes. Equipamentos de transporte permitem movimentação entre as máquinas ou centro de trabalhos.

Para Moreira (2008) um sistema de manufatura é um conjunto de atividades e operações necessárias à produção de produtos envolvendo os seguintes elementos: insumos, o processo de transformação e os produtos fabricados.

Já Black (1998) afirma que SM é um arranjo complexo de elementos físicos caracterizados por parâmetros mensuráveis. Dentre os elementos físicos importantes destacam-se: pessoas, processos, equipamentos, estoque e manuseio de materiais. Dentre os parâmetros mensuráveis destacam-se: taxa de produção, tempo de atravessamento, custo total ou unitário, entre outros.

Figura 2.1 - Sistema de manufatura

Fonte: Black (1998)

Um sistema de manufatura é composto de vários subsistemas como transportadores, células de produção, máquinas operatrizes, inspeção de controle da qualidade, no qual cada subsistema realiza uma determinada operação produtiva, que pode ser uma transformação física, ou um deslocamento físico, ou ainda uma

inspeção. Uma característica básica destes subsistemas é que por eles fluem diversos itens, como peças e materiais, que passam sequencialmente pelos subsistemas, onde transformações apropriadas são realizadas até obter o produto final requerido de acordo com especificações da qualidade (PALOMINO, 2001).

2.1.2 Importância do Sistema de Manufatura

A função da manufatura, na maior parte das empresas, representa a maior parte do seu ativo e do seu pessoal. A manufatura são os ossos, os nervos e os músculos da empresa. Uma manufatura saudável fornece à empresa a força para suportar o ataque da concorrência, para manter o melhoramento uniforme no desempenho competitivo e proporcionar a versatilidade operacional que pode responder aos mercados crescentemente voláteis e aos concorrentes (SLACK, 1993).

O sistema de manufatura é uma fonte gerador de vantagem competitiva numa organização. Skinner (1969) foi quem primeiro afirmou que decisões que parecem ser rotineiras concernentes à manufatura frequentemente acabam por limitar as opções estratégicas da organização, atrelando-a, através de instalações, equipamentos, pessoal, controles básicos e políticas, a uma postura não competitiva que pode levar anos para ser contornada.

De acordo com Agostinho (2000), o SM tem influência direta sobre os aspectos de desempenho competitivo de uma empresa, como: produção de produtos sem erros, manutenção invariável dos prazos de entregas prometidos, habilidade de lançar ao mercado novos produtos em tempos adequados, oferecimento de uma grande variedade de produtos ao mercado, habilidade de modificar quantidades a serem produzidas e/ou datas de entregas conforme a demanda.

Segundo Corrêa e Giansesi (1996), os sistemas de manufatura têm um papel fundamental no atendimento dos objetivos estratégicos das empresas. As razões para isso podem ser classificadas em três categorias principais:

- Uma crescente pressão por competitividade que os mercados têm demandado das empresas a partir do surgimento de novos concorrentes e novos produtos;
- O potencial competitivo que representa o recente desenvolvimento de novas tecnologias de processo e de gestão da manufatura, como os sistemas de manufatura integrada por computador e os sistemas flexíveis de manufatura;
- O melhor entendimento do papel estratégico que a produção pode e deve ter no atendimento dos objetivos globais das empresas.

2.1.3 Principais Tipos de Sistema de Manufatura

De acordo com Moreira (2008), a classificação dos sistemas de manufatura ocorre principalmente em função do fluxo do produto e os SM podem ser agrupados em três grandes categorias:

- Contínua ou de Fluxo em Linha: apresenta uma sequência linear para fazer o produto; os produtos são bastante padronizados e fluem de um posto de trabalho para outro em sequência prevista;
- Por Lotes: a fabricação é feita em lotes. Ao término da fabricação do lote de um produto, outros tomam o seu lugar nas máquinas. O produto original só voltará a ser feito depois de algum tempo, caracterizando-se assim uma produção intermitente de cada um dos produtos;
- Por Projeto: cada projeto é único, não havendo rigorosamente um fluxo do produto. Tem-se uma sequência de tarefas ao longo do tempo, geralmente de longa duração, com pouca ou nenhuma repetitividade.

Segundo Askin e Standridge (1993) os sistemas de manufatura classificam-se em:

- Processos Discretos: quando os produtos são distinguíveis;
- Processos Contínuos: quando os produtos não podem ser distinguidos individualmente.

Já Lucero (2001) afirma que as organizações são compostas por sistemas de manufatura complexos que têm o objetivo de produzir bens materiais ou produtos para serem utilizados pelo homem. Em seu conjunto, as empresas produzem os mais diversos produtos e por isso têm as mais variadas características. Isso fez com que cada área da ciência que necessitasse estudá-las tentasse classificá-las segundo seus paradigmas e entendimento, dando origem a várias formas de classificação, dentre elas as mais conhecidas são:

- Tipo de Indústria: classificação utilizada pela economia que divide em indústria automotiva, indústria química, dentre outras. É uma classificação baseada no produto ou bem material do processo de transformação produtivo e serve, dentre outros usos, para planos de promoção industrial e para as análises macroeconômicas dos governos e organismos nacionais e internacionais;
- Tamanho da Planta Industrial: utilizada nas legislações e pelos governos para aplicar suas políticas de promoção financeira e tributária, as dimensões do tamanho são geralmente duas: ganho em moeda vigente e quantidade de trabalhadores;
- Tipo de Processo Produtivo: classificação utilizada para determinar a posição da manufatura baseada no volume-variedade de produtos. Tal classificação determina a abordagem geral para gerenciar o processo de transformação. É dividida em processo por projeto; processo em lotes; processo de produção em massa; e processos contínuos;
- Tipo de Resposta à Demanda: utilizada para determinar a divisão segundo a forma de resposta à sociedade que podem ser: fabricar para estocar (*make-to-stock*), fabricar de acordo com a ordem de produção (*make-to-order*) e fabricar sobre pedido (*resource-to-order*). Cada tipo de demanda aponta para o grau de especialidade e variedade na fabricação dos seus produtos, e muitas empresas trabalham com mais de um tipo de resposta diante da demanda.

Nas próximas seções são discutidos os principais conceitos relacionados aos subsistemas de fabricação, transporte ou movimentação de materiais que compõem um sistema de manufatura.

2.1.4 Processos de Fabricação

Um procedimento ou atividade dedicada à produção de uma ou de um grupo de peças denomina-se processo de fabricação. Os processos de fabricação têm como objetivo a modificação de um material, com o fim de lhe conferir uma forma definida (AGOSTINHO *et al.*, 2004).

Cada processo de fabricação possui características específicas que o torna particular no que concerne às formas como trata os materiais. Conforme Resende (1993), a conveniência da escolha de um processo de fabricação está relacionada com a forma da peça a ser trabalhada, o que determina a utilização de máquinas que combinem movimentos apropriados para a peça.

Para Magalhães (2008), os processos de fabricação são modificações sucessivas na matéria-prima, normalmente um bloco de material ou pré-forma, que se sujeita até efetivamente assumir todas as exigências geométricas e tecnológicas definidas.

De acordo com Resende (1993), os principais processos para fabricação de peças brutas ou acabadas podem ser divididos em dois grupos: processos com remoção de material (usinagem) e processos sem remoção de material (fundição, metalurgia do pó, conformação mecânica e soldagem). A usinagem é dividida de acordo com a Figura 2.2 em convencional (torneamento, fresamento, furação, dentre outros) e não convencional (jato d'água, jato abrasivo, ultrasom, dentre outros).

De acordo com Halevi e Weill (1995), aproximadamente 80% dos processos efetuados nas indústrias são processos de redução de massa de materiais sólidos, ou seja, processos de usinagem. Além disso, esses processos têm uma grande flexibilidade, permitindo ao usuário conseguir uma larga escala de aplicações, com um número muito grande de soluções possíveis.

Este trabalho aprofundou-se nos processos de usinagem convencional. Segundo Ferraresi (1985), os processos de usinagem convencional compreendem aqueles que, ao conferir à peça a forma, as dimensões, o acabamento, ou ainda uma combinação qualquer destes três itens, produzem cavaco. Entende-se por cavaco a porção de material da peça retirada pela ferramenta, caracterizando-se por apresentar, em geral, uma forma geométrica irregular.

Figura 2.2 - Principais processos de fabricação

Fonte: Adaptado de Resende (1993)

O processo de usinagem possibilita atingir os seguintes objetivos (CHIAVERINI, 1986):

- Acabamento de superfícies de peças fundidas ou conformadas plasticamente de modo a obter-se melhor aspecto superficial e melhores tolerâncias dimensionais, de acordo com as especificações de fabricação e de acordo com o emprego;

- Obtenção de peculiaridades, impossíveis de conseguir por outros processos;
- Fabricação seriada de peças a um custo mais barato;
- Fabricação de uma ou poucas peças, praticamente com qualquer forma, a partir de um bloco de material metálico.

Para a utilização de processos de usinagem, as condições ideais de corte devem ser determinadas. Essas condições são complexas, pois dependem de um grande número de variáveis que são: materiais da ferramenta e da peça, geometria da ferramenta, parâmetros de corte (velocidade, avanço e profundidade de corte), processo de usinagem e características da máquina ferramenta.

De acordo com Agostinho *et al.* (2004), as principais condições para que ocorra o processo de usinagem são:

- Movimento de corte: movimento entre a peça e a ferramenta, o qual sem o movimento de avanço origina somente uma única remoção de cavaco, durante uma volta ou curso;
- Movimento de avanço: movimento entre a peça e a ferramenta que, juntamente com o movimento de corte, origina um levantamento repetido ou contínuo do cavaco durante várias revoluções ou cursos;
- Direção de corte: direção instantânea do movimento de corte;
- Direção de avanço: direção instantânea do movimento de avanço;
- Plano de trabalho: plano que contém as direções de corte e avanço;
- Velocidade de corte: velocidade instantânea de um ponto qualquer da aresta cortante de um ponto de referência, segundo a direção e sentido do corte;
- Avanço: é a distância percorrida pela ferramenta seguindo a direção de avanço em uma volta ou curso;
- Profundidade de corte: é a profundidade de penetração da aresta de corte, medida numa direção perpendicular ao plano de trabalho.

A partir das condições expostas anteriormente, tem-se os principais parâmetros de usinagem utilizados neste trabalho para análise de desempenho dos processos de fabricação que são relacionados ao custo e ao tempo.

Ferraresi (1985) e Diniz *et al.* (2001) dividem os custos de usinagem em três parcelas que são os custos relativos à mão-de-obra, à máquina-ferramenta e à

ferramenta. O custo de usinagem por peça pode ser calculado por meio da equação 2.1.

$$K_p = K_{us} + K_{um} + K_{uf} \quad (2.1)$$

Onde:

K_p - custo de produção por peça (R\$/peça)

K_{us} - custo de mão-de-obra envolvida na usinagem (R\$/peça)

K_{um} - custo da máquina-ferramenta (R\$/peça)

K_{uf} - custo da(s) ferramentas (R\$/peça)

O custo de mão-de-obra pode ser calculado por meio da equação 2.2.

$$K_{us} = t_t \frac{S_h}{60} \quad (2.2)$$

Onde:

T_t - tempo total de fabricação da peça (min)

S_h - salário do operador (R\$/hora)

Para calcular o custo da máquina-ferramenta, utiliza-se o salário máquina de acordo com as equações abaixo:

$$S_m = \frac{1}{H} \left[\left(V_{mi} - V_{mi} \times \frac{i_m}{M} \right) \times j + \frac{V_{mi}}{M} + K_m + (E_m \times K_e \times j) \right] \quad (2.3)$$

$$K_{um} = t_t \frac{S_m}{60} \quad (2.4)$$

Onde:

S_m - salário máquina (R\$/hora)

H - número de horas de trabalho previstas por ano (unid)

V_{mi} - valor inicial de aquisição da máquina-ferramenta (R\$)

i_m - idade da máquina-ferramenta (anos)

M - vida prevista para a máquina-ferramenta (anos)

j - taxa anual de juros (%)

E_m - espaço ocupado pela máquina-ferramenta (m²)

K_m - custo de manutenção (R\$/ano)

K_e - custo do m^2 ocupado pela máquina-ferramenta (R\$/ m^2 -ano)

Para o cálculo do custo da ferramenta por peça, considera-se a utilização de pastilhas intercambiáveis como ferramenta de corte; com isso calcula-se o custo de cada aresta da ferramenta para depois calcular o custo da ferramenta por peça propriamente dito, de acordo com as equações 2.5 e 2.6.

$$K_{ft} = \frac{V_{si}}{N_{fp}} + \frac{K_{pi}}{N_s} \quad (2.5)$$

$$K_{uf} = \frac{K_{ft}}{Z_t} \quad (2.6)$$

Onde:

K_{ft} - custo de cada aresta de corte do inserto (R\$)

V_{si} - custo de aquisição da porta-ferramenta (R\$)

N_{fp} - vida média para da porta ferramentas em número de trocas (unid)

K_{pi} - custo de aquisição do inserto (R\$)

N_s - número de arestas de corte disponíveis em cada inserto (unid)

Z_t - número de peças usinadas por vida da ferramenta (unid)

Com relação ao tempo da usinagem, Ferraresi (1985) o definiu como sendo o tempo gasto em todas as etapas da usinagem de uma peça ou de um lote de peças, que se constitui nas seguintes fases:

- Colocação e fixação da peça em bruto ou semi-acabada na máquina-ferramenta;
- Aproximação ou posicionamento da ferramenta para o início do corte;
- Corte propriamente dito;
- Afastamento da ferramenta;
- Inspeção (se necessário) e retirada da peça usinada.

De acordo com Ferraresi (1985), o tempo de usinagem pode ser classificado, quanto à finalidade:

- Tempo de acabamento: operação de usinagem destinada a obter na peça as dimensões finais, ou um acabamento superficial especificado, ou ambos. As condições de acabamento são leves, com profundidades de usinagem bastante reduzidas e avanços tão pequenos quantos os necessários para a obtenção da especificação da peça-obra;
- Tempo de desbaste: operação de usinagem anterior a de acabamento, visando obter na peça a forma e dimensões próximas das finais. A condição de desbaste deve ser tão severa quanto permitirem a espessura do cavaco, a vida da ferramenta, a potência do motor e a própria peça.

De acordo com Diniz *et al.* (2001), os tempos gastos na usinagem são geralmente convencionados da seguinte forma:

$$T_t = T_c + T_s + T_a + \left(\frac{T_p}{Z}\right) + \left(\frac{N_t}{Z}\right) \times T_{ft} \quad (2.7)$$

Onde:

T_t – tempo total de usinagem de uma peça (min)

T_c – tempo de corte (min)

T_s – tempo secundário (min)

T_a – tempo de aproximação e afastamento da ferramenta (min)

T_p – tempo de preparo da máquina (min)

T_{ft} – tempo de troca da ferramenta (min)

Z – lote de peça a ser usinado (unid)

N_t – número de trocas da ferramenta na usinagem do lote Z (unid)

A equação 2.8 evidência outra forma de calcular o tempo total de usinagem de uma peça, que pode ser dividido em duas partes (DINIZ *et al.*, 2001):

$$T_t = T_c + T_i \quad (2.8)$$

Onde:

T_t – tempo total de usinagem de uma peça (min)

T_c – tempo de corte propriamente (min)

T_i – tempo improdutivo que se refere à colocação, inspeção e retirada da peça, aproximação e afastamento da ferramenta, substituição da ferramenta e preparo da máquina para a usinagem (min)

Segundo Ferraresi (1985), define-se como tempo de corte (T_c) de uma operação de usinagem o espaço de tempo em que as ferramentas em avanço de usinagem trabalham na remoção de cavacos da peça. Complementa Diniz *et al.* (2001) dizendo que o tempo de corte (T_c) resume a totalidade dos tempos ativos, pois ele representa o tempo em que os movimentos de corte e ou avanço estão efetivamente ocorrendo de acordo com a equação 2.9.

$$T_c = \frac{\pi \cdot d \cdot I_f}{1000 \cdot f \cdot V_c} \quad (2.9)$$

Onde:

T_c – tempo corte (min)

d – diâmetro da ferramenta (mm)

I_f – movimento do avanço (mm)

f – avanço (mm/volta)

V_c – velocidade de corte (m/min)

De acordo com Diniz *et al.* (2001), os tempos improdutivos (T_i) estão divididos em duas parcelas de acordo com as equações 2.10 e 2.11:

- Tempo improdutivo referente à colocação, inspeção e retirada da peça, aproximação e afastamento da ferramenta e preparo da máquina para usinagem de um lote, que é independente da velocidade de corte;
- Tempo relacionado com a troca da ferramenta. Quanto maior a velocidade de corte, menor a vida da ferramenta e maior é o numero de paradas da máquina para sua substituição. Portanto, maior também esta parcela.

$$T_i = T_{im} + T_{fr} \quad (2.10)$$

$$T_{fr} = \left(\frac{T_c}{T} - \frac{1}{Z} \right) T_{ft} \quad (2.11)$$

Onde:

T_{im} – tempo improdutivo independente da velocidade de corte (min)

T_{fr} – tempo improdutivo para a troca da ferramenta (min)

T – tempo de vida da ferramenta (min)

Z – número de peças do lote (unid)

T_{ft} – tempo de troca de ferramenta (min)

2.1.4.1 Torneamento

Torneamento é um processo mecânico destinado à obtenção de superfícies de revolução com auxílio de uma ou mais ferramentas monocortantes (FERRARESI, 1985). Resende (1993) complementa dizendo que o torneamento consiste em perfilar em torno de um eixo um sólido em rotação arrancando material perifericamente.

Conforme Chiaverini (1986), o torneamento é a operação por intermédio da qual um sólido indefinido é feito girar ao redor do eixo da máquina operatriz que executa o trabalho de usinagem, ao mesmo tempo em que uma ferramenta de corte lhe tira material perifericamente, de modo a transformá-lo numa peça bem definida, tanto em relação à forma quanto às dimensões.

O torneamento acontece mediante a retirada progressiva do cavaco da peça a ser trabalhada. O cavaco é cortado por uma ferramenta que deve ter uma dureza superior à do material a ser cortado (BRAGA, 1999).

De acordo com Trent e Wright (2000), o torneamento é a operação de usinagem mais comumente empregada em trabalhos de corte de metal. O material a ser cortado é fixado ao mandril de um torno e rotacionado, enquanto a ferramenta, presa firmemente em uma porta ferramenta, move-se em um plano que, idealmente, contém o eixo de rotação da peça.

O torneamento é uma operação muito utilizada pela indústria mecânica por causa do grande número de formas que pode conseguir, além de sua alta taxa de remoção de cavaco (AGOSTINHO *et al.*, 2004).

A máquina que faz o torneamento é chamada de torno. Segundo Braga (1999), basicamente, todos os tornos, respeitando-se suas variações de dispositivos ou dimensões exigidas em cada caso, são compostos das seguintes partes:

1. Corpo da máquina: barramento, cabeçote fixo e móvel, caixas de mudança de velocidade.

2. Sistema de transmissão de movimento do eixo: motor, polia, engrenagens, redutores.
3. Sistema de deslocamento da ferramenta e de movimentação da peça em diferentes velocidades: engrenagens, caixa de câmbio, inversores de marcha, furos e varas.
4. Sistemas de fixação da ferramenta: torre, carro porta ferramenta, carro transversal, carro principal ou longitudinal e da peça: placas e cabeçote móvel.
5. Comandos dos movimentos e das velocidades: manivelas e alavancas.

O torno é uma máquina operatriz bastante flexível que pode realizar uma grande variedade de operações, como: tornear superfícies cilíndricas externas e internas; tornear superfícies cônicas externas e internas; roscar superfícies externas e internas; e perfilar superfícies (BRAGA, 1999).

Os tornos podem ser classificados como: torno universal, de placa, revolver, vertical, copiador, semi-automático, automático, CNC controlado por computador. Cada tipo de torno possui características específicas com relação às formas construtivas, potência e capacidade de produção.

2.1.4.2 Fresamento

Dentre os processos convencionais de usinagem, o fresamento é um dos que apresenta maior versatilidade e complexidade. De acordo com Stemmer (1992), o fresamento é um processo de usinagem no qual a remoção de material se realiza de modo intermitente, pelo movimento rotativo da ferramenta, gerando superfícies das mais variadas formas.

Segundo Ferraresi (1985), o fresamento é uma operação de usinagem cuja ferramenta, chamada fresa, possui arestas cortantes dispostas simetricamente em torno de um eixo que executa a retirada do material auxiliada pelo movimento cortante. O movimento de avanço, que permite o prosseguimento da operação, é feito pela própria peça em usinagem que é fixada na mesa da máquina. A mesa obriga a peça a passar sob a ferramenta em rotação, dando-lhe a forma e dimensões desejadas.

Usualmente a peça efetua o movimento de avanço, enquanto que a ferramenta, girando à velocidade relativamente alta, realiza o movimento de corte, mantendo o eixo de giro numa posição fixa. De modo geral, tanto a peça quanto a ferramenta podem assumir movimentos relativos, independentes ou combinados, permitindo a realização de uma ampla variedade de operações, gerando superfícies planas ou curvas, ranhuras, ressalto, roscas, engrenagens e outras configurações (STEMMER, 1992).

De acordo com Ferraresi (1985), o processo de fresamento caracteriza-se principalmente pelos seguintes fatores:

- A ferramenta usada no processo, chamada de fresa, é multicortante, sendo provida de arestas cortantes dispostas simetricamente ao redor de um eixo;
- O movimento de corte é proporcionado pela rotação da fresa ao redor do seu eixo, fazendo com que cada uma das arestas cortantes (dentes) retire uma porção de material;
- O movimento de avanço é geralmente feito pela própria peça em usinagem, que está fixada na mesa da máquina, o que obriga a peça a passar sob a ferramenta em rotação, dando-lhe a forma e a dimensão desejadas.

Ferraresi (1985) classifica os tipos de fresamento em fresamento tangencial e frontal. No fresamento tangencial os dentes ativos (arestas cortantes) estão na superfície cilíndrica da ferramenta, sendo o eixo da fresa paralelo à superfície gerada. No fresamento frontal os dentes ativos estão na superfície da ferramenta, sendo o eixo da fresa perpendicular à superfície.

O fresamento é feito por fresadoras e ferramentas especiais chamadas fresas. O fresamento consiste na retirada do excesso de metal ou sobremetal da superfície de uma peça, a fim de dar-lhe a forma e o acabamento desejados (BRAGA, 1999).

As fresadoras classificam-se em relação ao eixo-árvore em horizontal, vertical e universal. A fresadora é horizontal quando seu eixo-árvore é paralelo à mesa da máquina. Se o eixo-árvore for perpendicular à mesa da máquina, chama-se de fresadora vertical. Já a fresadora universal dispõe de dois eixos-árvore, um horizontal e outro vertical. O eixo vertical situa-se no cabeçote, parte superior da máquina. O eixo horizontal localiza-se no corpo da máquina (BRAGA, 1999).

2.1.4.3 Furação

Segundo Resende (1993), a furação consiste em abrir uma cavidade cilíndrica numa massa metálica, mediante uma ferramenta de arestas cortantes, denominada broca. Para este fim, é provida de movimento rotatório contínuo e de movimento de avanço retilíneo, segundo o eixo de rotação, realizados por uma furadeira.

Essa operação de usinagem tem por objetivo abrir furos em peças. Ela é, muitas vezes, uma operação intermediária de preparação de outras operações, como alargar furos com acabamentos rigorosos, serrar contornos internos e abrir roscas (BRAGA, 1999).

O processo de furação é o processo mais competitivo, flexível, com secção circular e o mais utilizado para a realização de furos na maioria dos materiais, e em condições variadas (FERRARESI, 1985; STEMMER, 1992).

Segundo Ferraresi (1985), o processo de furação pode ser classificado em:

- Furação em cheio: obtenção de um furo cilíndrico em uma peça, onde todo o material contido no volume do furo é retirado na forma de cavaco;
- Furo com rebaixo: obtenção de dois diâmetros diferentes no mesmo furo, porém com profundidades diferentes. O diâmetro menor é passante e o maior tem uma menor profundidade;
- Furo escariado: obtenção de um diâmetro em ângulo;
- Furo cego: furação onde uma parte do material contido no volume do furo não é removido, ou seja, a ferramenta não atravessa a peça.

De acordo com Stemmer (1992), na furação deve-se levar em conta os seguintes fatores: diâmetro do furo, profundidade, tolerâncias de forma e de medidas, volumes de produção.

De acordo com Braga (1999), os principais tipos de furadeiras são:

- Furadeira portátil: usadas em montagens, na execução de furos de fixação de pinos, cavilhas e parafusos em peças muito grandes como turbinas, carrocerias. Pode ser elétrica e também pneumática;
- Furadeira de coluna: seu suporte principal é uma coluna na qual estão montados o sistema de transmissão de movimento, a mesa e à base. A coluna permite deslocar e girar o sistema de transmissão e a mesa, segundo

o tamanho das peças. A furadeira de coluna pode ser: de bancada (motores de pequena potência para fazer furos pequenos); de piso (geralmente usada para a furação de peças com diâmetros grandes);

- Furadeira radial: é empregada para abrir furos em peças pesadas, volumosas ou de difícil alinhamento. Possui um potente braço horizontal que pode ser baixado e erguido e é capaz de girar em torno da coluna;
- Furadeiras especiais: 1. Furadeira múltipla: possui vários fusos alinhados para executar operações sucessivas ou simultâneas em uma única peça ou em diversas peças ao mesmo tempo. É usada em operações seriadas nas quais é preciso fazer furos de diversas medidas; 2. Furadeira de fusos múltiplos - os fusos trabalham juntos, em feixes. A mesa gira sobre seu eixo central. É usada em usinagem de uma peça com vários furos e produzida em grandes quantidades de peças seriadas.

Conforme Resende (1993), a escolha de uma furadeira, do método e da aparelhagem certa para executar a furação de um determinado elemento deve ser feita com base nos seguintes elementos: forma da peça; dimensões da peça; número de furos a ser aberto na peça; quantidade a ser produzida; diversidade no diâmetro dos furos de uma mesma peça e; grau de precisão requerida.

2.1.4.4 Retificação

A retificação é um processo de usinagem que consiste na remoção de material da peça mediante abrasivo. O abrasivo é constituído de grãos cortantes que removem cavacos muito pequenos de material, motivo pelo qual o acabamento superficial da peça é melhor que o obtido nos demais processos de usinagem. Dependendo da superfície a ser retificada tem-se: retificação cilíndrica externa e interna; retificação plana tangencial e frontal; retificação de perfis (AGOSTINHO *et al.*, 2004).

De acordo com Vieira Junior *et al.* (1999), o processo de retificação ao longo dos tempos vem sendo considerado como um dos mais importantes da manufatura,

tendo como objetivo melhorar o acabamento superficial e garantir a integridade das peças acabadas.

De acordo com Stoeterau (2004), a retificação é um processo de usinagem com ferramenta de geometria não definida, utilizado principalmente em operações de acabamento, visando a melhoria da qualidade superficial, das tolerâncias geométricas e dimensionais. A retificação se caracteriza pela ação de grãos mais ou menos disformes, de materiais duros que são postos em interferência com o material da peça. Sua diferença básica dos processos com ferramentas de geometria definida está na impossibilidade de definir geometricamente os grãos abrasivos responsáveis pela usinagem, sendo que estes podem ser classificados quanto à sua forma geral, tamanho médio e tipo de material.

A retificação pode ser dividida em tangencial ou frontal:

- Retificação tangencial: a retificação é executada com a superfície de revolução da ferramenta e pode ser: cilíndrica (a superfície usinada é cilíndrica); cônica (a superfície usinada é cônica); perfis (a superfície usinada é uma superfície qualquer gerada pelo perfil do rebolo); tangencial plana (a superfície usinada é uma superfície plana); cilíndrica sem centros (a peça sem fixação axial é usinada por ferramentas abrasivas de revolução, com ou sem movimento longitudinal da peça);
- Retificação frontal: a retificação é executada com a face do rebolo. É geralmente executada na superfície plana da peça, perpendicularmente ao eixo do rebolo. A retificação frontal pode ser com avanço retilíneo da peça, ou com avanço circular da peça.

Há basicamente três tipos de retificadora: a plana, a cilíndrica universal e a cilíndrica sem centro (*center less*). Quanto ao movimento, em geral as retificadoras podem ser manuais, semi-automáticas e automáticas.

2.1.4.5 Mandrilamento

É o processo de usinagem destinado à obtenção de superfícies de revolução com auxílio de uma ou mais ferramentas de barra. Para tanto, a ferramenta gira e a

peça ou a ferramenta se deslocam simultaneamente seguindo uma trajetória determinada (AGOSTINHO *et al.*, 2004).

Segundo Resende (1993), o mandrilamento consiste em alargar uma câmara cilíndrica ou um furo, a fim de levá-los à medida desejada. O mandrilamento é executado pela mandriladora. As operações em mandriladora são preferidas para as peças de notáveis dimensões, ou então de fácil manuseio.

O mandrilamento permite obter superfícies cilíndricas ou cônicas internas, segundo eixos perfeitamente paralelos entre si e dentro de apreciáveis tolerâncias dimensionais. As peças submetidas ao mandrilamento caracterizam-se por serem de grandes dimensões e, portanto, de manuseio e montagens difíceis nas placas giratórias dos tornos (AGOSTINHO *et al.*, 2004).

De acordo com Pangrácio (2003), a operacionalização do mandrilamento é da seguinte forma: a ferramenta de corte é fixa em uma barra de mandrilar, com um ângulo determinado pela operação a ser realizada. O mandrilamento ocorre com a rotação da barra de mandrilar, e o sentido de avanço do corte se dá no eixo paralelo ao eixo da ferramenta, ficando a profundidade do corte dada pelo deslocamento da ferramenta em relação ao furo.

Os principais tipos de mandrilamento são cilíndrico, radial, cônico e esférico. O mandrilamento cilíndrico é o processo em que a superfície usinada é cilíndrica e o seu eixo de rotação coincide com o eixo em torno do qual a ferramenta gira. O mandrilamento radial é o processo em que a superfície é plana e perpendicular ao eixo em torno do qual gira a ferramenta. O mandrilamento cônico é processo em que a superfície usinada é cônica e seu eixo de rotação coincide com o eixo em torno da qual a ferramenta gira. O mandrilamento esférico é o processo em que a superfície usinada é esférica e o eixo de rotação coincide com o eixo em torno do qual a ferramenta gira (AGOSTINHO *et al.*, 2004).

2.1.4.6 Brochamento

De acordo com Stemmer (1992), o brochamento é um processo de usinagem, em que o movimento de corte é basicamente linear, caracterizando-se, porém, pelo emprego de uma ferramenta de dentes múltiplos, de alturas crescentes, dispostos

em série, denominada brocha, relativamente comprida, que pode ser usada na usinagem de furos, utilizando esforços de compressão ou tração (brochamento interno), ou arrastada sobre a superfície de uma peça (brochamento externo).

Com exceção de certas operações de brochamento circular, o movimento é retilíneo, realizado hidráulica ou mecanicamente. A ferramenta pode trabalhar à tração ou à compressão. As brochadeiras que comprimem a brocha são geralmente verticais, enquanto aquelas que tracionam a brocha podem ser horizontais ou verticais (AGOSTINHO *et al.*, 2004).

Segundo Agostinho *et al.* (2004), as vantagens destes processo são:

- Rapidez de execução;
- Boa precisão;
- Baixo custo de produção para altas séries;
- Não necessita de mão-de-obra especializada;
- Permite executar numa só passagem as operações executadas por vários processos diferentes de usinagem;
- O brochamento interno permite executar furos com formas variadas.

2.1.4.7 Aplainamento

Aplainamento é uma operação de usinagem feita com máquinas chamadas plainas e que consiste em obter superfícies planas, em posição horizontal, vertical ou inclinada. As operações de aplainamento são realizadas com o emprego de ferramentas que têm apenas uma aresta cortante que retira o sobremetal com movimento linear (BRAGA, 1999).

De acordo com Agostinho *et al.* (2004), aplainamento é um processo mecânico de usinagem destinado à obtenção de superfícies planas, geradas por um movimento retilíneo alternativo da peça ou da ferramenta. O aplainamento pode ser horizontal ou vertical. Quanto à finalidade, as operações de aplainamento podem ser classificadas ainda em aplainamento de desbaste e aplainamento de acabamento.

O aplainamento apresenta grandes vantagens na usinagem de réguas, bases, guias e barramentos de máquinas, porque a passada da ferramenta é capaz de retirar material em toda a superfície da peça. O aplainamento usa ferramentas de

corte com uma só aresta cortante que são mais baratas, mais fáceis de afiar e com montagem mais rápida. Isso significa que o aplainamento é, em regra geral, mais econômico que outras operações de usinagem que usam ferramentas multicortantes (BRAGA, 1999).

Tanto a máquina quanto o ferramental de uma operação de aplainamento são relativamente baratos, mas ela é uma operação lenta, o que faz com que ela não seja muito utilizada na produção seriada e sim na fabricação de pequenos lotes de peças (AGOSTINHO *et al.*, 2004).

De acordo com Braga (1999), as operações de aplainamento são realizadas em dois tipos de máquinas:

- Plaina limadora: que pode ser: vertical ou horizontal. Apresenta movimento retilíneo alternativo (vaivém) que move a ferramenta sobre a superfície plana da peça retirando o material. Significa que o ciclo completo divide-se em duas partes: em uma (avanço da ferramenta) realiza-se o corte; na outra (recuo da ferramenta), não há trabalho, ou seja, é um tempo perdido;
- Plaina de mesa: executa os mesmos trabalhos que as plainas limadoras podendo também ser adaptada até para fresamento e retificação. A diferença entre as duas é que, na plaina de mesa, é a peça que faz o movimento de vaivém. A ferramenta, por sua vez, faz um movimento transversal correspondente ao passo do avanço. Usina qualquer superfície de peças como colunas e bases de máquinas, barramentos de tornos, blocos de motores diesel marítimos de grandes dimensões.

2.1.4.8 Alargamento

De acordo com Weinert *et al.* (1998), alargamento é definido como um tipo de furação utilizando uma ferramenta que produz cavacos com pequenas dimensões e proporciona superfícies com alta qualidade superficial e precisão dimensional e de forma.

O processo de furação com brocas helicoidais consiste numa operação de desbaste. Para calibrar o furo e melhorar o acabamento superficial originados pela furação com broca helicoidal empregam-se geralmente os alargadores para o

acabamento. São ferramentas multicortantes, geralmente de forma cilíndrica ou cônica, que, através dos movimentos de corte (rotativo) e de avanço (axial), servem para alargar e acabar os furos (AGOSTINHO *et al.*, 2004).

Segundo Resende (1993), o alargamento é a operação que tem por fim ajustar o diâmetro de um furo, produzindo um bom acabamento da superfície usinada, com grande precisão, removendo pequena quantidade de material. O alargador é uma ferramenta multicortante, geralmente de forma cilíndrica ou cônica que, no seu movimento rotatório, serve para alargar e acabar furos.

Conforme Dos Santos (2004), os principais parâmetros para a escolha do alargador são: material e geometria da peça; tipo de máquina; tipo de fixação da peça e da ferramenta; e acabamento e tolerâncias desejados.

2.1.5 Processos de Movimentação Interna

A movimentação interna na manufatura é responsável por toda a movimentação dos produtos ou peças dentro do sistema de manufatura. Essa movimentação ocorre desde o local de armazenamento de matéria-prima até o de produto acabado, passando pelas várias operações de fabricação.

De acordo com Arnold (1999), o manuseio de materiais consiste no transporte à curta distância que ocorre no interior ou em torno de uma edificação, como um centro de distribuição ou uma fábrica.

De acordo com Martins e Alt (2006), o tempo e o custo despendidos no transporte ou movimentação de materiais não agregam valor ao produto e devem, portanto, ser minimizados até atingir o indispensável para garantir o trânsito entre o armazém de entrada e o despacho do produto acabado.

Num sistema de manufatura, busca-se realizar a movimentação ou manuseio de materiais de forma mais rápida e de baixo custo, pois essa movimentação agrega valor e é um ponto importante na redução de custos em relação aos concorrentes. Equipamentos ou métodos de movimentação interna ineficientes podem acarretar altos custos para a organização devido ao fato de que as atividades de movimentação são repetidas muitas vezes ao longo de um período de tempo e

envolvem a segurança e integridade dos produtos e dos operários participantes do sistema de manufatura.

Segundo Martins e Alt (2006) e Moura (1997), para a realização da movimentação interna é necessária uma sequência de perguntas que os autores denominaram de equação da movimentação física (ver figura 2.3). A sequência de análise deve ser respeitada da esquerda para a direita.

Figura 2.3 - Equação da movimentação física

Fonte: Martins e Alt (2006)

Para Pozo (2007), os principais objetivos do manuseio de materiais são: otimizar a utilização cúbica das edificações de produção e armazenagem; otimizar a eficiência operacional; reduzir os custos de movimentação; otimizar a carga de cada transporte; melhorar o atendimento ao mercado.

De acordo com Tompkins *et al.* (2010), dez princípios são diretrizes para a movimentação de materiais:

1. Planejamento: é necessário determinar o melhor método, do ponto de vista econômico, planejando todas as suas atividades.
2. Padronização: padronizar significa uma menor variedade e customização dos métodos e equipamentos de movimentação.
3. Trabalho: o cálculo do trabalho a ser realizado por um equipamento de movimentação é realizado pela multiplicação do peso e do volume pela distância movimentada.
4. Ergonomia: a capacidade e limitações humanas precisam ser reconhecidas e respeitadas no projeto das tarefas e dos equipamentos.

5. Carga Unitizada: unitizar a carga possibilita transportar e armazenar como uma única entidade. Um exemplo é a paletização.
6. Utilização do espaço: aproveitamento dos espaços verticais contribui para a redução dos espaços requeridos e dos custos.
7. Sistema: as interações entre entidades interdependentes que formam um todo unificado gerando ganhos de produtividade e redução de custo.
8. Automação: as operações de movimentação de materiais podem ser automatizadas para melhorar o fluxo e a eficiência operacional.
9. Ambiental: considerar as possíveis implicações sobre o meio ambiente, tais como consumo exagerado de energia, desperdícios, dejetos e poluição, reduzindo-as ao máximo.
10. Custo do ciclo de vida: incluem todos os fluxos de caixa que irão ocorrer a partir do momento em que é gerado um gasto para planejar ou adquirir um novo equipamento, ou de por em prática um novo método, até que o equipamento ou método seja substituído.

Segundo Ballou (2006), o equipamento de movimentação ou manuseio de material é diferenciado pelo seu grau de uso especializado e pela extensão da força manual para operá-lo. Existem três categorias principais de equipamentos:

- Equipamento Manual: são equipamentos de baixo custo e com grande flexibilidade. No entanto, a utilização desses equipamentos é limitada em função da capacidade física dos operadores. Exemplos: carrinhos de duas rodas e as paleteiras de quatro rodas;
- Equipamento Misto: são equipamentos que unem a rapidez, eficiência com flexibilidade e custo relativamente baixo. Exemplos: guindastes, elevadores, guinchos, esteiras transportadoras e empilhadeira mecânica e suas variações;
- Equipamento Automatizado: são equipamentos controlados por computadores, códigos de barras e tecnologia de escaneamento. Exemplos: Veículos Guiados Automaticamente (AVGs).

De acordo com Bowersox *et al.* (2006), os principais equipamentos de movimentação interna são:

- Paletes: equipamento versátil e de baixo custo, mas com capacidade limitada. São muito utilizados na transferência de pequenas cargas em maiores distâncias dentro do sistema de manufatura;
- Esteiras Transportadoras: método convencional bastante utilizado. As esteiras são classificadas de acordo com o tipo de acionamento (energia ou gravidade) e o tipo de movimento (rolete ou correia);
- Empilhadeiras: podem movimentar produtos horizontalmente e verticalmente. Utilizadas em conjunto com o paleta podem empilhar vários produtos para uma melhor eficiência do transporte. Há vários tipos de empilhadeiras como: de longo alcance, com garras laterais e trilaterais;
- Veículos Guiados Automaticamente (AVGs): possuem trilhos que podem ser construídos com faixas ou por produtos químicos; podem ser inteligentes, sendo compostos por um microcomputador de bordo; a comunicação com o sistema de controle pode ser feita através de dispositivos ópticos ou magnéticos.

Moura (1997) classifica os equipamentos de movimentação de materiais da seguinte forma:

- Transportadores: correias, correntes, fitas metálicas, roletes, rodízios, roscas e vibratórios;
- Guindastes, talhas e elevadores: guindastes fixos e móveis, pontes rolantes, talhas, guinchos, monovias e elevadores;
- Veículos industriais: carrinhos de todos os tipos, tratores, trailers e veículos especiais para transporte a granel;
- Equipamento de posicionamento, pesagem e controle: plataformas fixas e móveis, rampas, equipamentos de transferência;
- Containers e estruturas de suporte: vasos, tanques, suportes e plataformas, estrados, paletes, suportes para bobinas e equipamento auxiliar de embalagem.

Segundo Dias (1993), os custos de movimentação afetam diretamente o custo do produto final, mas não contribuem em nada para agregar mais valor. Uma seleção adequada do método mais compatível à natureza e ao regime da produção,

gerando uma redução nos roteiros percorridos pelo produto em suas diversas etapas, do estoque à expedição, proporciona redução de custos.

Conforme Da Silva (2002), para o cálculo do custo de movimentação de materiais para um operador é correto utilizar a taxa do custo fixo baseada no número de horas trabalhada da mão-de-obra, o tempo de trabalho (movimentação) e o salário da mão-de-obra.

$$C_{mo} = S_o + \frac{C_F}{H_{mdo}} \times t_t \quad (2.12)$$

Onde:

C_{mo} – custo de movimentação (R\$)

S_o – salário do operador (hora)

C_F – custo fixo (R\$)

H_{mdo} – número total de horas de mão-de-obra utilizada (hora)

t_t – tempo de trabalho (hora)

De acordo com Da Silva (2002, p.73), “no caso da movimentação ocorrer por esteira, por empilhadeira ou AVG, ou por qualquer outro meio não-mensurável, pode-se, simplesmente, definir um custo por hora para a movimentação”.

Segundo Tompkins *et al.* (2010), o custo de movimentação de materiais influencia no *layout* da fábrica e pode ser minimizado (equação 2.13).

$$z = \sum_{i=1}^m \sum_{j=1}^m f_{ij} c_{ij} d_{ij} \quad (2.13)$$

Onde:

z – custo de movimentação entre equipamentos (R\$)

m – número de equipamentos (unid)

i e j – indexadores dos setores analisados

f_{ij} – fluxo de transportes do equipamento i para o equipamento j (unid)

c_{ij} – custo unitário do transporte entre os equipamentos (R\$)

d_{ij} – distância entre os equipamentos i e j (m)

2.2. TECNOLOGIA DE *FEATURES*

O objetivo deste tópico é apresentar uma revisão bibliográfica tecnologia de *features* aplicada a sistemas de manufatura. Inicialmente detalha-se a evolução da utilização das *features* na manufatura e a definição de *features*, após, aborda-se as vantagens e limitações da utilização das *features*, assim como os principais tipos de *features*.

Em seguida, são apresentados os cálculos para obtenção de tempos e custos de manufatura baseados nas *features* e como realizar o reconhecimento, projeto e mapeamento de *Features*. A última parte do tópico refere-se à aplicação de *features* na manufatura.

2.2.1 Evolução da Utilização de *Features* na Manufatura

De acordo com Schutzer *et al.* (1992), *features* foram introduzidas na segunda metade dos anos 70 no campo de engenharia, fornecendo uma linguagem para a descrição de geometria de uma peça.

Segundo Shah *et al.* (1994), o primeiro trabalho relacionado com as *features* foi realizado por Grayer durante seu doutorado em Cambridge, em 1976, onde *features* foram utilizadas para a automatização da geração de programas Comando Numérico (NC) com base em desenhos feitos em um sistema CAD (Projeto Assistido por Computador).

O aumento da utilização de *features* na manufatura se deu a partir do crescimento das técnicas de modelagem de sólidos que gerou o surgimento de uma grande quantidade de sistemas CAD sofisticados com interfaces tridimensionais (HAN *et al.*, 2000).

Em seguida os sistemas CAD foram interligados aos sistemas CAM (Manufatura Assistida por Computador) com a utilização das *features*. Bhandarkar *et al.* (2000) afirmam que as *features* permitiram o sucesso na ligação entre os sistemas CAD/CAM que foi um passo fundamental para a automatização do desenvolvimento do produto desde o estágio de projeto até à manufatura.

2.2.2 Definição de *Features*

Para transformar uma peça bruta em uma peça acabada são necessários processos de fabricação que podem ser definidos a partir de informações contidas na peça. Esse conjunto de informações pode ser armazenado no projeto da peça em forma de *features*.

O conceito de *features* assume várias definições cada uma formulada com base em conceitos de uma área específica. Segundo Lee (2004), a *feature* pode ser definida de diversos pontos de vista, tais como, projeto, análise, montagem, ou diversas funções relativas à manufatura.

De acordo com Zhang *et al.* (1999), pesquisadores fornecem definições diferentes para *features* através do estudo de diversos ângulos desde o desenvolvimento inicial do conceito de *features* em 1976.

De acordo com Shah e Mantyla (1995), *features* são formas geométricas definidas por um conjunto de parâmetros que têm significado especial para engenheiros de projeto e fabricação. A Figura 2.4 mostra exemplos de *features*.

Figura 2.4 - Exemplo de *features* em peças

Fonte: Benavente, (2007)

Segundo Gao *et al.* (2004), as *features* são usadas para dar um significado ao produto com relação aos seus atributos e sua geometria e melhoram a comunicação entre o projeto e a fabricação.

Features correspondem a uma forma de representar o significado de engenharia num produto ou de um processo de manufatura, através de entidades geométricas. Assim, as *features* descrevem um modelo de produto de maneira mais inteligente do que a geometria explícita.

Já Mayer *et al.* (1994) apresentam várias definições de *features*, cada uma aplicada a uma área distinta: de forma (geometria e topologia de uma peça), de tolerância (desvios aceitáveis nas dimensões de uma peça), de material (propriedades mecânicas de uma peça), funcional (funcionalidade da peça), *features* de montagem (operações de montagem).

Através das *features*, pode-se compor uma estrutura de dados para representar uma peça ou uma montagem em termos dos seus constituintes, onde (BENAVENTE, 2007):

- Cada *feature* tem uma identidade única;
- A forma pode ser expressa em termos de: parâmetros dimensionais; enumeração geométrica; relações e entidades topológicas; etapas de construção para produzir a geometria das *features*.

De acordo com Schutzer *et al.* (1992), *features* associam aspectos funcionais e semânticos com geometria do produto:

- O aspecto funcional está ligado à forma de agrupamento dos itens geométricos, que são combinados logicamente. Elas descrevem a forma geométrica de um produto. A geometria é descrita por meio de parâmetros. Um exemplo é o furo, que pode ser descrito pelos parâmetros de diâmetro, profundidade, plano de referencia e a localização;
- O aspecto semântico diz respeito ao significado das *features*. Este significado está diretamente relacionado com a aplicação. De acordo com a aplicação, existem diferentes semânticas que são conectadas com a geometria. O projetista busca através das *features* de forma descrever as funções de um componente, enquanto que o engenheiro de manufatura está interessado em definir a máquina-ferramenta e o processo como os quais ele pode fabricar a

geometria do produto. Portanto, existem diferentes tipos para as *features* de forma e as *features* de manufatura.

2.2.3 Vantagens e Limitações da Utilização de *Features*

De acordo com Maziero (1998), a principal vantagem da utilização de *features* é a maior rapidez em produzir um projeto e na interligação com a fabricação, havendo, conseqüentemente, redução do custo final do produto. Outras vantagens são: não induz a erros ou distorções de interpretação pelo sistema, pois o objeto modelado sempre corresponderá a uma representação validada pelo próprio usuário e possibilita o desenvolvimento de aplicações que verificam as condições de manufatura, de forma a validar o modelo geométrico antes da manufatura.

Segundo Shah e Mantyla (1995), as características das *features* asseguram algumas vantagens em relação aos modelos geométricos que são:

- Os modelos podem ser descritos usando entidades de alto nível definidos como abstrações genéricas de formas recorrentes. Os relacionamentos entre as entidades caracterizam parâmetros e restrições e disponibilizam uma base para a propagação de alterações no modelo;
- Conduzem à descrição do projeto em termos de entidades orientadas para a aplicação;
- Conduzem a uma interface melhorada para manipulação de geometrias de projeto;
- As *features* podem realizar um trabalho de validação de alterações muito melhor, uma vez que carregam a intenção do projeto.

Numa modelagem baseada em *features*, o usuário somente pode utilizar aquelas *features* que estão definidas na biblioteca, o que leva a uma limitação quanto ao universo a ser utilizado. Outra limitação da utilização das *features* é quanto a especialização de sistemas de manufatura. Quanto mais simples (alto nível) for a forma geométrica da *feature*, maior a flexibilidade de modelagem, porém menor a especialização do sistema (MAZIEIRO, 1998).

2.2.4 Principais Tipos de *Features*

De acordo com Yang e Wang (1992), para um projetista, uma *feature* representa um projeto específico com funcionalidades baseadas na forma. No entanto, para um processista uma *feature* ilustra características ligadas ao processo de manufatura. A Figura 2.5 detalha as *features* padrão de projeto e de manufatura.

De acordo com Han *et al.* (2000), o principal problema do projeto e modelagem utilizando *features* é que as *features* que são mais comuns para o uso durante o desenvolvimento do projeto, não são, necessariamente, as *features* de manufatura.

Figura 2.5 - *Features* padrão ao projeto e à manufatura

Fonte: Adaptado de Yang e Wang (1992)

Por exemplo, na Figura 2.6 a peça pode ser concebida pela adição de um bloco na base da peça, como ilustrado no modelo de *features* desenvolvido pelo setor de projetos. No entanto, a modelagem de *features* de manufatura para a peça pode ser definido subtraindo-se dois blocos da peça bruta (HAN *et al.*, 2000).

Figura 2.6 - Uma peça com diferentes modelos de *features*

Fonte: Adaptado de Han *et al.* (2000)

Para uma melhor compreensão das diferenças entre as *features* desenvolvidas na fase de projeto e as *features* utilizadas para a manufatura das peças detalha-se nas próximas seções as *features* de forma e de manufatura.

2.2.4.1 *Features* de Forma

Segundo Shah *et al.* (1994), *features* de projeto ou forma são elementos físicos de uma peça que podem ser identificados por uma forma e por alguns atributos. De acordo com Han e Reguicha (1998), *features* de forma podem ser tanto positivas como negativas, dependendo se são compostas de saliência ou de uma depressão.

De acordo com Yang e Wang (1992), *features* de forma são definidas por um conjunto de parâmetros que são encapsulados como atributos de classes específicas. Quando esses parâmetros são alterados, características intrínsecas das

features são atualizadas simultaneamente. As *features* de forma podem ser classificadas em três grupos:

- Um primeiro grupo de *features* são as que têm dimensões padrão. Por exemplo, uma ranhura possui dimensões padrões, ou ainda, os atributos de uma chaveta, como largura, altura e comprimento são encapsulados em uma classe chamada chaveta e não são acessíveis por outras classes;
- No segundo grupo, as *features* não têm dimensões padrão, como por exemplo, chanfro ou filete. As suas dimensões devem ser fornecidas pelo projetista e, em seguida, os dados armazenados e parametrizados;
- O terceiro é o grupo definido de *features* que são projetadas para expandir a categoria de *features* de forma para facilitar a o desenvolvimento do projeto.

De acordo com Gao e Huang (1996), *features* de forma são classificadas da seguinte maneira:

- Atômicas: são formas geométricas básicas, tais como: pontos, linhas, arcos, planos e superfícies, as quais, quando reunidas darão origem às *features* primitivas;
- Primitivas: são rasgos, furos e chavetas formadas a partir das *features* atômicas;
- Compostas: são coleções de primitivas e/ou atômicas, as quais juntas podem executar uma função fornecida ou serem produzidas por operações similares.

Features, conhecidas como de forma, representam a maioria dos tipos de *features* utilizadas na manufatura. Cada uma possui um conjunto de possíveis métodos de manufatura, por exemplo: um furo pode ser obtido por uma operação de furação ou mandrilamento (CANCIGLIERI, 1999).

Segundo Maziero (1998), a classificação das *features* é necessária para que sejam conhecidos realmente os tipos de *features* existentes e a sua localização dentro do contexto em análise.

As *features* de forma são classificadas, inicialmente, quanto ao modo de representação em: paramétricas, as quais podem ser instanciadas através de um ou mais conjuntos de parâmetros geométricos; por exemplo, um furo pode ser definido pelos seguintes parâmetros: diâmetro e profundidade; e não-paramétricas, que são definidas ou através de um conjunto de entidades geométricas de baixo nível

explicitamente identificadas ou através de conjuntos específicos de *features* paramétricas (LIMA, 1994).

De acordo com Maziero (1998), as *features* paramétricas são subdivididas em função da forma geral da *feature*, resultando, assim, em duas subdivisões principais, que são: as rotacionais e as prismáticas. A Figura 2.7 mostra exemplos dessas *features*.

Figura 2.7 - *Features* rotacional e prismática
Fonte: Maziero (1998)

2.2.4.2 *Features* de Manufatura

De acordo Farias (2008), entende-se por *features* de manufatura aquelas características geométricas extraídas do sistema CAD e que possam ser fabricadas com recursos disponíveis, como máquinas e ferramentas. Complementam Han e Requicha (1998) dizendo que uma *feature* de manufatura é tipicamente uma *feature* de forma negativa associada às características do processo de usinagem.

Segundo Lui e Wang (2007), *features* de manufatura são definidas como o volume retirado na operação de manufatura. Esse volume corresponde ao material que é removido na operação de usinagem. Essas *features* de manufatura têm uma forma padrão a partir de um conjunto de informações sobre como fabricar, ou seja, fornecem uma indicação quanto ao tipo de operação e às ferramentas que serão necessárias para fabricar a *feature*.

Jung (2002) classifica as *features* de manufatura em quatro classes que são: rotacional; prismática; cavidade e; circular. Para cada classe de *features* utiliza-se

processos de usinagem deferentes como, por exemplo, na *feature* circular furo utiliza-se o processo de fabricação furação ou alargamento. A Figura 2.8 detalha as quatros classes e suas principais *features*.

Figura 2.8 - Tipos de *features*

Fonte: Jung (2002)

De acordo com Álvares (2005), associado às *features* de manufatura, no caso de processos de fabricação com remoção de material, tem-se as *features* de usinagem. Uma *feature* de usinagem é constituída por: *features* volumétricas que é o volume de material removido pela operação de usinagem para transformar a peça bruta em peça acabada. O volume removido é denominado de volume delta ou volume removido e; *features* de superfície: é uma coleção de faces na peça que resulta da usinagem (subtração) de uma *feature* volumétrica.

Cada *feature* de usinagem é definida por uma coleção de parâmetros e combinação de fronteira. Do ponto de vista da usinagem, a forma como a remoção de material é feita depende do movimento de corte, do movimento de avanço, da forma do corte e dos parâmetros de configuração. As principais formas de *features* de usinagem são torneamento, fresamento, furação (ZHANG *et al.*, 1999).

2.2.5 Tempo e Custo de Manufatura das *Features*

Segundo Jung (2002), a vantagem de calcular o custo de fabricação baseado na tecnologia de *features* é a possibilidade de lidar com um número infinito de formas geométricas da peça manufaturada a partir das informações fornecidas pelas *features*. Estimar custos através do cálculo das operações de usinagem para fabricação da *feature* é um paradigma recente em comparação com a estimativa tradicional de experiência na área de usinagem.

O cálculo do custo de manufatura é baseado no tempo de usinagem e no custo do material. O tempo de usinagem é composto pelo tempo de *setup*, tempo de operação e tempo de não operação. O tempo de operação e de não operação são proporcionais à quantidade de produção. O tempo de *setup* é proporcional ao tamanho do lote, ou seja, o tempo de *setup* por peça é o tempo de *setup* dividido pelo tamanho do lote. O cálculo do tempo e custo de manufatura são realizados de acordo com as equações 2.13 e 2.14 (JUNG, 2002).

$$T_m = T_{su} + T_{ot} + T_{no} \quad (2.13)$$

$$C_m = (R_o + R_m) \left(\frac{T_{su}}{Q} T_{ot} + T_{no} \right) + C_{ma} \quad (2.14)$$

Onde:

T_m – tempo de manufatura (min)

T_{ot} – tempo total de operação (min)

T_{no} – tempo total de não-operação (min)

C_m – custo de manufatura (R\$/peça)

R_o – salário do operador (R\$/hora)
 R_m – salário da máquina (R\$/hora)
 T_{su} – tempo de *setup* (min)
 Q – tamanho do lote de produção (unid)
 C_{ma} – custo do material (R\$/peça)

O tempo de total de *setup* de um lote de peças é o somatório do tempo de *setup* das máquinas utilizadas e do tempo de *setup* de cada ferramenta utilizada na máquina (JUNG, 2002).

$$T_{su} = \frac{\sum_i ta_i + \sum_i \sum_j tb_{ij}}{Q} \quad (2.15)$$

Onde:

ta_i – tempo de *setup* da máquina (min)
 tb_{ij} – tempo de *setup* das ferramentas (min)
 i – quantidade total de máquinas (unid)
 j – quantidade total de ferramentas (unid)

De acordo com Jung *et al.* (2000), o tempo operacional é composto pelo tempo de corte (desbaste), o tempo de acabamento e o tempo de aproximação da ferramenta de acordo com a equação 2.16.

$$T_{ot} = \sum_{i=0}^n Tr_i + \sum_{i=0}^n Tf_i + \sum_{j=0}^m Ta_j \quad (2.16)$$

Onde:

T_{ot} – tempo total de operação (min)
 Tr_i – tempo de corte (desbaste) da *feature* (min)
 Tf_i – tempo de acabamento da *feature* (min)
 Ta_j – tempo de aproximação da ferramenta (min)
 n – quantidade de *features* na peça (unid)
 m – quantidade de ferramentas utilizadas (unid)

O tempo de corte (desbaste) é proporcional ao volume removido na usinagem da peça. O volume de usinagem removido é diretamente proporcional ao volume das

features de manufatura que pode ser obtido através de cálculos matemáticos de acordo com a Tabela 2.1.

Tabela 2.1 - Cálculo do volume das principais *features*
Fonte: Jung (2002)

Tipos de <i>Features</i>	Volume (mm ³)
Passo	$\pi \times (\text{DiametroMaior}^2 - \text{DiametroMenor}^2) \times (\text{Largura}/4)$
Chanfro	$\pi \times ((\text{DiametroMaior}^2 \times \text{DiametroMenor}^2)/4 - (\text{DiametroMenor}^2/12)) \times \text{Largura}$
Arredondamento	$\pi \times \text{RaioMaior}^2 \times (\text{RaioMaior} - \text{RaioMenor}) - \pi \times [(\text{RaioMaior} - \text{RaioMenor}) \times \text{RaioMenor}^2 + (\text{RaioMaior} - \text{RaioMenor})^2 \times \text{RaioMenor} \times \pi/2 + (\text{RaioMaior} - \text{RaioMenor})^3 \times 2/3]$
Plano	Altura x Comprimento x Largura
Degrau	Altura x Comprimento x Largura
Ranhura	Altura x Comprimento x Largura
Entalhe	Altura x Comprimento x Largura
Depressão	Altura x Comprimento x Largura
Rasgo	Altura x Comprimento x Largura
Furo	$\pi \times (\text{Diametro}^2 \times (\text{Altura}/4))$

Assume-se que o tempo de corte é calculado para a máxima potência da máquina e que a máquina possui força suficiente para cortar o material. Conforme Jung (2002), o tempo de corte (desbaste) é calculado de acordo com a equação 2.17:

$$T_r = \frac{V_r}{MRR} \quad (2.17)$$

$$MRR_r = d \times f \times s \quad (2.18)$$

$$MRR_p = D \times d \times s \quad (2.19)$$

$$MRR_{ca} = l \times d \times s \quad (2.20)$$

$$MRR_{ci} = A \times f \quad (2.21)$$

Onde:

V_r – volume do material removido (mm^3)

MRR_r – taxa de remoção do material de *features* rotacionais (mm^3/min)

MRR_p – taxa de remoção do material de *features* prismáticas (mm^3/min)

MRR_{ca} – taxa de remoção do material de *features* cavidades (mm^3/min)

MRR_{ci} – taxa de remoção do material de *features* circulares (mm^3/min)

d – profundidade do corte (mm)

D – diâmetro do corte (mm)

l – comprimento do corte (mm)

f – avanço (mm/v)

s – velocidade de corte (mm/min)

A – capacidade da broca (mm^2/min)

Além da operação de desbaste outro tipo de operação realizada na usinagem é o acabamento. Esta operação não está fortemente associada à remoção de material da peça, mas à precisão dimensional, à rugosidade da superfície e desvios geométricos. O tempo de acabamento é proporcional à área de corte das *features*, que pode ser obtida por equações matemáticas de acordo com a Tabela 2.2.

Tabela 2.2 - Cálculo da área das principais *features*

Fonte: Jung (2002)

Tipos de <i>Features</i>	Área (mm^2)
Passo	$\pi \times \text{DiâmetroMenor} \times \text{Largura} + \pi \times (\text{DiâmetroMaior}^2 - \text{DiâmetroMenor}^2)/4$
Chanfro	$\pi \times (\text{RaioMaior} + \text{RaioMenor}) \times \sqrt{\text{Largura}^2 (\text{RaioMaior} - \text{RaioMenor})^2}$
Arredondamento	$\pi^2 \times (\text{RaioMaior} - \text{RaioMenor}) \times \text{RaioMenor} + 2 \times \pi \times [(\text{RaioMaior} - \text{RaioMenor})^2]$
Plano	Comprimento x Largura
Degrau	Altura x Comprimento + Comprimento x Largura
Ranhura	Altura x Comprimento + Comprimento x Largura
Entalhe	Altura x Comprimento x+ Comprimento x Largura + Altura x Largura
Depressão	Altura x Comprimento + 2 Comprimento x Largura + Altura x Largura
Rasgo	Altura x Comprimento + 2 Comprimento x Largura + 2 Altura x Largura
Furo	$\pi \times \text{Diâmetro} \times \text{Altura}$

O tempo de acabamento é obtido a partir da equação 2.22 (JUNG,2002).

$$T_f = \frac{A_f}{R_t} \quad (2.22)$$

Onde:

A_f – área da *feature* (mm²)

R_t – taxa de remoção da área da *feature* (mm²/min)

Segundo Jung *et al.* (2000), o tempo de aproximação da ferramenta varia de acordo com o tipo de máquina, o diâmetro da peça e localização de repouso da ferramenta. Geralmente, o tempo de aproximação é muito pequeno comparado com o tempo de corte, sendo em muitas situações estimado baseado no tempo de corte.

O tempo de não operação ou tempo improdutivo foi definido na seção 2.1.4 por Diniz *et al* (2001) como sendo o tempo referente à colocação, inspeção e retirada da peça, substituição da ferramenta e preparo da máquina para usinagem de um lote, que é independente da velocidade de corte.

2.2.6 Reconhecimento, Projeto e Mapeamento de *Features*

De acordo com Han e Requicha (1998), modelos de *features* podem ser desenvolvidos através de dois métodos:

- Reconhecimento de *features*: peças projetadas através de operações de modelagem de sólidos exigem reconhecimento de *features*;
- Projeto baseado em *features*: permite que o projetista use *features* como blocos de construção para criar uma peça, e normalmente usa *features* de projeto (forma).

No reconhecimento das *features* de projeto, a peça é inicialmente construída com o uso de sistemas CAD. Posteriormente, são extraídas as *features* de manufatura utilizando-se informações geométricas do sistema CAD. Existem duas formas de reconhecimento de *features*:

- Reconhecimento manual: o usuário interage com o sistema que possibilita a identificação dos elementos que irão compor as *features* de manufatura;

- Reconhecimento automático: existem diferentes metodologias que procuram reconhecer automaticamente, a partir de um modelo geométrico da peça, o maior número possível de *features* de manufatura.

No Projeto Baseado em *Features*, as peças são criadas diretamente usando as *features* e o modelo geométrico é gerado do modelo de *features*. Isto requer que o sistema de projeto (CAD) tenha definições genéricas de *features* disponibilizadas pela biblioteca de *features*, permitindo a instânciação das *features* pela especificação de dimensões e parâmetros de localização (SHAH E MANTYLA, 1995).

De acordo com Shah e Mantyla (1995) e Álvares e Pereira (2003), as duas metodologias de Projeto por *Features* utilizadas são:

- Geometria Destrutiva de Sólidos (*Destructive Solid Geometry*): inicia-se com um modelo de peça bruta que sofre subtração de *features* que correspondem ao material removido por operações de usinagem. As *features* de usinagem estão diretamente disponíveis no modelo da peça não sendo necessário o reconhecimento ou mapeamento de *features*. Por outro lado, o projetista deve ter um amplo conhecimento de manufatura para pensar em termos de *features* de manufatura;
- Síntese por *Features* de Projeto (*Synthesis by Features*): o modelo é construído pela adição ou subtração de *features*, não sendo necessário iniciar com um modelo de peça bruta.

A atividade de mapeamento de *features* de usinagem depende da geometria da peça bruta e da peça acabada, dos métodos de usinagem e da sequência de operações de fabricação. Em função destas escolhas pode-se ter mapeadas diferentes *features* de usinagem.

Segundo Álvares e Ferreira (2003), as *features* de usinagem são mapeadas a partir das *features* de projeto disponíveis no banco de dados, através da decomposição volumétrica baseada em critérios geométricos, *setup* da peça e nas operações de usinagem.

De acordo com Álvares e Pereira (2003), modelos das *features* são dependentes do domínio, ou seja, da aplicação. Quando uma peça é projetada por *features*, o modelo resultante nem sempre é conveniente para uma dada aplicação

na usinagem. Assim é necessário a transformação de uma visão para outra, sendo este processo denominado de mapeamento de *features*. Esse mapeamento deve ser realizado porque uma mesma peça pode ser constituída com grupos de *features* de diferentes tipos, dependendo da tarefa de engenharia a ser executada.

2.2.7 Aplicações de *Features* na Manufatura

Vários autores desenvolveram ao longo dos anos aplicações com as *features* com o objetivo de automatizar a interface entre Projeto Assistido por Computador (CAD) e o Manufatura Assistida por Computador (CAM) para com isso conseguir a total integração de um sistema produtivo.

Autores como Kumar e Sing (2007), Sudarsan *et al.* (2005) e Zhou *et al.* (2007) desenvolveram trabalhos com enfoque na utilização de sistemas especialistas baseados em regras de produção em conjunto com a representação geométrica a partir de *features*.

Segundo Han *et al.* (2000), ao longo dos anos foram desenvolvidos vários algoritmos para detectar a informação a partir da inserção de *features* em modelos produzidos por sistemas CAD.

Sheu (1998) desenvolveu um sistema integrado para fabricação de peças rotacionais. As *features* foram utilizadas para especificar as informações de fabricação exigidas para o sistema. Os produtos foram projetados a partir de primitivas como cilindros, arcos e cones. Esse sistema tinha capacidade de converter um modelo *wireframe* em uma representação CSG (*Constructive Solid Geometry*) pela abordagem reconhecimento das *features*. Apenas um número limitado de *features* poderia ser reconhecidas pelo sistema e os produtos prismáticos não foram considerados.

Rezende (1996) desenvolveu um sistema de Planejamento do Processo Assistido por Computador (CAPP) baseado num sistema especialista para aplicação numa célula de manufatura para torneamento de peças. O sistema modela as informações através de *features*, organiza estas informações para o modelo de manufatura e então define as operações de usinagem bem como as ferramentas a serem utilizadas para ter como resultado o plano de processo.

Hoque e Szecsi (2008) desenvolveram uma aplicação para utilizar *features* de geometrias parametrizadas para descrever o processo de manufatura buscando mostrar as limitações geométricas do produto durante o seu desenho num sistema CAD.

Maziero (1998) desenvolveu um sistema CAD baseado em *features* que permite ao usuário modelar peças com as *features*. A partir dessas *features* realiza-se a identificação e análise das tolerâncias na peça.

Álvares (2005) desenvolveu um sistema para o projeto e fabricação de peças rotacionais à distância via Internet, através do qual se propõe uma metodologia que utiliza a modelagem baseada na síntese da peça por *features* de projeto, a fim de permitir as atividades de projeto, planejamento do processo e fabricação utilizando como referência o modelo de *features* de manufatura.

3. TECNICAS UTILIZADAS NO DESENVOLVIMENTO DO AMBIENTE DE SIMULAÇÃO

Este capítulo inicialmente conceitua simulação indicando quando usar e suas vantagens e desvantagens. Após, são detalhados os principais componentes, metodologias de estudo e ferramentas da simulação. Na última seção relacionada à simulação são apresentados exemplos reais de aplicações da simulação em sistemas de manufatura. A segunda parte do capítulo aborda os conceitos de realidade virtual. São detalhados os tipos e ferramentas da realidade virtual. Em seguida, são apresentadas aplicações de realidade virtual na manufatura e os conceitos de ambiente virtual tridimensional. A última parte do capítulo trata dos principais modelos de processo de *software* e suas etapas.

3.1 SIMULAÇÃO

O objetivo deste tópico é apresentar uma revisão bibliográfica de simulação computacional. Inicialmente conceitua-se modelo, sistema e simulação, após, aborda-se o porquê da utilização da simulação, assim como suas vantagens e desvantagens e os principais componentes da simulação.

Em seguida, são apresentados os principais tipos de ferramentas para simulação. A última parte do tópico refere-se à importância da utilização da simulação em sistemas de manufatura, além de exemplos práticos de aplicação de simulação na manufatura.

3.1.1 Conceituação de Modelos e Sistemas

Segundo Shannon (1998), os termos modelo e sistema são palavras-chave para a definição de simulação. A principal utilidade do modelo é resumir o funcionamento do sistema num pequeno número de variáveis que permita sua

compreensão pelo intelecto humano. Para se construir modelos e empregá-los para prever e explicar fenômenos com alto grau de precisão é necessário uma seleção correta das variáveis mais significativas para se descrever o comportamento do sistema.

Conforme Banks *et al.* (2009) e Law e Kelton (2000), existem várias classificações para modelos. As principais são:

- Estático ou Dinâmico: um modelo estático representa um sistema em um ponto particular no tempo ou um sistema no qual o tempo simplesmente não desempenha um papel importante. Um modelo dinâmico representa o sistema à medida que ele evolui ao longo do tempo;
- Discreto ou Contínuo: no modelo discreto as mudanças do sistema ocorrem em intervalos definidos e específicos de tempo. No modelo contínuo o comportamento ininterrupto de mudanças de estado pode ser representado;
- Determinístico ou Estocástico: no modelo determinístico os dados são considerados sem variabilidade estatística associada ao modelo. No modelo estocástico os efeitos da aleatoriedade dos dados são considerados no desenvolvimento do modelo;
- Analítico ou Simulação: modelo analítico é definido como aquele cuja estrutura é formada por uma série de equações matemáticas, através das quais o comportamento do sistema pode ser obtido pela atribuição de valores aos parâmetros do modelo e a solução de equações. Modelo de simulação pode ser definido como aquele representado por uma estrutura matemática/lógica, que pode ser exercitada de forma a mimetizar o comportamento do sistema;

Law e Kelton (2000), expõem a necessidade de se estudar os sistemas para tentar aferir algumas hipóteses sobre os relacionamentos entre vários componentes ou para predizer o desempenho do sistema sob novas condições. Algumas das diferentes maneiras existentes para se estudar um sistema estão representadas na Figura 3.1.

Figura 3.1 - Formas de estudo de um sistema

Fonte: Law e Kelton (2000)

Na experimentação com o sistema real, os efeitos da mudança são analisados no próprio sistema, após a sua implementação. Esta alternativa geralmente apresenta riscos e custos elevados, podendo, ainda, estar sujeita a restrições físicas e temporais. Na experimentação com modelos do sistema, os modelos representam uma simplificação da realidade, através do qual procura-se identificar e destacar elementos da realidade que sejam os mais importantes para a decisão a ser tomada. A experimentação apresenta, quando comparada com a experimentação real, menor custo, maior segurança e rapidez (LAW e KELTON, 2000).

Os modelos físicos (icônicos) normalmente se parecem com o que eles representam, mas diferem no tamanho, ou seja, são réplicas geralmente de dimensões reduzidas. Como exemplo tem-se: maquetes. Os modelos matemáticos (abstratos) usam símbolos em lugar de dispositivos físicos, ou seja, eles representam um sistema em termos de relacionamentos lógicos e quantitativos, os quais são manipulados para se observar como o modelo reage e então como o sistema reagiria (LAW e KELTON, 2000).

Uma vez construído um modelo matemático, ele deve ser examinado a fim de determinar como ele poderia ser utilizado na obtenção de respostas às questões de interesse a respeito do sistema que supostamente representa. Se o modelo é

suficientemente simples, pode ser possível trabalhar com suas relações e quantidades para obter uma solução analítica. Se uma solução analítica para um modelo matemático é viável e computacionalmente eficiente, esta é, frequentemente, a abordagem mais adequada. Entretanto, muitos sistemas são altamente complexos dificultando qualquer possibilidade de soluções deste tipo. Nesse caso, o modelo deve ser estudado através da simulação, isto é, numericamente experimentado para as entradas em questão, com o objetivo de determinar como elas afetam as saídas e as medidas de desempenho do sistema (LAW e KELTON, 2000).

3.1.2 Conceituação de Simulação

Simulação é um termo amplo e, basicamente, pode ser definido como o processo de elaboração de um modelo de um sistema real (ou hipotético) e a condução de experimentos com a finalidade de entender o comportamento de um sistema ou avaliar sua operação (SHANNON, 1998).

Pritsker (1986) definiu simulação computacional ou simulação como o processo de projetar um modelo lógico matemático de um sistema real e fazer experimentos deste sistema no computador.

De acordo com Freitas Filho (2008), “a simulação computacional de sistemas”, ou simplesmente simulação, consiste na utilização de determinadas técnicas matemáticas, empregadas em computadores digitais, as quais permitem imitar o funcionamento de operação ou processo (sistemas) do mundo real.

Para Prado (1999), simulação é a técnica de solução de um problema pela análise de um modelo que descreve o comportamento do sistema usando um computador digital. Complementa Banks *et al.* (2009) dizendo que a simulação é a imitação de uma operação de um processo real ou sistema.

Segundo Harrell e Tumay (1997), simulação é uma atividade por meio da qual se pode tirar conclusões sobre o comportamento de um dado sistema pelo estudo do comportamento do seu modelo correspondente no qual as relações de causas e efeitos são as mesmas ou simulares do sistema real.

Conforme Shannon (1998), simulação consiste no processo de desenvolver um modelo de um sistema real e realizar experimentos com este modelo com o propósito de entender o comportamento do sistema e/ou as evoluções das várias estratégias para a operação do mesmo. Pidd (1998) complementa dizendo que a simulação é utilizada devido a seu baixo custo, maior segurança e rapidez em comparação com a realização de experimentações na realidade.

Pegden *et al.* (1990) apresenta uma definição de simulação utilizando o conceito de modelo e sistema. Ele afirma que simulação é o processo de projetar um modelo conceitual de um sistema real e conduzir experimentos com este modelo com o propósito de entender o comportamento do sistema ou avaliar várias estratégias para sua operação. Assim pode-se pensar em modelagem de simulação como uma método experimental, posta em prática, que procura realizar o seguinte:

- Descrever os comportamentos do sistema;
- Construir teorias ou hipóteses que sejam responsáveis pela observação do comportamento;
- Usar o modelo para prever comportamento futuro, isto é, os efeitos produzidos pelas mudanças no sistema ou em seus métodos de operação.

Na definição de simulação verifica-se que os conceitos de sistema, modelo e de simulação estão intimamente relacionados. Na simulação, o modelo representa o objeto de estudo (sistema real) enquanto que a simulação é um método de análise e projeção (BARRÔNIO, 2000).

3.1.3 Quando Usar Simulação

Segundo Freitas Filho (2008), a simulação permite ao analista realizar estudos sobre os correspondentes sistemas para responder a questões do tipo “O que aconteceria se”. O principal apelo ao uso desta ferramenta é que tais questões podem ser respondidas sem que os sistemas sob investigação sofram qualquer perturbação, uma vez que os estudos são realizados no computador. A simulação permite que tais estudos sejam realizados sobre sistemas que ainda não existem,

levando ao desenvolvimento de projetos eficientes antes que qualquer mudança física tenha sido iniciada.

De acordo com Pidd (1998), a utilização da simulação é recomendável para sistemas ou ambientes com as seguintes características:

- Quando são dinâmicos: o comportamento do sistema varia ao longo do tempo. Esta variação pode estar associada a fatores que não se pode controlar, mas que podem ser determinados via análise estatística;
- Quando são interativos: os sistemas em questão possuem um número de componentes que interagem entre si e esta interação afeta o comportamento de todo o sistema;
- Quando são complicados: quando existem inúmeras variáveis que interagem no sistema e sua dinâmica precisa ser considerada e analisada.

Bertrand e Fransoo (2002) apontam que apesar da qualidade científica dos resultados da simulação ser menor que o caso de análise matemática, sua “relevância científica” é alta. A variedade de modelos científicos na qual a simulação é capaz de trabalhar é maior que uma análise matemática.

Strack (1984) recomenda a utilização da simulação quando:

- Não há uma formulação matemática completa para o problema;
- Não há método analítico para a resolução do modelo matemático;
- Resultados são mais fáceis de serem obtidos por simulação que por qualquer outro método analítico;
- Não existe habilidade técnica para a resolução do modelo matemático por técnica analítica ou numérica;
- Torna-se necessário observar o desenvolvimento do processo desde o início até o seu término;
- Quando são necessários detalhes específicos do sistema;
- Quando a experimentação na situação real apresenta inúmeros obstáculos ou não é possível.

3.1.4 Vantagens e Desvantagens da Simulação

As vantagens da simulação são citadas por Banks (1998), Banks *et al.* (2009), Schriber (1991), Kelton *et al.* (2009), Carson II (2004) e Centeno e Carrillo (2001), entre outros, e são relacionadas abaixo:

- Modelos mais realistas: maior liberdade na construção do modelo. A simulação não obriga a enquadrar um problema em determinado tipo de modelo padrão para que se possa obter uma solução;
- Processo de modelagem evolutivo: começa-se com o modelo relativamente simples e aumenta-se sua complexidade aos poucos, identificando de maneira mais clara as peculiaridades do problema;
- Uma vez criado, um modelo pode ser utilizado inúmeras vezes para avaliar projetos e políticas propostas;
- Os modelos de simulação podem ser quase tão detalhados quanto os sistemas reais; novas políticas e procedimentos operacionais, regras de decisão, fluxos de informação, etc., podem ser avaliados sem que o sistema real seja perturbado;
- Perguntas do tipo “e se?” (“*what if?*”): em diversos casos, em lugar de buscar uma solução, o objetivo resume-se em tornar mais claras as possíveis consequências de um conjunto de decisões;
- Aplicação a problemas mal-estruturados: diversos problemas da vida real referem-se a situações em que se tem apenas um conhecimento parcial sobre as suas variáveis ou relações. A simulação é uma das poucas ferramentas para o estudo deste tipo de problema;
- Grande flexibilidade: a simulação aplica-se aos mais variados problemas;
- Aquisição de visão sistêmica: visão do efeito que alterações locais terão sobre o desempenho global de todo o sistema;
- Exploração de possibilidades: uma vez desenvolvido um modelo de simulação válido, se pode explorar novas políticas sem perturbar o sistema real;
- Desenvolvimento de entendimento: estudos de simulação ajudam no entendimento dos componentes do sistema e de como ele realmente opera;
- Construção de consenso: o resultado de uma simulação, submetido a uma série de etapas de modelagem, teste, validação, tem melhor aceitação que a opinião de uma única pessoa.

Os mesmos autores citam também algumas desvantagens da simulação:

- A modelagem e a análise da simulação podem ser dispendiosas em termos de recursos financeiros e de tempo;
- A construção de modelos requer treinamento especial. Portanto, o aprendizado se dá ao longo do tempo, com aquisição de experiência. Dois modelos de um sistema construídos por dois indivíduos competentes terão similaridades, mas dificilmente serão iguais;
- A programação de um modelo de simulação pode se tornar altamente dispendiosa e desgastante se os recursos computacionais não forem apropriados;
- Os resultados da simulação são, muitas vezes, de difícil interpretação. Uma vez que os modelos tentam capturar a variabilidade do sistema, é comum que existam dificuldades em determinar quando uma observação realizada se deve a alguma relação significativa no sistema ou a processos aleatórios construídos e embutidos no modelo;
- Pode ser usada inapropriadamente, por exemplo, quando uma solução analítica é factível.

3.1.5 Principais Componentes e Resultados da Simulação

De acordo com Freitas Filho (2008), Harrel *et al.* (2003), Law e Kelton (2000), Banks *et al.* (2009), os principais componentes da simulação consistem em:

- Entidade: são os elementos do sistema que estão sendo simulados de forma a analisar seu comportamento. Eles podem ser: matéria-prima, produtos, pessoas, peças ou documentos;
- Atributos: são as propriedades de uma entidade, ou seja, características da entidade que influem de alguma forma no sistema;
- Estado do sistema: é uma coleção de variáveis que contêm toda informação necessária para descrever o sistema em um determinado instante;
- Evento: é uma ocorrência instantânea que altera o estado do sistema;
- Fila: é uma coleção permanente ou temporária de entidades associadas. Só ocorrem antes de atividades que requerem mais de uma entidade para serem processadas;

- Lista: é uma coleção de entidades permanentemente ou temporariamente associadas, “ordenadas” segundo um critério. Como por exemplo, *First Come, First Served* (FCFS) – primeiro a entrar, primeiro a ser processado;
- Lista de eventos: é uma lista de eventos ordenada contendo registros da sequência dos eventos futuros, ou seja, uma lista com evento que ocorrerá no próximo instante de tempo;
- Atividade: são operações ou procedimentos realizados no sistema e que demanda tempo. Sua duração pode ser definida de forma determinística, em termos de distribuição estatística, em função de atributos de variáveis e entidade ou em função do estado do sistema no instante do seu início;
- Tempo de espera: é a quantidade de tempo que uma entidade espera na fila. Ao contrário da atividade, sua duração não é determinada, mas depende das condições apresentadas pelo sistema em seu decorrer;
- Tempo entre chegadas: é a frequência com que as entidades chegam no sistema;
- Tempo de serviço: é o tempo necessário para o servidor executar uma atividade.

A partir da definição dos principais componentes da simulação é possível detalhar o processo de fila. Segundo Gavira (2003), o processo de fila é caracterizado pelos seguintes elementos:

- Especificação da população de clientes: finita ou infinita;
- Distribuição da probabilidade do intervalo de tempo entre chegadas: a distribuição de chegada descreve o padrão pelo qual os elementos chegam para serem servidos;
- A disponibilidade do serviço: alguns sistemas só atendem durante um certo intervalo de tempo, outros estão sempre disponíveis;
- A capacidade do sistema: o número de entidades atendidas simultaneamente; as entidades podem ser atendidas individualmente ou em grupos;
- A duração do tempo de serviço de cada entidade: a distribuição de serviço descreve o padrão pelo qual o servidor provê o serviço. Pode ser constante ou aleatório;

- Número de canais e fases: descreve o número de linhas que existem em um sistema; o número de fase se refere ao número de instalações de serviço através das quais uma entidade pode passar antes de se completar o serviço;
- Disciplina da fila: é o conjunto de regras que determinam a ordem em que os elementos da fila são atendidos. Há várias possibilidades, como atendimento pela ordem de chegada, atendimento aleatório ou datas de entregas prometidas.

De acordo com Freitas Filho (2008), os principais resultados (estatísticas) na simulação podem ser obtidos pelas seguintes equações:

$$TempoMédioEsperaFila = \frac{\sum TempoEsperaFila}{NúmeroTotalClientes} \quad (3.1)$$

$$Pr obabilidadeClienteEsperarFila = \frac{NúmeroClientesQueEsperaram}{NúmeroTotalClientes} \times 100 \quad (3.2)$$

$$Pr obabilidadeOperadorLivre = \frac{\sum TempoLivreOperador}{TempoTotalSimulação} \times 100 \quad (3.3)$$

$$TempoMédioServiço = \frac{\sum TempoServiço}{NúmeroTotalClientes} \quad (3.4)$$

$$TempoMédioDespendidoSistema = \frac{\sum TempoSistema}{NúmeroTotalClientes} \quad (3.5)$$

3.1.6 Metodologia de Estudos de Simulação

Um estudo de simulação baseia-se numa série de etapas segundo um conjunto de critérios que é chamado de roteiro ou metodologia de simulação. Diferentes metodologias têm sido apresentadas por vários autores, dentre eles Freitas Filho (2008), Lobão e Porto (1997), Law e Kelton (2000), Kelton *et al.* (2009)

e Banks *et al.* (2009). Essas metodologias são semelhantes, pois se baseiam no mesmo método científico. A seguir são detalhados os passos básicos dessas metodologias desses autores:

Figura 3.2 - Metodologia de estudo de simulação

Fonte: Adaptado de Freitas Filho (2008), Lobão e Porto (1997), Law e Kelton (2000), Kelton *et al.* (2009) e Banks *et al.* (2009).

1. **Formulação do Problema:** definição do propósito do estudo, assim como estabelecer os objetivos e o escopo do estudo. Devem ser feitas perguntas como: qual será o propósito do estudo e como será utilizado? Quem serão os usuários do modelo?

2. *Planejamento do Estudo*: levantamento da disponibilidade da infraestrutura do pessoal, dos recursos de *hardware* e *software* e do cronograma para a execução do trabalho.
3. *Coleta de Informações e Dados*: consiste na coleta de informações e dados disponíveis que serão processados quando houver necessidade. Essa coleta pode ser realizada no próprio sistema real, em catálogos de fabricantes, dados de sistemas similares e entrevistas com especialistas na área.
4. *Formulação do Modelo Conceitual*: elaboração de um modelo preliminar para a definição dos principais componentes e das interações entre os elementos do sistema: recursos, entidades e movimentações.
5. *Tradução do Modelo*: construção do modelo em forma gráfica ou algorítmica.
6. *Verificação e Validação*: procedimentos para confirmar a operacionalidade do modelo conforme as expectativas e a representatividade com os dados de saída do sistema real.
7. *Projeto Experimental Final*: delineamento de como o modelo verificado e validado irá reproduzir a informação desejada e como as corridas de simulação especificadas no projeto experimental serão executadas.
8. *Experimentação*: execução da simulação para gerar os dados desejados e executar a análise de sensibilidade.
9. *Interpretação e Análise Estatística dos Resultados*: análise e interpretação dos resultados gerados pela simulação, buscando identificar os parâmetros de maior influência nos resultados obtidos.
10. *Documentação, Apresentação dos Resultados e Implementação*: execução e apresentação do estudo no sistema real e registro dos procedimentos realizados ao longo de todo o projeto.

Pidd (1998) propõe cinco princípios básicos e indispensáveis em qualquer modelo:

- O modelo deve ser simples apesar de partir de pensamentos complicados. Os modelos não necessitam ser tão complicados quanto a realidade;
- Começar do simples e acrescentar complexidade na medida do necessário;
- Evitar grandes modelos pela dificuldade em entendê-los. A regra é dividir esses grandes modelos em menores;
- A definição dos dados a serem coletados deve ser orientada pelo modelo;

- O comportamento do modelador na construção do modelo busca organização dos problemas, pois a modelagem não é um processo desordenado.

3.1.7 Ferramentas de Simulação

Para o desenvolvimento do ambiente de simulação necessita-se de um programa computacional. Um programa computacional é uma sequência definida de operações necessárias para que o programador possa descrever um modelo do sistema. Esse programa utiliza-se de um computador para executar o modelo, de forma a gerar saídas que possam ser analisadas, a fim de que decisões possam ser tomadas (KHEIR, 1996).

Segundo Shannon (1998), os programas computacionais voltados para simulação podem ser implementados em diferentes ferramentas que são classificadas em três grandes abordagens, que são:

- Programas codificados em uma linguagem de propósito geral (linguagem de programação);
- Linguagem de simulação;
- Pacote de simulação.

Harrell e Tumay (1997) mostram na Figura 3.3 a redução do tempo de programação e execução requeridos à medida que as técnicas de simulação evoluíram. Essas ferramentas continuam evoluindo e tornando-se mais flexíveis e fáceis de usar, com melhores recursos visuais e interação com o usuário.

Em 1950, as ferramentas de simulação era desenvolvidas somente com a utilização de linguagens de programação (exemplo, Fortran e Pascal). Nos anos 60 foram incluídos as linguagens gerais de simulação (exemplo, GPSS). Nos anos 80 foram desenvolvidos pacotes ou programas de simulação (exemplo, Arena) (HARRELL e TUMAY, 1997). Atualmente, além de todas ferramentas utilizadas anteriormente também são desenvolvidos sistemas de simulação personalizados que permitem a simulação de situações específicas de acordo com a necessidade dos *stakeholders*.

Figura 3.3 - A evolução das ferramentas de simulação

Fonte: Harrell e Tumay (1997)

3.1.7.1 Linguagem de Propósito Geral

Na linguagem de propósito geral (linguagem de programação) um programa especial é escrito para a simulação de cada sistema a ser estudado. Essa ferramenta de simulação oferece ao programador a máxima flexibilidade em projeto e formulação do modelo matemático do sistema em estudo, tipo e formato de dados de saída gerados e espécies de experiências de simulação executadas com o modelo (NAYLOR *et al.*, 1971).

Segundo Law e Kelton (2000), algumas razões para o uso das linguagens de programação são:

- Flexibilidade oferecida na descrição matemática do sistema modelo;
- A maioria dos modeladores já conhecem as linguagens de programação;
- O modelo de simulação escrito eficientemente em uma linguagem de programação requer menos tempo de execução do que as outras duas abordagens;
- Os *softwares* de programação em geral custam menos, mas o projeto total pode custar mais do que com outras abordagens;
- Flexibilidade em termos de tipos de experimentos que podem ser desempenhados no sistema modelado.

Por outro lado, Naylor *et al.* (1971) afirmam que a principal deficiência da modelagem em linguagem de propósito geral é a dificuldade de confecção dos programas. O programador pode se ver com dificuldades no controle sequencial, o que resulta no surgimento de pequenos erros. Os equívocos nesta fase são responsáveis pelo surgimento de efeitos obscuros e difíceis de eliminar.

De acordo com Prado (1999), as principais linguagens de programação de uso geral são *Pascal*, *Fortran*, *Visual Basic*, *C*, *C++* e *Java*.

Para Bateman *et al.* (1997), as linguagens de propósito geral permitem a modelagem em várias e diferentes aplicações, o tempo e as habilidades de programação necessários desencorajam muito modeladores. O uso dessas linguagens está restrito a profissionais que tenham bons conhecimentos em programação.

3.1.7.2 Linguagem de Simulação

De acordo com Law e Kelton (2000), linguagens de simulação são pacotes computacionais de natureza genérica, porém com características especiais para determinados tipos de aplicações em simulação.

Assim como nas linguagens de programação, as linguagens de simulação também fazem uso de compiladores, mas são usadas especificamente para aplicações em simulação. Essas linguagens são bibliotecas compostas por macros de linguagens de programação.

Segundo Prado (1999), dentre as principais linguagens de simulação se destacam o Simula®, o Slam®, o Siman®, o GPSS®, o GASP® e o Dynamo®.

A vantagem das linguagens de simulação é a grande flexibilidade que permite modelar qualquer tipo de sistema. As desvantagens estão associadas à necessidade de conhecimentos específicos de programação e ao tempo demandado para a construção de modelos mais complexos (LAW e KELTON, 2000).

3.1.7.3 Programas de Simulação

Segundo Law e Kelton (2000), os programas de simulação ou simuladores são pacotes computacionais que favorecem a modelagem de sistemas específicos com pouca programação. As vantagens dos simuladores estão associadas ao tempo reduzido para construção do modelo e a facilidade de utilização com menus gráficos relacionados com o sistema específico. A desvantagem principal é a pouca flexibilidade de modelagem.

De acordo com Shannon (1998), os pacotes de simulação buscam diminuir o espaço entre a conceituação que o usuário tem do modelo e sua forma executável. Os pacotes de simulação são divididos em duas categorias: uma de propósito geral e outro de propósito específico. Na primeira categoria estão os pacotes que podem resolver problemas em geral de simulação, como ARENA®, AweSim®, GPSS/H™, Simscript II.5®, Extend™.

Na segunda categoria estão os pacotes utilizados na simulação de problemas específicos, tais como sistemas de manufatura e manuseio de material. Os pacotes SimFactory, ProModel®, AutoMod™, Taylor II® e Witness® se enquadram nesta categoria, assim como os pacotes para estudos de reengenharia de processos, como BPSimulator™, processModel™, SIMPROCESS® e Extend+BPR (SHANNON, 1998).

As principais características comuns dos pacotes de simulação são:

- Ferramentas estatísticas para tratamento dos dados da simulação;
- Animação do modelo e interação amigável entre usuário e simulação;
- Construções para aplicações específicas que proporcionam rapidez na modelagem;
- Fornecimento de relatórios em tempo real sobre a simulação executada.

3.1.8 Simulação de Sistemas de Manufatura

O uso de simulação em sistemas de manufatura tem aumentado significativamente nos últimos anos. Existem várias razões para este crescimento,

dentre elas, a busca para aumentar a qualidade e produtividade, reduzindo, conseqüentemente, os custos. Outro fator importante é a popularização da utilização da simulação nas empresas, assim como o aumento da capacidade de recursos (*hardware* e *software*) para realização da simulação.

De acordo com O'kane *et al.* (2000), a simulação tem se tornado uma das técnicas mais populares para se analisar problemas complexos em ambientes da manufatura. Os autores complementam dizendo que os sistemas de manufatura modernos consistem em muitas operações que ocorrem aleatória e não linearmente, dessa forma, a modelagem por simulação torna-se um dos mais populares e eficientes meios de análise desses sistemas.

Através da simulação pode-se prever o comportamento de sistemas de manufatura ou o efeito da mudança de um parâmetro em seu funcionamento. Para Harrell *et al.* (2003), a simulação fornece um meio para validar se as melhores decisões estão sendo tomadas e evita técnicas de tentativa e erro.

Para Pegden *et al.* (1990), a simulação é utilizada na manufatura de três formas distintas:

- Como ferramenta de projeto e análise de layouts, de compra de máquinas, de avaliação de políticas alternativas;
- Como ferramenta de planejamento, pois o seu uso permite ao tomador de decisão explorar, planejar e avaliar mudanças no planejamento a fim de obter a otimização do mesmo;
- Como parte de sistemas em tempo real com ou sem a intervenção humana.

Segundo Law e McComas (1999), a manufatura é uma das maiores aplicações da simulação, tendo como benefícios destacar a necessidade e quantidade de maquinário e/ou funcionários extras, avaliação de desempenho e avaliação dos procedimentos operacionais.

De acordo com Harrell *et al.* (2003), algumas contribuições que a simulação traz para os sistemas de manufatura são: análise de métodos, implantação de *layout*, análise do impacto na variação do tamanho de lote para fabricação, controle da produção, controle sobre o estoque em processo, planejamento da cadeia de suprimentos, programação da produção, controle da produção em tempo real, avaliação do impacto da introdução de um novo produto, previsão de custos, avaliação da eficácia dos planos de contingência e análises de novas tecnologias.

Porto e Palma (2000) listaram algumas das vantagens da aplicação da simulação na manufatura:

- Novas políticas, procedimentos operacionais, regras de projetos, fluxo de informação, procedimentos organizacionais podem ser explorados sem perturbar a continuidade das operações do sistema real;
- Hipóteses de “como” e “por que” certos fenômenos ocorrem podem ser testadas para verificar a viabilidade;
- Tempo pode ser comprimido ou expandido permitindo o aumento ou a redução da velocidade do fenômeno da investigação;
- Estudo da simulação pode ajudar o entendimento de como o sistema opera em vez de como os indivíduos pensam que o sistema opera.

Abdul-Kader e Gharbi (2002) utilizaram a simulação para estimar a capacidade de linhas produtivas que produzem diversos produtos diferentes. A simulação permitiu a representação no modelo de fatores como tempo de *setup*, tempo de reparo de falhas, tempo entre falhas.

Torga *et al.* (2006) exploraram em seu trabalho a simulação de uma linha de produção puxada através de um estudo de caso realizado em uma empresa automobilística. Os autores sugeriram a associação dos fundamentos da simulação e otimização como forma de minimizar o número de *kanbans*, com objetivo de reduzir o estoque em processo e o inventário de material acabado.

Oliveira *et al.* (2009) desenvolveram em seu trabalho um modelo de simulação de eventos discretos para identificar os pontos de melhoria do fluxo da manufatura incluindo ao seu *layout* em um centro de usinagem. Foi realizada uma análise detalhada do *lead time* e os custos de produção que diminuíram após inclusão do centro de usinagem.

Braghirolli (2009) desenvolveu um estudo comparativo entre a aderência do comportamento linear da mão-de-obra para as células de manufatura previsto na literatura e o comportamento observado nas implementações em empresas do setor metal-mecânico, através do uso da simulação computacional.

Pinho (2008) desenvolveu um método para a otimização de modelos de simulação aplicados a sistemas de manufatura, capaz de atingir os resultados de forma mais rápida e eficiente, quando comparado a uma ferramenta de otimização comercial.

3.2. REALIDADE VIRTUAL

O objetivo deste tópico é apresentar uma revisão bibliográfica sobre realidade virtual (RV). Inicialmente conceitua-se RV, em seguida, aborda-se os tipos de realidade virtual, as ferramentas da realidade virtual e a utilização da realidade virtual na manufatura. Por último, são apresentados conceitos relativos a ambientes virtuais tridimensionais.

3.2.1 Definição de Realidade Virtual

O termo Realidade Virtual (RV) foi proposto por *Jaron Lanier*, que nos anos 80 sentiu a necessidade de um termo para diferenciar as simulações tradicionais por computação dos mundos digitais que ele tentava criar (MACHOVER e TICE, 1994).

Segundo Banerjee e Zetu (2001), realidade virtual é a habilidade para criar e interagir no espaço cibernético, ou seja, um espaço representando um ambiente que possui uma grande similaridade com o ambiente ao nosso redor.

De acordo com Shukla *et al.* (1996), a realidade virtual é uma tecnologia que tem a finalidade de proporcionar ao usuário uma percepção do ambiente real e permite experiências interativas, facilitando a avaliação de diferentes cenários. Além da interação pode ser oferecida ao usuário a imersão, ou seja, o usuário tem a sensação de estar dentro do modelo e conseqüentemente um maior realismo e permitindo uma exploração mais rica do mesmo.

Para Cruz-Neira (1992), a RV possibilita a simulação de mundos reais e imaginários na tela do computador ou em outros dispositivos criando no usuário a sensação de presença em um mundo virtual.

Pimentel e Teixeira (1995) definem a realidade virtual como o uso de uma tecnologia para convencer o usuário de que ele está em outra realidade - um novo meio de “estar” e “tocar” em informações.

Para Latta e Oberg (1994), realidade virtual é uma avançada interface homem-máquina que simula um ambiente realístico e permite que participantes interajam com ele.

A realidade virtual pode ser definida como o uso de computadores e interfaces homem-máquina para criar o efeito de mundos tridimensionais que possuam objetos interativos com uma forte sensação de presença tridimensional (BRYSON, 1996).

3.2.2 Tipos de Sistemas de Realidade Virtual

Conforme Pimentel e Teixeira (1995), os sistemas de RV podem ser monoscópicos ou estereoscópicos, ou seja, cada um dos olhos pode visualizar ou não imagens diferentes. No caso de um sistema monoscópico, a mesma imagem é exibida para os dois olhos: apenas uma imagem passa pelo processo de renderização e é exibida para os dois olhos. Já no sistema estereoscópico, cada olho observa uma imagem ligeiramente diferente, sendo necessário um processo de renderização separado para cada uma das imagens.

De acordo com Jacobson (1994), os sistemas de RV diferem entre si levando em conta o nível de imersão, envolvimento e de interatividade proporcionado ao participante. São determinados de acordo com os tipos de dispositivos de entrada e saída de dados usados no sistema de RV além da velocidade e potência do computador que suporta o sistema de RV.

Segundo Cruz-Neira (1992), a imersão está ligada ao sentimento de estar dentro do ambiente. Esse sentimento de imersão é obtido com o uso de dispositivos como capacete de visualização (HMD) e cavernas digitais. Além do fator visual, os dispositivos ligados a outros sentidos também são importantes para a imersão, como por exemplo, o som e dispositivos de rastreamento.

Com relação ao senso de imersão, a RV é classificada em imersiva e não-imersiva (TORI e KIRNER, 2006).

- Imersiva: quando o usuário é transportado para o domínio da aplicação, através de dispositivos multisensoriais, que capturam seus movimentos e comportamentos e reagem a eles (por exemplo, HMD e Cavernas Digitais), provocando uma sensação de presença dentro do mundo virtual;

- Não-Imersiva: ocorre quando o usuário é transportado parcialmente ao mundo virtual, através de uma janela (por exemplo, monitor ou projeção) mas continuando a sentir-se no mundo real.

De acordo com Fraser *et al.* (2000), embora a RV imersiva tenha evoluído, a utilização da RV não imersiva com o monitor, apresenta pontos positivos, tais como: menor custo em relação a modelos imersivos; apresenta facilidade de uso e alta disponibilidade de equipamentos.

A idéia de envolvimento está ligada ao grau de motivação para o engajamento de uma pessoa em 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 RV 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 (BOTEGA e CRUVINEL, 2009).

Para Tori e Kirner (2006), a idéia de interação está ligada à capacidade do computador detectar entradas do usuário e modificar instantaneamente o mundo virtual e as ações sobre ele. Complementa Bowman *et al.* (2005) dizendo que a interação abrange a navegação, seleção, manipulação e controle do sistema.

A interação em ambientes tridimensionais não-imersivos pode ser realizada de duas formas: direta ou indiretamente. A manipulação indireta de objetos funciona com objetos gráficos de interface conhecidos (*Sliders*, *Dials* e *Widgets*). Fazendo uma analogia, a manipulação indireta funciona como se fosse uma espécie de controle remoto à distância. Cada ação do usuário no objeto de interface intermediário é refletida no objeto que se deseja manipular (JACOB, 2002).

A manipulação direta é realizada através de apontamento sobre a cena 3D. O usuário pode interagir com o mouse clicando sobre a cena representada com vista ortográfica ou em perspectiva, para realizar operações de translação, escala e rotação (JACOB, 2002).

Um sistema de RV também pode proporcionar três formas diferentes de envolvimento: Passivo, Exploratório ou Interativo (ADAMS, 1995).

- Passivo: proporciona ao usuário uma exploração do ambiente automática e sem interferência. A rota e os pontos de observação são controlados exclusivamente pelo programa;

- Exploratório: proporciona uma 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;
- Interativo: proporciona uma exploração do ambiente dirigida pelo usuário e as entidades virtuais do ambiente respondem e reagem às ações do usuário.

De acordo com Kirner (2005), o envolvimento passivo é como ler um livro ou assistir televisão e o exploratório (ativo) como participar de um jogo com algum parceiro.

Segundo Mine (1995), os sistemas de RV interativos podem ser definidos segundo três categorias, de acordo com o tipo de controle exercido pelo usuário:

- Interação direta: esta categoria inclui as técnicas que utilizam o corpo do usuário (mãos, braços ou cabeça) atuando diretamente sobre o objeto através de um “toque virtual”. Para isso é necessário que o sistema possua funções de suporte ao rastreamento das mãos e da direção do olhar, reconhecimento de gestos e detecção do apontamento de um objeto;
- Interação com controles físicos: esta categoria inclui o uso de botões, *joysticks*, *mouse*, pedais, dentre outros. Esses controles físicos podem aumentar a sensação de presença do usuário no mundo virtual, pois lhe permitem algum tipo de sensação tátil;
- Controles virtuais: nessa categoria é necessário representar visualmente um dispositivo físico. Qualquer objetivo que se imagine pode ser implementado como controle virtual. Essa categoria tem a maior flexibilidade, porém como desvantagem tem a falta de retorno sensorial e a dificuldade de interação com o objeto virtual.

3.2.3 Ferramentas da Realidade Virtual

Para o desenvolvimento de ambientes de realidade virtual necessita-se de ferramentas de *hardwares* e *softwares*. Os *hardwares* são basicamente os dispositivos de entrada e de saída de dados, dispositivos de interação e os *softwares* são programas específicos de modelagem em 3D de objetos e da cena.

3.2.3.1 Dispositivos de Entrada de Dados

De acordo com Pimentel e Teixeira (1995), os dispositivos de entrada de dados permitem a movimentação e interação do usuário com o mundo virtual. Sem o dispositivo de entrada de dados o usuário participa da experiência em RV apenas de forma passiva.

“Dispositivos de entrada de dados para sistemas de RV são utilizados para enviar informações sobre ações do usuário para o sistema. Eles são divididos em dois tipos: de interação e de rastreamento” (MACHADO e CARDOSO, 2006, p.40).

Os dispositivos de interação permitem ao usuário a movimentação e manipulação de objetos no mundo virtual. Os dispositivos de rastreamento monitoram partes do corpo do usuário, detectando os movimentos, para criar a sensação de presença no mundo virtual (PIMENTEL e TEIXEIRA, 1995).

De acordo com Machado e Cardoso (2006), os sistemas de rastreamento utilizam sensores sobre partes do corpo ou objetos, que emitem sinais a uma caixa controladora que processa estes sinais e os envia ao computador. Estes sistemas são classificados de acordo com a tecnologia que empregam. As mais comuns são técnicas mecânicas, eletromagnéticas, acústicas, óticas e de processamento de imagens.

O rastreamento em ambientes de realidade virtual tem a função de identificar a posição da mão, da cabeça, do próprio usuário ou de algo atrelado a ele, como uma placa. Com isto, o sistema permite que o usuário exerça um controle de posicionamento em ambientes virtuais, podendo, por exemplo, movimentar-se e tocar, agarrar, mover e soltar objetos virtuais (KIRNER e SISCOOTTO, 2007).

Os dispositivos convencionais de entrada são periféricos de usabilidade comum, destinados a usuários comuns e sistemas de propósito geral. São eles: teclado, *mouse*, *trackballs*, digitalizadores de mesa (*scanners*), canetas digitais e microfones (BOTEGA e CRUVINEL, 2009).

Os dispositivos não-convencionais de entrada de dados podem contribuir para o desenvolvimento de ambientes interativos e semi-imersivos, porém, não provêm o realismo e o envolvimento fornecidos por sistemas virtuais dotados de dispositivos não-convencionais, como, por exemplo, luvas digitais (BOTEGA e CRUVINEL, 2009).

3.2.3.2 Dispositivos de Saída de Dados

Os dispositivos de saída de dados são responsáveis pelo envio de informações ao usuário. Uma vez que os sistemas de RV buscam a exploração dos cinco sentidos, os dispositivos de saída de dados estimulam estes sentidos através de dispositivos específicos (MACHADO e CARDOSO, 2006).

Segundo Pimentel e Teixeira (1995), uma grande porção do cérebro é dedicada ao processamento e organização dos estímulos visuais. Devido a isto, os dispositivos visuais e o tipo de imagem gerada por um sistema de RV influenciam na determinação do nível de imersão de um sistema de RV.

Segundo Botega e Cruvinel (2009), os dispositivos convencionais de saída de dados são periféricos de comum utilização, destinados à visualização e análise de sistemas de propósito geral. São eles: monitor de vídeo, impressoras, auto-falantes, etc.

Envolvendo sistemas de RV, os dispositivos convencionais de saída de dados podem contribuir para o desenvolvimento de ambientes interativos e semi-imersivos, porém, não são capazes de prover realismo e o envolvimento fornecidos por sistemas virtuais dotados de dispositivos não-convencionais, como por exemplo, capacetes (HMD) e cavernas digitais (CAVE) (BOTEGA e CRUVINEL, 2009).

3.2.3.3 Programas para a Realidade Virtual

De acordo com Tori e Kirner (2006), os programas de realidade virtual são complexos e envolvem interações em tempo real entre muitos componentes de *hardware* e *software*. O *software* de realidade virtual atua na fase de preparação do sistema, no desenvolvimento de ambientes 3D e na fase de execução.

Segundo Tori e Kirner (2006) e Camargo (2008), os *softwares* para o desenvolvimento de sistemas baseados em RV podem envolver:

- Programas de Modelagem 3D como *Blender*¹, *3Ds Studio*², *EonStudio*:³ para criação de um mundo que envolve a modelagem 3D da aparência e comportamento dos objetos que o compõem;
- Linguagens para Modelagem de Realidade Virtual, como VRML, X3D: os modelos 3D podem ser gerados em um código para modelagem em ambientes de realidade virtual, através do padrão VRML (*Virtual Reality Modeling Language*) ou X3D que são linguagens de marcação escrita em modo texto (CAREY e BELL, 1997);
- Interface de Programação de Aplicações (API) como *OpenGL*, *Directx* e *Java3d*: são bibliotecas gráficas que contém rotinas e padrões estabelecidos para a utilização de suas funcionalidades (TORI e KIRNER, 2006);
- *Toolkits* implementados sob a forma de bibliotecas C/C++, como *WorldToolKit* que é uma arquitetura para aplicações de RV com suporte a grafo de cena (JACOBSON, 1994);
- Motores Gráficos (*Engines*), como OGRE 3D (JUNKER, 2006).

De acordo com Soares *et al.* (2006), as principais bibliotecas para desenvolvimento de aplicações em realidade virtual são:

- *CAVELib*: biblioteca gráfica de alto desempenho, que permite ao usuário desenvolver aplicativos para equipamentos de RV com amplo grau de abstração. Desenvolvido originalmente na Universidade de Illinois no *Electronic Visualization Laboratory* (EVL), é atualmente comercializado pela VRCO (TORI e KIRNER, 2006);
- *OpenSG*: é um sistema para geração de imagens em tempo real, baseado no princípio de gráfico de cena. Implementado em C++, tem suporte em vários sistemas operacionais. O grupo desenvolvedor é formado por várias empresas e universidades interessadas que auxiliam seu desenvolvimento (REINERS *et al.*, 2002);
- *OpenInventor*: é destinado a aplicações 3D pouco complexas. Implementado em C++, tem suporte em vários sistemas operacionais (TORI e KIRNER, 2006);

¹ Página web do Blender: <http://www.blender.org/>. Acessado em Agosto/2011

² Página web do 3Ds Studio: <http://www.autodesk.com.br/>. Acessado em Agosto/2011

³ Página web do EonStudio: <http://www.eonreality.com/>. Acessado em Agosto/2011

- *Open Scene Graph*: conjunto de ferramentas portáteis, voltadas ao desenvolvimento de aplicações de RV. Tem sua estrutura orientada a objetos baseada em *OpenGL* (MARTZ, 1997);
- *AVANGO*⁴: é um sistema de desenvolvimento orientado a objetos, para aplicações de RV distribuídas;
- *VrJuggler*: é uma biblioteca de desenvolvimento para aplicações de RV. Seus princípios de funcionamento são bem simples, uma função gráfica é especificada para o módulo principal e este fica responsável por todas as outras necessidades para ambientes virtuais imersivos (TORI e KIRNER, 2006);
- *Diverse*⁵: possui diversos pacotes de ferramentas para desenvolvimento de aplicações de RV, além de possuir um sistema próprio de leitura e visualização de arquivos;
- *Syzygy*: biblioteca desenvolvida na Universidade de *Illinois*, ela permite o uso de aglomerados gráficos no desenvolvimento de sistemas de realidade virtual, auxiliando também no tratamento da interação com o usuário. O desenvolvimento de aplicações não é trivial (SCHAEFFER e GOUDESEUNE, 2003);
- *JINX*: desenvolvido no Laboratório de Sistemas Integráveis da Universidade de São Paulo, foi desenvolvido na plataforma Linux. Permite o desenvolvimento de aplicações em aglomerados de computadores para basicamente qualquer tipo de projeção. Utiliza os protocolos XML, tanto para o gerenciamento como acesso a dados, por intermédio do formato X3D (SOARES e ZUFFO, 2004).
- *OpenCSG*: é uma biblioteca compacta que utiliza como base imagens CSG (*Constructive Solid Geometry*). CSG é uma técnica de modelagem 3D que consiste em combinar geometrias tridimensionais simples com complexas através de operações booleanas no espaço tridimensional. A *OpenCSG* foi escrita na linguagem C++, é multiplataforma e utiliza a *Opengl*. Com ela é possível desenvolver modelos complexos 3D usando formas primitivas, ou seja, duas formas primitivas são combinadas pela união, interseção ou

⁴ Página web AVANGO: <http://www.avango.org/>. Acessado em Agosto/2011.

⁵ Página web Diverse: <http://www.sv.vt.edu/> Acessado em Agosto/2011.

subtração formando uma forma final complexa. A OpenCSG utiliza algoritmos específicos para renderização de formas CSG sem a necessidade de um cálculo da fronteira geométrica das formas. Os principais algoritmos utilizados são o *Goldfather* e o SCS (KIRSCH e DOLLNER, 2005).

3.2.4 Utilização da Realidade Virtual na Manufatura

De acordo com Tori e Kirner (2006), a RV é utilizada na manufatura para: visualização de protótipos; treinamento; avaliação de fatores ergonômicos; simulação de montagens; simulação da dinâmica de estruturas articuladas; simulação de processos produtivos; estudo de técnicas de engenharia; planejamento e túnel de vento virtual.

Segundo Leston (1996), o setor industrial utiliza a realidade virtual nas seguintes situações: na venda e no marketing; na automação de projetos; na concepção de produtos; no planejamento e na manutenção, no treinamento de funcionários; nos testes e nas visualizações de dados e em simulações de utilização de máquinas e mão-de-obra.

Para Kelsick *et al.* (2003), a RV pode auxiliar com relação a sistemas de manufatura em quatro áreas: arranjo físico da fábrica, prototipagem de métodos de montagem, simulação de fluxo de peça e simulação de processo. Em seu trabalho, os autores desenvolveram um simulador para uma célula de manufatura baseado em realidade virtual.

Netto (1998) utilizou a realidade virtual para realizar a prototipação de um torno CNC com o foco principalmente no seu sistema de intertravamento (funcionalidade) e no seu modelo geométrico (*design* físico).

O Núcleo de Informática Educacional (NIEd) do Serviço Nacional de Aprendizagem Industrial (SENAI-RS) desenvolveu o sistema ASIMOV que é um simulador de célula robotizada em um ambiente não-imersivo. O sistema foi desenvolvido com o intuito de facilitar o aprendizado dos alunos que pretendem trabalhar com manipulação de robôs no ambiente de produção (ASIMOV, 1999).

Espíndola (2007) propôs uma metodologia para a visualização imersiva distribuída, a ser utilizada em ambientes de multiprojecao para a representação de cenários associados ao processo de construção naval/*offshore*.

Dos Santos (2006) em seu trabalho construiu um simulador do robô. Neste simulador o usuário controla um robô virtual dentro de uma célula de manufatura utilizando recursos de realidade virtual, como estereoscopia e rastreamento de posição.

3.2.5 Ambientes Virtuais Tridimensionais

Os Ambientes Virtuais Tridimensionais são ferramentas para o desenvolvimento de interfaces mais realistas e interessantes para o usuário, a partir da exploração das técnicas de Realidade Virtual (RV).

De acordo com Sementille (1999), Ambiente Virtual (AV) é um sistema que cria a ilusão de um mundo que não existe na realidade. Isto requer a combinação de entrada (interação do usuário), computação (simulação de processos) e saída (estímulos multisensoriais).

Os ambientes virtuais são projetados através de ferramentas especiais, criadas a partir de técnicas de computação gráfica. Uma vez que é possível interagir e explorar esse mundo por meio de dispositivos de entrada e de saída, ele se transforma em um ambiente virtual (NETTO, 1998).

Segundo Tori e Kirner (2006, p.10), “os AVs devem conter objetos virtuais que terão certos atributos a eles associados como: a geometria, cores, texturas, iluminação, características dinâmicas, restrições físicas e atributos acústicos”.

A geometria de um objeto está relacionada com sua representação, ou seja, quantos vértices, faces, lados e tamanho que um determinado objeto possui. De acordo com Foley (1990), modelagem geométrica consiste em um conjunto de métodos que visam descrever a forma e as características geométricas de um objeto.

A descrição geométrica do objeto pode ser classificada como estática ou dinâmica no tempo. A descrição geométrica estática é aquela cuja forma não varia no tempo, enquanto a dinâmica é aquela que pode ter sua forma modificada por

uma função relacionada ao tempo, bem como por ação de qualquer outro atributo da simulação, como um choque, o efeito de uma força, entre outros (FERREIRA, 1999).

De acordo com Ferreira (1999), o grau de complexidade da descrição geométrica é, em geral, diretamente proporcional à qualidade visual, porém inversamente proporcional à velocidade com que a imagem é gerada. Essa premissa nem sempre é válida se for levada em consideração a excelente qualidade visual obtida quando se faz uso de texturas (atributo de aparência da entidade), mesmo sobre malhas pouco refinadas.

De acordo com Machado e Cardoso (2006), os objetos dos ambientes virtuais geralmente podem mover-se com seis graus de liberdade (6DOF – *degrees of freedom*), o que implica na possibilidade de três rotações e três translações de acordo com a Figura 3.4.

Figura 3.4 - Seis graus de liberdade dos elementos de ambiente virtual

Fonte: Machado e Cardoso (2006)

Num ambiente virtual tridimensional uma cena é visualizada com o auxílio de uma câmera virtual. Tal câmera é posicionada e apontada para a parte da cena que se deseja visualizar. Com os objetos modelados na cena, a iluminação permite a visualização da cena.

A câmera num ambiente virtual desempenha um papel fundamental. De acordo com Hearn e Baker (1997), ao criar cenas tridimensionais, pode-se dizer que se está usando uma máquina fotográfica, onde o observador fica em um ponto de observação, vê através das lentes a cena, se posiciona de forma a obter a imagem e

define a posição da câmera, sua orientação e foco e a posição dos planos que formam a cena.

Algumas características possíveis a partir da iluminação são: maior realismo nas imagens e o destaque de partes dos objetos com sombreamento e realces. O ambiente virtual oferece aos seus usuários uma experiência virtual tanto em ambientes que representam espaços construídos, como em ambientes que só existem no meio virtual.

De acordo com Tori e Kirner (2006), a iluminação do ambiente virtual utiliza normalmente técnicas simplificadas, como uso de luz ambiente e luzes pontuais que garantem um bom grau de realismo, permitindo uma execução em tempo real.

3.3. MODELOS DE PROCESSO DE SOFTWARE

O objetivo deste tópico é apresentar uma revisão bibliográfica sobre os principais modelos de processo de desenvolvimento de *software* utilizados atualmente e suas etapas.

3.3.1 Principais Modelos de Processo de *Software*

De acordo com Pressman (2006), os modelos de processo foram propostos para colocar ordem no caos do desenvolvimento de *software*. A história tem indicado que esses modelos têm trazido uma estrutura útil e um roteiro efetivo para o desenvolvimento de *software*.

Conforme Sommerville (2011, p.43), “um modelo de processo de *software* é uma representação abstrata de um processo de *software*. Cada modelo de processo representa um processo sob determinada perspectiva”. Complementa Gruhn (2002), dizendo que um processo de *software* é um conjunto de atividades conduzidas no contexto de projeto, tais como, análise de requisitos, projeto, codificação, teste, instalação, etc, os recursos (*hardware*, *software* e pessoas) necessários e os procedimentos a serem adotados na realização de cada uma das atividades.

Pressman (2006), classifica os modelos da estrutura dos processos em etapas denominadas ciclos de vidas. Os modelos de ciclo de vida do *software* podem ocorrer em sequência ou sobrepostos e executados interativamente e podem ser divididos em etapas, atividades ou tarefas, a fim de ter um maior controle da execução do projeto.

Os modelos de processo de *software* mais citados na literatura são:

- Modelo em Cascata: modelo sistemático e seqüencial para o desenvolvimento de *softwares* que começa com a especificação dos requisitos pelo cliente e progride ao longo do planejamento, modelagem, construção e implantação, culminando na manutenção progressiva do *software* acabado (PRESSMAN, 2006);
- Modelo de Desenvolvimento Evolucionário: este modelo intercala as atividades de especificação, desenvolvimento e validação. Um sistema inicial é desenvolvido baseado em especificações. Este sistema é, então, refinado com as entradas do cliente para produzir um sistema que satisfaça as necessidades do cliente (SOMMERVILE, 2011);
- Modelo em Espiral: este modelo prevê a prototipação, o desenvolvimento evolutivo e cíclico combinados com os aspectos sistemáticos do modelo em cascata. É dividido em quatro estágios: estágio 1 são determinados objetivos, soluções alternativas e restrições; estágio 2 são analisados os riscos das decisões do estágio anterior; estágio 3 consiste no desenvolvimento (especificação, codificação e verificação); estágio 4 compreende a revisão das etapas anteriores e planejamento da próxima fase (BOEHM, 1988);
- Modelo Orientado ao Reuso: este modelo tem como base a existência de um número significativo de componentes reusáveis. O processo de desenvolvimento de sistemas se concentra na integração desses componentes, em vez de proceder ao desenvolvimento a partir do zero (SOMMERVILE, 2011);
- Modelo RAD (*Rapid Application Development* – Desenvolvimento Rápido de Aplicação): este modelo enfatiza um ciclo de desenvolvimento curto. É uma adaptação de alta velocidade do modelo em cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes (PRESSMAN, 2006);

- Modelo RUP (*Ration Unified Process* – Processo Unificado da *Rational*): é um modelo iterativo e incremental centrado na arquitetura dirigida por casos de uso, ou seja, por levantamento de requisitos baseados na visão do usuário (KRUTCHEN e KROLL, 2003);
- Modelos Ágeis: são modelos que visam reduzir o tempo de mudanças de requisitos em relação aos modelos tradicionais. Os principais modelos utilizados atualmente são o modelo *Extreme Programming* (XP) e o modelo *Scrum* (RISING e JANOFF, 2000).

Realizando uma combinação das classificações dadas para as atividades/fases que compõem os principais modelos de processo de *software* de acordo com Pfleeger e Atlee (2006), Pressman (2006) e Sommerville (2011), têm-se:

- Especificação: estabelecimento de uma solução geral para o problema, envolvendo questões extra *software*. Realiza análise de requisitos a partir do levantamento das necessidades do *software* a ser implementado. É realizada a especificação do sistema com a sua descrição funcional;
- Projeto: desenvolve um modelo conceitual para o sistema, composto de módulos independentes ou não. São projetadas as interfaces onde cada módulo tem sua interface de comunicação estudada e definida e os módulos em si são definidos, e possivelmente traduzidos para pseudo-código;
- Implementação: é realizado a codificação que é a implementação em si do sistema em uma linguagem de computador;
- Validação: são realizados testes para verificar a presença de erros e comportamento adequado a nível das funções e módulos básicos do sistema. São verificadas por meio de teste a integração dos diferentes módulos em um produto de *software* homogêneo, e a verificação da interação entre estes quando operando em conjunto;
- Manutenção e Evolução: nesta fase, o *software* em geral entra em um ciclo iterativo que abrange todas as fases anteriores.

Nas próximas seções são detalhadas as principais atividades que compõem os modelos de processo de desenvolvimento de *software*.

3.3.2 Especificação

A especificação de *software* ou engenharia de requisitos é o processo para compreender e definir quais serviços são necessários e identificar as restrições de operação e desenvolvimento do sistema. A análise de requisitos é uma fase crítica do processo de *software*, pois os erros nesse estágio conduzem inevitavelmente a problemas posteriores no projeto e na implementação do sistema (SOMMERVILLE, 2011).

De acordo com Kotonya e Sommerville (1998), os requisitos são descrições sobre o comportamento do sistema de *software*, informações sobre o domínio da aplicação, restrições de operação ou especificação de suas propriedades ou atributos. Os requisitos podem também representar restrições no processo de desenvolvimentos.

Segundo Kosciński e Soares (2007), os requisitos definem o que o *software* deve fazer. Mas, além disso, é preciso que fique claro como as operações são realizadas. Existem dois pontos de vista a esse respeito: o do cliente, que detalha a maneira como ele realiza seu trabalho e como o *software* deve lhe auxiliar, e o do desenvolvedor, que questiona a maneira como, internamente, o *software* deve funcionar.

Existem vários requisitos de *software*, porém esses requisitos podem ser divididos em funcionais e os não funcionais. Os requisitos funcionais especificam funções que o sistema deve ser capaz de executar. Definem o comportamento do sistema, através da transformação que os componentes de hardware e de *software* realizam nas entradas para produzirem as saídas. Os requisitos não funcionais determinam os comportamentos e restrições que os sistemas de software devem satisfazer (PRESSMAN, 2006).

3.3.3 Projeto

“Um projeto de *software* é uma descrição de estrutura de *software* a ser implementada, dos dados que são parte do sistema, das interfaces entre os

componentes do sistema e, algumas vezes, dos algoritmos utilizados” (SUMMERVILE, 2011, p. 50).

Segundo Pressman (2006), os principais componentes do projeto de *software* são: projeto de dados/classes; projeto arquitetural; projeto de interface e; projeto em nível de componentes. O projeto de dados/classes transforma modelos de análise-classe em realizações de classes de projeto e nas estruturas de dados dos requisitos para implementar o *software*. O projeto arquitetural é o arcabouço do sistema que deriva da especificação do sistema. O projeto de interface descreve como o *software* se comunica com os sistemas que interoperam com ele e com as pessoas que o utilizam. O projeto em nível de componentes transforma elementos estruturais da arquitetura em uma descrição procedural dos componentes de *software*.

De acordo com Schach (2009), a entrada para o processo de projeto é o documento de especificação, uma descrição do que o produto deve fazer. A saída é o documento de projeto, uma descrição de como o produto atingirá esse objetivo.

Segundo Booch *et al.* (2000), existem várias maneiras de se definir um projeto detalhado ou modelo. As duas maneiras mais comuns são provenientes da perspectiva de um algoritmo ou da perspectiva orientada a objetos.

Na perspectiva de um algoritmo, o principal bloco de construção é o procedimento ou a função. Essa perspectiva conduz os desenvolvedores a voltar seu foco de atenção para questões referentes ao controle e à decomposição de algoritmos maiores em outros menores. Não existe nenhuma grande desvantagem nessa solução, com exceção da tendência a permitir sistemas instáveis. À medida que os requisitos se modificam e o sistema cresce fica difícil a manutenção do sistema construído a partir do foco em algoritmos. Na perspectiva orientada a objetos, o principal bloco de construção de todos os sistemas é o objeto ou a classe (BOOCH *et al.*, 2000).

3.3.3.1 Linguagem Unificada de Modelagem (UML)

Segundo Rosseti *et al.* (2000), a UML é uma linguagem de modelagem para representação conceitual e física de sistemas orientados a objeto. A UML é muito

utilizada para modelar sistemas de informação. Complementam Rosseti e Chan (2003) dizendo que a UML é utilizada como padrão para modelagem orientada a objeto no desenvolvimento de *softwares* e na modelagem de sistemas em gerais.

Para Tiberti (2004), a grande vantagem da UML em relação a outros métodos é que ela não é um método em si, mas, sim, uma linguagem para representação de um sistema e pode ser aplicada de maneira independente nas fases do processo de *software*. Por se tratar de uma linguagem, a transição entre as fases de análise, projeto e codificação é natural, rápida, complementar e sem ambiguidade.

Para realizar a modelagem de sistemas, a UML utiliza diagramas. Conforme Furlan (1998), um diagrama é uma apresentação gráfica de uma coleção de elementos de modelo, frequentemente mostrado como um gráfico conectado de arcos (relacionamento) e vértices (outros elementos do modelo).

Esses diagramas da UML são classificados em estáticos e dinâmicos. Os diagramas estáticos existem para visualizar, especificar, construir e documentar os aspectos estáticos de um sistema. Os aspectos estáticos do sistema são considerados como uma representação de seu esqueleto e estrutura relativamente estáveis. Os diagramas dinâmicos são utilizados para visualizar, especificar, construir e documentar os aspectos dinâmicos de um sistema. Os aspectos dinâmicos de um sistema são considerados como uma representação de suas partes que sofrem alterações (BOOCH *et al.*, 2000).

Segundo Booch *et al.* (2000), a UML usa quatro tipos de diagramas estáticos (Classes, Objetos, Componentes e Implantação) e cinco tipos de diagramas dinâmicos (Casos de uso, Sequências, Colaborações, Gráficos de estados e Atividades) para a modelagem de sistemas.

3.3.4 Implementação

Com base nas especificações de requisitos e nos projetos de *software*, o desenvolvedor na fase de implementação é responsável por codificar os módulos referente a aplicação de acordo com a plataforma tecnológica e linguagem específica definida no projeto

Segundo Schach (2009), a implementação é o processo de converter o projeto de *software* em código. A implementação envolve a escolha da linguagem para o desenvolvimento e a programação do *software*.

De acordo com Sommerville (2011), a implementação de um sistema parte naturalmente dos processos de projeto do sistema. Pode ser utilizado ferramentas *Case* para gerar um esqueleto de programa com base no projeto. Isso inclui código para definir e implementar interfaces e, em vários casos, o desenvolvedor precisa apenas adicionar detalhes da operação de cada componente do programa.

Uma atividade fundamental na implementação é a programação. Segundo Sommerville (2011), a programação é uma atividade pessoal e normalmente não existe um processo geral a seguir. Alguns programadores começarão com componentes que compreendem bem e irão desenvolvê-los para, então, passarem para aqueles mais mal compreendidos. Outros preferem a abordagem oposta, deixando os componentes mais familiares para o fim.

3.3.5 Validação

“A validação de *software* ou, mais genericamente, verificação e validação (V & V) destina-se a mostrar que um sistema está em conformidade com sua especificação e que atende as expectativas do cliente” (SOMMERVILLE, 2011, p.52).

Conforme Sommerville (2011), dentro do processo de V & V, há duas abordagens complementares para a verificação e análise de sistemas:

- Inspeções de *software*: análise e verificações da representação do sistema com documento de requisitos, diagramas de projeto e código-fonte do *software*. As inspeções podem ser realizadas em todos os estágios do desenvolvimento;
- Testes de *software*: envolve executar o *software* com dados de teste. São examinadas as saídas do *software* e seu comportamento operacional para verificar se seu desempenho está conforme o necessário. O teste é uma técnica dinâmica de verificação e validação.

Segundo Pressman (2006), o teste de *software* é um elemento crítico da garantia de qualidade de *software* e representa a última revisão da especificação, do projeto e da codificação.

De acordo com Sommerville (2011), o teste é destinado a mostrar que um programa faz o que é proposto e para descobrir os defeitos do programa antes do uso. Quando se testa o *software*, o programa é executado usando dados fictícios. Os resultados do teste são verificados à procura de erros, anomalias ou informações sobre os atributos não funcionais do programa.

Para Myers (1979), o teste é o processo de executar um programa com a intenção de descobrir um erro através de um bom caso de teste. Um bom caso de teste pode ser definido como aquele que tem alta probabilidade de revelar defeitos no *software* e um caso de teste bem sucedido é aquele capaz de revelar erros ainda não descobertos.

Segundo Sommerville (2011), o processo de teste tem dois objetivos distintos:

- Demonstrar ao desenvolvedor e ao cliente que o *software* atende a seus requisitos. Esse objetivo leva a testes de validação, nos quais se espera que o sistema execute suas funções corretamente usando determinado conjunto de casos de teste que refletem o uso esperado do sistema;
- Descobrir situações em que o *software* se comporta de maneira incorreta, indesejável ou de forma diferente das especificações. Esse objetivo leva a testes de defeitos, nos quais os casos de teste são projetados para expor os defeitos.

De acordo com Pressman (2006), o desenvolvimento do teste de *software* envolve quatro etapas: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados. Essas etapas são desenvolvidas ao longo do desenvolvimento do *software* e se concretizam em quatro fases: teste de unidade, teste de integração, teste de validação e teste de sistema.

O teste de unidade testa individualmente cada módulo da arquitetura do *software* buscando revelar erros de lógica e fornecer evidências de que o módulo funciona adequadamente. O teste de integração é uma técnica sistemática para integrar os módulos componentes da estrutura do *software*, visando à identificação de erros de interface entre tais módulos. No teste de validação os requisitos estabelecidos como parte da análise de requisitos são validados em contraste com o

programa que acabou de ser construído. O teste de sistema verifica se todos os elementos do sistema se combinam adequadamente e se o desempenho do sistema foi atingido na perspectiva dos requisitos determinados (PRESSMAN, 2006).

Para Sommerville (2011), durante o desenvolvimento, o teste pode ocorrer em três níveis de granularidade:

- Teste unitário: as unidades individuais de programa ou classes de objetos são testadas individualmente. Esse teste deve centrar-se em testar a funcionalidade dos objetos ou métodos;
- Teste de componentes: várias unidades individuais são integradas para criar componentes compostos. Esse teste deve centrar-se em testar as interfaces dos componentes;
- Teste de sistema: alguns ou todos os componentes de um sistema estão integrados e o sistema é testado como um todo. Esse teste deve centrar-se em testar as interações entre os componentes.

Para que as etapas de teste de *software* sejam realizadas com sucesso é importante definir técnicas para o seu desenvolvimento. Segundo Maldonado (1991), os testes são estabelecidos de acordo com três técnicas: funcional, estrutural, baseada em erros e máquinas de estados finitos. Na técnica funcional, os requisitos de teste são estabelecidos a partir da especificação funcional do *software*. Na técnica estrutural derivam-se os requisitos de teste a partir dos aspectos de implementação do *software*. Na técnica baseada em erros, os elementos requeridos para caracterizar a atividade de teste são baseados em erros comuns que podem ocorrer durante o processo de desenvolvimento de *software*. Na técnica baseada em máquinas de estados finitos, uma máquina de estados finitos é utilizada para modelar o comportamento do sistema.

Conforme Pressman (2006), o teste funcional (ou caixa preta) se refere a testes que são conduzidos na interface do *software*. Examina algum aspecto fundamental do sistema, pouco se preocupando com a estrutura lógica interna do *software*. O teste estrutural ou caixa branca é baseado em um exame rigoroso dos detalhes procedimentais. Caminhos lógicos internos ao *software* e colaborações entre componentes são testados, definindo-se casos de testes que exercitam conjuntos específicos de condições e/ou ciclos.

3.3.6 Manutenção

De acordo com Schach (2009), assim que o produto passa pelo teste de aceitação, ele é entregue ao cliente final. O produto é instalado e usado para a finalidade para qual foi construído. Porém, qualquer produto útil, será submetido à manutenção pós-entrega, seja para corrigir falhas (manutenção corretiva), seja para estender a funcionalidade do produto (evolução/aperfeiçoamento).

Pressman (2006) afirma que apenas 20% do trabalho de manutenção é relativo à manutenção corretiva, os outros 80% são de adaptação de sistemas ao ambiente externo, de melhorias solicitadas pelos usuários e de processos de reengenharia.

Conforme Stark *et al.* (1999), na atividade de manutenção um dos pontos importante que deve ser analisado é a alteração dos requisitos. Essas alterações impactam nos custos e na qualidade do produto. Os autores definiram duas técnicas para lidar com as mudanças de requisitos durante a manutenção. A primeira é a análise exploratória de dados para o entendimento dos tipos de mudanças realizadas. A segunda é a utilização de um modelo de regressão para ajudar na comunicação sobre os custos e cronogramas afetados pelas mudanças.

4. UMA ABORDAGEM PARA SIMULAÇÃO DE SISTEMAS DE MANUFATURA A PARTIR DE *FEATURES* DE UMA PEÇA

Este capítulo inicialmente apresenta as formas de abordagens para simulação de sistemas manufatura utilizadas tradicionalmente. Em seguida, detalha a utilização da tecnologia de *features* nos sistemas de manufatura. Após, é mostrada a abordagem proposta. A última parte do capítulo trata do funcionamento da abordagem proposta.

4.1 ABORDAGENS PARA SIMULAÇÃO DE SISTEMA DE MANUFATURA

A simulação é uma ferramenta muito utilizada, ao longo dos anos, em sistemas de manufatura conforme visto na seção 3.1. Ela permite que o usuário realize testes e experimentos sem alterar o cotidiano da empresa possibilitando uma tomada de decisão com maior acurácia.

Devido à flexibilidade e ao poder de análise da simulação, existe um grande número de exemplos de aplicações em sistemas de manufatura. Porém, as principais abordagens para simulação de sistemas de manufatura estão ligadas a fatores específicos como:

- Tipo de Processo: os processos de manufatura são analisados pela simulação para projeto, planejamento, programação e melhoria de seus desempenhos relacionados ao planejamento do recursos produtivos e da mão-de-obra, do tempo e custo de produção, da quantidade de produção, do fluxo de produção, dos níveis de estoque, das filas, do recurso restrição, da automação dos processos, dentre outros;
- Tipo de Peça/Produto: as peças ou produtos na simulação de sistemas de manufatura são referências quando se busca determinar o *mix* a ser manufaturado, o sequenciamento, a manufaturabilidade, o tipo de inspeção, estocagem e movimentação, o tempo de *setup* e as condições de corte;
- Tipo de *Layout*: a simulação é utilizada para simular *layouts* de sistemas de manufatura inexistentes ou para realizar mudanças em *layout* existentes.

Nessas simulações são analisados os caminhos/distâncias dos fluxos de material, os tipos e as quantidades de equipamentos utilizados na movimentação e fabricação e a necessidade e localização dos estoques.

4.2 UTILIZAÇÃO DA TECNOLOGIA DAS *FEATURES* NOS SISTEMAS DE MANUFATURA

A representação de um produto é um dos itens mais importantes para sua produção. O produto precisa ser detalhado ao nível dos componentes para sua fabricação. As principais características de um componente mecânico para ser fabricado podem ser descritas como geométricas, topológicas e tecnológicas. As *features* fornecem essas informações quando inseridas numa peça. *Feature* é considerado um paradigma para a integração das atividades de engenharia, desde o projeto até a manufatura.

O conceito de *features* aplicada a sistemas de manufatura iniciou com os sistemas CAD (*Computer Aided Desing* - Projeto Assistido por Computador), a partir da necessidade da utilização de um conjunto de tecnologias para fabricar objetos físicos diretamente das fontes de dados gerados pelo sistema CAD. Para isso foi integrado o sistema CAD com o sistema CAM (*Computer Aided Manufacturing* - Fabricação Assistida por Computador) gerando o sistema CIM (*Computer Integrated Manufacturing* - Sistema de Manufatura Integrada). Esses sistemas utilizam as *features* para reduzir os custos e os erros na integração do projeto com manufatura.

Com a continuação das pesquisas na área de *features* aplicada à manufatura surgiram os sistemas CAPP (*Computer Aide Process Planning* - Planejamento de Processo Auxiliado pelo Computador) que são sistemas que buscam automatizar o processo de planejamento do processo na interligação do sistema CAD com o sistema CAM. Esses sistemas geram o plano do processo que são roteiros de produção composto pela sequência de produção, máquinas, tempos e seleção de ferramenta, dentre outros.

Os sistemas CAPP buscam gerar um plano de processo otimizado, ou seja, o melhor plano de processo a partir dos recursos disponíveis. Para isso, esses

sistemas possuem regras definidas ou uma base de conhecimento tornando-os sistemas especialistas.

A partir do exposto, observa-se que a utilização da tecnologia de *features* na manufatura até o momento é voltada para integração entre o projeto da peça e a sua manufatura e na automatização do planejamento do processo eliminando erros e reduzindo o tempo de planejamento.

4.3 ABORDAGEM PROPOSTA

A abordagem deste trabalho foi proposta a partir da constatação da inexistência de trabalhos acadêmicos anteriores que utilizem a técnica da simulação para construir cenários com sistemas de manufatura e analisar sua dinâmica a partir das *features* contidas na peça.

A abordagem utiliza a peça como referência para simulação de sistemas de manufatura. A peça é composta de *features* que são características geométricas que dão forma à peça. Cada *feature* possui um volume e uma área que são utilizados como base para os cálculos de tempos e custos de fabricação da peça. O volume da *feature* é utilizado como base para o cálculo do tempo de debaste e a área da *feature* como base para o tempo de acabamento. Os custos são calculados baseados nos tempos de processamento de cada recurso.

Além dos recursos de fabricação responsáveis pelas *features*, a abordagem considera os recursos de transporte de materiais que são responsáveis por movimentar a peça de um recurso de fabricação para outro e a dinâmica das filas em cada recurso do sistema.

A abordagem possui a flexibilidade e a versatilidade da simulação computacional com a coerência da descrição detalhada da geometria fornecida pelas *features*. Ela possibilita realizar simulações a partir do projeto da peça com as *features*, o que torna confiável a simulação, porém não tira a possibilidade de desenvolvimento de diferentes cenários para uma melhor tomada de decisão.

Os diferenciais da abordagem proposta são:

- É um paradigma diferente para realizar a simulação de sistema de manufatura definindo o roteiro de produção e os processos (recursos) a partir das *features*

da peça. A abordagem tradicional considera a peça como um todo e na abordagem proposta são considerados todas as *features* contidas na peça separadamente;

- É uma forma diferente de calcular tempo e custo de usinagem. A forma tradicional utiliza a peça como base para os cálculos e na abordagem proposta utiliza o volume e a área das *features* como base para os cálculos;
- É voltada para a simulação do sistema da manufatura como um todo considerando a dinâmica do sistema, porém são calculados indicadores gerais do sistema e também específicos dos recursos;
- É uma abordagem genérica podendo ser usada para diferentes tipos de sistemas de manufatura;
- Não utiliza base de conhecimento ou regras específicas para a tomada de decisão. Busca a flexibilidade para realização de testes e experimentos.

As limitações da abordagem são:

- Os recursos de fabricação têm que ser máquinas que realizam os processos de usinagem convencional (Torneamento, Fresamento, Furação, Retificação, Mandrilamento, Brochamento, Aplainamento, Alargamento);
- As *features* devem ser do tipo (Rotacional, Prismática, Cavidade ou Circular) de acordo com o apresentado na seção 2.2;
- Não é possível simular dois recursos de fabricação ou transporte paralelamente. Cada recurso é único e pode ter sua capacidade alterada de acordo com o cenário proposto;
- Somente um tipo de peça por vez pode ser simulado para cada sistema de manufatura modelado.

4.3 FUNCIONAMENTO DA ABORDAGEM PROPOSTA

Para um melhor detalhamento, o funcionamento da abordagem proposta foi dividida em quatro partes: chegada de peça no sistema de manufatura; execução da simulação; processamento das filas e cálculo dos tempos e dos custos dos recursos. Nas próximas seções são apresentadas cada uma dessas etapas.

4.3.1 Chegada de Peça

As peças entram no sistema de manufatura a ser simulado de acordo com o tempo de chegada de peças (TCP). A primeira peça entra no instante zero. A segunda peça entra no processo no instante igual ao TCP. É verificado se o recurso está livre. Caso o recurso (fabricação ou transporte) esteja livre, a peça entra no recurso para ser atendida. Caso o recurso esteja ocupado, a peça é encaminhada para uma fila do recurso e após é liberada para o processamento. As atividades descritas são realizadas para todas as peças a serem simuladas. A Figura 4.1 detalha essa estrutura.

Figura 4.1 - Chegada de peças no sistema de manufatura

4.3.2 Execução da Simulação

Essa etapa inicia-se com o cálculo do passo da simulação (incremento). O sistema calcula o tempo que cada recurso leva para produzir uma *feature* ou para

transportar uma peça. O passo é igual ao menor tempo dentre todos os tempos dos recursos calculados, ou seja, a simulação será atualizada a cada passo de tempo que é igual ao mínimo tempo dos recursos. Em seguida, iguala o tempo de simulação ao passo.

A cada novo passo da simulação são atualizados os *status* de cada recurso (Livre e Ocupado) e calculados os tempos de processamento e os tempos de espera (fila) em cada recurso. A simulação termina quando o tempo da soma dos passos da simulação atingir o tempo total de simulação predeterminado ou quando simula a quantidade de peças predeterminada. A Figura 4.2 detalha essa estrutura.

Figura 4.2 - Execução da simulação

4.3.3 Processamento das Filas

Cada recurso tem um local de espera (fila) antes de seu processamento. A prioridade adotada na fila é o FIFO (*First In, First Out*). A cada passo da simulação o sistema verifica o *status* de cada recurso. Caso não tenha peça na fila o sistema passa para o próximo passo. Caso tenha peça na fila, verifica se o recurso está livre.

Caso o recurso esteja livre, processa a peça. Caso o recurso esteja ocupado, insere a peça na fila e depois processa a peça de acordo com a prioridade adotada. A Figura 4.3 detalha essa estrutura.

Figura 4.3 - Processamento das filas

4.3.4 Cálculo dos Tempos e dos Custos dos Recursos

As peças ao longo da simulação sofrem alterações geométricas com inserção das *features* na sua estrutura pelos recursos de fabricação ou podem ser transportadas pelos recursos de transporte.

Para cada uma dessas atividades são calculados um tempo e um custo. O tempo e o custo de fabricação das *features* variam conforme o tipo da *feature*. As fórmulas de calcular o tempo e o custo de transporte foram as mesmas para todos os tipos de equipamentos ou operários. As fórmulas usadas nesses cálculos foram desenvolvidas a partir da adaptação da revisão bibliográfica das seções 2.1 e 2.2 e são detalhadas a seguir.

1. Cálculo do Tempo de Fabricação

$$T_f = T_c + T_a + T_i + T_{fr} \quad (4.1)$$

Onde:

T_f – tempo de fabricação por recurso (min)

T_c – tempo de corte (desbaste) da *feature* (min)

T_a – tempo de acabamento da *feature* (min)

T_i – tempo de improdutivo (min)

T_{fr} – tempo de troca da ferramenta (min)

$$T_c = \frac{V_r}{MRR} \quad (4.2)$$

$$\text{Feature Tipo Rotacional} - MRR_r = d \times f \times s \quad (4.3)$$

$$\text{Feature Tipo Prismática} - MRR_p = D \times d \times s \quad (4.4)$$

$$\text{Feature Tipo Cavidade} - MRR_{ca} = l \times d \times s \quad (4.5)$$

$$\text{Feature Tipo Circular} - MRR_{ci} = d \times D \times v \quad (4.6)$$

Onde:

T_c – tempo de corte (desbaste) da *feature* (min)

V_r^6 – volume da *feature* (mm³)

MRR – taxa de remoção do material (mm³/min)

d – profundidade do corte (mm)

D – diâmetro do corte (mm)

l – comprimento do corte (mm)

f – avanço por revolução (mm)

s – velocidade de corte (mm/min)

v – velocidade de avanço (mm/min)

$$T_a = \frac{A_f}{R_t} \quad (4.7)$$

Onde:

T_a – tempo de acabamento da *feature* (min)

A_f^7 – área da *feature* (mm²)

R_t – taxa de remoção da área da *feature* (mm²/min)

$$T_i = T_c \times \left(\frac{P_i}{100} \right) \quad (4.8)$$

Onde:

T_i – tempo improdutivo (min)

P_i – percentual de tempo improdutivo (%)

$$T_{fr} = \left(\frac{T_c}{T} - \frac{1}{Z} \right) T_{ft} \quad (4.9)$$

Onde:

T_{fr} – tempo de troca da ferramenta (min)

T_c – tempo de corte (desbaste) da *feature* (min)

T – tempo de vida da ferramenta (min)

Z – número de peças do lote (unid)

T_{ft} – tempo de troca de ferramenta (min)

⁶ O volume das *features* é calculado de acordo com a Tabela 2.1 da Seção 2.2 do Capítulo 2.

⁷ A área das *features* é calculada de acordo com a Tabela 2.2 da Seção 2.2 do Capítulo 2.

2. Cálculo do Custo de Fabricação

$$C_f = \left(\frac{S_h}{60} \right) \times T_f + \left(\frac{S_m}{60} \right) \times T_f + C_{fr} \quad (4.10)$$

Onde:

C_f – custo de fabricação (R\$)

T_f – tempo de fabricação por recurso (min)

S_h – salário do operador (R\$/hora)

S_m – salário da máquina (R\$/hora)

C_{fr} – custo da(s) ferramenta(s) (R\$)

3. Cálculo do Tempo de Transporte

$$T_{to} = T_t + T_i \quad (4.11)$$

$$T_t = \frac{D_t}{V_t} \quad (4.12)$$

$$T_i = T_t \times \left(\frac{P_i}{100} \right) \quad (4.13)$$

Onde:

T_{to} – tempo total de transporte (min)

T_t – tempo de transporte (min)

D_t – distância do percurso do transporte (m)

V_t – velocidade do transportador (m/min)

T_i – tempo improdutivo (min)

P_i – percentual de tempo improdutivo (%)

4. Cálculo do Custo de Transporte

$$C_t = C_m \times T_{to} \quad (4.14)$$

Onde:

C_t – custo de transporte (R\$)

C_m – custo de movimentação por minuto (R\$)

T_{to} – tempo total de transporte (min)

5. O AMBIENTE DE SIMULAÇÃO

Este capítulo inicialmente apresenta o ambiente de simulação desenvolvido. Após uma curta introdução, são detalhadas as etapas de desenvolvimento do ambiente de simulação. Logo em seguida são apresentados os requisitos do sistema, sua arquitetura em seus seis módulos, acompanhada do detalhamento das interfaces e da modelagem do sistema. Após, é tratada a implementação do ambiente. A última parte do capítulo é a documentação.

5.1 INTRODUÇÃO

O ambiente de simulação desenvolvido neste trabalho é uma ferramenta para a simulação de sistemas de manufatura que permite construir, testar e experimentar modelos baseados em sistemas reais ou não. Ele fornece capacidade de simulação aos sistemas de manufatura permitindo avaliações rápidas e econômicas de diversas alternativas de produção.

O sistema torna possível que os responsáveis pela área de manufatura com pouco ou nenhum conhecimento de programação e simulação efetuem estudos de maneira simples e correta, sem um treinamento prévio muito elaborado. No ambiente é possível modelar e simular os elementos importantes da manufatura.

As peças fabricadas no ambiente de simulação apresentam tamanhos e formas variadas, contudo é possível descrever cada peça em termos de *features* partindo de um modelo sólido. Quando se seleciona uma *feature* suas características, como por exemplo diâmetro, comprimento, largura e altura, devem ser fornecidas para que representem a situação real expressa por um projeto da engenharia ou pelo conhecimento de especialistas.

Segundo Han *et al.* (2000), é árdua a tarefa de reconhecer as características de um produto, não só reconhecer, mas assegurar que estas características possam ser fabricadas com recursos disponíveis. Ademais, há vários tipos de *features* a serem consideradas: furos, rasgos, rebaixos, cavidades, ressaltos, depressões, entalhes, dentre outros.

O ambiente de simulação procura tornar a simulação a mais cômoda possível utilizando as *features*, pois a partir das informações fornecidas pelas *features* é possível identificar os recursos necessários, assim como a sequência de produção para a simulação de um sistema de manufatura.

5.2 ETAPAS DO DESENVOLVIMENTO

O desenvolvimento do ambiente de simulação foi dividido em cinco etapas de acordo com a Figura 5.1. Essa divisão ocorreu a partir de estudo prévio da literatura de modelos de processo de *software* na seção 3.3.

Figura 5.1 - Etapas do desenvolvimento do ambiente de simulação

A primeira etapa foi análise de requisitos em que foram definidos os requisitos funcionais e não funcionais do sistema. A segunda etapa foi o projeto que foi dividido no projeto da arquitetura com suas interfaces, projeto do funcionamento e modelagem do sistema. A terceira etapa foi a implementação, onde o sistema foi codificado numa linguagem computacional. A quarta etapa foi a verificação e validação a partir dos testes do programa. Ao longo de todas as etapas foi realizada a documentação do sistema.

Nas próximas seções serão detalhadas as etapas análise de requisitos, projeto, implementação e documentação. No capítulo 6 será detalhada a etapa de verificação e validação do sistema.

5.3 ANÁLISE DE REQUISITOS

Iniciou-se o trabalho definindo os propósitos e os requisitos do ambiente de simulação de modo a entender quais parâmetros influenciam o sistema e o que está modelando. Para isso se buscou entender o funcionamento de diferentes tipos de sistemas de manufatura.

Inicialmente, os esforços foram direcionados para o entendimento de diferentes tipos de sistemas de manufatura, assim como seus principais parâmetros e restrições. Para obtenção da familiarização e conhecimento adequado dos elementos que compõem um sistema de manufatura utilizou-se a forma matemática/lógica em forma de fluxogramas.

Segundo Pidd (1998), grande parte do trabalho de construção de um modelo de simulação concentra-se no entendimento da lógica de funcionamento do sistema a ser simulado em termos de entidades e interações.

Após o período de familiarização com a lógica dos sistemas de manufatura, observou-se a importância dos aspectos geométricos da peça, assim como os aspectos de fabricação e movimentação para um bom funcionamento de um sistema de manufatura. A partir desta constatação, buscou-se na literatura trabalhos voltados para a simulação de sistemas de manufatura baseado na geometria da peça, porém não foram encontrados trabalhos específicos que tratassem do assunto. Neste momento foi respondida a pergunta sobre o porquê da existência deste trabalho e o que ele procura solucionar.

Ao aprofundar os estudos em modelagem geométrica da peça para sistemas de manufatura foram encontrados trabalhos sobre sistemas CAD (Projeto Assistido por Computador) que utilizam *features* na modelagem de formas geométricas das peças. Observou-se diversos trabalhos na literatura voltados para interface entre o projeto no sistema CAD e o sistema de manufatura CAM (Manufatura Assistida por Computador). Com isso foi respondida a pergunta sobre a viabilidade de realizar simulações da fabricação de peças baseada na tecnologia de *features*.

Como as *features* são características geométricas das peças e atualmente os sistemas CAD que as modelam na etapa de projeto são desenvolvidos em 3D, optou-se por incluir no escopo do ambiente de simulação a visualização final do sistema de manufatura a ser simulado também em 3D utilizando os conceitos da

realidade virtual. Com essa interface o usuário pode validar a simulação, sentir uma maior interação com o sistema e observar a fabricação da peça detalhadamente com as *features* e a movimentação ao longo do sistema de manufatura.

Outros fatores que foram analisados que influenciam o sistema foram: a melhor forma de armazenamento dos dados, os principais tipos de *features* e de recursos a serem incluídos no sistema, as melhores soluções para o ambiente de implementação do trabalho e do sistema visualização tridimensional e a escolha da linguagem de programação para a codificação.

Na análise de requisitos também foram definidos os principais usuários, o paradigma de desenvolvimento, as formas de mensuração do desempenho e os requisitos do sistema, assim como suas limitações.

O sistema é voltado para os responsáveis por sistemas de manufatura que necessitam analisar projeções de seus sistemas sem alterar seu cotidiano. Esses usuários têm amplos conhecimentos em sistemas de manufatura, como, por exemplo, sobre os parâmetros dos recursos existentes em seus sistemas, porém não possuem conhecimentos em simulação e programação.

O ambiente permite a modelagem hierárquica dos elementos que compõem o sistema de manufatura, o uso do paradigma de orientação a objetos e, na elaboração, possibilita a reutilização de *features* e recursos para novos cenários.

O ambiente propõe cenários para analisar o desempenho de sistemas de manufatura. Para mensurar esse desempenho foram definidos indicadores baseados em custo e tempo. Esses indicadores podem ser gerais do sistema, como, por exemplo, o tempo de produção de uma peça ou, pode serem específicos, o tempo e o custo por recurso de fabricação e/ou o transporte (movimentação).

Os principais requisitos do ambiente de simulação são:

- Facilidade de Uso: possuir uma interface auto-explicativa que permite que um usuário execute uma simulação no primeiro contato com a ferramenta;
- Desempenho: executar as simulações sem demonstrar lentidão ou travar;
- Qualidade das Informações: gerar dados trabalhados e transformados em informações que possibilitem a tomada de decisão;
- Similaridade com a Realidade: ao ver o sistema em funcionamento é possível entender como tal funcionamento ocorre;
- Nível de Detalhamento: ser aplicável a processos diversos, ou seja, deve proporcionar uma flexibilidade para simular a realidade das empresas;

- Adaptabilidade: ser possível realizar alterações em partes do sistema sem prejudicar o todo;
- Documentação: ter uma documentação do código do sistema assim como um manual de utilização e do funcionamento do mesmo.

Na análise dos requisitos do sistema foi definido que não era esperado como resposta decisões de otimizações, mas sim fazer simulações de situações reais. O ambiente se limita a realizar projeções a partir de dados fornecidos pelo usuário do sistema.

A principal limitação do sistema esta relacionada com a quantidade de *features* constante na sua base de dados. A variedade de peças a serem simuladas dependem da quantidade de *features* armazenadas, pois a descrição detalhada da peça é desenvolvida a partir delas. As *features* fornecem o acesso às informações da peça num nível aprofundado e preciso. Por exemplo, quando se modela um furo em uma peça, geram-se algumas informações geométricas a seus respeito, como localização, diâmetro e profundidade .

A biblioteca de *features* desenvolvida inicialmente no ambiente de simulação é mostrada na Figura 5.2. Ela é composta por *features* rotacionais (passo, chanfro e arredondamento) prismáticas (plano, degrau e ranhura), cavidades (entalhe, depressão e rasgo) e circulares (furo).

Figura 5.2 - Tipos de *features* do ambiente de simulação

A variedade de recursos de fabricação e de recursos de transporte na base de dados também são limitadores do sistema. A quantidade de recursos armazenados determina a possibilidade de produzir os produtos.

Outra limitação definida é o escopo do sistema de visualização tridimensional. O sistema mostra o funcionamento da simulação do sistema de manufatura numa cena tridimensional sem a possibilidade de inserções, alterações ou remoções de elementos ao longo da simulação.

5.4 PROJETO DO AMBIENTE DE SIMULAÇÃO

5.4.1 Arquitetura do Ambiente de Simulação

A arquitetura básica do ambiente de simulação é composta por seis módulos que são interligados e interdependentes de acordo com a Figura 5.3.

Figura 5.3 - Arquitetura do ambiente de simulação

O módulo banco de dados corresponde ao armazenamento dos dados relativos às peças, às *features* e aos recursos que compõem o sistema.

O módulo modelador corresponde à modelagem do ambiente com a composição da sequência de produção e da cena tridimensional a partir dos objetos armazenados na base de dados.

O módulo simulador executa as atividades que ocorrerão ao longo da simulação. O módulo funciona como um núcleo da simulação e é responsável pela coordenação da simulação, controlando as relações entre os componentes do sistema ao longo da simulação.

O módulo interface gráfica com usuário corresponde à interface gráfica que permite a interação amigável entre o usuário e ambiente de simulação gerando acesso aos outros módulos.

O módulo de visualização tridimensional corresponde à cena 3D do sistema de manufatura simulado baseado nos conceitos de realidade virtual.

O módulo resultados é responsável por fornecer ao usuário do sistema os resultados relativos à simulação realizada.

5.4.1.1 Módulo Banco de Dados

O banco de dados é um componente fundamental para o ambiente de simulação. Ele é o responsável pelo armazenamento dos dados das peças, assim como a formação da biblioteca das *features* e dos recursos.

Os dados contidos na base de dados são de natureza estática, ou seja, os dados não podem ser alterados quando a simulação estiver sendo executada. Os dados podem ser inseridos, alterados ou excluídos pelo usuário na interface gráfica do sistema de acordo com o exemplo da Figura 5.4.

Na Figura 5.4 na parte esquerda da interface se localiza a tabela com os dados de todas as peças cadastradas no banco de dados do sistema. Nas colunas ficam os parâmetros e nas linhas as diferentes peças. Na localização acima da tabela tem-se o botão “+” para inserção de uma nova linha na tabela (peça) e o botão “-” para exclusão de uma peça da base de dados. Na parte direita da interface estão os espaços para preenchimento dos parâmetros da peça para inserção ou alteração de dados na tabela. O botão salvar é responsável pela inserção dos dados no banco de dados.

Figura 5.4 - Exemplo da interface peças do banco de dados

Antes do desenvolvimento propriamente dito do banco de dados realizou-se a sua modelagem para identificar todos os atributos que o compõem, assim como suas características. Essa modelagem é mostrada na Figura 5.5 e em seguida detalham-se os dados de cada tabela do banco de dados.

Figura 5.5 - Modelagem do banco de dados

A Tabela de Peças refere-se às informações relativas às peças que o sistema de manufatura produzirá. Cada peça possui características e informações particulares que são armazenadas nessa tabela de acordo com a Figura 5.4. Os dados que compõem essa tabela são:

- Código: refere-se ao código da peça para identificação no banco de dados. Com o código é possível formar famílias de peças com similaridades;
- Descrição ou Nome: refere-se ao espaço composto por vinte caracteres que estão disponíveis para um melhor detalhamento da peça;
- Material: refere-se a qual tipo de material é feita a peça. Essa característica é importante para a determinação dos parâmetros do recurso a ser utilizado na fabricação;
- Dimensões: refere-se às dimensões da peça que são o comprimento, a largura e a altura.

A Tabela *Features* refere-se às características de todas as *features* que o usuário pode utilizar na fabricação de uma peça. Cada *feature* possui um conjunto de parâmetros que não se alteram, independentemente das outras *features* existentes na biblioteca e de outras que venham a ser criadas. A Figura 5.6 mostra a interface *features* do banco de dados. Os dados que compõem essa tabela são:

- Código: refere-se ao código de cadastro da *feature* na base de dados. O código é dividido por tipo de *feature*. A Tabela 5.1 mostra os tipos diferentes de *features* cadastrados no banco de dados;

Tabela 5.1 - Codificação das *features* no banco de dados

Código	Tipo Feature	Nome Feature
1	Circular	Furo
2	Rotacional	Chanfro
3	Rotacional	Arredondamento
4	Prismática	Plano
5	Prismática	Degrau
6	Prismática	Ranhura
7	Cavidade	Entalhe
8	Cavidade	Depressão
9	Cavidade	Rasgo
10	Rotacional	Passo

- Descrição ou Nome: refere-se ao espaço composto por vinte caracteres que estão disponíveis para um melhor detalhamento da *Feature*;
- Tipo: refere-se aos quatro possíveis tipos das *feature* armazenadas na base de dados que são: rotativa, prismática, cavidade, circular;
- Dimensões: refere-se às dimensões da *feature* que são o Diâmetro, o Comprimento, a Largura, a Altura, o Diâmetro maior e o Diâmetro menor;
- Posição: refere-se à localização (X, Y, Z) da *feature* na peça. Os valores variam de zero um a um;
- Orientação: refere-se à face da localização da *feature* na peça. Em cada peça são consideradas seis possíveis faces.

	codigo	name	Cavid
1	9	Rasgo	Cavid
2	1	Furo	Circul
3	5	Degrau	Prism
4	7	Entalhe	Cavid
5	2	Chanfro	Rotac
6	9	Rasgo1	Cavid
7	1	Furo1	Circul
8	1	Furo2	Circul

Código: 9
 Nome: Rasgo1
 Tipo: Cavidade
 Largura (mm): 100,00
 Comprimento (mm): 100,00
 Altura (mm): 20,00
 Diâmetro(mm): 0,00
 Diâmetro Maior (mm): 0,00
 Diâmetro Menor (mm): 0,00
 Posição X(mm): 0,80
 Posição Y (mm): 0,90
 Posição Z(mm): 0,00
 Orientação: 1

Salvar Limpar Sair

Figura 5.6 - Interface *features* do banco de dados

A Tabela Recursos refere-se aos recursos que compõem o sistema de manufatura. Os recursos são divididos em fabricação e transporte. A Figura 5.7 mostra a interface recursos do banco de dados. Os dados armazenados nessa tabela são:

- Código: refere-se ao código de cadastro do recurso na base de dados. O código é dividido por tipo de recurso. A Tabela 5.2 mostra os tipos diferentes de *features* cadastrados no banco de dados;

Tabela 5.2 - Codificação dos recursos no banco de dados

Código	Tipo Recurso	Nome Recurso	Código	Tipo Recurso	Nome Recurso
1	Fabricação	Torno	9	Transporte	Operário
2	Fabricação	Fresa	10	Transporte	Esteria 1M
3	Fabricação	Furadeira	11	Transporte	Esteria 2M
4	Fabricação	Retífica	12	Transporte	Esteria 3M
5	Fabricação	Mandriladora	13	Transporte	Esteria 4M
6	Fabricação	Brochadeira	14	Transporte	Empilhadeira
7	Fabricação	Plaina	15	Transporte	AVG
8	Fabricação	Alargador	16	Transporte	Paleteira

- Descrição ou Nome: refere-se ao espaço composto por vinte caracteres que estão disponíveis para um melhor detalhamento do recurso;
- Parâmetros para o Cálculo do Custo da Fabricação: refere-se ao Custo de Mão-de-Obra; ao Custo da Máquina-Ferramenta e ao Custo da(s) Ferramenta(s);
- Parâmetros para o Cálculo do Tempo de Corte da Fabricação: refere-se ao Diâmetro do Corte; à Profundidade do Corte; ao Comprimento do Corte, ao Avanço por Revolução; à Velocidade de Corte e; à Velocidade de Avanço;
- Parâmetro para o Cálculo do Tempo de Acabamento da Fabricação: Taxa de Acabamento da Peça;
- Parâmetro para o Cálculo do Tempo Improdutivo da Fabricação e do Transporte: Percentual do Tempo Improdutivo de Fabricação e de Transporte;
- Parâmetros para o Cálculo do Tempo de Troca de Ferramenta da Fabricação: o Tempo de Vida da Ferramenta e o Tempo de Troca da Ferramenta;
- Parâmetros para Cálculo do Tempo do Transporte: refere-se à Velocidade e à Distância de Transporte das Peças;
- Parâmetro para Cálculo do Custo do Transporte: refere-se ao Custo por Unidade de Tempo.

Banco de Dados Recursos

	codigo	name	
1	1	Torno Universal	Fabric
2	14	Empilhadeira	Transp
3	3	Furadeira de Co...	Fabric
4	2	Fresa	Fabric
5	10	Esteira1m	Transp
6	9	Operario	Transp
7	1	Torno1	Fabric
8	3	Furadeira1	Fabric

Código: 1

Nome: Torno Universal

Tipo: Fabricacao

Fabricação - Dados Custo

Salario Op (R\$/h): 8,00

Salario Maq (R\$/h): 25,00

Custo Ferr (R\$/mm3): 0,08

Transporte

Velocidade (m/min): 0,00

Distância (m): 0,00

Custo (R\$/min): 0,00

Tempo Impr(%): 30,00

Fabricação - Dados Tempo

Diam. Corte (mm): 0,00

Prof. Corte (mm): 20,00

Compr. Corte (mm): 100,00

Avanço (mm): 0,00

Vel. Corte (mm/min): 15,00

Vel. Avanço (mm/min): 0,00

Vida Ferr (min): 5,00

Troca Fer (min): 0,80

Taxa Acab. (mm2/min): 8000,00

Salvar **Limpar** **Sair**

Figura 5.7 - Interface recursos do banco de dados

A modelagem destes objetos em uma base de dados não está diretamente ligada às relações destes objetos. O nível de abstração das relações deve ser ainda maior, pois, a relação entre dados é diferente das relações entre objetos. Uma base de dados liga os dados na representação e não denota o grau de relação entre os objetos.

5.4.1.2 Módulo Modelador

Os usuários do ambiente de simulação devem primeiramente construir um modelo a ser simulado. Para isto, o simulador tem um sistema de edição gráfica interativo. O usuário deve fornecer apenas os dados necessários à simulação sem desenvolver nenhuma programação ou ter conhecimento prévio de sistemas de simulação.

O sistema de manufatura é modelado por uma lista de instâncias de objetos pré-definidos no sistema. Cada instância representa um elemento do sistema, como por exemplo, uma *feature* ou um recurso de fabricação. Todos os componentes

dessa lista devem ter sido previamente cadastrados no banco de dados, ou seja, devem ter sido armazenados seus atributos particulares. A modelagem consiste em organizar esses componentes de forma a ser realizada a sequência para a simulação da produção de uma peça.

A Figura 5.8 detalha a interface de modelagem da simulação. Na parte superior tem-se o local de seleção da peça, do tempo entre entrada das peças no sistema, da quantidade a ser produzida, além do tempo de simulação. Na parte inferior esquerda está o tipo do recurso e o nome do recurso a ser selecionado.

Na parte inferior direita da Figura 5.8 é a lista da sequência de simulação da peça. Para inserir o recurso na lista de simulação é necessário pressionar no botão “+” que se encontra em cima do filtro do tipo de recurso e para excluir um recurso da lista é necessário pressionar o botão “-”.

Figura 5.8 - Interface de modelagem com recurso transporte

Após a inserção do recurso na lista se o mesmo for do tipo transporte, ele é normalmente inserido na lista, porém se for do tipo fabricação surgirá uma complementação da interface de acordo com a Figura 5.9. Na complementação seleciona-se o tipo da *feature* e em seguida o nome da *feature* a ser produzida pelo recurso de fabricação anteriormente selecionado.

Figura 5.9 - Interface de modelagem com recurso fabricação

5.4.1.3 Módulo Simulador

O módulo simulador inicia no instante que usuário pressionar o botão “ok” na interface do módulo modelador. Uma vez especificada as configurações de simulação do sistema de manufatura a ser simulado bem como o conjunto de parâmetros de cada objeto que compõem o sistema, a simulação pode ser imediatamente processada. O simulador não permite alterações durante o processamento da simulação.

Esse módulo funciona como um núcleo de simulação que atua sobre a lista de simulação modelada anteriormente para a peça selecionada. Esse núcleo é responsável pelo gerenciamento e execução da simulação.

Inicialmente, o núcleo seleciona o primeiro recurso da lista de simulação. Em seguida insere a primeira peça no sistema, baseado no tempo de chegada das peças e no passo da simulação. O passo da simulação é o incremento de tempo de atualização da simulação.

Na próxima etapa, o núcleo de simulação verifica se o recurso está “ocupado” ou “livre”. Caso esteja ocupado, insere a peça na espera (fila). Caso o recurso esteja livre realiza o atendimento da peça. Após realização do atendimento os indicadores do sistema são atualizados. Esses indicadores são baseados no tempo e custo de fabricação e transporte e nos tempos de espera nos recursos. Essas etapas descritas anteriormente são realizadas para todos os recursos da lista de simulação.

O núcleo coordena a alocação das atividades de acordo com a lista de simulação, o incremento (passo da simulação) de tempo, a quantidade de produção, o tempo de simulação e o envio dos resultados para o módulo resultados até o final da simulação. A Figura 5.10 detalha o funcionamento do módulo simulador.

As condições que resultam no término da simulação podem ser a quantidade de produção pré-determinada ou tempo de simulação finalizado. Nos dois casos, o controle do programa retorna à janela principal. Caso o tempo de simulação inserido na interface seja zero o sistema terminará após a fabricação da quantidade de peças pré-determinada. Caso a quantidade de peças a serem produzidas inseridas na interface for igual a zero, a quantidade de peças será determinada pelo tempo de simulação.

Antes de iniciar a simulação propriamente dita, o sistema promove a compilação dos parâmetros fornecidos verificando a consistência do modelo. Se algum problema for detectado, o pedido de simulação é cancelado e o usuário é informado do erro ocorrido. Se nenhuma inconsistência for encontrada a simulação é então iniciada.

A lista de simulação é criada para cada peça, pois só pode ser iniciada a simulação de uma peça após o término da produção da anterior. Essa restrição está relacionada à gestão do núcleo de simulação e ao *layout* físico da cena 3D.

Após o término da simulação, o usuário pode modificar os parâmetros, acrescentar novos elementos ao sistema de manufatura e recarregar o modelo. É possível retornar a qualquer das etapas anteriores e requisitar nova simulação.

As informações geradas pela simulação não são armazenadas no banco de dados, mas armazenadas segundo o formato de extensão *XML* (*Extensible Markup Language*) que pode ser lido pela interface gráfica com o usuário no módulo de resultados para uma melhor análise após a simulação ou por outros programas que importem esse formato de arquivo.

Figura 5.10 - Módulo simulador do ambiente de simulação

5.4.1.4 Módulo Interface Gráfica com Usuário

A qualidade das interfaces gráficas com usuário (GUI) é um dos grandes responsáveis pelo sucesso dos sistemas. As interfaces são responsáveis por realizar a comunicação entre o homem e o computador. Nesse sentido, desenvolveu-se uma interface amigável e comunicativa ao usuário. Para tornar a interface simples e de fácil aprendizado, utilizou-se o conceito de menus, janelas, caixas de diálogo, barra de ferramentas e barras de rolagem.

A interface foi desenvolvida a partir da análise do usuário; nesse caso especialistas em sistemas de manufatura, para evitar o desinteresse na utilização do sistema. A partir da definição do usuário a interface foi construída sem a necessidade de manipular, inserir ou retirar os objetos da cena 3D, pois para o perfil do usuário do sistema se tornaria complexo e lento. O sistema se encarrega de modelar a cena sem a necessidade de especificação na interface gráfica.

A consistência da interface entre as atividades similares foi considerada no desenvolvimento, como, por exemplo, a inserção de dados das peças, das *features* e dos recursos na base dados.

Um exemplo da interface gráfica do ambiente de simulação é detalhado na Figura 5.11 que é a interface principal do ambiente de simulação. Essa interface contém o cabeçalho, localizado no topo; abaixo aparecem os menus de acesso às diversas atividades desempenhadas pelo programa.

A partir da interface principal é possível acessar as interfaces secundárias para a utilização das principais funções do ambiente de simulação de acordo com a Figura 5.12.

A interface principal é composta pelos seguintes submenus:

- Banco de Dados: responsável pelo acesso ao desenvolvimento de um novo banco, pela conexão com um banco existente e pelo cadastro de peças, *features* e recursos;
- Simulação: responsável pelo acesso a interface de modelagem da lógica para simulação;
- Resultados: responsável pelo acesso a interface de resultados gerados a partir de dados da simulação;

- Gerenciar Acesso: responsável pelo acesso as interfaces de cadastro de novo usuário e de alteração de senhas;
- Sair: responsável pelo acesso a interface de saída do sistema;
- Sobre: responsável pelo acesso a interface ajuda que contém informações de como usar o sistema e a interface sobre que detalha a versão, os direitos e o responsável pelo desenvolvimento do sistema.

Figura 5.11 - Interface principal do ambiente de simulação

Figura 5.12 - Menus secundários do ambiente de simulação

5.4.1.5 Módulo Sistema de Visualização Tridimensional

O sistema de visualização tridimensional com o usuário é responsável pela simulação do chão-de-fábrica gerenciando peças, *features*, recursos e a execução de trocas de materiais entre eles.

As finalidades principais do sistema de visualização gráfico tridimensional desenvolvido neste trabalho são mostrar a simulação baseada na tecnologia de *features* (características geométricas) nas peças no formato 3D e a fácil identificação de possíveis erros na simulação. Caso o responsável pela simulação detecte algum erro, ele poderá voltar ao módulo modelador e alterar o programa e simular novamente.

O ambiente de simulação utiliza a realidade virtual não-imersiva que possibilita diferentes tipos de experiência com baixo custo e fácil manuseio. O desenvolvimento da visualização 3D foi realizado de forma que o usuário não se preocupe com ela na modelagem. Isso foi feito para que os usuários executem somente a modelagem da simulação e o sistema se encarregue de transformar essa modelagem numa cena 3D.

Por se tratar de um sistema de realidade não-imersiva, o ambiente de simulação faz uso de dispositivos convencionais para visualização e manipulação do ambiente, como monitor, *mouse* e teclado que fornecem suporte ao conjunto de estímulos para imersão. O ambiente 3D pode ser exibido em uma janela ou em tela cheia.

A interação do usuário com o ambiente de simulação é feita de forma explorativa, onde ele tem total controle sobre o direcionamento e os pontos de observação, porém não manipula diretamente os objetos durante a simulação, ou seja, não pode realizar alterações nos objetos. O usuário pode selecionar a direção e os pontos de observação durante a simulação. O estímulo básico do participante é o visual na observação da cena e suas animações, porém é possível navegar por entre a cena o que faz com que o usuário tenha um maior sentimento de presença no ambiente virtual.

A cena tridimensional contém animações para apresentar o comportamento dinâmico das principais partes do sistema de manufatura. Estas animações

propiciam ao simulador características de imersão facilitando a visão e compreensão do sistema.

A Figura 5.13 mostra uma peça modelada com quatro *features* (furo, rasgo, degrau e depressão) no ambiente de simulação e a Figura 5.14 mostra uma cena 3D composta por um recurso de fabricação e a peça com as *features* sendo fabricadas e transportadas no ambiente de simulação.

Figura 5.13 - Peça modelada com *features* no ambiente de simulação

Figura 5.14 - Exemplo de uma cena 3D do ambiente de simulação

5.4.1.6 Módulo Resultados

O módulo resultados é responsável por mostrar ao usuário final os dados resultantes da simulação. Os dados podem ser mostrados de duas formas:

- A primeira é ao final da simulação numa tabela com todos os indicadores e seus resultados. Esses dados podem ser salvos no formato XML que é o formato utilizado pela maioria dos programas estatísticos que possibilita uma posterior análise. O Apêndice A mostra um exemplo dos resultados salvo pelo sistema. A Figura 5.15 detalha a planilha de resultado mostrada após a simulação;
- A segunda é após salvar a simulação no formato XML, o usuário acessa a interface principal no menu resultados, seleciona a interface resultados e importa os dados do arquivo XML. Essa interface permite, além da visualização da tabela completa como anterior, plotar três tipos diferentes de gráficos: pizza, barra e linha. Esses gráficos são para os indicadores relativos aos tempos e aos custos de cada recurso. O sistema também permite salvar os gráficos plotados no formato XML e exportar os dados da tabela no formato CSV, podendo ser aberto posteriormente na forma de planilha do *Microsoft Excel®* ou de seus concorrentes. A Figura 5.16 detalha a interface de resultado para análise dos dados da simulação.

	Nome	Tempo Proc.	Custo Proc.	Tempo Fila	Taxa Util.Rec	
1	Furadeira1	112.434	45.4391	0	15.402	89
2	Empilhadeira	11.214	6.1677	0	1.53616	
3	Torno1	446.85	159.117	0	61.2123	
4	Esteira	18.156	3.6312	0	2.48712	
5	Furadeira de Co...	328.547	162.222	0	45.0065	
6	Operario	8.544	0.68352	0	1.17041	
7	Fresa	626.903	199.677	37.096	85.8771	
8	Operario	8.544	0.68352	0	1.17041	

Figura 5.15 - Tabela de resultados da simulação

Figura 5.16 - Interface de resultados da simulação

Os resultados do ambiente de simulação podem ser do sistema de manufatura como um todo ou particular de partes integrantes do sistema. Nos resultados gerais são fornecidos, por exemplo, informações sobre o tempo total de produção. Os resultados individuais indicam, por exemplo, o percentual de utilização de recursos e o tempo, o custo e o tempo de espera por recurso.

Os resultados para a análise do ambiente de simulação são baseados em indicadores que consistem em:

- Tempo Processamento por Recurso: é o tempo por recurso para a fabricação ou o transporte da quantidade de peças simuladas. Esse indicador é importante para analisar os diferentes tempos de cada recurso/*feature* e transporte/peça possibilitando a tomada decisão sobre qual máquina deve fabricar qual *feature* e qual transportador deve realizar o transporte da peça;
- Custo Processamento por Recurso: é o custo por recurso para produção da quantidade de peça simulada. Esse indicador permite uma melhor análise das possibilidades de produção;
- Tempo de Espera por Recurso: é o tempo improdutivo das peças paradas na fila dos recursos. Esse indicador sinaliza qual recurso está gerando fila e o tempo dessa fila;

- Taxa de Utilização de Recursos: é o percentual de ocupação de cada recurso participante do sistema de manufatura simulado. Esse percentual é calculado pela razão do total de tempo que o recurso foi utilizado pelo total de tempo de produção. Esse indicador possibilita uma análise da necessidade de aumento ou redução da capacidade de cada recurso para balancear o sistema de manufatura;
- Quantidade Produzida: é a quantidade total de peças produzidas;
- Tempo Médio de Processamento: é a divisão da soma dos tempos de processamento de todos os recursos utilizados na produção das peças pela quantidade de peças produzidas;
- Custo Médio de Processamento: é a divisão da soma dos custos de processamento de todos os recursos utilizados na produção das peças pela quantidade de peças produzidas;
- Tempo Médio de Fila: é a divisão da soma dos tempos de esperas nas filas de todos os recursos pela quantidade de peças produzidas;
- Tempo Total de Produção: é a soma dos tempos de processamento e de espera de todos os recursos do sistema;
- Tempo Médio de Produção: é o tempo total de produção dividido pela quantidade de peças produzidas. Esse indicador informa em quanto tempo em média uma peça é produzida.

5.4.2 Funcionamento do Ambiente de Simulação

Nesse tópico se busca detalhar a lógica de funcionamento do simulador desde o acesso do usuário no sistema até a visualização dos resultados na interface gráfica após a simulação. A Figura 5.17 apresenta o fluxo completo do funcionamento do sistema e em seguida são detalhadas as partes desse fluxograma.

Na Figura 5.17, os diagramas de cor laranja são relacionados ao acesso e saída do programa e os diagramas de cor vermelha são relacionados à entrada de dados no sistema. O diagrama de cor azul é relacionado à modelagem, o de cor cinza à simulação e os de cor verde são relacionados à saída de dados.

Figura 5.17 - Funcionamento do ambiente de simulação

5.4.2.1 Acesso e Saída do Sistema

A lógica do funcionamento do ambiente de simulação inicia quando o usuário solicita acesso ao sistema de acordo com a interface da Figura 5.18. O sistema verifica se ele tem autorização e em caso positivo é liberado o acesso ao sistema. Se a autorização for negativa, o usuário solicita *login* e senha para o administrador do sistema.

A interface de login é uma janela com o título "Login". Ela possui dois campos de entrada: "Login" e "Senha". Abaixo dos campos, há dois botões: "Entrar" e "Cancelar".

Figura 5.18 - Interface de acesso ao ambiente de simulação

Para criar um novo *login* o administrador do ambiente de simulação solicita o cadastro do novo usuário clicando no submenu novo a partir da interface acessada no menu Gerenciar Acesso da interface principal. Na interface da Figura 5.19, o administrador preenche o formulário com o *login* e uma senha padrão (123456) e pede para gravar clicando no botão salvar. O sistema verifica se o novo *login* e senha não existem e os armazena. O sistema envia uma mensagem "Cadastro gerado com sucesso".

A interface de gerenciamento de acesso é uma janela com o título "Gerenciador de Acesso". Ela possui uma lista de usuários à esquerda com os nomes "Administrador", "Heráclito" e "Teste". À direita, há campos para "Administrador", "Senha" e "Confirmar Senha". Abaixo, há cinco botões: "Novo", "Salvar", "Excluir", "Cancelar" e "Fechar".

Figura 5.19 - Interface de gerenciamento de acesso

O usuário do ambiente de simulação pode, após o cadastro realizado, alterar a senha padrão inicial do usuário clicando no submenu alterar senha a partir da interface acessada no menu Gerenciar Acesso da interface principal. Na interface da Figura 5.20, o usuário digita a senha antiga, a senha nova e confirma a senha nova. O usuário pede para gravar a senha clicando no botão confirmar. Ao confirmar a alteração da senha aparece uma janela com a mensagem “Senha alterada com sucesso”. Caso no momento da confirmação da senha nova o usuário tenha digitado a senha errada, o sistema avisa com a mensagem “Senha incorreta. Entre novamente com a senha”.

A screenshot of a Windows-style dialog box titled "Alterar Senha". It contains four text input fields: "Usuário:" with "Administrador" entered, "Inserir senha antiga", "Inserir nova senha", and "Confirmar nova senha". To the right of the last two fields are two buttons: "Confirmar" (highlighted in blue) and "Fechar".

Figura 5.20 - Interface de alteração de senha

Quando o usuário do ambiente de simulação deseja sair do programa, ele vai até o menu Sair na interface principal e clica no submenu sair do programa, em seguida aparece uma janela com a mensagem “Tem certeza que deseja sair do programa?”, e depois é só clicar em sim para sair do programa ou cancelar para continuar no programa. A Figura 5.21 mostra a interface de saída do programa.

A screenshot of a Windows-style dialog box titled "Sair do Programa". It contains a single text input field with the message "Tem certeza que deseja sair do programa?". Below the field are two buttons: "Sim" (highlighted in blue) and "Não".

Figura 5.21 - Interface de saída do ambiente de simulação

5.4.2.2 Entrada de Dados

Após entrar no ambiente de simulação, o usuário acessa o menu Banco de Dados (BD) na interface principal que é responsável por todas as atividades de entrada de dados no sistema. Essas atividades são descritas no fluxograma da Figura 5.22.

Figura 5.22 - Fluxograma da entrada de dados no ambiente de simulação

1. Caso não exista um banco de dados utilizado previamente, o usuário deve clicar no submenu Novo, preencher um nome para o novo banco de dados e salvar no local desejado.
2. Caso exista um banco de dados ou já tenha sido criado o novo, o usuário deve clicar no submenu Conectar para que seja feita a conexão do banco de dados e o ambiente de simulação.
3. Após conectar o BD, o usuário acessa o submenu Peças para verificar se a peça a ser simulada já está cadastrada na base de dados e se não necessita editar seus parâmetros. Caso a resposta seja negativa, o usuário deve cadastrar ou editar a peça e salvar no final.
4. Após conectar o BD, o usuário acessa o submenu *Features* para verificar se a *feature* a ser simulada já está cadastrada na base de dados e se não necessita editar seus parâmetros. Caso a resposta seja negativa, o usuário deve cadastrar ou editar a *feature* e salvar no final.
5. Após conectar o BD, o usuário acessa o submenu Recursos para verificar se o recurso a ser simulado já está cadastrado na base de dados e se não necessita editar seus parâmetros. Caso a resposta seja negativa, o usuário deve cadastrar ou editar o recurso e salvar no final.

5.4.2.3 Modelagem da Simulação

Para realizar a Modelagem o usuário acessa o submenu Modelagem no menu Simulação na interface principal. A modelagem é uma atividade intermediária que organiza os elementos do banco de dados para que a simulação ocorra corretamente.

A primeira etapa da modelagem é a seleção da peça dentre todas as peças do banco de dados. Todas as peças, *features* e recursos fornecidos para modelagem devem ser cadastrados antes da modelagem na base de dados.

Na etapa seguinte, determina-se os parâmetros da simulação que são: a quantidade de peças a serem simuladas, o tempo entre a chegada das peças no sistema e o tempo total de simulação. A próxima etapa é a escolha do primeiro recurso (fabricação ou transporte). Após escolher e inserir o recurso na lista de

simulação, caso seja de fabricação seleciona-se a *feature* a ser fabricada. Um recurso de fabricação pode fabricar mais de uma *feature*, desde que seja novamente inserido na lista de simulação e seja selecionada a nova *feature* a ser fabricada.

Uma vez definido o recurso de fabricação e a *feature*, o usuário verifica a necessidade de inserir um transporte entre os recursos de fabricação de duas *features*. Caso seja necessário, insere o recurso de transporte. Caso não seja necessário, insere um novo recurso de fabricação e seleciona uma nova *feature* ou encerra a modelagem.

As etapas citadas anteriormente se repetiram até finalizar todas as *features* que compõem a peça. Para cada peça a ser simulada, o usuário deve modelar todas as etapas descritas anteriormente. A Figura 5.23 detalha o processo de modelagem.

Figura 5.23 - Funcionamento da modelagem da simulação

A modelagem será realizada a partir do conhecimento prévio pelo usuário do sistema modelado ou baseado no projeto da peça modelada. O sistema fornece flexibilidade na escolha dos seus componentes, porém é necessário que o usuário tenha conhecimento dos diversos componentes, assim como dos parâmetros que os cercam.

5.4.2.4 Simulação

Ao terminar a modelagem o usuário clica no botão “ok” para que inicie a simulação. A partir deste momento o sistema inicia a simulação lógica/matemática e o sistema de visualização da cena 3D que é mostrada para o usuário.

A primeira parte da simulação, simulação lógica/matemática, é realizada de acordo com abordagem proposta e detalhada no capítulo 4. A segunda parte da simulação é a visualização da cena tridimensional do sistema de manufatura modelado. A simulação da cena 3D ocorre ao mesmo tempo em que o programa realiza a simulação lógica/matemática.

A construção da cena 3D inicial com a criação geométrica e a aparência da peça baseada nos dados da base de dados (largura, comprimento e altura). Também baseadas na lista de simulação são criadas as formas geométricas das *features* a serem fabricadas nas peças. Em seguida, baseados na lista de simulação são adicionados os recursos (fabricação e transporte) na cena. Os recursos são desenvolvidos externamente ao ambiente de simulação e são importados no formato OBJ para a cena. A ligação entre a geometria e o tipo de objeto é realizado a partir do código de cadastro do recurso no banco de dados. Por exemplo, todos os tornos têm o mesmo código mostrado na seção 5.4.1 e são representados pela mesma geometria (um objeto no formato OBJ).

A cena é desenhada com a adição da iluminação e da câmera. A iluminação desenvolvida foi uniformemente aplicada a todas as primitivas de cena. A câmera disponibiliza seis posições diferentes de visualização que são: frontal, em cima, embaixo, direita, esquerda e por trás. Também foi implementada uma função para dar o *zoom* no local de interesse do usuário.

Os últimos passos antes da renderização da cena são a adição das animações. O funcionamento da animação da cena inicia quando a peça entra no sistema baseado no tempo entre a chegada de peças. Em seguida, a peça é processada de acordo com o tempo de inserção da *feature* na passagem por um recurso de fabricação e o tempo de transporte da peça entre dois recursos de fabricação. A duração da visualização é definida de acordo com o tempo de simulação ou a quantidade de peças a serem simuladas. A velocidade da animação é definida na interface de visualização tridimensional e pode variar de 0 a 100% do tempo de processamento dos recursos que compõem a cena. A Figura 5.24 apresenta a sequência de atividades para a simulação da cena 3D.

Figura 5.24 - Desenvolvimento da cena 3D

5.4.2.5 Saída de Dados

Ao concluir a etapa de simulação, o sistema automaticamente gera uma tabela com os resultados da simulação. Os resultados apresentados nesta tabela

são os indicadores de desempenho do sistema de manufatura simulado, ou seja, são a saída de dados do ambiente de simulação. Esses resultados podem ser salvos num arquivo de formato XML para uma análise posterior.

Os resultados salvos após a simulação podem ser acessados pelo usuário para uma análise através de gráficos gerados pelo sistema ou para serem exportados no formato CSV. Os dados da tabela de resultados podem ser exportados para uma planilha. Os dados da tabela que podem ser plotados em gráficos são os de tempo e de custo de processamento de cada recurso. Esses indicadores foram escolhidos para uma melhor análise devido mostrarem as relações na fabricação das *features* no sistema de manufatura, que é o propósito principal do ambiente de simulação.

Para gerar e visualizar os gráficos é necessário que o usuário selecione o submenu resultados do menu resultados para determinar os parâmetros no formulário resultados mostrado na Figura 5.16. Eles podem ser apresentados de três formas diferentes que são: barra, pizza e linha. Nas Figuras 5.25, 5.26 e 5.27 são apresentados exemplos de cada um desses gráficos. Cada gráfico pode ser salvo no formato XML para posterior análise. O Apêndice B mostra exemplo de gráficos salvos pelo sistema.

Figura 5.25 - Gráfico de barra do ambiente de simulação

Figura 5.26 - Gráfico de pizza do ambiente de simulação

Figura 5.27 - Gráfico de linha do ambiente de simulação

5.4.3 Modelagem do Ambiente de Simulação

A modelagem é a criação de um conjunto de modelos que representam diversas visões do sistema que está sendo construído, ou seja, a modelagem é a transcrição do modelo que está na mente do responsável pelo desenvolvimento do sistema (modelo abstrato) baseado nos requisitos analisados que é representado de acordo com alguma técnica de representação, a fim de torná-lo um modelo conceitual, de modo que outras pessoas possam entendê-lo.

A modelagem tem por objetivo desenvolver modelos que descrevem estática e dinamicamente o que o sistema deve fazer. Estes modelos expressam os requisitos, possibilitando um maior entendimento do domínio da aplicação, servindo para determinar se a especificação está completa e consistente.

Nesse trabalho como forma de representação do modelo utilizou-se a Modelagem Orientada a Objeto (MOO). Foram definidos os componentes e descritas as variáveis e interações lógicas que constituem o sistema. A representação através de objetos permite uma maior facilidade de visualização do funcionamento e do fluxo de informação deste, por ser mais próxima da ótica humana.

A modelagem orientada a objetos do sistema foi realizada com o programa *Rational Rose* que é uma ferramenta para desenvolvimento de sistemas orientados a objetos. Ela possibilita o desenvolvimento de análise e projetos utilizando metodologias de desenvolvimento, como por exemplo, a UML.

Foi inicialmente desenvolvida pela empresa *Rational Machines* como o nome *Rational Rose* em 1992, porém foi adquirida pela empresa IBM em 2003, se tornando *IBM Rational Architect*. A versão utilizada neste trabalho foi uma versão acadêmica do *Rational Rose*.

Essa ferramenta foi utilizada para a modelagem dos diagramas da UML. Ela foi selecionada devido às seguintes características: facilidade do uso, utilização da metodologia UML e possibilidade de interação com a linguagem C++.

Neste trabalho foi utilizado o método da Linguagem Unificada de Modelagem (UML) para modelagem devido a seu potencial na representação visual do sistema orientado a objeto. Dentre os vários diagramas da UML foram utilizados o diagrama de caso de uso, o diagrama de classe e o diagrama de sequência.

5.4.3.1 Diagrama de Caso de Uso

Os diagramas de caso de uso foram utilizados neste trabalho com o objetivo de produzir uma descrição clara de como o usuário e o administrador do sistema interagem com o sistema e fornecer uma descrição sobre as responsabilidades que devem ser cumpridas.

Foram desenvolvidos dez diagramas de estudo de casos com funções diferentes. Os nomes e funções desses diagramas são mostrados na Tabela 5.3.

Tabela 5.3 - Diagramas de caso de uso do ambiente de simulação

N	Caso de Uso	Descrição
01	Cadastrar	Cadastra o usuário no sistema
02	Validar Login e Senha	Permite o acesso do usuário no sistema
03	Alterar Senha	Realiza a alteração da senha de acesso
04	Cadastrar Peça	Realiza o cadastro da peça no banco de dados
05	Cadastrar Feature	Realiza o cadastro da feature no banco de dados
06	Cadastrar Recurso	Realiza o cadastro do recurso no banco de dados
07	Modelar	Realiza a modelagem da simulação
08	Simular	Realiza a simulação
09	Exibir Resultados	Exibe os resultados da simulação
10	Exportar Resultados	Exporta os resultados da simulação

Na Tabela 5.4 e na Figura 5.28 têm-se um exemplo de diagramas de caso de uso desenvolvido nesse trabalho. Para maiores detalhes todos os diagramas de caso de uso desenvolvidos nesse trabalho encontram-se no Apêndice C.

Tabela 5.4 - Exemplo de diagrama de caso de uso para cadastrar operador do sistema

No	Descrição	Eventos	Caso de Uso	Resposta
01	Administrador solicita cadastro do usuário	DadosUsuário	Cadastrar	msg01

Figura 5.28 - Exemplo de diagrama de caso de uso para cadastrar operador do sistema

Descrição: Este Caso de Uso é responsável em cadastrar um novo usuário do simulador.

Cenário Sucesso Principal:

- (a) O administrador solicita o cadastro do novo usuário na interface.
- (b) O administrador pede para gravar o novo *login* e senha.
- (c) O sistema verifica se o novo *login* e senha não existem e armazena.
- (d) O sistema envia Msg01 "Cadastro gerado com sucesso".

Cenário Alternativo 1:

Passo (b). O administrador pede para fechar.

- (b.1) O sistema aborta a operação sem gravar os dados.

Cenário Alternativo 2:

Passo (c). O sistema avisa ao usuário Msg01 "Dados inconsistentes".

- (c.1) O administrador corrige os dados.
- (c.2) vai para o Curso Normal passo (b).

Cenário Alternativo 2.1:

Passo (c.1) O administrador pede para cancelar.

- (c.1.1) O sistema aborta a operação sem gravar os dados.

5.4.3.2 Diagrama de Classes

O diagrama de classes em um modelo representado através da UML é o responsável pela visão estrutural e representa a estrutura estática de um sistema por se mostrar válido a partir de qualquer ponto do ciclo de vida do sistema.

Uma classe (conjunto de objetos que apresentam as mesmas características e operações identificáveis no sistema) é representada por um retângulo dividido em três partes: superior (utilizada para individualizar este elemento), intermediária (onde constam seus atributos) e inferior (onde constam as operações efetuadas por este elemento).

As classes de um sistema relacionam-se entre si, trocando informações (representadas pelos atributos) e disparando operações. Estas relações são representadas pelas setas, onde é possível identificar a quantidade de correspondências de objetos de uma classe para objetos equivalentes em outra classe.

Neste trabalho o diagrama de classe foi utilizado como base para visualização, especificação e documentação do modelo conceitual. Ele mostrou as classes, interfaces, colaborações e seus relacionamentos.

Para uma melhor organização e visualização o desenvolvimento do diagrama de classes do sistema foi dividido em três pacotes de acordo com a Figura 5.29. Dentro de cada pacote são detalhados seus objetos, atributos e métodos, além da interação entre eles. Conforme Booch *et al.* (2000), pacote é um mecanismo de agrupamento para organizar elementos semanticamente relacionados em grupos. Todos os modelos de elementos ligados ou referenciados por um pacote são chamados de conteúdo do pacote.

Figura 5.29 - Diagrama de pacotes do sistema

O pacote *Core*, coração do sistema, é composto pelas classes lógicas para o funcionamento da simulação. O pacote *GUI* é composta pelas interfaces gráficas com usuário e seus relacionamentos. O pacote *Scene* é composta pelos objetos que formam a cena tridimensional da simulação sistema. Nas Figuras 5.30, 5.31 e 5.32 têm-se os diagramas do conteúdo do pacote *Core*, *GUI* e *Scene* respectivamente.

As principais classe do pacote *Core* são relativas a simulação, banco de dados, peças, *features*, recursos e resultados. No pacote *Gui*, as principais classes são relativas as interfaces principal, de resultados, de cadastro do banco de dados das peças, das *features* e dos recursos, além das interfaces de login, cadastro de usuário, alteração de senha, ajuda e sobre. No pacote *Scene*, as principais classes são relacionadas ao funcionamento da cena como a câmera, iluminação, construir e mostrar as primitivas, localização dos recursos na cena, desenho da cena e definição de comandos de manipulação.

Figura 5.30 - Diagrama de classe do pacote Core

Figura 5.31 - Diagrama de classe do pacote GUI

Figura 5.32 - Diagrama de classe do pacote *Scene*

As classes constantes nos diagramas de classe das Figuras 5.30 e 5.31 foram desenvolvidas neste trabalho. No diagrama da Figura 5.32 foram desenvolvidas neste trabalho as classes *Camera*, *GIWidget*, *DisplayListPrimitive* e *Scene*, e as classes *OpenCsg*, *Glxew*, *Glew*, *Wglew*, *Utility*, *BoundingBox*, *Quat*, *Vec3* foram importadas de bibliotecas pré-existentis.

5.4.3.3 Diagrama de Sequência

Todos os sistemas possuem uma dinâmica representada pela troca de informações entre os subsistemas que o compõem e/ou o ambiente no qual está inserido. Esta dinâmica pode ser representada na UML pelo diagrama de sequência.

O diagrama de sequência foi utilizado neste trabalho no sentido de demonstrar a funcionalidade do sistema através de possíveis situações que

ocorrerão no seu uso. Na Figura 5.33, tem-se o exemplo de diagrama de sequência geral. No Apêndice D são detalhados os diagramas de sequência construídos na modelagem do sistema.

Figura 5.33 - Diagrama de sequência geral do ambiente de simulação

5.5 IMPLEMENTAÇÃO DO AMBIENTE DE SIMULAÇÃO

A implementação do ambiente de simulação foi a tradução dos requisitos e da modelagem num programa computacional. Ela foi dividida em partes principais que tiveram suas particularidades de desenvolvimento, assim como suas ferramentas detalhadas. Ao longo da seção são mostrados pseudos-códigos da implementação e no Apêndice E e no Anexo A são detalhadas todas as classes que compõem a codificação do programa.

Para o desenvolvimento do ambiente de simulação escolheu-se o programa *Qt* por ser um ambiente de desenvolvimento integrado (IDE) que possui interfaces com o sistema de banco de dados *Sqlite* e com a biblioteca gráfica *OpenGL*, sendo este voltado ao desenvolvimento de aplicações com a linguagem de programação C++, possibilitando ainda a utilização em multiplataforma. Outros fatores considerados na escolha foram o limite de tempo de desenvolvimento, os recursos financeiros, a boa documentação, a simplicidade da interface, e os esforços de programação.

Qt foi desenvolvido pela empresa *Trolltech* em 1995, porém foi adquirido pela Nokia em 2008, que atualmente o distribui com licença *open source*. Possui recursos que permitem a integração das aplicações com a plataforma preferida do usuário (Windows®, Linux/X11e MAC® OS X) e possibilita o tratamento modularizado de eventos, através do mecanismo desinais (“*signals*”) e encaixes (“*slots*”). Em particular, um objeto (“*signal*”) do *Qt* pode ser associado a um número grande de funcionalidades (“*slots*”) (BLANCHETTE e SUMMERFIELD, 2008).

A programação do ambiente de simulação foi realizada na linguagem C++. De acordo com Deitel e Deitel (2001), a linguagem C++ foi desenvolvida por *Bjarne Stroustrup* no início dos anos 80 no *Bell Laboratories*, com o objetivo de melhorar a linguagem de programação C. Os principais objetivos ao desenvolver o C++ foram: ser tão eficiente e portátil quanto a linguagem C; suportar múltiplos paradigmas de programação, como, por exemplo, a programação estruturada e a programação orientada a objetos e fornecer ao programador múltiplas escolhas, mesmo que seja possível escolher a opção errada.

A linguagem C++ foi selecionada devido à existência de suporte ao paradigma de orientação a objetos e de algumas características que foram importantes no desenvolvimento do sistema como: a abstração de dados, o encapsulamento, a herança, o polimorfismo e a modularidade.

Outra funcionalidade do C++ utilizada foram as funções e classes *templates*, pois permitem a definição de um modelo de função ou classe que poderá ser utilizada por vários objetos de tipos diferentes. Isso facilitou a implementação da simulação, pois possibilita a utilização de várias estruturas de dados, tais como: filas, listas e vetores, reduzindo a necessidade de codificação da estrutura de cada tipo de objeto a ser armazenado.

5.5.1 Banco de Dados

O *software* utilizado para o armazenamento do banco de dados foi o *Sqlite*. O *Sqlite* permite armazenar os dados da aplicação em tabelas e manipular esses dados através de comandos SQL.

Os dados do banco de dados acessado pelo ambiente de simulação podem ser gerados, alterados e acessados pela interface do usuário do sistema ou por um programa gerenciador de *Sqlite*. Essa característica gera compatibilidade com outros programas, pois os gerenciadores de *Sqlite* importam e exportam em outros formatos, como por exemplo, no formato texto (*txt*).

A principal vantagem que resultou na escolha do *Sqlite* neste trabalho foi ser um sistema multiplataforma que não necessita de instalação ou administração. Essa característica torna o banco de dados portátil. Algumas outras vantagens da utilização do *Sqlite* no ambiente simulação foram:

- Não precisa acessar um Sistema Gerenciador de Banco de Dados;
- É Gratuito, simples e fácil de usar;
- Banco de dados completo é armazenado em apenas um arquivo de sistema;
- Suporta bases de dados de até dois *terabytes* de tamanho;
- Mais rápido do que os populares bancos de dados cliente/servidor para a maioria das aplicações simples.

O banco de dados implementado não permite alterações durante a simulação. A sua função principal é a entrada de dados no sistema possibilitando a simulação. Os principais métodos utilizados para o banco de dados permitem:

- Criação de um novo banco de dados. Esse método é detalhado na Figura 5.34 que consiste na geração de um banco de dados (*Sqlite*) e a inserção das tabelas peça, *features* e recursos, assim como seus atributos;
- Ligação com um banco de dados existente. Esse método é detalhado na Figura 5.35 que permite a conexão de um banco de dados (*Sqlite*) existente com o sistema;
- Salvar e Carregar atributos das peças, *features* e recursos. A Figura 5.36 detalha o métodos relativos ao salvar e carregar os dados das peças da base de dados. As *features* e os recursos segue a mesma lógica das peças.

```

// Método para criar um Banco de Dados caso nao exista.
void DBManager::createNewDB(const std::string &dbName)
{
    QSqlDatabase db = QSqlDatabase::addDatabase("SQLITE");
    db.setDatabaseName(QString::fromStdString(dbName));
    if (db.open())
    {
        // Inserção da Tabela Peça e seus atributos.
        QSqlQuery query(db);
        query.exec("create table Piece(id int primary key, "
            "codigo int,"
            "name text,"
            "material text,"
            "largura float,"
            "comprimento float,"
            "altura float)");
        // Inserção da Tabela Features e seus atributos.
        query.exec("create table Feature(id int primary key,"
            "codigo int,"
            "name text, "
            "type text,"
            "largura float,"
            "comprimento float,"
            "altura float,"
            "diametro float,"
            "diametroMaior float,"
            "diametroMenor float,"
            "posicaoX float,"
            "posicaoY float,"
            "posicaoZ float,"
            "orientacao int)");
        //Inserção da Tabela Recursos e seus atributos.
        query.exec("create table Resource(id int primary key,"
            "codigo int,"
            "name text,"
            "type text, "
            "salarioOp float,"
            "salarioMaq float,"
            "custoFerr float,"
            "diamCorte float,"
            "profCorte float,"
            "compCorte float,"
            "avanco float,"
            "velCorte float,"
            "tempImpr float,"
            "vidaFerr float,"
            "trocaFerr float,"
            "velAvanco float,"
            "taxaAcab float,"
            "velMovi float,"
            "distMovi float,"
            "cusMovi float)");
        if (!query.exec())
            qCritical() << query.lastError();
    } else
    {
        qCritical() << db.lastError();
    }
}

```

Figura 5.34 - Código para criação de um novo banco de dados

```

// Conectar o ambiente de simulação num banco de dados Sqlite existente.
void MainWindow::on_action_Open_triggered()
{
    // Abrir pastas do sistema.
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open Data
Base File"), QDir::currentPath());
    if(!fileName.isNull())
    {
        // Conectar ao banco de dados selecionado.
        QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
        db.setDatabaseName(fileName);
    }
}

```

Figura 5.35 - Código para conectar o sistema num banco de dados existente

```

// Carrega da base de dados os atributos relativos da peça.
Piece* DBManager::loadPiece(const std::string &objName)
{
    //Carrega os atributos do Banco de Dados relativo a Peca a simular.
    QSqlQuery query("SELECT * FROM Piece WHERE name=\"\" +
QString::fromStdString(objName) + "\"");
    if(query.first())
    {
        Properties prop;
        prop.set<string>("name", objName);
        prop.set<float>("largura", query.value(4).toFloat());
        prop.set<float>("comprimento", query.value(5).toFloat());
        prop.set<float>("altura", query.value(6).toFloat());
        return new Piece(prop);
    }
    return 0;
}

// Salvar os atributos da peça na base de dados.
PieceEditor::PieceEditor(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::PieceEditor)
{
    //Os dados e posições na tabela peças do banco de dados.
    ui->setupUi(this);
    mapper = new QDataWidgetMapper(this);
    mapper->setModel(model);
    mapper->setSubmitPolicy(QDataWidgetMapper::ManualSubmit);
    mapper->addMapping(ui->codspinBox, 1);
    mapper->addMapping(ui->nameEdit, 2);
    mapper->addMapping(ui->matEdit, 3);
    mapper->addMapping(ui->largSpinBox, 4);
    mapper->addMapping(ui->compSpinBox, 5);
    mapper->addMapping(ui->altSpinBox, 6);
}

void PieceEditor::on_saveButton_clicked()
{
    //Salvar os dados ao clicar no botão salvar da interface de cadastro do
    banco de dados.
    mapper->submit();
}

```

Figura 5.36 - Código para salvar e carregar da base de dados os atributos da peça

5.5.2 Interfaces

Além de ser a IDE que organizou o programa, o programa Qt também foi utilizado no desenvolvimento da interface gráfica com usuário. Ele é conhecido como um *widget toolkit*, ou seja, um programa usado para o desenvolvimento de aplicações com interface gráfica.

O Qt proporcionou facilidades operacionais com recursos interativos, possuindo suporte interno para vários objetos (*widgets*) da interface com usuário: janelas, ícones, menus, botões, *layouts*, quadro de diálogos, barras de rolamentos, gráficos e tabelas. Ela fornece um painel de controle para o usuário visualizar informações sobre o ambiente e alterar as configurações.

A Figura 5.37 mostra alguns desses *widgets* utilizados no trabalho. A Figura 5.38 mostra uma interface, como exemplo, utilizando esses objetos. Os objetos utilizados na interface resultados são:

- Os botões (*Buttons*) do tipo *Push Button* para carrregar, salvar, limpar dados, dentre outros. Os botões do tipo *Check Box* para selecionar o eixo e o *grid* para mostrar no gráfico;
- O *layout* horizontal para organizar os botões (gráfico tempo e gráfico custo) horizontalmente e *layout* vertical para organizar os botões (carrregar, salvar, limpar dados, dentre outros) verticalmente;
- A Tabela (*Table View*) para mostrar o resultados como o tempo e o custo de processamento/fabricação;
- Os objetos de entrada (*Input Widgets*) como o *Combo Box* para escolher a customizar entre o cabeçalho, a legenda, o eixo X ou o eixo Y. Ou ainda, *widget Font Combo Box* para definir o tipo da fonte e o *Spin Box* para definir o tamanho da fonte do cabeçalho do gráfico. Também foi utilizado o *widget Line Edit* para informar o nome do rótulo do eixo X e Y e as unidades de cada eixo;
- Os objetos de exibição (*Display Widgets*) como o *Label* para inserir os textos em cada seção, como por exemplo, configurar dados, exportar dados, criar gráfico, customizar e mostrar gráfico.

Figura 5.37 - Exemplos de widgets do Qt

Figura 5.38 - Exemplo da interface resultados do ambiente de simulação com seus *widgets*

Além dos *widgets* para o desenvolvimento das formulários das interfaces, o *Qt* possui um conjunto de classes e métodos prontos para utilizar na programação das interfaces. A Figura 5.39 mostra, como exemplo, uma parte do código da interface principal do sistema.

```

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}
MainWindow::~MainWindow()
{
    delete ui;
}
// Carrega o formulário de interface das Feature.
// Para carrega usa um método Sql relacional do Qt.
// Carregamento dos formulários interfaces Peças e Recursos é similar.
void MainWindow::on_actionEditFeature_triggered()
{
    QSqlRelationalTableModel *model = new QSqlRelationalTableModel(this,
QSqlDatabase::database());
    model->setTable("Feature");
    FeatureEditor *editor = new FeatureEditor(model);
    EditorDialog *dialog = new EditorDialog(editor, this);
    dialog->exec();
    delete dialog;
}
// Carrega o formulário da interface Modelagem da Simulação.
void MainWindow::on_action_Configure_triggered()
{
    SimulationDialog *dialog = new SimulationDialog(this);
    if(dialog->exec())
    {
        Simulation* simulation = dialog->getSimulation();
        Report report = simulation->simulate();
        SimulationResults *dialog = new SimulationResults(report, this);
        dialog->exec();
    }
    delete dialog;
}
// Carrega o formulário da interface Ajuda.
// O carregamento do formulário da interface Sobre é similar.
void MainWindow::on_actionHelp_triggered()
{
    AboutDialog *dialog = new AboutDialog(this);
    dialog->exec();
    delete dialog;
}
// Carrega o formulário da interface Resultados.
void MainWindow::on_actionReport_triggered()
{
    Chart *dialog = new Chart(this);
    dialog->exec();
    delete dialog;
}

```

Figura 5.39 - Exemplo código da interface principal

O Qt também permite o uso do mecanismo *signal/slot*. A Figura 5.40 mostra um exemplo do mecanismo *signal/slot* utilizado no trabalho em que, se clicar (*clicked()*) no botão salvar (*savePushButton*), então, aceita (*accept()*) o que foi definido no *CSVExporter* (exportar a planilha de resultados) e ao clicar no botão (*cancelPushButton*) rejeita o que foi definido no *CSVExporter*.

Figura 5.40 - Exemplo da utilização do *signal/slot*

5.5.3 Simulação

A simulação inicia após a modelagem (ao clicar no botão “ok”) com armazenagem dos dados de entrada do sistema (tempo entre chegada das peças, quantidade de peças a ser simulado e tempo total de simulação) e a lista de simulação conforme o código da Figura 5.41.

A simulação conforme mostrado no capítulo 4 foi dividida em quatro partes que são:

- Chegada de peças no sistema;
- Execução da simulação;
- Processamento das filas e;
- Cálculo dos Tempos e dos custos dos Recursos.

A Figura 5.42 detalha o código da chegada das peças no sistema. A Figura 5.43 mostra o código da execução da simulação. A Figura 5.44 apresenta o processamento das filas e por último a Figura 5.45 detalha o código do cálculo do tempo e custo dos recursos de fabricação.

```

// Método acionado quando o usuário clica "ok" na interface de modelagem.
void SimulationDialog::on_buttonBox_accepted()
{
    // Cria uma nova simulação.
    sim = new Simulation();
    // Carrega os dados da peça selecionada e inseri na nova simulação.
    sim->setPiece(DBManager::loadPiece(ui->pieceComboBox
    >currentText().toStdString()));
    //Armezena em sim o Tempo de Chegada das Peças.
    sim->setArrivalTime(ui->arrivalTimeSpinBox->value());
    // Armazena em sim a Quantidade de Peça a ser simulado.
    sim->setQuantity(ui->quantitySpinBox->value());
    // Armazena em sim o Tempo Total Simulação.
    sim->setSimulationTime(ui->SimulationTimespinBox->value());
    // Percorre a lista de simulação.
    for(int i=0; i < ui->resourcesList->count(); ++i)
    {
        // Carrega os dados dos recursos da lista a partir do banco de dados.
        QString resName = ui->resourcesList->item(i)->text();
        ResourceType resType = static_cast<ResourceType>(ui->resourcesList-
        >item(i)->type());
        Resource* resource=DBManager::loadResource(resType,resName.toStdString());
        // Se o recurso for do tipo fabricação inserir uma feature por recurso.
        if(resType == FABRICATION_RESOURCE)
        {
            QList<QListWidgetItem*> features = featuresMap.values(resName);
            for(int j=0; j < features.size(); ++j)
            {
                // Carrega os dados das features do banco de dados.
                Feature *feature = DBManager::loadFeature(static_cast<FeatureType>
                (features[j]->type()), features[j]->text().toStdString() );
                static_cast<FabricationResource*>(resource)->addFeature(feature);
            }
        }
        sim->addResource(resource);
    }
    QDialog::accept();
}

```

Figura 5.41 - Código do planejamento da simulação

```

// Estrutura da chegada de peças.
struct PieceSource : public ResourceNode
{
    // As peças entram no sistema de acordo com tempo de chegada(ArrivalTime).
    PieceSource(float arrivalTime) :
    //Peças armazenadas antes de entrar no sistema.
    ResourceNode(-1, arrivalTime)
    {
    }
}
// Atualiza a fonte de peças a cada passo dt.
virtual bool update(float dt, float* rest)
{
    // Para não iniciar com uma peça no fila.
    if(m_inQueueSize == -1)
        return false;
    m_currentTime += dt;
}

```

```

// São realizados testes entre o tempo atual da simulação e o tempo de
// chegada das peças para determinar se retorna falso ou verdadeiro para
// entrada da peça no sistema.
// Se o tempo atual for maior ou igual ao tempo de chegada da peça
// entra no "if", se não retorna falso.
    if(equal(m_currentTime,arrivalTime) || (m_currentTime>arrivalTime))
    {
// Se tempo atual for maior que o tempo de chegada da peça entra no "if"
// e retorna verdadeiro.
        if(equal(m_currentTime, arrivalTime))
        {
// Quando o sistema é atualizado no passo dt a peça pode ter entrado
// no sistema, para corrigir essa distorção utiliza o rest que é
// diferente de zero no caso do tempo atual ser maior do que o tempo
// de chegada da peça
            *rest = 0.f;
            m_currentTime = 0.f;
        }
        else
        {
// O rest é calculado subtraindo o tempo atual pelo tempo de chegada
// da peça no sistema.
            *rest = m_currentTime - arrivalTime;
            if(m_inQueueSize > 0)
                m_currentTime = *rest;
            else
                m_currentTime = 0.f;
        }
// Retira a peça do total de peças a ser simuladas.
        --m_inQueueSize;
        return true;
    }
    return false;
}
}

```

Figura 5.42 - Código da chegada peças no ambiente de simulação

```

// Realiza a simulacao e envia para classe report resultados da simulacao.
Report Simulation::simulate()
{
    Report report;
    report.totalTime = 0.0;
    report.totalCost = 0.0;
    report.averageTime = 0.0;
    report.averageCost = 0.0;
    report.averageQueue = 0.0;
    report.totalQueue = 0.0;
    float Cost = 0.0;
    float Time = 0.0;
    if(m_resources.empty())
        return report;
// Sao criados vetores para armazenamento de indicadores para cada recurso.
    report.resourceNames = vector<string>(m_resources.size());
    report.resourceCosts = vector<float>(m_resources.size());
    report.resourceTimes = vector<float>(m_resources.size());
    report.resourceQueueTimes = vector<float>(m_resources.size());
    report.utilizationRate = vector<float>(m_resources.size());
}

```

```

//Vetor de nós. Um para cada recurso.
vector<ResourceNode*> resourceNodes;
// Inicia a lista de ResourceNode com a fonte de peças inicializada com o
//tempo de chegada de peças.
resourceNodes.push_back(new PieceSource(m_arrival));
int i = 0;
// Calcula o passo da simulação a partir do tempo entre os recursos.
float dt = numeric_limits<float>::max();
for(list<Resource*>::iterator it = m_resources.begin(); it !=
m_resources.end(); ++it){
    Properties prop = (*it)->getProperties();
    report.resourceNames[i] = prop.get<string>("name");
    float t = (*it)->computeTime(m_piece);
    resourceNodes.push_back(new ResourceNode(i, t));
    dt = min(dt, t);
    ++i;
}
// dt e o menor valor entre o dt encontrado no loop anterior
// e o tempo de chegada de uma nova peça.
// Nao funciona se tempo de chegada das peças no sistema for igual a zero.
dt = min<float>(dt, m_arrival);
//Inicializa o um recurso "0" com todas as peças a serem simuladas.
// As peças ficam em fila e são inseridas no sistema de acordo com o tempo.
resourceNodes[0]->m_inQueueSize = m_quantity;
int finishedPieces = 0;
float rest = 0;
float t = 0;
// A simulação sera encerrada quando termina o tempo de simulação e
// toda a quantidade de peças sejam simuladas.
for(t = dt; t < m_simulationTime || m_quantity >= finishedPieces; t += dt)
{
    // Percorre a lista de recursos começando pelo último.
    for(int r = resourceNodes.size()-1; r >= 0 ; --r){
        // Atualiza o estado do recurso r e entra no if se ele terminar uma peça
        //com essa atualizacao.
        if(resourceNodes[r]->update(dt, &report, &rest))
        {
            //Cria evento usado na animação.
            SimulationEvent e(t, r - 1, r);
            events.push_back(e);
        }
        // Se r nao for o último.
        if(r < resourceNodes.size()-1)
        {
            // Adiciona uma peça na fila do proximo recurso.
            resourceNodes[r+1]->m_inQueueSize ++;
            // Se tiver rest para propagar, atualiza o proximo com esse rest.
            if(rest != 0.f)
            {
                // E realizado um update no rest e no report.
                resourceNodes[r+1]->update(rest, &report, &rest);
            }
            else
            {
                // Incrementa a quantidade de peças simuladas.
                finishedPieces++;
                break;
            }
        }
    }
}
}

```

Figura 5.43 - Código da execução da simulação

```

//Estrutura dos recursos baseado se estão com status ocupados ou livres,
//de acordo com o processamento das peças nas filas.
struct ResourceNode
{
    enum Status{
        READY,
        BUSY
    };
    // Principais variáveis utilizadas no processamento das filas: id recurso
    // tempo processamento, tempo atual, tamanho da fila, status.
    ResourceNode(int id, float timeCost) : m_id(id), m_timeCost(timeCost),
    m_currentTime(0.f), m_inQueueSize(0), m_status(READY){}
    //Avança na simulacao com o passo dt.
    // Retorna true sempre que termina uma peça.
    // Os dados são atualizados em enviados para a classe report
    // que são os resultados da simulacao.
    virtual bool update(float dt, Report* report, float* rest){
        if(m_status == READY){
            if(m_inQueueSize > 0){
                --m_inQueueSize;
                m_status = BUSY;
                m_currentTime = 0.f;
            }
            else
                return false;
        }
        // Incrementa na variavel m_currentTime o passo da simulação.
        // Sempre verifica qual o objeto tem fila e o tempo dessa fila.
        m_currentTime += dt;
        if(equal(m_currentTime, m_timeCost) || (m_currentTime > m_timeCost))
        {
            if(equal(m_currentTime, m_timeCost)){
                *rest = 0.f;
                m_status = READY;
                m_currentTime = 0.f;
                report->resourceQueueTimes[m_id] += m_inQueueSize * dt;
                report->totalQueue += m_inQueueSize * dt;
            }
            else{
                *rest = m_currentTime - m_timeCost;
                if(m_inQueueSize > 0){
                    // Calcula o tempo de fila por recurso de acordo com passo e rest.
                    // Incrementa o tempo total de fila.
                    report->resourceQueueTimes[m_id] += m_inQueueSize * (dt - *rest);
                    report->totalQueue += m_inQueueSize * (dt - *rest);
                    --m_inQueueSize;
                    report->resourceQueueTimes[m_id] += m_inQueueSize * (*rest);
                    report->totalQueue += m_inQueueSize * (*rest);
                    m_currentTime = *rest;
                }
                else{
                    m_currentTime = 0.f;
                    m_status = READY;
                }
                return true;
            }
        }
        return false;
    }
}

```

Figura 5.44 - Código do processamento das filas


```

// Cálculo do Tempo de Fabricação de uma Peça.
float FabricationResource::computeTime(Piece*)
{
    TempoFab =0;
    float TempoTrFerr = 0.0;
    float MRRr = 0.0;
    float MRRp = 0.0;
    float MRRca = 0.0;
    float MRRci = 0.0;
    float MRR = 0.0;
    float TempoCorte = 0.0;
    float TempoImprod = 0.0;
    float TempoAcabamento = 0.0;
    float FeatVol = 0.0;
    float FeatArea = 0.0;
    int quant = 0;
    // Seleciona a feature e calcula o volume e área a ser fabricada.
    for(list<Feature*>::iterator it = m_features.begin(); it !=
m_features.end(); ++it)
    {
        Feature *feature = *it;
        FeatVol = feature->volumeFeature();
        FeatArea = feature->areaFeature();
    }
    // Cálculo do tempo de corte (desbaste).
    MRRr = m_profCorte * m_avanco * m_velCorte;
    MRRp = m_diamCorte * m_profCorte * m_velCorte;
    MRRca = m_compCorte * m_profCorte * m_velCorte;
    MRRci = m_profCorte * m_diamCorte * m_velAvanco;
    MRR =(MRRr + MRRp + MRRca + MRRci);
    TempoCorte = FeatVol / MRR;
    // Cálculo do tempo improdutivo.
    TempoImprod = TempoCorte * (m_temImpr/100);
    // Cálculo do tempo de troca de ferramenta.
    TempoTrFerr = ((TempoCorte/m_vidaFerr)-(1/quant))*m_trocaFerr;
    // Cálculo do tempo de acabamento.
    if (m_taxaAcab != 0)
    {
        TempoAcabamento = FeatArea /m_taxaAcab;
    }
    // Cálculo do tempo fabricação total.
    TempoFab = TempoCorte + TempoAcabamento + TempoImprod +
TempoTrFerr;
    }
    else{
        TempoFab = TempoCorte + TempoImprod + TempoTrFerr;
    }
    }
    return TempoFab;
}

// Cálculo do Custo de Fabricação de uma Peça.
float FabricationResource::computeCost(Piece*)
{
    float CustoFab =0;
    float TempFab =0;
    TempFab = TempoFab;
    CustoFab = ((m_salarioOp/60) *TempFab)+((m_salarioMaq/60)
*TempFab)+ m_custoFerr;
    return CustoFab;
}

```

Figura 5.45 - Código do cálculo do tempo e custo de fabricação

5.5.4 Desenvolvimento da Visualização 3D

A modelagem dos recursos de fabricação e transporte tridimensionais utilizados no ambiente de simulação foi realizada utilizando o programa *Blender* ou *Blender 3D*.

O *Blender* foi selecionado por ser um programa gratuito, com boas funcionalidades, com uma vasta bibliografia composta de tutoriais, manuais, livros e artigos. Porém o principal motivo da escolha foi a possibilidade de exportar os objetos criados no programa em diversas extensões que podem ser utilizadas em outros programas para serem abertos ou integrados em outros programas gráficos por meio de diversas extensões, como por exemplo, a extensão “obj” utilizada no ambiente de simulação.

Cada recurso utilizado no cena tridimensional precisa estar cadastrado no banco de dados com um código que é interligado a uma geometria no formato “obj”. Na Tabela 5.1 foram definidos os códigos para os respectivos recursos. A Figura 5.46 e 5.47 mostram respectivamente um recurso de fabricação (torno) e um recurso de transporte (empilhadeira) utilizado no ambiente de simulação. O Anexo B apresenta todas a geometrias utilizadas pelo sistema. Podem ser ampliadas esses objetos de acordo com a necessidade do usuário.

Figura 5.46 - Exemplo de recurso de fabricação no formato “obj”

Figura 5.47 - Exemplo de recurso de transporte no formato “obj”

A Figura 5.48 detalha o código implementado para importar objetos no formato “obj”. Os objetos importados têm que ter seus vértices e normais definidos. Ao serem importados são inseridos em uma estrutura “*BoundingBox*” que são malhas/ caixas que facilitam a manipulação do objeto.

```
// Ler os objetos importados no formato OBJ.
Mesh* loadObj(string filename) {
    static map<string, Mesh*> loadedMeshes;
    ifstream file(filename.c_str(), ios::in);
    if(!file) {
        cerr << "File \"" << filename << "\" not found." << endl;
        return NULL;
    }
    // Se um obj com nó filename já tiver sido carregado, apenas retorna o ponteiro.
    map<string, Mesh*>::iterator it = loadedMeshes.find(filename);
    if(it != loadedMeshes.end())
        return it->second;
    Mesh* m = new Mesh;
    loadedMeshes.insert(make_pair(filename, m));
    string type;
    while(file >> type) {
        // Importa os Vertices.
        if(type == "v") {
            float x, y, z;
            file >> x >> y >> z;
            m->vertices.push_back(Vec3(x, y, z));
        }
        // Texturas não foram importadas.
        else if(type == "vt")
            {}
    }
}
```

```

// Importa as Normais.
else if(type == "vn"){
    float x, y, z;
    file >> x >> y >> z;
    m->normals.push_back(Vec3(x, y, z));
}else if(type == "f"){
    int vi[3];
    int ni[3];
    int ti[3];
    string token;
    file >> token;
    int count = sscanf(token.c_str(), "%d/%d/%d", &vi[0], &ti[0], &ni[0]);
    if(count == 1){
        count = sscanf(token.c_str(), "%d//%d", &vi[0], &ni[0]);
        if(count == 1)
            cerr << "Objeto sem normais" << endl;
        file >> token;
        count = sscanf(token.c_str(), "%d/%d/%d", &vi[1], &ti[1], &ni[1]);
        if(count == 1){
            count = sscanf(token.c_str(), "%d//%d", &vi[1], &ni[1]);
            if(count == 1)
                cerr << "Objeto sem normais" << endl;
            file >> token;
            count = sscanf(token.c_str(), "%d/%d/%d", &vi[2], &ti[2], &ni[2]);
            if(count == 1){
                count = sscanf(token.c_str(), "%d//%d", &vi[2], &ni[2]);
                if(count == 1)
                    cerr << "Objeto sem normais" << endl;
            }
        }
    }
// Faces com vertices e normais.
    Face face;
    face.vertices[0] = vi[0]-1;
    face.vertices[1] = vi[1]-1;
    face.vertices[2] = vi[2]-1;
    face.normals[0] = ni[0]-1;
    face.normals[1] = ni[1]-1;
    face.normals[2] = ni[2]-1;
    m->faces.push_back(face);
}
//Calcula a boundingbox onde e inserido o objeto importado.
m->boundingbox.pMin = Vec3(99999, 99999999, 9999999);
m->boundingbox.pMax = Vec3(-99999, -99999999, -9999999);
for(unsigned int i = 0; i < m->vertices.size(); i++){
//Pmin da bounding box nos eixos X, Y e Z.
    if(m->vertices[i].x < m->boundingbox.pMin.x)
        m->boundingbox.pMin.x = m->vertices[i].x;
    if(m->vertices[i].y < m->boundingbox.pMin.y)
        m->boundingbox.pMin.y = m->vertices[i].y;
    if(m->vertices[i].z < m->boundingbox.pMin.z)
        m->boundingbox.pMin.z = m->vertices[i].z;
//Pmax da bounding box nos eixos X, Y e Z.
    if(m->vertices[i].x > m->boundingbox.pMax.x)
        m->boundingbox.pMax.x = m->vertices[i].x;
    if(m->vertices[i].y > m->boundingbox.pMax.y)
        m->boundingbox.pMax.y = m->vertices[i].y;
    if(m->vertices[i].z > m->boundingbox.pMax.z)
        m->boundingbox.pMax.z = m->vertices[i].z;
}
m->boundingbox.center = (m->boundingbox.pMax + m->boundingbox.pMin);
return m;
}

```

Figura 5.48 - Código para importar objetos no formato "obj"

Para o desenvolvimento das interseções entre a peça e as *features* foi utilizada a OpenCSG. A OpenCSG é uma biblioteca compacta que utiliza como base imagens CSG (*Constructive Solid Geometry*). CSG é uma técnica de modelagem 3D que consiste em combinar geometrias tridimensionais simples com complexas através de operações booleanas no espaço tridimensional.

A OpenCSG foi selecionada neste trabalho devido à necessidade de mostrar na cena 3D a transformação geométrica da peça com a inserção das *features* ao longo do sistema manufatura. Em cada recurso de fabricação a peça recebe uma nova *feature* e sofre uma alteração geométrica na sua composição que foi modelada a partir dessa ferramenta. Na Figura 5.49 é mostrado diferentes tipos de *features* implementadas inseridas numa peça.

Figura 5.49 - Exemplo de diferentes tipos de *features* inseridas numa peça

As *features* podem ser inseridas nas seis faces da peça a partir da inserção do atributo orientação na base de dados. A Tabela 5.5 detalha as seis faces que compõem a peça e número da orientação de cada face. A posição da *feature* na peça é definida pelos atributos posição X, Y, Z. Esses atributos podem variar de zero a um.

Tabela 5.5 - Código da face para inserção das *features*

Face	Orientação
Face Cima (XZ+)	1
Face Baixo (XZ-)	2
Face Direita (YZ+)	3
Face Esquerda (YZ-)	4
Face Frente (XY+)	5
Face Trás (XY-)	6

Para o desenvolvimento da cena tridimensional foi utilizada a *OpenGL* (*Open Graphics Library*). Ela é uma API (*Application Programming Interface*), ou seja, é uma interface de programação gráfica em 3D de grande utilização no desenvolvimento de aplicações em computação gráfica.

A *OpenGL* é uma sofisticada biblioteca de códigos para gerenciamento e desenvolvimento de aplicações tridimensionais, multiplataforma, portátil e segue a convenção de chamada de bibliotecas da linguagem de programação C/C++ com diferentes linguagens.

A *OpenGL* foi utilizada neste trabalho devido oferecer mecanismos para o desenvolvimento da cena tridimensional em multiplataforma utilizando todos os recursos disponíveis numa biblioteca gráfica 3D como por exemplo: tipos diferentes de projeção, navegação, iluminação e criação de efeitos, transparência e/ou mistura de cores, além da animação dos objetos definidos pelo programador.

Algumas características da implementação do ambiente de simulação são:

- A iluminação da cena foi definida a partir de uma luz difusa vermelha e duas luzes posicionais;
- O plano de fundo da cena foi definido como sendo de cor degradê;
- A *zoom* na cena é realizada por meio do *trackball* inserido no *mouse*. Girando o *trackball* é realizada a aproximação e o afastamento na cena. Quando pressionado permite navegar na cena com funções como mover e girar a cena com o acompanhamento da câmera;
- A câmera foi implementada com projeção perspectiva. Foi definido o ângulo de visualização (parâmetro *fovy*) de 60 graus na direção y com uma distância do observador até o plano frontal de 0,05 e a distância do observador até o plano de corte traseiro de 20.000;

- O usuário tem duas opções de visualização tridimensional: a primeira é visualização da cena como um todo e a segunda é visualização somente da peça com as *features*. Para alternar entre os dois tipos de visualização basta clicar com o botão direito do *mouse* em cima da interface de visualização que surgirá o menu conforme mostrado na Figura 5.50.

Figura 5.50 - Opções de visualização da cena

Alguns exemplos da utilização da *OpenGL* neste trabalho são detalhados na Figura 5.51 que mostra a codificação para construção da geometria de uma peça para a cena. A Figura 5.52 o mostra o código responsável pela visualização da cena completa com a animação ou da peça com as *features*.

```
//Construir uma peça no ambiente de simulação
int makePiece(float width, float length, float height)
{
    //Considerando x = largura, y = comprimento, z = altura.
    int id = glGenLists(1);
    glNewList(id, GL_COMPILE);
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glPushMatrix();

    // Define que é uma quadrilátero.
    glBegin(GL_QUADS);
```

```
//Definição dos vertices e das normais da peça.
```

```
    glNormal3f(-1, 0, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 0, height);
    glVertex3f(0, length, height);
    glVertex3f(0, length, 0);
    glNormal3f(0, 1, 0);
    glVertex3f(0, length, 0);
    glVertex3f(0, length, height);
    glVertex3f(width, length, height);
    glVertex3f(width, length, 0);
    glNormal3f(1, 0, 0);
    glVertex3f(width, length, 0);
    glVertex3f(width, length, height);
    glVertex3f(width, 0, height);
    glVertex3f(width, 0, 0);
    glNormal3f(0, -1, 0);
    glVertex3f(width, 0, 0);
    glVertex3f(width, 0, height);
    glVertex3f(0, 0, height);
    glVertex3f(0, 0, 0);
    glNormal3f(0, 0, 1);
    glVertex3f(0, 0, height);
    glVertex3f(width, 0, height);
    glVertex3f(width, length, height);
    glVertex3f(0, length, height);
    glNormal3f(0, 0, -1);
    glVertex3f(0, length, 0);
    glVertex3f(width, length, 0);
    glVertex3f(width, 0, 0);
    glVertex3f(0, 0, 0);
    glEnd();
    glPopMatrix();
    glPopAttrib();
    glEndList();

    return id;
}
```

Figura 5.51 - Exemplo da codificação da geometria de uma peça

```
//Visualização da cena ou da peça com as features
```

```
void Scene::draw()
{
//Define o parametro de projeção.
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
// Define visualizar o modo animação.
    if(mode == ANIMATION)
    {
// Determina o frame da animação baseado no tempo de simulação
// multiplicado pelo fator de animação definido na interface.
        while(currFrame == (int)(events[currEvent].time*factor))
        {
            if(events[currEvent].from > -1)
                objs[events[currEvent].from]->removePiece();
            if(events[currEvent].dest < objs.size())
                objs[events[currEvent].dest]->addPiece();
            currEvent++;
        }
    }
}
```



```

// Para iniciar a animação clicar em iniciar (play) na interface.
for(unsigned int i = 0; i < objs.size(); i++)
{
    objs[i]->play();
}
if(playing)
    currFrame++;
}
// Define visualizar o modo peça com as features.
else if(mode == FEATURES)
{
    // A peça será mostrada na cor vermelha.
    glColor3f(0.7, 0.7, 0.7);
    render(m_primitives);
    glDepthFunc(GL_EQUAL);
    // Inserir as features na peça.
    for(vector<Primitive*>::iterator iter = m_primitives.begin(); iter
!= m_primitives.end(); iter++)
    {
        (*iter)->render();
    }
    glDepthFunc(GL_LESS);
}
glPopMatrix();
}

```

Figura 5.52 - Exemplo da codificação da visualização de uma cena ou da peça com as *features*

5.6 DOCUMENTAÇÃO

A documentação do sistema foi realizada de duas formas: no próprio ambiente de simulação (o programa) e em um relatório de progresso.

Ao longo do desenvolvimento deste trabalho foi documentada cada versão do ambiente de simulação de acordo com o seu desenvolvimento e no próprio ambiente de simulação comentou-se todas as atividades para permitir o rápido e claro entendimento por outros usuários não familiarizados com o programa.

Simultaneamente ao desenvolvimento do ambiente foi criado um relatório com a descrição das atividades, fluxos, diagramas e a lógica do ambiente de simulação.

A documentação do ambiente de simulação é realizada com dois objetivos principais que são:

- Primeiro para servir como um guia para que alguém, familiarizado ou não com o simulador e com os experimentos realizados, possa fazer uso do mesmo e dos resultados já produzidos;

- Segundo, porque se forem necessárias futuras modificações no ambiente de simulação, toda a documentação existente vem facilitar bastante os novos trabalhos.

Devido à facilidade de modificação do sistema, por pessoas alheias ao processo, que podem involuntariamente (ou não) alterar a configuração do simulador, instalou-se *login* e senha de acesso ao ambiente e senha de acesso ao arquivo fonte do programa.

6. VERIFICAÇÃO E VALIDAÇÃO

Este capítulo aborda o verificação e validação do ambiente de simulação. Inicialmente foi redigida uma introdução para uma melhor definição do significado do verificação, validação e teste de *software*, seguida de uma apresentação de como desenvolver os testes. Após, foram planejados e executados três casos de testes e a última parte mostra a análise dos resultados obtidos na aplicação do ambiente de simulação a partir dos casos de teste.

6.1 INTRODUÇÃO

O processo de desenvolvimento de *software* envolve uma série de atividades nas quais, apesar das técnicas, métodos e ferramentas empregadas, erros no programa final podem ocorrer. A verificação e validação (V & V) de *software* é a última etapa do processo de modelo de *software* antes do programa ser entregue ao cliente final. Essa etapa é realizada, em muitos casos, em paralelo com a implementação, onde são inspecionados e testados os códigos desenvolvidos.

De acordo com a literatura estudada (seção 3.3) a V & V busca mostrar que o sistema desenvolvido atende as especificações do cliente. Para isso são utilizadas duas técnicas para garantir essa V & V: inspeções e testes de *software*.

As inspeções no trabalho em estudo consistiu na verificação ao longo do desenvolvimento de erros. A técnica do *debugging* (depuração) foi utilizada para encontrar eventuais erros e corrigi-los.

A segunda técnica, o teste de *software*, foi utilizada ao longo da implementação do trabalho e no final com o desenvolvimento de casos de teste. Os principais propósitos da utilização de testes no ambiente de simulação foram:

- Analisar a funcionalidade do sistema: observar se o sistema atende as necessidades do usuário para a realização de suas tarefas;
- Avaliar a interface para o usuário: observa-se aspectos como a facilidade de aprendizagem, a usabilidade e a relação entre interface tridimensional e a

GUI. A usabilidade pode ser entendida como a qualidade de uso da interface e descreve o esforço necessário para se utilizar o *software*;

- Identificar e especificar os problemas com o sistema: observar possíveis erros, falhas ou resultados errados.

6.2 DESENVOLVIMENTO DE TESTE

De acordo com a literatura estudada, os testes do ambiente de simulação foram realizados segundo as seguintes etapas, fases e técnicas:

- As etapas desenvolvidas foram: planejamento dos testes, execução dos testes e avaliação dos resultados. Na fase de planejamento foram organizados casos de testes que contemplam características diferentes e importantes para o bom funcionamento do sistema. Após, foi realizada a execução de cada caso de teste de acordo com o planejamento. A última etapa foi analisar os resultados gerados pelos casos de teste;
- Os testes foram realizados em três fases: teste de unidade, teste de componentes e teste do sistema. O teste de unidade consistiu em analisar cada classe do programa observando seus métodos e atributos. Foram revisadas as declarações e implementações de cada classe, assim como as estruturas, a herança, a definição de variáveis locais e globais e a utilização de bibliotecas padrão. Na fase de componentes foi realizada uma busca por erros na interface entre os três grandes pacotes do sistema: *Core*, *Gui*, *Scene*. Essa interligação foi importante para a qualidade do sistema. Foi desenvolvido um caso de teste para verificar o funcionamento integrado das principais interfaces do sistema. O último teste realizado foi o do sistema como um todo. Para essa fase foram construídos casos de teste detalhando possíveis situações de erro no programa;
- Em relação às técnicas de teste foram realizados testes funcional e estrutural. O teste funcional consistiu na análise da entrada de dados, processamento e saída de dados. Foi construído um caso de teste em que foram projetados três cenários com alteração das entradas de dados e visualização das mudanças na saída de dados após o processamento no sistema. O teste

estrutural consistiu na criação de dois casos de teste para detectar erros na parte lógica/estrutural do programa.

6.3 PLANEJAMENTO E EXECUÇÃO DE CASOS DE TESTE

Nesse tópico foram definidos casos de testes a serem simulados no ambiente de simulação. A familiarização com o ambiente de simulação e o entendimento da influência das variáveis no comportamento do sistema foi alcançada através da realização dos casos de teste.

Para permitir uma boa qualidade da análise dos resultados, cada caso de teste foi documentado. Os dados de entrada e de saída foram organizados em forma de tabelas e armazenados para a construção de um histórico de dados.

Foram desenvolvidos três casos de testes para uma melhor análise do ambiente de simulação. Cada teste tem suas particularidades nos aspectos composição e funcionalidades a serem testados.

6.3.1 Caso de Teste 1

O primeiro caso de teste foi desenvolvido com o intuito de testar a integração entre os pacotes que compõem o sistema que são: *Core*, *Gui*, *Scene*. O pacote *Core* contempla todas as classes de implementação da lógica do sistema. O pacote *GUI* é composto pelas classes das interfaces gráficas com o usuário. O pacote *Scene* é composto pelas classes que geram a cena tridimensional da simulação do sistema de manufatura.

O desenvolvimento do programa foi realizado de forma incremental, ou seja, cada pacote foi desenvolvido e testado individualmente e depois integrado com os demais. Essa prática incremental possibilitou identificar e corrigir erros ao longo do desenvolvimento de cada pacote.

Esse caso de teste foi dividido em cinco etapas que são:

- *Cadastro de usuário*: a primeira etapa foi testar a integração entre as classes lógicas do cadastro e alteração da senha de usuário com a interface gráfica. O teste consistiu em: o administrador cria um novo *login* e senha para o usuário e verifica se a classe responsável pela criação da tabela e armazenamento dos dados estava correta. Outra verificação realizada foi a alteração da senha padrão para uma nova senha e o armazenamento desses dados. Os testes foram realizados a partir da interface gráfica com o usuário;
- *Banco de dados*: nesta etapa foram testadas a integração entre as classes lógicas do banco de dados e a interface gráfica com o usuário. O primeiro teste foi criar um novo banco de dados do tipo *Sqlite* a partir da interface. Em seguida, foi testada a conexão do sistema via interface com um banco de dados existente. Os últimos testes foram relacionados ao cadastro dos elementos na base de dados. Foram testadas as interfaces peças, *features* e recursos nas funções de inserção, alteração e remoção de dados. Em todas as fases desse teste da base dados foi utilizado o programa *Sqlite Database Browser* para a validação. Tal programa foi utilizado quando, por exemplo, durante a geração de um novo banco de dados pelo ambiente de simulação, o arquivo do banco de dados foi aberto pelo *Sqlite Database Browser* para a validação dos seus atributos e relacionamentos corretamente;
- *Modelagem*: nesta etapa foi testada a integração da interface de modelagem da simulação com a base de dados e com a classe lógica da simulação. Os dados que compõem a interface de modelagem são divididos em dados provenientes da base de dados (peça, recursos e *features*) e dados fornecidos na própria interface (tempo entre a chegada de peças, quantidade de peças a serem produzidas e tempo total de simulação). Todos esses dados da interface são utilizados pela classe lógica de simulação do sistema. O teste consistiu em verificar se os dados referentes à base de dados estavam acessíveis na interface de modelagem, se a lista de simulação proveniente desses dados estava sendo construída corretamente e se esses dados estavam sendo lidos sem erros pela classe simulação;
- *Visualização da cena 3D*: nesta etapa foram testados os relacionamentos das classes do pacote lógico da simulação com as classes do pacote da cena 3D. A cena 3D recebe do pacote de simulação as seguintes informações: da base de dados, informações para a composição geométrica das peças e das

features; da simulação, a lista de simulação com a sequência do processo de simulação, os tempos de processamento nos recursos de fabricação e transporte, os tempos de espera nas filas, o tempo entre a chegada de peças no sistema, a quantidade de peças a serem simuladas e o tempo total de simulação. O teste consistiu em verificar se os dados estavam sendo transmitidos para o bom funcionamento da cena 3D;

- *Resultados*: nesta etapa foi testado o relacionamento entre a classe resultados da lógica do sistema e a interface gráfica com usuário resultados. Ao longo da simulação os indicadores de desempenho do sistema são armazenados numa classe lógica chamada resultados. No final da simulação esses resultados podem ser salvos ou não. Os resultados são salvos num arquivo no formato *XML*. A interface gráfica com usuário resultados obtém seus dados a partir do arquivo salvo anteriormente. O teste consistiu em verificar a consistência dos dados armazenados no arquivo XML e a sua integridade no momento de mostrar na interface gráfica. Também foram testados o funcionamento dos gráficos dos resultados.

6.3.2 Caso de Teste 2

O segundo caso de teste foi desenvolvido com o objetivo de verificar erros nas transações entre os objetos e nos cálculos que compõem o ambiente de simulação. Para isso foi realizada a simulação de um sistema de manufatura composto por dois recursos de fabricação e um recurso de transporte seguindo uma sequência de produção de acordo com a Figura 6.1. Cada recurso de fabricação produz uma *feature* e o recurso de transporte movimenta as peças entre os recursos de fabricação. Os objetos participantes desse caso de teste são detalhados na Tabela 6.1.

Tabela 6.1 - Objetos do caso de teste 2

Peça	Tipos de <i>Feature</i>	Recurso Fabricação	Recurso Transporte
Platô	Rasgo e Furo	Torno e Furadeira	Esteira

Figura 6.1 - Sequência de produção do caso de teste 2

Os parâmetros de entrada definidos na modelagem da simulação, a peça e as *features* são mostrados nas Tabelas 6.2, 6.3 e 6.4. Os dados relativos aos recursos de fabricação e transporte são detalhados na Tabela 6.5.

Tabela 6.2 - Parâmetros de entrada do caso de teste 2

Tempo Entre Chegada de Peças(min)	Quantidade de Peças Simuladas (unidades)	Tempo Total de Simulação (min)
5,50	100	850

Tabela 6.3 - Parâmetros da peça do caso de teste 2

PEÇA				
Descrição	Material	Comprimento (mm)	Largura (mm)	Altura (mm)
Platô	Aço ABNT 1040	170,00	100,00	60,00

Tabela 6.4 - Parâmetros das *features* do caso de teste 2

FEATURES						
Rasgo						
Comprimento (mm)	Largura (mm)	Altura (mm)	Pos. X	Pos. Y	Pos. Z	Orientação
100,00	60,00	20,00	0,30	0,30	0,0	5 (Face XY)
Furo						
Diâmetro (mm)	Altura (mm)		Pos. X	Pos. Y	Pos. Z	Orientação
20,00	50,00		0,20	0,50	0,0	5 (Face XY)

Tabela 6.5 - Parâmetros dos recursos do caso de teste 2

FABRICAÇÃO			TRANSPORTE	
Parâmetros	Torno	Furadeira	Parâmetros	Esteira
Salário Operador (R\$/hora)	8,00	8,00	Velocidade (m/min)	20,00
Salário Máquina (R\$/hora)	25,00	20,00	Distância (m)	4,00
Custo Ferramenta (R\$/mm ³)	0,08	0,10	Per.Tempo Improdutivo (%)	20,00
Diâmetro Corte (mm)	0,00	20,00	Custo Unidade por Tempo (R\$/min)	0,55
Profundidade Corte (mm)	20,00	50,00		
Comprimento Corte (mm)	100,00	0,00		
Velocidade de Corte (mm/min)	15,00	0,00		
Velocidade de Avanço (mm/min)	0,00	10,00		
Per.Tempo Improdutivo (%)	30,00	25,00		
Tempo Vida Ferramenta (min)	5,00	7,00		
Tempo Troca Ferramenta (min)	0,80	1,40		
Taxa Acabamento (mm ² /min)	8.000,00	2.200,00		

A execução do caso de teste foi dividida em duas etapas:

- A primeira etapa consistiu na verificação dos erros na simulação lógico/matemática do sistema. Ao longo da simulação foram analisados os cálculos dos tempos e dos custos de processamento (fabricação e transporte) em cada recurso, dos tempos de espera em fila e dos indicadores de desempenho do sistema. Para realização desses cálculos foram também verificados possíveis erros nos cálculos dos volumes e das áreas das *features*;
- A segunda etapa consistiu na verificação de erros na visualização e funcionamento da cena 3D. A cena é composta das peças, das *features* e dos recursos que tiveram seus parâmetros de entrada previamente determinados.

6.3.3 Caso de Teste 3

O terceiro caso de teste foi desenvolvido para verificar erros lógicos comparando os dados de entrada com os dados de saída do sistema. Para realização desse caso de teste foram construídos três cenários similares, porém com alteração da organização do *layout* para visualização das mudanças na saída de dados após o processamento no sistema. Em seguida, foi construído um quarto cenário para comparar mudanças a partir da alteração do tempo de chegada de peças no sistema.

A diferença entre o primeiro, o segundo e o terceiro cenários é mostrada na Figura 6.2. A composição da sequência dos recursos de produção é alterada mantendo o restante dos dados similares, para a verificação da variação nos indicadores de saída do sistema.

Os três cenários foram compostos por três recursos de fabricação e dois recursos de transporte. Cada recurso de fabricação produz uma *feature* e o recurso de transporte movimenta as peças entre os recursos de fabricação.

Figura 6.2 - Sequência de produção do caso de teste 3

Os objetos participantes do caso de teste são detalhados na Tabela 6.6. e na Figura 6.3 mostra o modelo base do primeiro cenário.

Figura 6.3 - Modelo do primeiro cenário caso de teste 3

Tabela 6.6 - Objetos do caso de teste 3

Peça	Tipos de <i>Feature</i>	Recurso Fabricação	Recurso Transporte
Barra	Degrau, Entalhe e Furo	Fresa, Torno e Furadeira	Esteira e Operário

Os parâmetros de entrada, da peça, das *features* e dos recursos a serem simulados são mostrados nas Tabelas 6.7, 6.8, 6.9 e 6.10.

Tabela 6.7 - Parâmetros de entrada do caso de teste 3

Tempo Entre Chegada de Peças (min)	Tempo Total de Simulação (min)
6,0	730

Tabela 6.8 - Parâmetros da peça do caso de teste 3

PEÇA				
Descrição	Material	Comprimento (mm)	Largura (mm)	Altura(mm)
Barra	Ferro	350,00	160,00	90,00

Tabela 6.9 - Parâmetros das *features* do caso de teste 3

FEATURES						
Degrau						
Comprimento (mm)	Largura (mm)	Altura (mm)	Pos. X	Pos. Y	Pos. Z	Orientação
80	160	20	0,16	0,0	0,4	1 (Face XZ)
Entalhe						
Comprimento (mm)	Largura (mm)	Altura (mm)	Pos. X	Pos. Y	Pos. Z	Orientação
100	60	40	0,30	0,0	0,25	1 (Face XZ)
Furo						
Diâmetro(mm)	Altura (mm)		Pos. X	Pos. Y	Pos. Z	Orientação
25	90		0,80	0,0	0,60	1 (Face XZ)

Tabela 6.10 - Parâmetros dos recursos do caso de teste 3

FABRICAÇÃO				TRANSPORTE		
Parâmetros	Fresa	Torno	Furadeira	Parâmetros	Esteira	Operário
Salário Operador (R\$/hora)	4,00	4,00	4,00	Velocidade (m/min)	15,00	50,00
Salário Máquina (R\$/hora)	10,00	9,00	6,00	Distância (m)	3,00	5,00
Custo Ferramenta (R\$/mm ³)	0,60	0,70	0,30	Per.Tempo Improdutivo (%)	2,00	20,00
Diâmetro Corte (mm)	120,00	0,00	25,00	Custo Unidade por Tempo (R\$/min)	0,20	0,08
Profundidade Corte (mm)	20,00	40,00	90,00			
Comprimento Corte (mm)	0,00	100,00	0,00			
Velocidade de Corte (mm/min)	30,00	28,00	0,00			
Velocidade de Avanço (mm/min)	0,00	0,00	7,00			
Per.Tempo Improdutivo (%)	12,00	24,00	17,00			
Tempo Vida Ferramenta (min)	15,00	8,00	20,00			
Tempo Troca Ferramenta (min)	0,80	1,10	0,90			
Taxa Acabamento (mm ² /min)	5.000	10.000	0,00			

6.4 AVALIAÇÃO DOS RESULTADOS

O objetivo deste tópico foi traçar inferências sobre os resultados obtidos na execução dos diferentes casos de testes executados. A análise de resultados foi uma etapa crítica do desenvolvimento do sistema, pois para sua realização foi

necessário um entendimento aprofundado dos possíveis erros nos testes e seus significados para o sistema.

A análise de resultados teve início com a interpretação dos resultados, seguida pela fase de identificação de oportunidades de resolução do erro e, por fim, o registro e a documentação das soluções.

6.4.1 Caso de Teste 1

O primeiro caso de teste foi executado após uma divisão em etapas. A análise dos resultados também foi realizada para cada etapa da execução da seguinte forma:

1. Cadastro de usuário

Os testes foram realizados sem erros. Foram testados o acesso do usuário ao sistema, o cadastro de um novo usuário e a alteração da senha do usuário.

2. Banco de dados

Os testes foram realizados sem erros lógicos. Foram testados a criação de um novo banco de dados, a conexão de um banco existente, a inserção, a alteração e a remoção de dados da base de dados.

Nesta etapa foram observadas e corrigidas duas falhas na utilização do sistema que foram:

- A primeira falha foi no momento de salvar o novo banco de dados. O sistema permitiu salvar o banco em qualquer local do computador gerando a possibilidade de perda do banco por parte do usuário numa utilização posterior. Para que não haja essa falha o sistema foi corrigido para salvar o banco de dados somente no diretório onde se localiza o ambiente de simulação. Também foi determinado que a conexão deve ser realizada

preferencialmente com o arquivo de banco de dados presente no diretório do programa;

- A segunda falha detectada durante o teste foi a não definição na interface da quantidade de casas decimais após a vírgula em números fracionados. Essa falha pode acarretar em variações nos cálculos matemáticos do sistema. Essa falha foi corrigida com a determinação pela interface de aceitar somente dois dígitos após a vírgula.

3. Modelagem

Nesta etapa foi detectado um erro na formação da lista de simulação. O erro ocorreu na seleção de um recurso de fabricação responsável por fabricar duas *features* em sequência. Caso a lista de simulação fosse construída conforme estava o sistema (Figura 6.4) ocorria um erro de sobreposição da *feature* na lista. Para solucionar o erro, a lista de simulação foi construída conforme a Figura 6.5, inserindo o recurso de fabricação duas vezes com *features* diferentes. Essa correção evita a sobreposição da *feature* nos cálculos matemáticos e permite a visualização da cena tridimensional numa sequência onde cada recurso de fabricação produz uma *feature*.

Figura 6.4 - Erro na modelagem da lista de simulação

Figura 6.5 - Correção na modelagem da lista de simulação

4. Visualização da cena 3D

Os testes foram realizados sem erros. Todos os dados da interface trocados entre as classes da lógica da simulação e as classes da cena tridimensional foram transmitidos com sucesso.

5. Resultados

Nesta etapa foi detectado um erro no momento de salvar os resultados da simulação no formato XML. Quando o arquivo com os resultados no formato XML foi salvo após a simulação, ele indicou duplicidade em alguns indicadores que só precisam de uma linha para serem apresentados. O erro é mostrado no exemplo da Figura 6.6. Cada linha da tabela (recurso) é representada pela *ModelDataItem*.

Os indicadores referentes aos recursos são mostrados em todas as cinco linhas do exemplo, porém os indicadores gerais do sistema (em negrito) só deveriam aparecer na primeira linha. A Figura 6.7 detalha um exemplo do arquivo XML após a correção do erro.

```

<ModelRoot>
<ModelDataItem Nome="Fresa" Tempo Proc.="704.38" Custo Proc.="224.35" Tempo
Fila="85.19" Util.Recurso="96.49" Quantidade="100" Tempo M. Proc="13.62" Custo M.
Proc.="4.59" Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>

<ModelDataItem Nome="Esteira" Tempo Proc.="20.40" Custo Proc.="4.08" Tempo Fila="0"
Util.Recurso="2.79" Quantidade="100" Tempo M. Proc="13.62" Custo M. Proc.="4.59"
Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>

<ModelDataItem Nome="Torno" Tempo Proc.="502.07" Custo Proc.="178.78" Tempo Fila="0"
Util.Recurso="68.77" Quantidade="100" Tempo M. Proc="13.62" Custo M. Proc.="4.59"
Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>

<ModelDataItem Nome="Operario" Tempo Proc.="9.6" Custo Proc.="0.76" Tempo Fila="0"
Util.Recurso="1.315" Quantidade="100" Tempo M. Proc="13.62" Custo M. Proc.="4.59"
Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>

<ModelDataItem Nome="Furadeira" Tempo Proc.="126.33" Custo Proc.="51.05" Tempo Fila="0"
Util.Recurso="17.30" Quantidade="100" Tempo M. Proc="13.62" Custo M. Proc.="4.59"
Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>
</ModelRoot>

```

Figura 6.6 - Arquivo XML resultados com erro

```

<ModelRoot>
<ModelDataItem Nome="Fresa" Tempo Proc.="704.38" Custo Proc.="224.35" Tempo
Fila="85.19" Util.Recurso="96.49" Quantidade="100" Tempo M. Proc="13.62" Custo M.
Proc.="4.59" Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>

<ModelDataItem Nome="Esteira" Tempo Proc.="20.40" Custo Proc.="4.08" Tempo Fila="0"
Util.Recurso="2.79"/>

<ModelDataItem Nome="Torno" Tempo Proc.="502.07" Custo Proc.="178.78" Tempo Fila="0"
Util.Recurso="68.77"/>

<ModelDataItem Nome="Operario" Tempo Proc.="9.6" Custo Proc.="0.76" Tempo Fila="0"
Util.Recurso="1.315"/>

<ModelDataItem Nome="Furadeira" Tempo Proc.="126.33" Custo Proc.="51.05" Tempo Fila="0"
Util.Recurso="17.30"/>
</ModelRoot>

```

Figura 6.7 - Arquivo XML com resultados corretos

6.4.2 Caso de Teste 2

O segundo caso de teste teve como objetivo verificar erros na simulação lógico/matemática do sistema e na visualização e no funcionamento da cena tridimensional.

Inicialmente foram realizados testes nos cálculos dos tempos e dos custos de processamento em cada recurso, nos cálculos dos volumes e das áreas das *features*, nos cálculos dos tempos de espera em fila e nos cálculos dos indicadores de desempenho do sistema. Não foram encontrados erros nas verificações realizadas e os resultados da simulação apresentados nas Tabelas 6.11 e 6.12 se mostraram condizentes com os dados de entrada do sistema.

Tabela 6.11 - Resultados da simulação por recurso do caso de teste 2

Nome	Torno / Rasgo	Esteira	Furadeira / Furo
Tempo Processamento (min)	788,20	24,00	369,15
Custo Processamento (R\$)	441,51	13,20	182,27
Tempo de Fila (min)	301,15	0,00	0,00
Taxa Utilização do Recurso (%)	92,72	3,03	43,42

Tabela 6.12 - Resultados da simulação gerais do caso de teste 2

Tempo Médio Processamento (min)	Custo Médio Processamento (R\$)	Tempo Médio Fila (min)	Tempo Total Produção (min)	Tempo Médio Total (min)
11,81	6,36	3,01	1.482,50	14,82

A segunda etapa verificou a formação e o funcionamento da cena 3D. Foram detectados dois erros. O primeiro foi na inserção da *feature* do tipo circular (furo) na peça. A inserção da *feature* na peça é baseada nas posições X, Y, Z e na orientação da face da peça. A falha ocorreu no momento de determinar o ponto de referência na *feature* da posição zero (X=0, Y=0, Z=0). No caso da *feature* do tipo cavidade (rasgo) o ponto de referência é no ponto inferior esquerdo no início da peça. Para solucionar essa falha convencionou-se que a *feature* furo teria como ponto de referência o seu centro. Essa solução possibilitou determinar a posição correta do

furo na peça. A Figura 6.8 detalha a peça com as *features* rasgo e furo simuladas nesse caso de teste.

Figura 6.8 - Peça com as *features* do caso de teste 2

O segundo erro foi na escala dos objetos (recursos) importados no formato OBJ. No momento de adição dos objetos na cena eles tiveram seus tamanhos diferentes entre si formando uma cena com grandes distorções de escala. Para resolver o problema cada objeto do ambiente de simulação foi trabalhado antes da importação para o sistema. Essa padronização de tamanho foi realizada no *software* Blender, onde foram desenhados os recursos de fabricação e o transporte que compõem o ambiente de simulação. A Figura 6.9 apresenta a cena 3D simulada nesse caso de teste.

Figura 6.9 - Cena 3D simulada no caso de teste 2

Para o bom funcionamento da cena 3D do ambiente de simulação foi analisado também o *hardware* a ser utilizado. Quando se desenvolve um simulador com recursos gráficos, três componentes de *hardware* são importantes para o bom funcionamento do sistema que são: o processador, a memória e a placa de vídeo.

No sistema deste trabalho o processador e a memória foram testados em três computadores com *hardware* diferentes (1. Processador *Intel* Celeron 723 1.3 Ghz e memória 2gb Ram 2. Processador *Intel* Core2duo t8100 2.10Ghz e memória 2gb Ram, 3. Processador *Intel* Core i5-2300 2.8Ghz e memória 4gb Ram) e não houve nenhuma restrição ou limitação. Porém durante a execução da cena tridimensional a placa gráfica foi um fator limitante. Foi necessário utilizar uma placa de no mínimo 128mb de memória dedicada para conseguir renderizar a cena sem travamento e/ou lentidão. Essa limitação foi imposta pela complexidade da cena e pela nitidez imposta pelo sistema para uma boa visualização da cena.

6.4.3 Caso de Teste 3

O terceiro caso de teste teve como objetivo verificar erros comparando os dados de entrada com os dados de saída do sistema. As Tabelas 6.13 e 6.14 mostram comparativos entre os resultados da simulação dos três cenários propostos.

Tabela 6.13 - Comparativo dos resultados da simulação por recurso dos cenários do caso de teste 3

Recurso / Feature	Indicadores	Cenário 1	Cenário 2	Cenário 3
Recurso Fresa / Feature Degrau	Tempo Processamento (min)	704,38	704,38	704,38
	Custo Processamento (R\$)	224,35	224,35	224,35
	Tempo de Fila (min)	85,19	83,88	74,20
	Taxa Utilização do Recurso (%)	96,49	96,49	96,49
Recurso Esteira	Tempo Processamento (min)	20,40	20,40	20,40
	Custo Processamento (R\$)	4,08	4,08	4,08
	Tempo de Fila (min)	0,00	0,00	0,00
	Taxa Utilização do Recurso (%)	2,79	2,79	2,79
Recursos Torno / Feature Entalhe	Tempo Processamento (min)	502,07	502,07	502,07
	Custo Processamento (R\$)	178,78	178,78	178,78

	Tempo de Fila (min)	0,00	0,00	0,00
	Taxa Utilização do Recurso (%)	68,77	68,77	68,77
Recurso Operário	Tempo Processamento (min)	9,60	9,60	9,60
	Custo Processamento (R\$)	0,76	0,76	0,76
	Tempo de Fila (min)	0,00	0,00	0,00
	Taxa Utilização do Recurso (%)	1,31	1,31	1,31
Recurso Furadeira / Feature Furo	Tempo Processamento (min)	126,33	126,33	126,33
	Custo Processamento (R\$)	51,05	51,56	51,05
	Tempo de Fila (min)	0,00	0,00	0,00
	Taxa Utilização do Recurso (%)	17,30	17,47	17,30

Tabela 6.14 - Comparativo dos resultados da simulação gerais dos cenários do caso de teste 3

Indicadores	Cenário 1	Cenário 2	Cenário 3
Tempo Médio Processamento (min)	13,62	13,62	13,62
Custo Médio Processamento (R\$)	4,59	4,59	4,59
Tempo Médio Fila (min)	0,85	0,83	0,74
Tempo Total Produção(min)	1.447,99	1.460,31	1.437,00
Tempo Médio Total (min)	14,47	14,45	14,37

A partir dos resultados das Tabelas 6.13 e 6.14 foi realizada uma análise detalhada do ambiente de simulação. Os resultados mostraram que a mudança de posição dos recursos na sequência de produção gerou uma alteração no tempo de fila do sistema. Quanto mais no final do processo se localizou o recurso “restrição fresa”, menor ficou o tempo de fila. Com isso o tempo médio de produção de uma peça caiu de 14,47 minutos do primeiro cenário para 14,37 minutos do terceiro cenário.

O recurso fresa foi a restrição do sistema de manufatura simulado devido ao tempo de chegada de uma nova peça no sistema (6 minutos) ser menor do que seu tempo de processamento médio (7,04 minutos). Os outros recursos do sistema de manufatura tiveram seus tempos de processamento abaixo do tempo de chegada das peças e não houve formação de fila.

Para uma melhor análise do ambiente de simulação foi realizada a simulação de um quarto cenário com os mesmos dados de entrada do terceiro cenário, escolhido por ter melhores resultados, porém com alteração do tempo de chegada de uma nova peça no sistema de 6 minutos para 4 minutos. O tempo de simulação permaneceu 730 minutos. O objetivo dessa simulação foi mostrar a reação do

sistema com um tempo de chegada de peça mais baixo exigindo mais capacidade dos recursos do sistema (menor tempo de processamento por recurso). Na Tabela 6.15 e 6.16 são apresentados os resultados dessa simulação.

Tabela 6.15 - Resultados da simulação por recurso do cenário 4 do caso de teste 3

Recurso / Feature	Indicadores	Cenário 4
Recurso Fresa / Feature Degrau	Tempo Processamento (min)	718,47
	Custo Processamento (R\$)	228,84
	Tempo de Fila (min)	196,41
	Taxa Utilização do Recurso (%)	98,42
Recurso Esteira	Tempo Processamento (min)	20,80
	Custo Processamento (R\$)	4,16
	Tempo de Fila (min)	0,00
	Taxa Utilização do Recurso (%)	2,85
Recursos Torno / Feature Entalhe	Tempo Processamento (min)	512,12
	Custo Processamento (R\$)	182,35
	Tempo de Fila (min)	246,16
	Taxa Utilização do Recurso (%)	70,15
Recurso Operário	Tempo Processamento (min)	9,79
	Custo Processamento (R\$)	0,78
	Tempo de Fila (min)	0,00
	Taxa Utilização do Recurso (%)	1,34
Recurso Furadeira / Feature Furo	Tempo Processamento (min)	128,85
	Custo Processamento (R\$)	52,07
	Tempo de Fila (min)	0,00
	Taxa Utilização do Recurso (%)	17,65

Tabela 6.16 - Resultados da simulação gerais do cenário 4 do caso de teste 3

Tempo Médio Processamento (min)	Custo Médio Processamento (R\$)	Tempo Médio Fila (min)	Tempo Total Produção (min)	Tempo Médio Total (min)
13,62	4,59	4,33	1.832,63	17,96

Comparando os resultados entre os cenários três e quatro é possível observar um aumento do tempo médio de fila de 0,74 minutos para 4,33 minutos, o que significa um aumento de 82,90%. A fila não ficou restrita somente ao recurso fresa

(196,41 minutos) sendo também formada no primeiro recurso do sistema o torno (246,16 minutos). Esse fato ocorreu devido o tempo de chegada de uma nova peça (4 minutos) ser menor que o tempo de processamento médio do recurso torno (5,12 minutos). Como consequência do aumento do tempo de espera houve um aumento do tempo médio de produção da peça passando de 14,37 minutos para 17,96, o que significa um aumento de 19,82%

Apesar do aumento do tempo de produção de uma peça, o quarto cenário simulou a produção de 102 peças enquanto o terceiro cenário produziu 100 peças. Esse aumento foi devido à chegada de mais peças no sistema ao longo da simulação devido a seu menor tempo de chegada de uma nova peça no sistema.

7. APLICAÇÃO DO AMBIENTE DE SIMULAÇÃO EM SISTEMAS DE MANUFATURA REAIS

Este capítulo apresenta aplicações do ambiente de simulação em sistemas de manufatura reais. Inicialmente foi redigida uma introdução para uma melhor definição do propósito e dos passos do desenvolvimento das aplicações. Após, são detalhadas as duas aplicações com seus cenários e análises de seus resultados.

7.1 INTRODUÇÃO

A aplicação do ambiente de simulação em sistemas de manufatura reais teve como objetivo mostrar que as funcionalidades do ambiente decorrente da abordagem proposta no trabalho são aplicáveis em sistemas reais e podem prover benefícios para as empresas de diferentes setores.

O desenvolvimento das aplicações foram realizados a partir da adaptação de metodologias para estudos de simulação existentes na literatura e apresentadas na seção 3.1.6. A Figura 7.1 mostra os passos sequenciais que foram utilizados nas aplicações e em seguida é detalhado o desenvolvimento de cada passo.

Figura 7.1 - Passos da aplicação do ambiente de simulação em sistemas de manufatura reais

- *Formulação do Problema:* iniciou-se definindo os propósitos e objetivos da aplicação do ambiente de simulação em cada um dos sistemas de manufatura em estudo de modo a entender quais parâmetros influenciam o sistema e o que está se modelando. Para isso buscou-se entender o funcionamento de cada sistema de manufatura com seus principais parâmetros e restrições. Para obtenção da familiarização e conhecimento adequado dos processos e recursos que compõem um sistema de manufatura utilizou-se a forma matemática/lógica em forma de fluxogramas. Também foi definido que não era esperado como resposta destas aplicações decisões de otimizações, mas fazer simulações de sistemas reais. As aplicações se limitariam a realizar projeções a partir de dados coletados no sistema real;
- *Coleta de Informações e Dados:* foi realizada uma pesquisa a partir de visitas técnicas nas empresas em estudo com perguntas (entrevistas) às pessoas direta ou indiretamente envolvidas no sistema de manufatura, além do exame de elementos documentais como desenhos técnicos e ordens de produção;
- *Formulação do Modelo Conceitual:* foram definidos os principais componentes (peça, *features*, recursos de fabricação e movimentação) e as interações entre esses recursos. Para cada sistema de manufatura foram inseridas as informações sobre os componentes no banco de dados do sistema, assim como na modelagem da simulação;
- *Verificação e Validação:* a verificação e a validação foi realizada para confirmar que o modelo de cada aplicação estava operando sem erros de sintaxe e lógica, além de garantir que os resultados por ele fornecidos sejam confiáveis e representativos de cada sistema real. Na verificação foi utilizada a técnica do teste por parâmetros. O teste por parâmetros consistiu na execução da simulação através da variação dos parâmetros de entrada, verificando os resultados obtidos. A validação foi realizada por especialistas que trabalham no dia-a-dia de cada sistema de manufatura;
- *Experimentação:* foram definidos, para cada sistema de manufatura em estudo, cenários a serem testados no ambiente de simulação. Para permitir uma boa qualidade da análise dos resultados, cada cenário testado foi cuidadosamente documentado. Os dados de entrada e de saída foram organizados em forma de tabelas e armazenados em uma base de dados para posterior análise;

- *Análise dos Resultados:* foram analisados os resultados gerados pela simulação, buscando testar soluções para o problema de cada aplicação. O objetivo desta etapa foi traçar inferências sobre os resultados alcançados pela simulação dos cenários. A partir dos resultados obtidos na simulação tiram-se conclusões sobre o sistema real modelado. Esse passo foi baseado nos indicadores de desempenho do ambiente de simulação que foram projetados a partir das variações dos parâmetros de cada aplicação;
- *Documentação:* foi realizado ao longo do desenvolvimento das aplicações. Para cada aplicação foram geradas tabelas, fluxo e figuras, além do detalhamento da aplicação. No final do desenvolvimento das aplicações gerou-se um registro (documento) geral que está sendo detalhado nesse capítulo do trabalho.

7.2 DESENVOLVIMENTO DAS APLICAÇÕES

7.2.1 Aplicação 1

A primeira aplicação foi desenvolvida baseada num sistema de manufatura de uma empresa do setor metalúrgico que trabalha a vinte anos com o desenvolvimento de produtos como escadas, cadeiras, mesas, corrimãos e cobertura metálicas para os setores residenciais e comerciais.

A peça em estudo faz parte de um corrimão (guarda corpo) detalhado na amostra da Figura 7.2. Esse corrimão é comercializado para utilização em áreas internas ou externas tendo como a principal função a segurança de seus usuários. Ele é composto de três peças diferentes (base de fixação, montante, parte superior) que são soldadas formando o produto final. Cada peça do corrimão é manufaturada em processos de fabricação e movimentação separados realizados em paralelo. A altura do corrimão depende do tamanho do montante e a largura do montante define o tamanho da base de fixação.

Figura 7.2 - Amostra do corrimão da aplicação 1

A peça em estudo é a base de fixação do corrimão. Essa base pode ser quadrada ou circular dependendo da necessidade do usuário. A peça (base) é soldada no montante de acordo com a Figura 7.3 para dar sustentação ao corrimão. O sistema de manufatura responsável por produzir a peça é composto pelas etapas mostradas na Figura 7.4.

Figura 7.3 - Base de fixação soldada no montante

Figura 7.4 - Sistema de manufatura da peça da aplicação 1

O primeiro processo do sistema de manufatura é o corte da placa. O material da placa é aço inoxidável (esse material é muito utilizado em corrimãos devido à sua resistência à corrosão, às suas propriedades higiênicas e estéticas). A empresa compra a matéria-prima em barras de seis metros de comprimento e corta de acordo com as medidas do projeto da peça. Esse processo é realizado numa máquina de corte industrial. Em seguida, a peça cortada é encaminhada para o processo de furação. Esse processo é único que produz *features*, o furo. Esse processo é fundamental para a peça, pois permite fixação do corrimão na superfície para sustentação do mesmo. A peça é composta de quatro furos (*features*) onde são inseridos os parafusos de acordo com a Figura 7.5. O último processo é o polimento da barra. Esse processo tem a função de dar o brilho/proteção à peça e utiliza uma sequência e grão de abrasivos numa máquina politriz.

A peça em estudo foi escolhida devido a um problema exposto pelo gerente da empresa em estudo no momento da visita técnica. A máquina (furadeira) que produz as *features* furos na peça é antiga, tendo sido comprada no início do funcionamento da empresa a vinte anos atrás. Com isso, gerou-se uma dúvida com relação a eficiência dessa máquina, ou seja, de acordo com a produção mensal da

peça e os tempos e custos de produção da máquina atual, vale a pena comprar uma nova máquina ou continuar produzindo com a atual.

Figura 7.5 - A peça da aplicação 1

Para o desenvolvimento do modelo foi realizada a coleta de dados mostrada nas Tabelas 7.1, 7.2, 7.3 e 7.4. Os dados coletados são: as dimensões da peça, as dimensões e as localizações das *features*, os parâmetros de tempo e de custo do recurso furadeira atual, a quantidade de peças produzidas por mês e o tempo entre chegada de peças no sistema.

Tabela 7.1 - Parâmetros de entrada aplicação 1

Tempo Entre Chegada de Peças (min)	Quantidade de Peças Produzidas (mês)
2,0	300

Tabela 7.2 - Parâmetros da peça da aplicação 1

PEÇA				
Descrição	Material	Comprimento (mm)	Largura (mm)	Altura(mm)
Base de Fixação	Aço Inoxidável	75	75	5

Tabela 7.3 - Parâmetros das *features* da aplicação 1

FEATURES					
FUROS					
Diâmetro(mm)	Altura (mm)	Pos. X	Pos. Y	Pos. Z	Orientação
10	5	0,10	0,0	0,90	1 (Face XZ)
10	5	0,10	0,0	0,10	1 (Face XZ)
10	5	0,90	0,0	0,90	1 (Face XZ)
10	5	0,90	0,0	0,10	1 (Face XZ)

Tabela 7.4 - Parâmetros do recurso atual da aplicação 1

Parâmetros	Furadeira
Salário Operador (R\$/hora)	4,30
Salário Máquina (R\$/hora)	15,00
Custo Ferramenta (R\$/mm ³)	0,10
Diametro de Corte (mm)	10,00
Profundidade de Corte (mm)	5,00
Velocidade de Avanço (mm/min)	6,50
Per.Tempo Improdutivo (%)	20,00
Tempo Vida Ferramenta (min)	15,00
Tempo Troca Ferramenta (min)	0,52

O próximo passo desenvolvido foi a construção do modelo. Os dados coletados foram inseridos no banco de dados do ambiente de simulação seguido da modelagem da simulação. A Figura 7.6 mostra o modelo da peça simulada na aplicação 1. A modelagem do situação atual foi mostrada ao responsável pelo sistema de manufatura para validação do modelo.

Figura 7.6 - Modelo da peça da aplicação 1

A próximo passo foi a experimentação de um cenário com um recurso furadeira com parâmetros diferentes dos atuais. Para isso, foi realizada uma pesquisa em empresas especializadas na comercialização de furadeiras industriais. A Tabela 7.5 detalha os parâmetros da furadeira do cenários proposto.

Tabela 7.5 - Parâmetros do novo recurso da aplicação 1

Parâmetros	Furadeira
Salário Operador (R\$/hora)	4,30
Salário Máquina (R\$/hora)	12,00
Custo Ferramenta (R\$/mm ³)	0,10
Diametro de Corte (mm)	10,00
Profundidade de Corte (mm)	5,00
Velocidade de Avanço (mm/min)	9,00
Per.Tempo Improdutivo (%)	15,00
Tempo Vida Ferramenta (min)	20,00
Tempo Troca Ferramenta (min)	0,48

O último passo desenvolvido foi a análise dos resultados da simulação do cenário atual e o proposto. As Figuras 7.7 e 7.8 mostram a interface resultados do ambiente de simulação com o gráfico comparando o tempo e o custo de fabricação por furo dos dois cenário modelados. A Tabela 7.6 detalha o comparativo dos cenários atual e proposto.

Tabela 7.6 - Comparativo dos resultados da simulação dos cenários da aplicação 1

Indicadores	Cenário Atual	Cenário Proposto
Tempo Fabricação por Furo (min)	1,48	1,01
Custo Fabricação por Furo (R\$)	0,57	0,37
Tempo Fabricação por Furo Mensal (min)	445,98	305,90
Custo Fabricação por Furo Mensal (R\$)	173,45	113,104
Tempo Total de Produção Peça (min)	1.783,93	1.223,61
Custo Total de Produção Peça (R\$)	693,83	452,41

Figura 7.7 - Comparativo do tempo de fabricação por furo entre cenários da aplicação 1

Figura 7.8 - Comparativo do custo de fabricação por furo entre cenários da aplicação 1

A partir dos resultados da Tabela 7.6 foi realizada uma análise da aplicação. Os resultados mostram que a inserção de uma nova máquina no cenário proposto gerou uma redução de 31,75 % no tempo de fabricação por furo e uma redução de

35,08% no custo de fabricação por furo. Considerando quatro furos e a produção total de trezentas peças houve uma redução de tempo de 560,32 minutos por mês e uma redução de custos de R\$241,42 reais por mês.

Essa melhoria é devido à nova furadeira proporcionar uma redução no tempo improdutivo ficando menos tempo em manutenção corretiva, conseqüentemente, reduzindo o salário máquina. Há também uma redução no tempo de troca de ferramenta devido a dispositivos modernos já acoplados à nova furadeira e um aumento no tempo de vida da ferramenta. Outro fator importante é o aumento do avanço por minuto gerado a partir de uma maior rotação por minuto (rpm) do motor da nova furadeira

Apesar das melhorias mostradas, para comprar uma nova furadeira é necessário desembolsar em torno de R\$8.000,00. De acordo com o responsável pelo sistema de manufatura, a furadeira atual pode ser vendida por R\$3.000,00, sendo assim necessário desembolsar R\$5.000,00. Com os resultados obtidos na simulação foi possível analisar o ganho por mês relativo ao tempo e ao custo para cada peça a ser fabricada na furadeira e comparar com o valor a ser gasto, gerando uma tomada de decisão com maior acuracidade.

7.2.2 Aplicação 2

A segunda aplicação foi desenvolvida baseada num sistema de manufatura de uma empresa fabricante de máquinas para embalagem plásticas. A empresa atua neste mercado a dezessete anos tendo no seu portfólio máquinas para embalagens do tipo sacoleira, laminadora e blocadora.

A principal máquina manufaturada pela empresa em estudo é a blocadora. Ela produz sacos e sacolas de plástico de diferentes tamanhos e larguras. O processo produtivo da blocadora consiste no corte e na soldagem dos sacos a partir de bobinas de tamanhos variados. O corte pode ser do tipo solto ou serrilhado e a soldagem pode ser do tipo de fundo, de beira lateral ou lateral convencional.

Apesar de ser uma máquina complexa composta por diversos componentes, alguns componentes tem uma maior relevância devido à função exercida. A peça em estudo é desses componentes essenciais para uma máquina blocadora, pois tem

como função estabilizar o eixo da máquina. Sem essa peça uma máquina blocadora não executa suas atividades básicas de corte e soldagem com sucesso. Ela é chamada de travessa do eixo do blocador e a Figura 7.9 mostra uma amostra da peça. O material da peça é o aço SAE 1020. Esse tipo de aço é muito utilizado em componentes mecânicos devido às suas características como a boa plasticidade, usinabilidade e soldabilidade.

Figura 7.9 - Peça da aplicação 2

A peça é produzida a partir de um projeto mecânico desenvolvido num sistema CAD. A Figura 7.10 detalha o projeto da peça que é composta de seis *features* que são: dois rasgos e quatro furos.

Figura 7.10 - Projeto da peça da aplicação 2

Apesar de na Figura 7.9 ser possível ver somente dois furos, a parte inferior do projeto da Figura 7.10 mostra que são inseridos dois furos sobrepostos sendo assim necessário fabricar quatro furos. Esses furos são do tipo cego, detalhado no projeto, com o diâmetro maior e altura menor ou do tipo passante que são os de diâmetro menor e altura maior. Com relação ao rasgo, devido ao arredondamento das laterais, a sua fabricação consiste na confecção de um rasgo quadrado seguidos de dois semi-círculos nas laterais.

Para a fabricação das *features* o sistema de manufatura da empresa é composto de uma fresa (fabrica os rasgos) e uma furadeira (fabrica os furos). A movimentação das peças entre as máquinas é realizada por um operário. A Figura 7.11 apresenta os processos do sistema de manufatura.

Figura 7.11 - Sistema de manufatura da aplicação 2

A peça em estudo foi escolhida devido a um problema observado na visita técnica à empresa. O sistema de manufatura apresenta um desbalanceamento de tempo entre os seus processos. A máquina fresa responsável pela fabricação das *features* rasgo têm um tempo de processamento muito maior do que os outros processos (movimentação e furação) gerando uma fila de peças nesse recurso. Além do tempo, o processamento também apresenta um alto custo em relação ao

processo de furação. Para minimizar esse problema a empresa inseriu um estoque pulmão produzido num turno extra de trabalho da máquina fresa. Porém essa solução gera altos custos devido ao pagamento de horas extras diariamente. Com isso, gerou-se uma dúvida com relação aos ganhos com a mudança na alocação da máquina furadeira para produzir os semi-furos que compõem os rasgos reduzindo assim o tempo e o custo do processo de fresamento.

Após a definição do problema realizou-se a coleta de dados e informações que são mostrados nas Tabelas 7.7, 7.8, 7.9 e 7.10. Os dados coletados são: as dimensões da peça; as dimensões e as localizações das *features*; os parâmetros de tempo e de custo do recurso de fabricação e movimentação; a quantidade de peças produzidas por mês e; o tempo entre chegada de peças no sistema.

Tabela 7.7 - Parâmetros de entrada aplicação 2

Tempo Entre Chegada de Peças (min)	Quantidade de Peças Produzidas (mês)
5,0	125

Tabela 7.8 - Parâmetros da peça da aplicação 2

PEÇA				
Descrição	Material	Comprimento (mm)	Largura (mm)	Altura(mm)
Travessa do Eixo	Aço SAE 1020	210	25,40	25,40

Tabela 7.9 - Parâmetros das *features* da aplicação 2

FEATURES							
Rasgo							
Comprimento (mm)	Largura (mm)	Altura (mm)	Diametro (mm)	Pos. X	Pos. Y	Pos. Z	Orientação
10,00	10,00	24,50	10,00	0,50	0,00	0,04	1 (Face XZ)
10,00	10,00	24,50	10,00	0,50	0,00	0,50	1 (Face XZ)
Furo							
Diâmetro(mm)	Altura (mm)			Pos. X	Pos. Y	Pos. Z	Orientação
15,00	13,00			0,50	0,00	0,65	1 (Face XZ)
15,00	13,00			0,50	0,00	0,94	1 (Face XZ)
10,00	25,40			0,50	0,00	0,65	1 (Face XZ)
10,00	25,40			0,50	0,00	0,94	1 (Face XZ)

Tabela 7.10 - Parâmetros dos recursos atuais da aplicação 2

FABRICAÇÃO				TRANSPORTE	
Parâmetros	Fresa	Furadeira		Parâmetros	Operário
		D=10	D=15		
Salário Operador	16,00	16,00	16,00	Velocidade (m/min)	30,00

(R\$/hora)					
Salário Máquina (R\$/hora)	60,00	30,00	30,00	Distância (m)	3,00
Custo Ferramenta (R\$/mm ³)	1,60	1,20	1,20	Per.Tempo Improdutivo (%)	30,00
Diâmetro Corte (mm)	10,00	15,00	10,00	Custo Unidade por Tempo (R\$/min)	0,20
Profundidade Corte (mm)	26,00	13,00	26,00		
Comprimento Corte (mm)	10,00	0,00	0,00		
Velocidade de Corte (mm/min)	8,00	0,00	0,00		
Velocidade de Avanço (mm/min)	4,00	8,00	8,00		
Per.Tempo Improdutivo (%)	25,00	15,00	15,00		
Tempo Vida Ferramenta (min)	5,00	6,00	6,00		
Tempo Troca Ferramenta (min)	1,20	0,55	0,55		

O próximo passo desenvolvido foi a construção do modelo. Os dados coletados foram inseridos no banco de dados do ambiente de simulação seguido da modelagem da simulação. A Figura 7.12 e 7.13 mostram os modelos da peça simulada e do sistema de manufatura na aplicação 2. A modelagem da situação atual foi mostrada ao responsável pelo sistema de manufatura para validação do modelo.

Figura 7.12 - Modelo da peça da aplicação 2

Figura 7.13 - Modelo do sistema de manufatura da aplicação 2

O próximo passo foi a experimentação a partir da implementação de um novo cenário do sistema de manufatura em estudo. Nesse cenário foi determinado que o recurso furadeira ficaria responsável pela produção dos semi-furos dos rasgos, reduzindo assim as responsabilidades do recurso fresa que é o recurso restrição do sistema de manufatura.

Com essa alteração, busca-se reduzir os custos com estoques pulmões e horas extras, além de balancear os tempos e custos do sistema de manufatura. Esse cenário foi sugerido pelo responsável do sistema de manufatura que não tinha uma ferramenta para realizar essa análise.

O último passo desenvolvido foi a análise dos resultados da simulação do cenário atual e do proposto. As Tabelas 7.11 e 7.12 detalham o comparativo dos resultados específicos por recurso e geral do sistema entre os cenários atual e o proposto para a produção de 125 peças por mês. Já a Tabela 7.13 detalha o tempo e o custo de fabricação para cada *feature* nos dois cenários simulados.

Tabela 7.11 - Resultados da simulação por recurso da aplicação 2

Recurso / Feature	Indicadores	Cenário Atual	Cenário Proposto
Recurso Fresa / Feature Rasgo	Tempo Processamento (min)	1.282,50	927,50
	Custo Processamento (R\$)	2.225,00	1.575,00
	Tempo de Fila (min)	518,75	327,50
Recurso Operário	Tempo Processamento (min)	16,25	16,25
	Custo Processamento (R\$)	3,75	3,75
	Tempo de Fila (min)	0,00	0,00
Recurso Furadeira / Feature Furo	Tempo Processamento (min)	750,00	897,50
	Custo Processamento (R\$)	1.172,50	1.435,00
	Tempo de Fila (min)	0,00	0,00

Tabela 7.12 - Resultados da simulação gerais da aplicação 2

Indicadores	Cenário Atual	Cenário Proposto
Tempo Médio Processamento (min)	2.048,50	1.841,25
Custo Médio Processamento (R\$)	3.401,25	3.013,75
Tempo Médio Fila (min)	4,15	2,62
Tempo Total Produção(min)	2.567,75	2.168,75
Tempo Médio Produção por Peça (min)	20,53	17,35

Tabela 7.13 - Resultados por *features* da aplicação 2

Recurso / Feature	Features	Cenário Atual		Cenario Proposto	
		Tempo(m)	Custo(R\$)	Tempo(m)	Custo (R\$)
Fresa / Rasgo	Rasgo	7,42	12,60	7,42	12,60
	Semi-Furo	2,84	5,20		
Furadeira / Furo	Furo (D=15)	3,64	5,18	3,64	5,18
	Furo (D=10)	2,36	4,20	2,36	4,20
	Semi-Furo			1,18	2,10

A partir dos resultados das Tabelas 7.11, 7.12 e 7.13 foi realizada uma análise da aplicação. Os resultados mostram que o cenário proposto gerou uma redução de tempo e custo em relação ao cenário atual da empresa.

A Tabela 7.11 detalha que os processos de fresamento (927,50 min) e de furação (897,50 min), no cenário proposto, ficaram com os tempos de

processamento mais balanceados em relação ao cenário atual que tem os tempos de fresamento (1.282,50 min) e a furação (750,00 min). A tabela ainda mostra que houve uma redução no tempo de fila do recurso restrição fresa de 518,75 minutos para 327,50 minutos.

Já a Tabela 7.12 mostra a redução entre o cenário atual e proposto no tempo e no custo de processamento em torno de 10,11% e 11,39 %, respectivamente. Tal tabela também detalha a redução do tempo médio de fila por peça no sistema de manufatura de 36,86% e, conseqüentemente, a redução do tempo médio de produção de uma peça de 20,53 minutos para 17,35, o que significa uma redução 15,49%.

Prosseguindo, a Tabela 7.13 detalha o porquê das melhorias do cenário proposto em relação ao cenário atual. O recurso furadeira produz as *features* semi-furo em um menor tempo e custo do que o recurso fresa. Essa redução foi de 58,45% no tempo e 59,61% no custo de fabricação dos semi-furos. No cenário proposto o recurso fresa produz somente os rasgos num tempo de 7,42 minutos ao custo de R\$12,60 enquanto o recurso furadeira produz as *features* circulares (furos e semi-furos) num tempo de 7,18 minutos ao custo de R\$11,48. Portanto, no cenário proposto, os tempos e os custos ficaram balanceados reduzindo assim o tempo de fila e, conseqüentemente, o tempo de produção de uma peça.

A solução proposta nessa aplicação não necessita de investimento para ser implantada. A implantação depende somente de um treinamento dos operários do recurso furadeira para produzir os semi-furos nas laterais dos rasgos com a mesma qualidade que eram fabricados no recurso fresa.

8. CONCLUSÃO

Este capítulo apresenta as principais conclusões e comentários do trabalho, tais como: objetivos alcançados, importância do trabalho, diferenciais e contribuições, limitações e dificuldades encontradas. Por fim, apresenta-se sugestões para realização de trabalhos futuros.

8.1 CONCLUSÕES E COMENTÁRIOS

Pode-se afirmar que o objetivo proposto na seção 1.3, ou seja “propor um ambiente para a simulação da manufatura, a partir de uma nova abordagem baseada nas *features* das peças a serem simuladas e na técnica de realidade virtual”, foi plenamente atingido. Esta afirmação baseia-se na proposta apresentada no capítulo 4 e nos desenvolvimentos e testes apresentados nos capítulos 5, 6 e 7.

Para o desenvolvimento da abordagem proposta foi especificado, modelado, implementado e testado um programa computacional. Foi construída uma ferramenta que é um ambiente voltado para simulação de sistemas de manufatura com facilidade de uso, custos menores e com mais flexibilidade do que a experimentação no sistema real.

O ambiente de simulação tem como objetivo permitir aos responsáveis por sistemas de manufatura tomar decisões baseadas em indicadores de desempenho do sistema com relação ao tempo e ao custo sem alterar o dia-a-dia da empresa.

O sistema foi desenvolvido a partir de pesquisas em sistemas de manufatura para definir os seus principais componentes e os relacionamentos entre eles. Sua arquitetura foi dividida em módulos que são: banco de dados, modelagem da simulação, simulação, interface gráfica com usuário e sistema de visualização tridimensional. Cada módulo foi detalhado nesse trabalho mostrando seu desenvolvimento e funcionamento. Para esse desenvolvimento foi necessário identificar e selecionar os recursos de *software* adequados às especificações predeterminadas na definição do sistema.

Entre os diferenciais e contribuições do ambiente de simulação apresentado neste trabalho pode-se destacar:

1. A união de tecnologias de diferentes áreas como tecnologia de *features*, simulação computacional e realidade virtual para análise de desempenho de sistemas de manufatura existentes ou não.
2. Confiabilidade na simulação. O sistema é simulado baseado no projeto geométrico da peça.
3. Flexibilidade na experimentação de sistemas de manufatura. O sistema permite ao usuário realizar diversas combinações de recursos para a fabricação de uma *feature*, assim como de uma sequência de produção.
4. A utilização de interfaces gráficas com usuário simples e amigáveis que permitem a manipulação dos dados de entrada e saída e a realização da simulação fácil e eficazmente. Não é preciso ter conhecimentos de simulação ou programação para usar o sistema.
5. O sistema proporciona uma visualização 3D que fornece à simulação um maior controle, pois não fica dependente somente da lógica matemática desenvolvida e ainda fornece ao usuário uma sensação de integração com o sistema.
6. Existe a possibilidade de ampliação da biblioteca de *features* e de recursos e conseqüentemente da quantidade de peças a serem simuladas.

O ambiente de simulação também apresentou limitações que são:

1. Desenvolvimento de modelos de sistemas de manufatura que dependem da biblioteca de *features* e recursos.
2. Necessidade de conhecimentos em interpretação de projetos mecânicos de peças e de conhecimento do sistema de manufatura (recursos) a ser simulado.
3. Impossibilidade de simular a fabricação com recursos em paralelo.
4. Na simulação foram considerados os tempos de *setup* e os tempos de falhas dos recursos de fabricação e a movimentação como tempo improdutivo não fazendo distinção entre eles.
5. O sistema de visualização tridimensional permite somente a visualização durante a simulação, não possibilitando manipulações dos objetos da cena.

A principal dificuldade encontrada no desenvolvimento do trabalho foi a familiarização com áreas diferentes do conhecimento. Foi necessário adquirir conhecimentos nas áreas de manufatura (projeto e produção), simulação e realidade virtual. Além dos conceitos teóricos também foi necessário utilizar recursos de *software* da área computacional para modelagem e implementação do sistema. Alguns *softwares* já faziam parte do conhecimento do autor e outros foram aprendidos ao longo do desenvolvimento do trabalho.

8.2 SUGESTÕES PARA TRABALHOS FUTUROS

Com base nos conhecimentos adquiridos na elaboração desta tese, são apresentadas algumas sugestões para trabalhos futuros:

- Aplicar o ambiente de simulação em sistemas de manufatura reais ampliando o banco de dados das *features* e dos recursos e comparar os resultados das simulações;
- Ampliar as funções do sistema de visualização tridimensional permitindo uma maior interação com o usuário e possibilitando modificações da cena ao longo da simulação;
- Adaptar ao ambiente de simulação dispositivos de entrada e de saída de dados disponíveis na realidade virtual para uma maior imersão do usuário. Alguns exemplos de dispositivos disponíveis no mercado são: áudio, rastreadores, *joysticks* e cavernas digitais;
- Integrar o ambiente de simulação a um sistema computacional de projetos (CAD) no que diz respeito à peça modelada com as *features*. Essa integração permitiria que os dados da peça fossem transmitidos automaticamente para o banco de dados do ambiente de simulação.
- Inserir no ambiente de simulação *features* com características geométricas mais complexas possibilitando simular uma maior quantidade de sistemas de manufatura.
- Adicionar no ambiente de simulação a localização (X, Y, Z) de cada recurso (fabricação e transporte) na cena tridimensional com o objetivo de possibilitar análise de *layout* (distâncias) do sistema de manufatura.

REFERENCIAS

ABDUL-KADER, W.; GHARBI, A. **Capacity estimation of a multi-product unreliable production line**. International Journal of production research, n.40, p.4815-4834, 2002.

ADAMS, L. **Visualização e realidade virtual**. São Paulo: Makron Books, 1995.

AGOSTINHO, L. O. **Sistemas de Informações Aplicadas a Estratégia da Manufatura**. Apostila da Disciplina Sistemas de Informações Aplicadas a Estratégia da Manufatura. Departamento de Engenharia de Fabricação, Faculdade de Engenharia Mecânica Universidade Estadual de Campinas, 2000.

AGOSTINHO, L. O.; VILELLA, R. C.; BUTTON, S. T. **Processos de fabricação e planejamento de processos**. Apostila da disciplina Introdução à Engenharia de Fabricação da Engenharia Mecânica da Unicamp, 2004.

ÁLVARES, A. J. **Uma metodologia para integração CAD/CAPP/CAM voltada para manufatura remota de peças rotacionais baseada na internet**. Tese de Doutorado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina, 2005.

ÁLVARES, A. J.; FERREIRA, J. C. E. **Uma metodologia para integração CAD/CAPP/CAM voltada para manufatura remota de peças rotacionais via Web**. 2º Congresso de Engenharia de Fabricação. Uberlândia-MG, 2003.

ARNOLD, J. R. T. **Administração de materiais**. São Paulo: Atlas, 1999.

ASIMOV. **Simulador de robótica – células de produção**. Manual do Usuário, Porto Alegre: Serviço Nacional de Aprendizagem Industrial (SENAI-RS), 1999.

ASKIN, R. G.; STANDRIDGE, C. R. **Modeling and analysis of manufacturing systems**. New York: John Wiley & Sons, 1993.

BALLOU, R. H. **Gerenciamento da Cadeia de Suprimentos**. São Paulo: Bookman, 2006.

BANKS, J. **Handbook of simulation: principles, methodology, advances, applications, and practice**. New York: John Wiley & Sons, 1998.

BANKS, J.; CARLSON, J. S.; NELSON, B. L.; NICOL, D. M. **Discrete-event system simulation**. 5^a.ed New Jersey: Prentice-Hall, 2009.

BANERJEE, P.; ZETU, D. **Virtual manufacturing**. New York: John Wiley & Sons, 2001.

BARRÔNIO, S. **Desenvolvimento de modelos de simulação computacional para análise e melhorias de sistemas produtivos**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal do Rio Grande do Sul, 2000.

BATEMAN, R. E.; BOWDEN, R.; GOGG, T.; HARRELL, C.; MOTT, J. **System improvement using simulation**. 5th ed, Utah: Promodel Corporation, 1997.

BENAVENTE, J. C. T. **Um sistema para o projeto e fabricação remota de peças prismáticas via internet**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina, 2007.

BERTRAND, J. W. M; FRANSOO, J. C. Modeling and Simulation: Operations management research methodologies using quantitative modeling. **International Journal of Operations & Production Management**, v.22, n.2, p.241-264, 2002.

BHANDARKAR, M. P.; DOWNIE, B.; HARDWICK, M.; NAGI, R. Migrating from IGES to STEP: one to one translation of IGES drawing to STEP drafting data. **Computers in Industry** , v. 41 n.3, pp.261–277, 2000.

BLACK, J. T. **Projeto da fábrica com futuro**. Porto Alegre: Bookman, 1998.

BLANCHETTE, J.; SUMMERFIELD, M. **C++ GUI programming with Qt4**. 2^a.ed New Jersey: Prentice-Hall, 2008.

BOEHM, B. A Spiral Model of Software Development and Enhancement. **IEEE Computer**, v.21, n.5, p. 61-72,1988.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML - Guia do Usuário**. São Paulo: Campus, 2000.

BOTEGA, L. C. CRUVINEL, P. E. **Realidade Virtual: Histórico, Conceitos e Dispositivos**. Porto Alegre: IX Symposium on Virtual and Augmented Reality, 2009.

BOWERSOX, D. J.; CLOSS, D. J.; COOPER, M. B. **Gestão Logística de Cadeias de Suprimentos**. Porto Alegre: Bookman, 2006.

BOWMAN, D. A.; KRUIJFF, E.; LAVIOLA, J. J.; POUPYREV, I. **3D user interfaces: theory and practice**. Boston: Addison-Wesley, 2005.

BRAGA, P. S. T. **Processos de fabricação**. Apostila do Programa de Certificação de Pessoal de Manutenção. Espírito Santo: Serviço Nacional de Aprendizagem Industrial (SENAI-ES), 1999.

BRAGHIROLI, L. F. **Estudo da linearidade da produção em células de manufatura através de simulação a eventos discretos**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal do Rio Grande do Sul, 2009.

BRYSON, S. Virtual reality in scientific visualization. **Communications of the ACM**, , v.39, n.5, p. 62-71, 1996.

CAMARGO, M. A. R. **Geração de objetos virtuais a partir de imagens**. Dissertação de Mestrado. Programa de Pós-Graduação em Ciências da Computação da Universidade Metodista de Piracicaba, 2008.

CANCIGLIERI, O. Product Model Based Translation Mechanism To Support Multiple Viewpoints In The Design For Manufacture Of Injection Molded Products. **Doctoral Thesis**, Loughborough University - Department of Manufacturing Engineering, Loughborough, United Kingdom, 1999.

CAREY, R.; BELL, G. **The annotated VRML 2.0 reference manual**. UK: Addison-Wesley, 1997.

CARSON II, J. S. Introduction to modeling and simulation. **Proceedings** of the 2004 Winter Simulation Conference, p.01-08, 2004.

CENTENO, M. A.; CARRILLO, M. Challenges of introducing simulation as a decision making tool. **Proceedings** of the 2001 Winter Simulation Conference, p.17-21, 2001.

CHIAVERINI, V. **Tecnologia mecânica: processos de fabricação e tratamento**. 2ª ed. São Paulo, Makron Books, 1986.

CHRYSSOLOURIS, G. **Manufacturing systems – theory and practice**. 2ª ed New York: Springer-Verlag, 2006.

CORRÊA, H. L.; GIANESI, I. G. N. **Just in time, MRP II e OPT: um enfoque estratégico**. 2ª ed, São Paulo: Atlas, 1996.

CRUZ-NEIRA, C.; SANDIN, D. J.; DEFANTI, T. A.; KENYON, R. V.; HART, J. C. The CAVE áudio visual experience automatic virtual environment. **Communication of the ACM**, n.35, p.64-72, 1992.

DA SILVA, A. R. Y. **Modelagem de custos em sistemas de manufatura utilizando redes de petri**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade de São Paulo, 2000.

DEITEL, H. M.; DEITEL, P. J. **C++: como programar**. 3ª ed, Porto Alegre: Bookman, 2001.

DIAS, M. A. P. **Administração de materiais: uma abordagem logística**. São Paulo: Atlas, 1993.

DINIZ, A. E.; MARCONDES, F. C.; COPPINI, N. L. **Tecnologia de usinagem dos materiais**. 3ª ed. São Paulo: MM Editora, 2001.

DOROZHKIN, D.; VANCE, J. M.; REHN, G. D.; LEMESSI, M. Coupling of interactive manufacturing operations simulation and immersive virtual reality. **Virtual Reality**, v. 16, p.15.23, 2012.

DOS SANTOS, R. **Avaliação do processo de alargamento de guias de válvulas**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal do Paraná, 2004.

DOS SANTOS, M. C. C. **Avaliação do uso de realidade virtual na robótica**. Dissertação de Mestrado. Programa de Pós-Graduação em Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul, 2006.

ESPÍNDOLA, D. B. **O uso da realidade virtual na manufatura da industria de construção naval/offshore**. Programa de Pós-Graduação em Engenharia Oceânica da Fundação Universidade do Rio Grande, 2007.

FARIAS, J. B. S. **Desenvolvimento de um sistema semi-generativo para planejamento de processo auxiliado por computador para ambiente de usinagem**. Dissertação de Mestrado.

Programa de Pós-Graduação da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, 2008.

FERRARESI, D. **Fundamentos da usinagem dos metais**. São Paulo: Edgard Blucher, 1985.

FERREIRA, A. G. **Uma arquitetura para visualização distribuída de ambientes virtuais**. Dissertação de Mestrado. Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Rio de Janeiro, 1999.

FOLEY, J. D. **Computer Graphics: Principles and Practice**. Boston: Addison-Wesley, 1990.

FRASER, M.; GLOVER, T.; VAGHI, I.; BENFORD, S.; GREENHALGH, C.; HINDMARSH, J.; HEATH, C. Revealing the realities of collaborative virtual reality. **Proceedings** of Third International Conference on Collaborative Virtual Environments, New York: ACM Press, 2000.

FREITAS-FILHO, P. J. **Introdução à modelagem e simulação de sistemas: com aplicação em Arena**. 2ª ed, Florianópolis: Visual Books, 2008.

FURLAN, J. D. **Modelagem de Objetos através da UML**. São Paulo: Makron Books, 1998.

GAO, J.; ZHENG, D.; GINDY, N. Mathematical representation of feature conversion for CAD/CAM system integration. **Computer Integrated Manufacturing**, n.20, p.457–467, 2004.

GAO, J. X.; HUANG, X. X. *Product and manufacturing capability modeling in an integrated CAD/Process planning environment*. The international Journal of Advanced Manufacturing Technology, London: Springer-Verlag. v. 11, p.43-51, 1996.

GAVIRA, M. O. **Simulação computacional como uma ferramenta de aquisição de conhecimento**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia de Produção da Escola de Engenharia de São Carlos da Universidade de São Paulo, 2003.

GIL, A. C. **Métodos e técnicas de pesquisa social**. 2ª ed. São Paulo: Atlas, 2008.

GRUHN, V. Process centered software engineering environments: A brief history and future challenges. In: **Annals** of Software Engineering 14, p. 363-382, 2002.

GUERRINI, F. M.; COELHO, R. T. **Tecnologias avançadas de manufatura: Coleção fábrica do milênio -volume I**. Jaboticabal: Novos talentos, 2005.

HALEVI, G.; WEILL, R. D. **Principles of process planning: a logical approach**. Chapman & Hall, 1995.

HAN, J.; PRATT, M.; REGLI, W. C. Manufacturing features recognition from solid models: a status report. **IEEE Transactions on Robotics and Automation**, v. 16, n. 6, p.782-796, 2000.

HAN J.; REQUICHA, A. A. G. Feature recognition from CAD models. **IEEE Computer Graphics and Applications**, p. 80–94, 1998.

HARRELL, C.; TUMAY, K. Simulation made easy. **IIE Solutions**, p.39-41, July, 1997.

HARRELL, C. R.; GHOSH, B. K.; BOWDEN, R. **Simulation Using Promodel**. 2^a ed. McGraw-Hill, 2003.

HEARN, D.; BAKER, M. P. **Computer graphics – C version**. 2^a ed. Ed. Prentice Hall, 1997.

HOQUE, A. S. M.; SZECSI, T. Designing using manufacturing feature library. **Journal of Materials Processing Technology**, v. 201, p.204-208, 2008.

KELSICK, J.; VANCE, J. M.; BUHR, L.; MOLLER, C. Discrete Event Simulation Implemented in a Virtual Environment. **ASME Journal of Mechanical Design**, v. 125, n. 3, 428–433, 2003.

KELTON, W. D.; SADOWSKI, R. P.; SWETS, N. **Simulation with arena**. 5^a ed. New York: WCB/McGraw-Hill, 2009.

KHEIR, N. A. **Systems modeling and computer simulation**. 2^a.ed. New York: Marcel Dekker, Inc, 1996.

KIRNER, C.; SISCOOTTO, R. A. **Fundamentos de realidade virtual e aumentada**. Livro do Pré-Simpósio, IX Symposium on Virtual and Augmented Reality. Porto Alegre: SBC, 2007.

KIRSCH, F.; DOLLNER, J. OpenCSG: A library for image-base CSG rendering. **Proceedings of USENIX Annual Technical Conference**, Anaheim – CA, 2005.

KOTONYA, P.; SOMMERVILLE, I. **Requirements engineering: processes and techniques**. Chichester, UK: John Wiley & Sons, 1998.

KOSCIANSKI, A.; SOARES, M. **Qualidade de software**. 2ª. Ed. Rio de Janeiro: Novatec, 2007.

KUMAR, S.; SINGH, R. A short note on an intelligent system for selection of materials for progressive die components. **Journal of Materials Processing Technology**, Elsevier. v.182, p.456–461, 2007.

JACOB, L. J. **Avaliação de técnicas egocêntricas e equipamentos especiais em ambientes virtuais**. Dissertação de Mestrado. Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande do Sul, 2002.

JACOBSON, L. **Realidade virtual em casa**. Rio de Janeiro: Berkeley, 1994.

JUNG, C. F. **Metodologia Científica: Ênfase em pesquisa tecnológica**. 3ª ed. Santa Maria: CDROM, 2003.

JUNG, J. Y. Manufacturing cost estimation for machined parts based on manufacturing features. **Journal of Intelligent Manufacturing**, p. 227-238, 2002.

JUNG, J. Y.; CHOI, S. J.; MOON, D. H. Cost estimating system for machined parts. **Proceedings of International Conference on Production Research**, Bangkok, Thailand, 2000.

JUNKER, G. **Pro OGRE 3D Programming**. Apress, 2006.

LATTA, J. N.; OBERG, D. J. A Conceptual Virtual Reality Model. **IEEE Computer Graphics & Applications**, n.14, p.23-29, 1994.

LAW, A. M.; KELTON, W. D. **Simulation modeling and analysis**. 4ª.ed. New York: McGraw-Hill, 2000.

LAW, A. M.; MCCOMAS, M. G. Simulation of manufacturing systems. **Proceedings of the 1999 Winter Simulation Conference**, 1999.

LEE, H. Rule-based process planning by grouping features. **KSME International Journal**, v.18, n.12, p.2095-2103, 2004.

LESTON, J. **Virtual reality: the it perspective**. Computer Bulletin, june, p. 12-13, 1996.

LIMA, C. P. **Modelamento baseado em features em um conceito de projeto para fabricação e montagem.** Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina, 1994.

LOBÃO, E. C.; PORTO, A. J. V. Proposta para sistematização de estudos de simulação. Anais do **Anais** do XVII Encontro Nacional de Engenharia de Produção -ENEGEP, Piracicaba-SP, 1997.

LUCERO, A. G. R. **Um Método de otimização para a programação da manufatura em pequenos lotes.** Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina, 2001.

LUI, Z.; WANG, L. Sequencing of interacting prismatic machining features for process planning. **Computers in Industry**. n.58. pp. 295-303, 2007.

MACHADO, L. S.; CARDOSO, A. **Dispositivos de entrada e saída para sistemas de realidade virtual.** Livro de Pré-Simpósio. VII Symposium on Virtual Reality. Belém-PA.p.39-50, 2006.

MACHOVER, C.; TICE, S. E. *Virtual reality*. **IEEE Computer Graphics & Applications**, v. 14, n. 1, p.15-16, 1994.

MAGALHÃES, J. S. F. **Sistema variante para seleção de máquinas-ferramenta.** Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal do Paraná, 2008.

MAYER, R. J.; SU, C. J.; SUN, T.; WYSK, R. A. ECTOF: a Feature Representation Technique for Concurrent Engineering Applications. **Journal of Design and Manufacturing**, v.4, n.1, p.49-65, 1994.

MALDONADO, J. C. **Critérios pontenciais usos: um contribuição ao teste estrutural de software.** Tese de Doutorado. Programa de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, 1991.

MARTINS, P. G.; ALT, P. R. C. **Administração de materiais e recursos patrimoniais.** 2ª ed, São Paulo: Saraiva, 2006.

MARTZ, P. **OpenSceneGraph Quick Start Guide.** Skew Matrix Software: LLC, 1997.

MATTOS, M. A. C. **Sistemas de informação: uma visão executiva.** São Paulo: Saraiva, 2005.

MAZIERO, N. L. **Um sistema computacional inteligente de suporte ao projeto, manufatura e montagem de peças baseado em features: uma abordagem com sistemas especialistas**. Tese de Doutorado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina, 1998.

MINE, M. **Virtual enviroment interaction techniques**. UNC Chapel Hill CS: Technical Report TR. 95-018, 1995.

MOREIRA, D. A. **Administração da produção e operações**. 2ª ed. São Paulo: Cengage Learning, 2008.

MOURA, R. A. **Sistemas e técnicas de movimentação e armazenagem de materiais**. São Paulo: IMAM, 1997.

MYERS, G. **The art of software testing**. John Wiley and Sons, 1979.

NAYLOR, T. H.; BALINTFY, J. L.; BURDICK, D. S.; CHU, K. **Computer simulation techniques**. New York: John Wiley & Sons, 1971.

NETTO, A. V. **Prototipação de um torno CNC utilizando realidade virtual**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Escola de Engenharia de São Carlos da Universidade de São Paulo, 1998.

O'KANE, J. F.; SPENCELEY, J. R.; TAYLOR, R. Simulation as an essential tool for advanced manufacturing technology problems. **Journal of Materials Processing Technology**, v.107, p.412-424, 2000.

OLIVEIRA, J. E.; RANGEL, J. J. A.; VIANNA, D. S. Um modelo de simulação para análise de usinagem metal mecânica do tipo job shop – um estudo de caso. **Anais do XVI Simpósio de Engenharia de Produção – SIMPEP**, Bauru-SP, 2009.

PAGE-JONES, M. **Fundamentals of object-oriented design in UML**. New York: Addison-Wesley, 2000.

PANGRÁCIO, M. L. **Análise da qualidade de furos realizados por fresamento helicoidal interpolado**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Paraná, 2003.

PALOMINO, R. C. **Um modelo baseado em redes de petri para o planejamento e a programação da produção em ambientes do tipo job shop**. Tese de Doutorado. Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina, 2001.

PASSOS, C. A. S. **Simulação de sistemas de manufatura**. Tese de Doutorado. Programa Pós-Graduação em Engenharia Elétrica da Universidade Estadual de Campinas, 1986.

PEGDEN, C. D.; SHANNON, R. E.; SADOWSKI, R. P. **Introduction to simulation using siman**. New York: McGraw-Hill, 1990.

PERERA, T.; LIYANAGE, K. Methodology for rapid identification and collection of input data in the simulation of the manufacturing systems. **Simulation Practice and Theory**, n.7, p.645–56, 2000.

PFLEEGER, S. L.; ATLEE, J.M. **Software engineering – theory and practice**. Prentice Hall: USA, 2006.

PIDD, M. **Modelagem empresarial: ferramentas para tomada de decisão**. Porto Alegre: Artes Médicas, 1998.

PIMENTEL, K.; TEIXEIRA, K. **Virtual reality - through the new looking glass**. 2.d. New York: McGraw-Hill, 1995.

PINHO, A. F. **Metodologia para utilização de algoritmos genéticos em modelos de simulação computacional em ambientes de manufatura**. Tese de Doutorado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Estadual Paulista, 2008.

PORTO, A. J. V.; PALMA, J. G. Da **Manufatura Virtual à Simulação e à Realidade Virtual**. In: *Fábrica do Futuro: entenda hoje como sua indústria vai ser amanhã*, Editora Banas, 2000.

POZO, H. **Administração de recursos materiais e patrimoniais: uma abordagem logística**. 4ª ed, São Paulo: Atlas, 2007.

PRADO, D. S. **Teoria das filas e da simulação**. 2 vol. Belo Horizonte: Editora de Desenvolvimento Gerencial, 1999.

PRESSMAN, R. S. **Engenharia de Software**. 6ª ed. São Paulo: McGraw-Hill, 2006.

PRITSKER, A. A. B. **Introduction to simulation and SLAM II**. New York : John Wiley & Sons, 1986.

REINERS, D.; VOSS, G.; BEHR, J. **Opensg: basic concepts**. In: OpenSG Forum, 2002.

RESENDE, M. O. **Princípios de processos de produção**. *Volume 1 e 2*. Apostila do Departamento de Engenharia Mecânica. São Carlos: Universidade de São Paulo, 1993.

REZENDE, D. F. **Planejamento de processos de fabricação assistido por computador através de um sistema especialista baseado na tecnologia de features: um modelo de desenvolvimento voltado para a realidade industrial**. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia Mecânica da Universidade Federal de Santa Catarina, 1996.

RISING, L.; JANOFF, N. S. The scrum software development process for small teams. **IEEE Software**, v.17, n.4, p.26-32, 2000.

ROSSETI, M.D.; AYLOR, B.; JACOBY, R.; PROROCK, A.; WHITE, A. Simfone: an object-oriented simulation framework. In: **Proceedings** of the 2000 Winter Simulation Conference, p.1855-1864, 2000.

ROSSETI, M. D.; CHAN, H. T. A prototype object-oriented supply chain simulation framework. In: **Proceedings** of the 2003 Winter Simulation Conference, p.1612-1620, 2003.

RYAN, J.; HEAVEY, C. Process modeling for simulation. **Computers in Industry**, v.57, p.437-450, 2006.

SALOMONS, O. W.; HOUTEN F. J. A. M.; VAN KALS H. J. J. Review of research in feature-based design. **Journal of Manufacturing Systems**, v.12, n.2, 113-132, 1993.

SCHACH, S. R. **Engenharia de software: os paradigmas clássico & orientado a objetos**. São Paulo: McGraw-Hill, 2009.

SCHAEFFER, B.; GOUDESEUNE, C. Syzygy: native PC cluster VR. **IEEE VR Conference**, 2003.

SCHRIBER, T. J. **An introduction to simulation using GPSS/H**. New York: John Wiley & Sons, 1991.

SCHUTZER, K.; SCHULZ, H.; WEIDMANN, A. *Design by Features: Feature Based Design Prismatic Parts*. In: **Proceedings** of the Workshop on Flexibility through Integrated Design, Process Planning and Scheduling, CETIM, Selins, 1992.

SEMENTILLE, A. C. **A utilização da arquitetura CORBA na construção de ambientes virtuais distribuídos**. Tese de Doutorado. Programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, 1999.

SKINNER, W. Manufacturing – missing link in corporate strategy. Boston: **Harvard Business Review**, v. 47, n. 3, p. 136-145, 1969.

SHAH, J. J.; MANTYLA, M. **Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications**. New York: John Wiley Sons Inc, 1995.

SHAH, J. J.; SHEN, Y.; SHIRUR, A. Determination of Machining Volumes From Extensible Sets of Design Features. **Advances in Feature Based Manufacturing**. Amsterdam: Elsevier, p.129-157, 1994.

SHANNON, R. E. Introduction to the art and science of simulation. In: **Proceedings** of the 1998 Winter Simulation Conference, p.07-14, 1998.

SHEU, J. J. A computer integrated manufacturing system for rotational parts. **International Journal of Computer Integrated Manufacturing**, n.11, p.538–547, 1998.

SHUKLA, C.; VAZQUEZ, M.; CHEN, F. Virtual Manufacturing: an Overview. **Computers and Industrial Engineering**, n.31, p.79-82, 1996.

SILVA, E. L.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. Florianópolis: LED/UFSC, 2005.

SLACK, N. **Vantagem Competitiva em Manufatura: atingindo competitividade nas operações industriais**. São Paulo: Atlas, 1993.

SOARES, L. P.; CABRAL, M. C.; ZUFFO, M. K. **Sistemas avançados de realidade virtual**. Porto Alegre: SBC, 2006.

SOARES, L. P.; ZUFFO, M. K. JINX: an X3D browser for VR immersive simulation based on clusters of commodity computers. Ninth International **Conference** on 3D Web Technology, 2004.

SOMMERVILLE, I. **Engenharia de software**. 9ª ed. São Paulo: Pearson Prentice Hall, 2011.

STAIR, R. M.; REYNOLDS, G. W. **Princípios de sistemas de informação**. 6.ed. São Paulo: Cengage Learning, 2008.

STARK, G.; OMAN, P.; SKILLICORN, A.; AMEELE, R. An examination of the effects of requirements changes on software maintenance releases. **Journal of Software Maintenance Research and Practice**, v. 11, p.293-309, 1999.

STEMMER, G. E. **Ferramentas de Corte II: brocas, alargadores, ferramentas de roscas, fresas, brochas, rebolos e abrasivos**. Florianópolis: Editora da UFSC, 1992.

STOETERAU, R. L. **Apostila de tribologia**. Florianópolis: Departamento de Engenharia Mecânica, Universidade Federal de Santa Catarina. p.71-92, 2004.

STRACK, J. **GPSS: Modelagem e simulação de sistemas**. Rio de Janeiro: LTC, 1984.

SUDARSAN, R.; FENVES, S. J.; SRIRAM, R. D.; WANG, F. A product information modeling framework for product lifecycle management. **Computer-Aided Design**, n.37. p.1399–1411, 2005.

TOMPKINS, J. A.; WHITE, J. A.; BOZER, Y. A.; TANCHOCO, J. M. A. **Facilities Planning**. 4ª ed. New Jersey: John Wiley & Sons, 2010.

TIBERTI, A. J. **Desenvolvimento do software de apoio ao projeto de arranjo físico de fábrica baseado em um framework orientado a objeto**. Tese de Doutorado.: Programa de Pós-Graduação em Engenharia Mecânica da Universidade de São Paulo, 2004.

TORGA, B. L. M.; MONTEVECHI, J. B.; PINHO, A. F. Modelagem, simulação e otimização em sistemas puxados de manufatura. XIII **Simpósio** de Engenharia de Produção – SIMPEP, Bauru – SP, 2006.

TORI, R.; KIRNER, C. **Fundamentos e tecnologia de realidade virtual e aumentada**. Porto Alegre: SBC. v.1, 422p, 2006.

TRENT, E. M.; WRIGHT, P. K. **Metal cutting**. 2ª ed. Woburn: Butterworth-Heinemann, 446 p, 2000.

VIEIRA JUNIOR, M.; LIBARDI, R.; CANCELIERI, H. A.; LIMA, A. Como o calor pode afetar a integridade superficial das peças. **Revista Máquinas e Metais**, fevereiro, p. 28-36, 1999.

WEINERT, K.; ADAMS F. J.; BIERMANN, D.; THAMKE, D. Alargadores de corte único: corte mais eficiente, além de vantagens econômicas. **Revista Máquinas e Metais**, São Paulo, n.392, p.38-57, 1998.

YANG, F. C.; WANG, M. T. An object-oriented feature-based computer-aided design system for concurrent engineering. **IEE Emerging Technologies and Factory Automation**, p. 393-398, 1992.

ZHANG, F.; HUAMING, L.; CHUNHUA, S. The disposal to face-based representation of design/manufacturing feature integration. Vehicle Electronics Conference. **Proceedings of the IEEE International**, v.1, p. 414-417, 1999.

ZHOU, X.; QIU, Y.; HUA,G.; WANG,H.; RUAN, X. A feasible approach to the integration of CAD and CAPP. **Computer-Aided Design**, v.39, p.324–338, 2007.

APÊNDICE A

Neste trabalho os arquivos com os resultados da simulação foram salvos no formato XML. Os dados que constam nesse arquivo e um exemplo de arquivo composto por cinco recursos são mostrados a seguir.

1. Nome - nome do recurso de fabricação ou transporte
2. Tempo Proc - tempo de processamento da peça (fabricação ou transporte)
3. Custo Proc - custo de processamento da peça (fabricação ou transporte)
4. Tempo Fila - tempo de espera da peça em cada recurso
5. Util.Recurso - percentual de utilização de cada recurso
6. Quantidade - quantidade de peças simuladas
7. Tempo Médio Processamento - tempo de processamentos médio por peça
8. Custo Médio Processamento - custo de processamento médio por peça
9. Tempo Total Fila - tempo de fila médio por peça
10. Tempo Total - soma do tempo total de processamento com o tempo total de fila
11. Tempo Médio - tempo médio que uma peça leva para ser produzida

```
<ModelRoot>
<ModelDataItem Nome="Fresa" Tempo Proc.="704.38" Custo Proc.="224.35" Tempo
Fila="85.19" Util.Recurso="96.49" Quantidade="100" Tempo M. Proc.="13.62" Custo M.
Proc.="4.59" Tempo M. Fila="0.85" Tempo Total="1447.99" Tempo Médio="14.47"/>

<ModelDataItem Nome="Esteira" Tempo Proc.="20.40" Custo Proc.="4.08" Tempo Fila="0"
Util.Recurso="2.79"/>

<ModelDataItem Nome="Torno" Tempo Proc.="502.07" Custo Proc.="178.78" Tempo Fila="0"
Util.Recurso="68.77"/>

<ModelDataItem Nome="Operario" Tempo Proc.="9.6" Custo Proc.="0.76" Tempo Fila="0"
Util.Recurso="1.315"/>

<ModelDataItem Nome="Furadeira" Tempo Proc.="126.33" Custo Proc.="51.05" Tempo Fila="0"
Util.Recurso="17.30"/>
</ModelRoot>
```

APÊNDICE B

Neste trabalho os dados gerados pelos gráficos (pizza, linha, barra) da interface resultados podem ser salvos no formato XML. Os dados podem ser relativos ao tempo ou ao custo de fabricação de cada recurso. São mostrados a seguir um gráfico salvo com dados de tempo e outro gráfico salvo com dados de custos.

1. Tempo de Fabricação por Recurso

```
<ModelRoot>
  <ModelDataItem Tempo="4.27" Nome="Brochadeira"/>
  <ModelDataItem Tempo="2.10" Nome="Esteira"/>
  <ModelDataItem Tempo="5.69" Nome="Furadeira de Coluna"/>
  <ModelDataItem Tempo="3.23" Nome="Operario"/>
  <ModelDataItem Tempo="13.12" Nome="Torno Universal"/>
</ModelRoot>
```

2. Custo de Fabricação por Recurso

```
<ModelRoot>
  <ModelDataItem Custo="2.08" Nome="Brochadeira"/>
  <ModelDataItem Custo="0.79" Nome="Esteira"/>
  <ModelDataItem Custo="3.94" Nome="Furadeira de Coluna"/>
  <ModelDataItem Custo="1.03" Nome="Operario"/>
  <ModelDataItem Custo="8.21" Nome="Torno Universal"/>
</ModelRoot>
```

APÊNDICE C

Diagramas de Casos de Uso (Uso Case)

O diagrama caso de uso (Use Case) é um diagrama baseado em cenários para obtenção de requisitos. Ele mostra um conjunto de casos de uso, atores e seus relacionamentos.

Os diagramas de caso de uso são importantes principalmente para a organização e modelagem dos comportamentos de um sistema. Eles fazem com que sistemas, subsistemas e classes fiquem acessíveis e compreensíveis, por apresentarem uma visão externa sobre como esses elementos podem ser utilizados no contexto.

Neste trabalho os diagramas de caso de uso foram utilizados da seguinte forma:

1.1 Definições dos atores

1.2 Lista de Eventos

No	Descrição	Eventos	Caso de Uso	Resposta
01	Administrador solicita cadastro do operador	DadosOperador	Cadastrar	msg01
02	Operador acessa o Simulador	DadosAcesso	ValidarLoginSenha	msg02
03	Operador altera a senha	DadosSenha	AlterarSenha	msg03
04	Operador cria cadastro Peca	DadosPeca	CadastrarPeca	msg04
05	Operador cria cadastro <i>Feature</i>	Dados Feature	CadastrarFeature	msg05
06	Operador cria cadastro Recurso	DadosRecurso	CadastrarRecurso	msg06
07	Operador faz a Modelagem	DadosModelagem	Modelagem	msg07
08	Operador faz Simulação	DadosSimulacao	Simular	msg08
9	Operador exhibi Relatórios	DadosExibicao	ExibirResultados	msg09
10	Operador exporta Relatórios	DadosExportacao	ExportarResultados	msg10

01 – Cadastrar

Este Caso de Uso é responsável em cadastrar um novo operador do ambiente de simulação.

Cenário Sucesso Principal:

- (a) O administrador solicita o cadastro do novo operador na interface.
- (b) O administrador pede para gravar o novo *login* e senha.
- (c) O sistema verifica se o novo *login* e senha não existem e armazena.
- (d) O sistema envia Msg01 "Cadastro gerado com sucesso".

Cenário Alternativo 1:

Passo (b). O administrador pede para fechar.

- (b.1) O sistema aborta a operação sem gravar os dados.

Cenário Alternativo 2:

Passo (c). O sistema avisa ao usuário Msg01 "Dados inconsistentes".

- (c.1) O administrador corrige os dados.
- (c.2) Vai para o Curso Normal passo (b).

Cenário Alternativo 2.1:

Passo (c.1) O administrador pede para cancelar.

- (c.1.1) O sistema aborta a operação sem gravar os dados.

02 – ValidarLoginSenha

Este Caso de Uso permite e verifica o acesso do operador do sistema de manufatura no ambiente de simulação.

Cenário Sucesso Principal:

- (a) O operador informa o *login* e a senha.
- (b) O sistema verifica se o operador é autorizado.
- (c) O sistema cria uma instância do operador e emite Msg02"Acesso Liberado".

Cenário Alternativo 1:

Passo (b). O operador digitou *login* errado.

- (b.1) O sistema emite a Msg02 "*Login Incorreto*".
- (c) Abandona o caso de uso.

Cenário Alternativo 2:

Passo (b). O operador digitou senha errada.

- (b.1) O sistema emite a Msg02 "*Senha Incorreta*".
- (c) Abandona o caso de uso.

Cenário Alternativo 3:

Passo (b) O operador digitou o *login* e a senha errados.

- (b.1) O sistema emite a Msg02 "*Login e Senha Incorretos*".
- (c) Abandona o caso de uso.

03 – Alterar Senha

Este Caso de Uso é responsável por alterar uma senha do operador do ambiente de simulação já existente.

Cenário Sucesso Principal:

- (a) O operador solicita alteração de senha na interface.
- (b) O operador digita a senha antiga, a senha nova e confirma a senha nova.
- (c) O operador pede para gravar a senha.
- (d) O sistema verifica a não existência da senha.
- (e) O sistema envia Msg03 "Senha alterada com sucesso".

Cenário Alternativo 1:

Passo (c) O operador pede para fechar.

- (c.1) O sistema aborta a operação sem gravar os dados.

Cenário Alternativo 2:

Passo (d) O sistema avisa ao usuário Msg03 "Senha incorreta. Entre novamente com a senha".

- (d.1) O operador digita novamente a senha.
- (d.2) vai para o Curso Normal passo (c).

Cenário Alternativo 2.1:

Passo (d.1) O operador pede para cancelar.

- (d.1.1) O sistema aborta a operação sem gravar os dados.

04 – CadastrarPeca

Este Caso de Uso permite o cadastro das peças com suas informações gerais na base de dados do ambiente de simulação.

Cenário Sucesso Principal:

- (a) O operador preenche os dados referente à peça na interface.
- (b) O operador pede para gravar.
- (c) O sistema verifica a entrada de dados.
- (d) O sistema cria um arquivo Peca no banco de dados.
- (e) O sistema envia Msg04 "Arquivo gerado com sucesso".

Cenário Alternativo 1:

Passo (b) O operador pede para cancelar.

- (b.1) O sistema aborta a operação sem gravar os dados.

Cenário Alternativo 2:

Passo (c) O sistema avisa ao usuário Msg04 "Dados inconsistentes"

- (c.1) O operador conserta os dados.
- (c.2) vai para o Curso Normal passo (b).

Cenário Alternativo 2.1:

Passo (c.1) O operador pede para cancelar.

- (c.1.1) O sistema aborta a operação sem gravar os dados.

05 – CadastrarFeature

Este Caso de Uso permite o cadastro as *features* a serem utilizadas na produção da peça.

Cenário Sucesso Principal:

- (a) O operador preenche os dados da feature na interface.
- (b) O operador pede para gravar.
- (d) O sistema verifica a entrada de dados.
- (e) O sistema cria uma *Feature* no banco de dados.
- (f) O sistema envia Msg05 "Arquivo gerado com sucesso".

Cenário Alternativo 1:

Passo (c). O operador pede para cancelar.

- (c.1) O sistema aborta a operação sem gravar os dados.

Cenário Alternativo 2:

Passo (d). O sistema avisa ao usuário Msg05 "Dados inconsistentes".

- (d.1) O operador corrige os dados.
- (d.2) vai para o Curso Normal passo (b).

Cenário Alternativo 2.1:

Passo (d.1) O operador pede para cancelar.

- (d.1.1) O sistema aborta a operação sem gravar os dados.

06 – Cadastrar Recurso

Este Caso de Uso permite o cadastro das informações dos recursos a serem utilizados na produção de uma peça.

Cenário Sucesso Principal:

- (a) O operador preenche os dados do recurso na interface.
- (c) O operador pede para gravar.
- (d) O sistema verifica a entrada de dados.
- (e) O sistema cria um Recurso no banco de dados.
- (f) O sistema envia Msg06 "Recurso salvo com sucesso".

Cenário Alternativo 1:

Passo (c). O operador pede para cancelar.

- (c.1) O sistema aborta a operação sem gravar os dados.

Cenário Alternativo 2:

Passo (d). O sistema avisa ao usuário Msg06 "Dados inconsistentes".

- (d.1) O operador corrige os dados.
- (d.2) vai para o Curso Normal passo (b).

Cenário Alternativo 2.1:

Passo (d.1) O operador pede para cancelar.

- (d.1.1) O sistema aborta a operação sem gravar os dados.

07 – Modelar

Este Caso de Uso é responsável em executar a modelagem do sistema de manufatura.

Cenário Sucesso Principal:

- (a) Na interface de modelagem, o operador monta a lista de simulação da peça a partir dos dados anteriormente cadastrados no banco de dados.
- (b) O operador após concluir a modelagem da lista pede para simular.
- (c) O sistema cria um arquivo com uma lista com os recursos e *features* a serem simuladas.
- (d) O sistema envia Msg07 "Iniciar simulação?". Caso o operador deseje realizar a simulação aceita pressionando "sim" na caixa de texto. Caso o operador

Cenário Alternativo 1:

Passo (b). O operador pede para cancelar.

- (b.1) O sistema aborta a operação sem iniciar a simulação.

Cenário Alternativo 2:

Passo (c). O sistema avisa ao usuário Msg07 "Dados inconsistentes".

- (c.1) O operador corrige os dados da lista de simulação.
- (c.2) vai para o Curso Normal passo (a).

Cenário Alternativo 2.1:

Passo (c.1) O operador pede para cancelar.

- (c.1.1) O sistema aborta a operação sem realizar a simulação.

08 – Simular

Este Caso de Uso é responsável em executar a simulação do sistema de manufatura testando uma gama de variáveis distintas.

Cenário Sucesso Principal:

- (a) Na interface simulação, o operador fornece a lista de simulação das peças.
- (b) O sistema cria uma instância a partir do núcleo de simulação com os dados da modelagem e chama a interface gráfica de visualização 3D do simulador.
- (d) Após a exibição da simulação, o sistema cria um arquivo com todos os dados gerados (resultados) e emite a Msg08 "Sucesso na Simulação".

Cenário Alternativo 1:

Passo (a) A lista de simulação não existe ou não esta consistente.

- (a.1) O sistema emite a Msg08 "Lista de Simulação inconsistente, Simulação abordada".
- (a.2) O operador abre um novo formulário de modelagem da simulação.
- (a.3) O sistema continua no Curso Normal.

Cenário Alternativo 2:

Passo (a) A lista de simulação não existe ou não esta consistente.

- (a.1) O sistema emite a Msg08 " Lista de Simulação inconsistente, Simulação abordada".
- (a.2) O operador cancela a simulação.
- (a.3) O sistema fecha a simulação e retorna a interface principal.

09 – Exibir Resultados

Este Caso de Uso é responsável exibir os resultados da simulação realizada.

Cenário Sucesso Principal:

- (a) O operador acessa a interface resultados solicita a exibição da tabela com os dados da simulação.
- (b) Após a exibição dos resultados, o sistema emite a Msg09 "Exibição de Resultados com Sucesso".

Cenário Alternativo 1:

Passo (a). O arquivo resultados não existe ou não pode ser aberto.

(a.1) O sistema emite a Msg09 "O resultado não pode ser aberto".

(a.2) O operador acessa novamente a interface de resultado e seleciona um novo arquivo.

(a.3) O sistema continua no Curso Normal.

Cenário Alternativo 2:

Passo (a). O arquivo não existe ou não pode ser aberto.

(a.1) O sistema emite a Msg09 "O resultado não pode ser aberto".

(a.2) O operador cancela abertura dos resultado.

(a.3) O sistema fecha o arquivo e sai da interface resultado.

10 – Exportar Resultados

Este Caso de Uso é responsável por exportar os resultados da simulação realizada em forma de arquivo com extensão CSV.

Cenário Sucesso Principal Exibir Relatório:

- (a) O operador acessa a interface resultados, abre a tabela com os dados da simulação e solicita exportar para arquivo com extensão CSV.
- (b) Após a exportação do relatório, o sistema emite a Msg10 "Resultado Exportado com Sucesso".

Cenário Alternativo 1:

Passo (a). O arquivo resultado não existe ou não pode ser aberto.

(a.1) O sistema emite a Msg10 "O resultado não pode ser aberto".

(a.2) O operador acessa novamente a interface de resultados e seleciona um novo arquivo.

(a.3) O sistema continua no Curso Normal.

Cenário Alternativo 2:

Passo (a). O arquivo não existe ou não pode ser aberto.

(a.1) O sistema emite a Msg10 "O resultado não pode ser aberto".

(a.2) O operador cancela a exportação do resultado.

(a.3) O sistema fecha o arquivo e sai da interface resultados.

APÊNDICE D

Diagramas de Sequência

Neste apêndice são detalhados os diagramas de sequência utilizados na modelagem para o desenvolvimento do ambiente de simulação

01 – Acessar o Sistema

Para acessar o ambiente de simulação o usuário deve ter seu *login* e senha cadastrados. Esses dados são armazenados para que o usuário possa acessar o sistema nas próximas vezes sem a necessidade de realizar um novo cadastro. O responsável pelo cadastro é o administrador do sistema.

02 – Cadastrar Elementos no Banco De Dados

O banco de dados é a forma de entrada dos dados no sistema. Esses dados podem ser inseridos, alterados ou removidos. Os dados armazenados são utilizados na modelagem, na simulação e na cena tridimensional. As três tabelas constantes no banco de dados são a de peças, a de *features* e a de recursos. Todas as três tabelas seguem a mesma forma de cadastrar suas atividades.

03 – Modelagem e Simulação

A modelagem e simulação são duas atividades fundamentais do ambiente de simulação. A modelagem é responsável por ler os dados contidos na base de dados e organizá-los para a simulação. Para realizar a modelagem deve ser fornecido os dados que são: a quantidade de peças a ser fabricadas, o tempo entre a chegada de peças no sistema, o tempo de simulação e uma lista de simulação que corresponde à sequência de atividades que serão realizadas ao longo da simulação.

A simulação é responsável por coordenar as diversas atividades do sistema a partir dos dados de entrada fornecidos pela modelagem e pelo banco de dados e transforma esses dados em resultados para análise de sistemas de manufatura.

04 – Resultados

Os resultados são as saídas do ambiente de simulação. Os resultados são relativos ao tempo e ao custo. Eles podem ser visualizados no próprio sistema ou exportados para outros programas nas extensões XML e CSV.

APENDICE E

Neste apêndice são detalhadas as declarações das classes que implementam o ambiente de simulação. As classes são divididas em três grandes pacotes: *Core*, *GUI* e *Scene*.

1. Core

1. Classe DBManager.h

```
#ifndef DBMANAGER_H
#define DBMANAGER_H
#include "Object.h"
#include "Resource.h"
#include "Feature.h"
class Piece;
class Resource;
class Feature;
#include <string>
#include <QSqlTableModel>
#include <QSqlDatabase>
class DBManager
{
public:
    static void createNewDB(const std::string &dbName);
    static Piece* loadPiece(const std::string &objName);
    static Resource* loadResource(ResourceType type, const std::string &objName);
    static Feature* loadFeature(FeatureType type, const std::string &objName);
private:
    QSqlDatabase db;
};
#endif // DBMANAGER_H
```

2. Classe Feature.h

```
#ifndef FEATURE_H
#define FEATURE_H
#include "Object.h"
#include <list>
#include <string>
enum FeatureType
{
    CIRCULAR_FEATURE,
    ROTACIONAL_FEATURE,
    PRISMATICA_FEATURE,
    CAVIDADE_FEATURE
};
class Feature : public Object
{
public:
    Feature(const Properties& prop, FeatureType featType);
    virtual Properties getProperties();
    virtual void setProperties(const Properties &);
```

```

    virtual float volumeFeature() = 0;
    virtual float areaFeature() = 0;
    FeatureType type;
    std::string m_name;
};
#endif // FEATURE_H

```

3. Classe Piece.h

```

#ifndef PIECE_H
#define PIECE_H
#include "Object.h"
#include <list>
#include <vector>
#include <string>
class Feature;
class Piece : public Object
{
public:
    enum Status{
        WAITING,
        RUNNING
    };
    Piece(const Properties& prop);
    virtual void setProperties(const Properties &);
    virtual Properties getProperties();
    float volumePiece();
private:
    float PieceVol;
    std::list<Feature*> m_features;
    std::string m_name;
    float m_largura;
    float m_comprimento;
    float m_altura;
    Status m_status;
    std::vector<int> m_waitingTimes;
    std::vector<int> m_runningTimes;
};
#endif // PIECE_H

```

4. Classe Resource.h

```

#ifndef RESOURCE_H
#define RESOURCE_H
#include "Object.h"
#include <string>
class Piece;
enum ResourceType
{
    FABRICATION_RESOURCE,
    TRANSPORT_RESOURCE
};
class Resource : public Object
{
public:
    Resource(const Properties& prop, ResourceType resType);
    virtual void setProperties(const Properties &);
    virtual Properties getProperties();
    virtual float computeTime(Piece*) = 0;
    virtual float computeCost(Piece*) = 0;
};

```

```

    ResourceType type;
    std::string m_name;
};
#endif // RESOURCE_H

```

5. Classe Simulation.h

```

#ifndef SIMULATION_H
#define SIMULATION_H
#include <list>
#include <string>
#include "Report.h"
class Piece;
class Resource;
class Feature;
class Report;
class SimulationDialog;
class Simulation
{
public:
    Simulation();
    void setPiece(Piece*);
    void setArrivalTime(float);
    void setQuantity(int);
    void setSimulationTime(float);
    void addResource(Resource*);
    Report simulate();
    Piece* m_piece;
    std::list<Resource*> m_resources;
private:
    float m_arrival;
    int m_quantity;
    float m_simulationTime;
};
#endif // SIMULATION_H

```

6. Classe Report.h

```

#ifndef REPORT_H
#define REPORT_H
#include <string>
#include <vector>
class Report
{
public:
    Report();
    std::vector<std::string> resourceNames;
    std::vector<float> resourceTimes;
    std::vector<float> resourceCosts;
    std::vector<float> resourceQueueTimes;
    std::vector<float> utilizationRate;
    int Quantity;
    float totalTime;
    float totalCost;
    float totalQueue;
    float averageTime;
    float SimulationTime;
};
#endif // REPORT_H

```

7. Classe FabricationResource.h

```

#ifndef FABRICATIONRESOURCE_H
#define FABRICATIONRESOURCE_H
#include "Resource.h"
#include "Feature.h"
#include "Properties.h"
#include <list>
#include <string>
class Feature;
class Simualtion;
class FabricationResource : public Resource
{
public:
    FabricationResource(const Properties& prop, ResourceType resType);
    virtual void setProperties(const Properties &);
    void setQuantity(int);
    virtual Properties getProperties();
    void addFeature(Feature*);
    virtual float computeTime(Piece*);
    virtual float computeCost(Piece*);
    float TempoFab;
    std::list<Feature*> m_features;
private:
    int m_quantity;
    float m_salarioOp;
    float m_salarioMaq;
    float m_custoFerr;
    float m_diamCorte;
    float m_profCorte;
    float m_compCorte;
    float m_avanco;
    float m_velCorte;
    float m_rotacao;
    float m_taxaAcab;
    float m_temImpr;
    float m_vidaFerr;
    float m_trocaFerr;
};
#endif // FABRICATIONRESOURCE_H

```

8. Classe TransportResource.h

```

#ifndef TRANSPORTRESOURCE_H
#define TRANSPORTRESOURCE_H
#include "Properties.h"
#include "Resource.h"
#include "Piece.h"
class Piece;
class TransportResource : public Resource
{
public:
    TransportResource(const Properties& prop, ResourceType resType);
    virtual void setProperties(const Properties &);
    virtual Properties getProperties();
    virtual float computeTime(Piece*);
    virtual float computeCost(Piece*);
    float TempoMovi;
private:
    Piece piece;

```

```

float m_velMovi;
float m_distMovi;
float m_cusMovi;
};
#endif // TRANSPORTRESOURCE_H

```

9. Classe PrismaticFeature.h

```

#ifndef PRISMATICFEATURE_H
#define PRISMATICFEATURE_H
#include "Feature.h"
#include <string>
class PrismaticFeature : public Feature
{
public:
    PrismaticFeature(const Properties& prop, FeatureType featType);
    virtual float volumeFeature();
    virtual float areaFeature();
    virtual Properties getProperties();
    virtual void setProperties(const Properties &);
private:
    int m_id;
    float m_largura;
    float m_comprimento;
    float m_altura;
    float m_posicaoX;
    float m_posicaoY;
    float m_posicaoZ;
    int m_orientacao;
};
#endif // PRISMATICFEATURE_H

```

10. Classe RevolvingFeature.h

```

#ifndef REVOLVINGFEATURE_H
#define REVOLVINGFEATURE_H
#include "Feature.h"
#include <string>
class RevolvingFeature : public Feature
{
public:
    RevolvingFeature(const Properties& prop, FeatureType featType);
    virtual float volumeFeature();
    virtual float areaFeature();
    virtual Properties getProperties();
    virtual void setProperties(const Properties &);
private:
    int m_id;
    float m_altura;
    float m_diametro;
    float m_posicaoX;
    float m_posicaoY;
    float m_posicaoZ;
    int m_orientacao;
};
#endif // REVOLVINGFEATURE_H

```

11. Classe RotationalFeature.h

```

#ifndef ROTATIONALFEATURE_H

```



```

#define ROTATIONALFEATURE_H
#include "Feature.h"
#include <string>
class RotationalFeature : public Feature
{
public:
    RotationalFeature(const Properties& prop, FeatureType featType);
    virtual float volumeFeature();
    virtual float areaFeature();
    virtual Properties getProperties();
    virtual void setProperties(const Properties &);
private:
    int m_id;
    float m_diaMaior;
    float m_diaMenor;
    float m_largura;
    float m_posicaoX;
    float m_posicaoY;
    float m_posicaoZ;
    int m_orientacao;
};
#endif // ROTATIONALFEATURE_H

```

12 Classe SlabFeature.h

```

#ifndef SLABFEATURE_H
#define SLABFEATURE_H
#include "Feature.h"
#include <string>
class SlabFeature : public Feature
{
public:
    SlabFeature(const Properties& prop, FeatureType featType);
    virtual float volumeFeature();
    virtual float areaFeature();
    virtual Properties getProperties();
    virtual void setProperties(const Properties &);
private:
    int m_id;
    float m_largura;
    float m_comprimento;
    float m_altura;
    float m_posicaoX;
    float m_posicaoY;
    float m_posicaoZ;
    int m_orientacao;
};
#endif // SLABFEATURE_H

```

2.GUI

1. Classe MainWindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
namespace Ui {
    class MainWindow;
}

```

```

}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void on_actionEditPiece_triggered();
    void on_action_Open_triggered();
    void on_actionEditFeature_triggered();
    void on_actionEditResource_triggered();
    void on_action_Configure_triggered();
    void on_action_New_triggered();
    void on_actionHelp_triggered();
    void on_actionAbout_triggered();
    void on_actionReport_triggered();
    void on_actionNovo_triggered();
    void on_actionAlterar_Senha_triggered();
    void on_actionExit_triggered();
private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H

```

2. Classe PieceEditor.h

```

#ifndef PIECEEDITOR_H
#define PIECEEDITOR_H
#include <QWidget>
#include "Editor.h"
class QDataWidgetMapper;
namespace Ui {
class PieceEditor;
}
class PieceEditor : public QWidget, public Editor
{
    Q_OBJECT

public:
    explicit PieceEditor(QSqlRelationalTableModel*, QWidget *parent = 0);
    ~PieceEditor();
    virtual void edit(int index);
    virtual QWidget* widget();

private slots:
    void on_clearButton_clicked();
    void on_saveButton_clicked();
private:
    Ui::PieceEditor *ui;
    QDataWidgetMapper* mapper;
};
#endif // PIECEEDITOR_H

```

3. Classe FeatureEditor.h

```

#ifndef FEATUREEDITOR_H
#define FEATUREEDITOR_H
#include <QWidget>
#include "Editor.h"

```

```

class QDataWidgetMapper;
namespace Ui {
class FeatureEditor;
}
class FeatureEditor : public QWidget, public Editor
{
    Q_OBJECT

public:
    explicit FeatureEditor(QSqlRelationalTableModel*, QWidget *parent = 0);
    ~FeatureEditor();
    virtual void edit(int index);
    virtual QWidget* widget();
private slots:
    void on_clearButton_clicked();
    void on_saveButton_clicked();

private:
    Ui::FeatureEditor *ui;
    QDataWidgetMapper* mapper;
};
#endif // FEATUREEDITOR_H

```

4. Classe ResourceEditor.h

```

#ifndef RESOURCEEDITOR_H
#define RESOURCEEDITOR_H
#include <QWidget>
#include "Editor.h"
class QDataWidgetMapper;
namespace Ui
{
class ResourceEditor;
}
class ResourceEditor : public QWidget, public Editor
{
    Q_OBJECT

public:
    explicit ResourceEditor(QSqlRelationalTableModel*, QWidget *parent = 0);
    ~ResourceEditor();
    virtual void edit(int index);
    virtual QWidget* widget();
private slots:
    void on_clearButton_clicked();
    void on_saveButton_clicked();

private:
    Ui::ResourceEditor *ui;
    QDataWidgetMapper* mapper;
};
#endif // RESOURCEEDITOR_H

```

5. Classe EditorDialog.h

```

#ifndef EDITORDIALOG_H
#define EDITORDIALOG_H
#include <QDialog>
class Editor;
class QSqlRelationalTableModel;

```

```

class QModelIndex;
namespace Ui
{
class EditorDialog;
}
class EditorDialog : public QDialog
{
    Q_OBJECT

public:
    explicit EditorDialog(Editor*, QWidget *parent = 0);
    ~EditorDialog();

private slots:
    void on_addButton_clicked();
    void on_removeButton_clicked();
    void on_tableView_clicked(const QModelIndex &index);
private:
    Ui::EditorDialog *ui;
    Editor* m_editor;
    QSqlRelationalTableModel* m_model;
};
#endif // EDITORDIALOG_H

```

6. Classe Editor.h

```

#ifndef EDITOR_H
#define EDITOR_H
class QWidget;
class QSqlRelationalTableModel;
class Editor
{
public:
    Editor(QSqlRelationalTableModel* model);
    QSqlRelationalTableModel* getModel();
    virtual void edit(int index) = 0;
    virtual QWidget* widget() = 0;
protected:
    QSqlRelationalTableModel* model;
};
#endif // EDITOR_H

```

7. Classe SimulationDialog.h

```

#ifndef SIMULATIONDIALOG_H
#define SIMULATIONDIALOG_H
#include <QDialog>
#include <QMultiMap>
#include <QList>
class Simulation;
namespace Ui {
class SimulationDialog;
}
class QListWidgetItem;
class SimulationDialog : public QDialog
{
    Q_OBJECT

public:
    explicit SimulationDialog(QWidget *parent = 0);

```

```

~SimulationDialog();
Simulation* getSimulation()
{ return sim; }

private slots:
    void on_resourceTypeComboBox_currentIndexChanged(int index);
    void on_featureTypeComboBox_currentIndexChanged(const QString &arg1);
    void on_addResourceButton_clicked();
    void on_removeResourceButton_clicked();
    void on_resourcesList_currentRowChanged(int currentRow);
    void on_addFeatureButton_clicked();
    void on_removeFeatureButton_clicked();
    void on_buttonBox_accepted();
private:
    Ui::SimulationDialog *ui;
    QList<QString> resources;
    typedef QMap<QString, QListWidgetItem*> FeaturesMap;
    FeaturesMap featuresMap;
    Simulation* sim;
};
#endif // SIMULATIONDIALOG_H

```

8. Classe SimulationResults.h

```

#ifndef SIMULATIONRESULTS_H
#define SIMULATIONRESULTS_H
#include <QDialog>
#include "Chart/model/Chartmodel.h"
class ChartModel;
namespace Ui
{
    class SimulationResults;
}
class Report;
class QModelIndex;
class SimulationResults : public QDialog
{
    Q_OBJECT
public:
    SimulationResults(const Report& report, QWidget *parent = 0);
    ~SimulationResults();
private:
    Ui::SimulationResults *ui;
    int m_activeRows;
    QStandardItemModel *model;
private slots:
    void on_SalvarTempoButton_clicked();
};
#endif // SIMULATIONRESULTS_H

```

9. Classe Chart.h

```

#ifndef CHART_H
#define CHART_H
#include <QDialog>
#include "Chart/view/Chartview.h"
#include "Chart/model/Chartmodel.h"
class QItemSelectionModel;
class ChartModel;
class ChartView;

```

```

namespace Ui
{
    class Chart;
}
class Chart : public QDialog
{
    Q_OBJECT
public:
    explicit Chart(QWidget *parent = 0);
    ~Chart();
private slots:
    void on_exportButton_clicked();
    void createChart();
    void createChart1();
    void toggleLabelInputs(int p_index);
    void setLabelsAndFonts();
    void setAxesPensAndGrid();
    void showChartInWidget();
    void showChartInWindow();
private:
    Ui::Chart *ui;
    ChartModel *m_dataModel;
    ChartView *m_Chart;
    QItemSelectionModel *m_selectionModel;
    int m_labelInputComboIndex;
};
#endif // CHART_H

```

10. Classe ChartModel.h

```

#ifndef CHARTMODEL_H
#define CHARTMODEL_H
#include <QStandardItemModel>
class ChartModel : public QStandardItemModel
{
    Q_OBJECT
public:
    explicit ChartModel(QObject *parent = 0);
    virtual ~ChartModel();
public slots:
    void saveData();
    void loadData();
    void clearData();
    void consolidateCategoryData();
    void consolidateCategoryData1();
signals:
private:
    int m_activeRows;
};
#endif // CHARTMODEL_H

```

11. Classe ChartView.h

```

#ifndef CHARTVIEW_H
#define CHARTVIEW_H
#include <QAbstractItemView>
#include <QGraphicsScene>
class QGraphicsView;
const int NOME = 0;
const int TEMPO = 1;

```

```

const int CUSTO = 2;
enum ChartType {BAR, PIE, LINE, UNKNOWN};
class ChartView : public QAbstractItemView
{
    Q_OBJECT
public:
    virtual ~ChartView();
    QRect visualRect(const QModelIndex &p_index) const;
    QModelIndex indexAt(const QPoint &p_point) const;
    void scrollTo(const QModelIndex &p_index, ScrollHint p_hint = EnsureVisible);
public slots:
    void setHeaderText(const QString &p_text);
    void setXLabelText(const QString &p_text);
    void setYLabelText(const QString &p_text);
    void setHeaderFont(const QFont &p_font);
    void setYFont(const QFont &p_font);
    void setXFont(const QFont &p_font);
    void setHeaderTextAlignment(Qt::Alignment p_align);
    void setXTextAlignment(Qt::Alignment p_align);
    void setYTextAlignment(Qt::Alignment p_align);
    void setXUnits(QString p_units);
    void setYUnits(QString p_units);
    void setXIntervals(int p_xLines);
    void setYIntervals(int p_yLines);
    void setXMargin(qreal p_margin);
    void setYMargin(qreal p_margin);
    void setLegendFont(const QFont &p_font);
    void setLegendColumnWidth(qreal p_width);
    void setGridPen(const QPen &p_pen);
    void setAxesPen(const QPen &p_pen);
    void setLoggingOn(bool p_logging);
    void drawChart();
    void drawChart1();
    void saveChart();
    void loadChart();
    void saveChart1();
    void loadChart1();
signals:
    void lastLogMsg(QString p_lastLogMsg);
protected slots:
    void dataChanged(const QModelIndex &topLeft, const QModelIndex &bottomRight);
    void rowsInserted(const QModelIndex &parent, int start, int end);
    void rowsAboutToBeRemoved(const QModelIndex &parent, int start, int end);
protected:
    explicit ChartView(QWidget *parent = 0);
    virtual void generateGraphicsItems() = 0;
    virtual void generateGraphicsItems1() = 0;
    qreal stripSpace(qreal p_perc);
    QString yUnits();
    QModelIndex moveCursor(QAbstractItemView::CursorAction p_cursorAction,
Qt::KeyboardModifiers p_modifiers);
    QRegion visualRegionForSelection(const QItemSelection &p_selection) const;
    int horizontalOffset() const;
    int verticalOffset() const;
    bool isIndexHidden(const QModelIndex &p_index) const;
    void setSelection(const QRect &p_rect, QItemSelectionModel::SelectionFlags p_command);
    ChartGraphItems *m_graphItems;
    ChartGrid *m_grid;
    ChartType m_type;
    QRectF m_innerScene;

```

```

    qreal m_totalValue;
    int m_activeRows;
private:
    void calculateGeometries();
    void resizeEvent(QResizeEvent *p_event);
    QGraphicsView *m_graphicsView;
    QGraphicsScene *m_graphScene;
    ChartTextItem m_headerItem;
    ChartTextItem m_yLabelItem;
    ChartTextItem m_xLabelItem;
    qreal m_xMargin;
    qreal m_yMargin;
    qreal m_legendColWidth;
    qreal m_maxValue;
};
#endif // CHARTVIEW_H

```

12. Classe CvsExporter.h

```

#ifndef CSVEXPORTER_H
#define CSVEXPORTER_H
#include <QtGui/QDialog>
#include <QtGui/QTableView>
#include <QtGui/QFileDialog>
#include <ui_Csvexporter.h>
class CSVExporter : public QDialog
{
    Q_OBJECT
public:
    CSVExporter(QTableView *p_tableView_2, QWidget *parent=0);
    virtual ~CSVExporter();
    virtual int exec();
private slots:
    virtual void on_savePushButton_clicked();
private:
    Ui_CSVExporter ui;
    QTableView *tableView_2;
    QAbstractItemModel *model;
};
#endif

```

13. Classe AboutDialog.h

```

#ifndef ABOUTDIALOG_H
#define ABOUTDIALOG_H
#include <QtGui/QDialog>
namespace Ui {
    class AboutDialog;
}
class AboutDialog : public QDialog {
    Q_OBJECT
    Q_DISABLE_COPY(AboutDialog)
public:
    explicit AboutDialog(QWidget *parent = 0);
    virtual ~AboutDialog();
protected:
    virtual void changeEvent(QEvent *e);
private:
    Ui::AboutDialog *m_ui;
};

```



```
#endif // ABOUTDIALOG_H
```

14. Classe HelpDialog.h

```
#ifndef HELPDIALOG_H
#define HELPDIALOG_H
#include <QtGui/QDialog>
namespace Ui
{
    class HelpDialog;
}
class HelpDialog : public QDialog {
    Q_OBJECT
    Q_DISABLE_COPY(HelpDialog)
public:
    explicit HelpDialog(QWidget *parent = 0);
    virtual ~HelpDialog();
protected:
    virtual void changeEvent(QEvent *e);
private:
    Ui::HelpDialog *m_ui;
};
#endif // HELPDIALOG_H
```

15. Classe ACESSManager.h

```
#ifndef ACESSMANAGER_H
#define ACESSMANAGER_H
#include <QDialog>
namespace Ui {
    class ACESSManager;
}
class ACESSManager : public QDialog
{
    Q_OBJECT
public:
    explicit ACESSManager(QWidget *parent = 0);
    ~ACESSManager();
private slots:
    void on_pushButtonNew_clicked();
    void on_pushButtonSave_clicked();
    void on_pushButtonDelete_clicked();
    void on_pushButtonCancel_clicked();

private:
    Ui::ACESSManager *ui;
};
#endif // ACESSMANAGER_H
```

16. Classe PasswordChange.h

```
#ifndef PASSWORDCHANGE_H
#define PASSWORDCHANGE_H
#include <QDialog>
namespace Ui {
    class PasswordChange;
}
class PasswordChange : public QDialog
{
    Q_OBJECT
```

```

public:
    explicit PasswordChange(QWidget *parent = 0);
    ~PasswordChange();
private slots:
    void on_pushButtonConfirm_clicked();
    void on_pushButtonClose_clicked();

private:
    Ui::PasswordChange *ui;
};
#endif // PASSWORDCHANGE_H

```

17. Classe Login.h

```

#ifndef LOGIN_H
#define LOGIN_H
#include <QDialog>
namespace Ui {
    class Login;
}
class Login : public QDialog
{
    Q_OBJECT
public:
    explicit Login(QWidget *parent = 0);
    ~Login();
private slots:
    void on_pushButtonEnter_clicked();
    void on_pushButtonCancel_clicked();

private:
    Ui::Login *ui;
};
#endif // LOGIN_H

```

3. Scene

1. Classe Scene.h

```

#ifndef SCENE_H
#define SCENE_H
#include <vector>
#include "BoundingBox.h"
#include "Vec3.h"
class Simulation;
class Piece;
class Feature;
namespace OpenCSG {
    class Primitive;
}
struct Face {
    int vertices[3];
    int normals[3];
};
struct Mesh {
    std::vector<Vec3> vertices;
    std::vector<Vec3> normals;
    std::vector<Face> faces;
    BoundingBox boundingbox;
    float dispx;
};

```

```

};
class Scene
{
public:
    Scene(Simulation*);
    void draw();
    BoundingBox bbox;
private:
    void makePiece(Piece*);
    void makeFeature(Feature*);
    void Animations();
    void Collisions();
    std::vector<OpenCSG::Primitive*> m_primitives;
    std::vector<Mesh> meshes;
};
#endif // SCENE_H

```

2. Classe GLWidget.h

```

#ifndef GLWIDGET_H
#define GLWIDGET_H
#include <QGLWidget>
#include <QMouseEvent>
#include <GL/glu.h>
#include "src/Camera.h"
#include "displaylistPrimitive.h"
class Scene;
enum mode { PAN, ROTATE, TRANSLATE };
class GLWidget : public QGLWidget
{
    Q_OBJECT
public:
    explicit GLWidget(QWidget *parent = 0);
    Camera *camera;
    void setScene(Scene *s);
private:
    void initializeGL();
    void paintGL();
    void resizeGL(int width, int height);
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
    void wheelEvent(QWheelEvent *event);
    int camera_mode;
    void drawBackground();
    void setProjection();
    void drawAxis();
    Scene *scene;
    void createCylinder(GLdouble radius, GLdouble height, GLint slices, GLint stacks);
    std::vector<OpenCSG::Primitive*> primitives;
    int currWinWidth, currWinHeight;
};
#endif // GLWIDGET_H

```

3. Classe Camera.h

```

#ifndef CAMERA_H_
#define CAMERA_H_
#include "BoundingBox.h"
#include "math/Vec3.h"
#include "math/Quat.h"

```

```

typedef struct{
    float fleft;
    float fright;
    float fbottom;
    float ftop;
    float fnear;
    float ffar;
} Frustum;
enum view { TOP, BOTTOM, FRONT, BACK, RIGHT, LEFT, DEFAULT };
class Camera {
public:
    Camera();
    void setView(int type);
    void setupViewMatrix(float *m);
    void setProjection(float l, float r, float b,
        float t, float n, float f);
    void updateRotation(int mouseX, int mouseY,
        int windowWidth, int windowHeight);
    void fit(BoundingBox bbox);
    void reset();
    void resetPanVector();
    void resetOrientation();

    void mapTrackball(int mouseX, int mouseY,
        int windowWidth, int windowHeight);
    void rotate(const Vec3 &axis, float angle);
    void zoom(float zoomFactor);
    void move(int mouseX, int mouseY);
    Frustum frustum;
    Vec3 pos;
    Quat orientation;
    bool initialized;
    Vec3 lastPos;
    Vec3 currPos;
    Vec3 moveDir;
    Vec3 currMvPos;
    Vec3 lastMvPos;
};
#endif

```

4. Classe DisplayPrimitive.h

```

#include "opencsg/opencsg.h"
namespace OpenCSG {
    class DisplayListPrimitive : public Primitive {
    public:
        DisplayListPrimitive(unsigned int displayListId_, Operation, unsigned int convexity);
        void setDisplayListId(unsigned int);
        unsigned int getDisplayListId() const;
        virtual void render();
    private:
        unsigned int mDisplayListId;
    };
} // namespace OpenCSG

```

5. Classe Utility.h

```

#ifndef UTILITY_H
#define UTILITY_H
#include <string>

```

```

#include <sstream>
template <class T>
inline std::string toString( const T& type)
{
    std::stringstream s;
    s << type;
    return s.str();
}
#endif

```

6. Classe BoundingBox.h

```

#ifndef _BOUNDINGBOX_H_
#define _BOUNDINGBOX_H_
#include "math/Vec3.h"
// #include "Mesh.h"
class BoundingBox {
public:
    BoundingBox();
    BoundingBox(const Vec3& pMin, const Vec3& pMax);
    Vec3 pMin, pMax;
    Vec3 center;
};
#endif

```

ANEXO A

Esse anexo detalha a composição da biblioteca OpenCSG que foi utilizada no trabalho. Inicialmente é apresentado uma figura com o diagrama das classes utilizadas. Em seguida é detalhada a declaração de cada classe que compõe o diagrama.

1. Classe OpenCSG.h

```

#ifndef __OpenCSG_opencsg_h__
#define __OpenCSG_opencsg_h__
#include <vector>
#define OPENCSG_VERSION 0x0132
#define OPENCSG_VERSION_STRING "OpenCSG 1.3.2"
namespace OpenCSG {
    enum Operation { Intersection, Subtraction };
    class Primitive {
    public:
        Primitive(Operation, unsigned int convexity);
        virtual ~Primitive();
        void setOperation(Operation);
        Operation getOperation() const;
        void setConvexity(unsigned int);
        unsigned int getConvexity() const;
        void setBoundingBox(float minx, float miny, float minz,
                           float maxx, float maxy, float maxz);
        void getBoundingBox(float& minx, float& miny, float& minz,
                           float& maxx, float& maxy, float& maxz) const;
        virtual void render() = 0;
    private:
        Operation mOperation;
        unsigned int mConvexity;
        float mMinx, mMiny, mMinz, mMaxx, mMaxy, mMaxz;
    };
    void render(const std::vector<Primitive*>& primitives);
    enum OptionType {
        AlgorithmSetting = 0,
        DepthComplexitySetting = 1,
    };
}

```

```

    OffscreenSetting      = 2,
    DepthBoundsOptimization = 3,
    OptionTypeUnused      = 4
};
void setOption(OptionType option, int newSetting);
int  getOption(OptionType option);
enum Algorithm {
    Automatic      = 0,
    Goldfeather    = 1,
    SCS            = 2,
    AlgorithmUnused = 3
};
enum DepthComplexityAlgorithm {
    NoDepthComplexitySampling = 0,
    OcclusionQuery            = 1,
    DepthComplexitySampling   = 2,
    DepthComplexityAlgorithmUnused = 3
};
enum OffscreenType {
    AutomaticOffscreenType = 0,
    FrameBufferObject      = 1,
    PBuffer                = 2,
    FrameBufferObjectARB    = 3,
    FrameBufferObjectEXT    = 4,
    OffscreenTypeUnused    = 5
};
enum Optimization {
    OptimizationDefault = 0,
    OptimizationForceOn = 1,
    OptimizationOn      = 2,
    OptimizationOff     = 3,
    OptimizationUnused  = 4
};
void setContext(int context);
/// Returns the current context.
int  getContext();
/// Releases the OpenGL resources allocated by OpenCSG for the current
/// context.
void freeResources();

void render(const std::vector<Primitive*>& primitives,
            Algorithm,
            DepthComplexityAlgorithm = NoDepthComplexitySampling);
} // namespace OpenCSG
#endif // __OpenCSG__opencsg_h__

```

2. Classe Render Textura.h

```

#ifndef __RENDETEXTURE2_HPP__
#define __RENDETEXTURE2_HPP__
#ifdef __APPLE__
#include <glew/GL/glew.h>
#else
#include <GL/glxew.h>
#endif
#include <string>
#include <vector>
class RenderTexture

```

```

{
public:
    enum UpdateMode
    {
        RT_RENDER_TO_TEXTURE,
        RT_COPY_TO_TEXTURE
    };

public:
    RenderTexture(const char *strMode="rgb tex2D");
    ~RenderTexture();
    bool Initialize(int width, int height,
        bool shareObjects=true,
        bool copyContext=false);
    bool Reset(const char* strMode,...);
    bool Resize(int width, int height);
    bool BeginCapture();
    bool BeginCapture(RenderTexture* current);
    bool EndCapture();
    void Bind() const;
    void BindDepth() const;
    bool BindBuffer( int iBuffer );
    void EnableTextureTarget() const
    { if (!_bInitialized) glEnable(_iTextureTarget); }
    void DisableTextureTarget() const
    { if (!_bInitialized) glDisable(_iTextureTarget); }
    unsigned int GetTextureID() const { return _iTextureID; }
    unsigned int GetDepthTextureID() const { return _iDepthTextureID; }
    unsigned int GetTextureTarget() const { return _iTextureTarget; }
    operator unsigned int()const{return _iTextureID;}
    int GetWidth() const { return _iWidth; }
    int GetHeight() const { return _iHeight; }
    int GetMaxS() const { return IsRectangleTexture() ? _iWidth : 1; }
    int GetMaxT() const { return IsRectangleTexture() ? _iHeight : 1; }
    int GetRedBits() const { return _iNumColorBits[0]; }
    int GetGreenBits() const { return _iNumColorBits[1]; }
    int GetBlueBits() const { return _iNumColorBits[2]; }
    int GetAlphaBits() const { return _iNumColorBits[3]; }
    int GetDepthBits() const { return _iNumDepthBits; }
    int GetStencilBits() const { return _iNumStencilBits; }
    bool IsInitialized() const { return _bInitialized; }
    bool IsTexture() const { return _bIsTexture; }
    bool IsDepthTexture() const { return _bIsDepthTexture; }
    bool IsFloatTexture() const { return _bFloat; }
    bool IsDoubleBuffered() const { return _bDoubleBuffered; }
    bool IsRectangleTexture() const { return _bRectangle; }
    bool HasDepth() const { return (_iNumDepthBits > 0); }
    bool HasStencil() const { return (_iNumStencilBits > 0); }
    bool IsMipmapped() const { return _bMipmap; }
    static bool IsPowerOfTwo(int n) { return ((n&(n-1))==0); }

protected:
    bool _Invalidate();
    typedef std::pair<std::string, std::string> KeyVal;
    void _ParseModeString(const char *modeString,
        std::vector<int> &pixelFormatAttribs,
        std::vector<int> &pbufferAttribs);
    std::vector<int> _ParseBitVector(std::string bitVector);
    KeyVal _GetKeyValuePair(std::string token);
    bool _VerifyExtensions();
    bool _InitializeTextures();

```



```

void _MaybeCopyBuffer();
bool _ReleaseBoundBuffers();
bool _MakeCurrent();
bool _BindDepthBuffer( ) const;
protected:
    int      _iWidth;
    int      _iHeight;
    bool     _bIsTexture;
    bool     _bIsDepthTexture;
    bool     _bHasARBDepthTexture;
    UpdateMode _eUpdateMode;
    bool     _bInitialized;
    unsigned int _iNumAuxBuffers;
    bool     _bIsBufferBound;
    int      _iCurrentBoundBuffer;
    unsigned int _iNumComponents;
    unsigned int _iNumColorBits[4];
    unsigned int _iNumDepthBits;
    unsigned int _iNumStencilBits;
    bool     _bFloat;
    bool     _bDoubleBuffered;
    bool     _bPowerOf2;
    bool     _bRectangle;
    bool     _bMipmap;
    bool     _bShareObjects;
    bool     _bCopyContext;
#ifdef _WIN32
    HDC      _hDC;
    HGLRC    _hGLContext;
    HPBUFFERARB _hPBuffer;
    HDC      _hPreviousDC;
    HGLRC    _hPreviousContext;
#else
    Display * _pDisplay;
    GLXContext _hGLContext;
    GLXPbuffer _hPBuffer;
    GLXDrawable _hPreviousDrawable;
    GLXContext _hPreviousContext;
#endif
    GLenum    _iTextureTarget;
    unsigned int _iTextureID;
    unsigned int _iDepthTextureID;
    unsigned short* _pPoorDepthTexture;
    std::vector<int> _pixelFormatAttribs;
    std::vector<int> _pbufferAttribs;
private:
    RenderTexture(const RenderTexture&);
    RenderTexture& operator=(const RenderTexture&);
};
#endif // __APPLE__
#endif // __RENDERTEXTURE2_HPP__

```

3. Classe Area.h

```

#ifndef __OpenCSG__area_h__
#define __OpenCSG__area_h__
#include "opencsgConfig.h"
namespace OpenCSG {
    struct PCArea {
        PCArea(

```

```

        int minxx = 0,
        int minyy = 0,
        int maxxx = 0,
        int maxyy = 0
    ) :
        minx(minxx),
        miny(minyy),
        maxx(maxxx),
        maxy(maxyy)
    { }
    int minx, miny, maxx, maxy;
};
struct NDCVolume {
    NDCVolume(
        float minxx = 0.0f,
        float minyy = 0.0f,
        float minzz = 0.0f,
        float maxxx = 0.0f,
        float maxyy = 0.0f,
        float maxzz = 0.0f
    ) :
        minx(minxx),
        miny(minyy),
        minz(minzz),
        maxx(maxxx),
        maxy(maxyy),
        maxz(maxzz)
    { }
    operator PCArea() const;
    float minx, miny, minz, maxx, maxy, maxz;
};
}
#endif // __OpenCSG__area_h__

```

4. Classe Batch.h

```

#ifndef __OpenCSG__batch_h__
#define __OpenCSG__batch_h__
#include "opencsgConfig.h"
#include <vector>
namespace OpenCSG {
    class Primitive;
    typedef std::vector<Primitive*> Batch;
    class Batcher {
    public:
        Batcher(const std::vector<Primitive*>& primitives);
        std::vector<Batch>::const_iterator begin() const;
        std::vector<Batch>::const_iterator end() const;
        unsigned int size() const;
    private:
        std::vector<Batch> mBatches;
    };
} // namespace OpenCSG
#endif // __OpenCSG__batch_h__

```

5. Classe ChannelManager.h

```

#ifndef __OpenCSG__channel_manager_h__
#define __OpenCSG__channel_manager_h__
#include "opencsgConfig.h"

```

```

#include <utility>
#include <vector>
namespace OpenCSG {
    namespace OpenGL {
        class OffscreenBuffer;
    }
    class Primitive;
    enum Channel {
        NoChannel = 0, Alpha = 1, Red = 2, Green = 4, Blue = 8
    };
    class ChannelManager {
    public:
        ChannelManager();
        virtual ~ChannelManager();
        bool init();
        Channel find() const;
        Channel request();
        Channel current() const;
        std::vector<Channel> occupied() const;
        void free();
        virtual void merge() = 0;
        void renderToChannel(bool on);
        void setupProjectiveTexture();
        void resetProjectiveTexture();
        static void setupTexEnv(Channel channel);
    private:
        static OpenGL::OffscreenBuffer* gOffscreenBuffer;
        static bool glnUse;
        bool mInOffscreenBuffer;
        Channel mCurrentChannel;
        int mOccupiedChannels;
    };
    class ChannelManagerForBatches : public ChannelManager {
    public:
        ChannelManagerForBatches();
        void store(Channel channel, const std::vector<Primitive*>& primitives, int layer);
        const std::vector<Primitive*> getPrimitives(Channel channel) const;
        int getLayer(Channel channel) const;
        void clear();
    private:
        std::vector<std::pair<std::vector<Primitive*>, int>> mPrimitives;
    };
}
#endif // __OpenCSG__channel_manager_h__

```

6. Classe Context.h

```

#ifndef __OpenCSG__context_h__
#define __OpenCSG__context_h__
#include "opencsgConfig.h"
#include <opencsg.h>
namespace OpenCSG {
    void setContext(int context);
    int getContext();
    void freeResources();
} // namespace OpenCSG
#endif // __OpenCSG__context_h__

```

7. Classe FrameBufferObject.h

```

#ifndef __OpenCSG__frame_buffer_object_h__
#define __OpenCSG__frame_buffer_object_h__
#include "opencsgConfig.h"
#include "offscreenBuffer.h"
#include <glew/GL/glew.h>
namespace OpenCSG {
    namespace OpenGL {
        class FrameBufferObject : public OffscreenBuffer {
        public:
            FrameBufferObject();
            virtual ~FrameBufferObject();
            virtual bool Initialize(int width, int height, bool shareObjects=true, bool copyContext=false);
            virtual bool IsInitialized() const { return initialized; }
            virtual bool Reset();
            virtual bool Resize(int width, int height);
            virtual bool BeginCapture();
            virtual bool EndCapture();
            virtual void Bind() const;
            virtual void EnableTextureTarget() const { if (initialized) glEnable(textureTarget); }
            virtual void DisableTextureTarget() const { if (initialized) glDisable(textureTarget); }
            virtual unsigned int GetTextureTarget() const { return textureTarget; }
            virtual int GetWidth() const { return width; }
            virtual int GetHeight() const { return height; }
            virtual bool HaveSeparateContext() const { return false; }
        protected:
            int width;
            int height;
            GLenum textureTarget;
            unsigned int textureID;
            unsigned int depthID;
            unsigned int framebufferID;
            int oldFramebufferID;
            bool initialized;
        };
    }
}
#endif // __OpenCSG__frame_buffer_object_h__

```

8. Classe FrameBufferObjectExt.h

```

#ifndef __OpenCSG__frame_buffer_object_ext_h__
#define __OpenCSG__frame_buffer_object_ext_h__
#include "opencsgConfig.h"
#include "offscreenBuffer.h"
#include <glew/GL/glew.h>
namespace OpenCSG {
    namespace OpenGL {
        class FrameBufferObjectExt : public OffscreenBuffer {
        public:
            FrameBufferObjectExt();
            virtual ~FrameBufferObjectExt();
            virtual bool Initialize(int width, int height, bool shareObjects=true, bool copyContext=false);
            virtual bool IsInitialized() const { return initialized; }
            virtual bool Reset();
            virtual bool Resize(int width, int height);
            virtual bool BeginCapture();
            virtual bool EndCapture();
            virtual void Bind() const;
            virtual void EnableTextureTarget() const { if (initialized) glEnable(textureTarget); }
            virtual void DisableTextureTarget() const { if (initialized) glDisable(textureTarget); }
        };
    }
}

```

```

        virtual unsigned int GetTextureTarget() const { return textureTarget; }
        virtual int GetWidth() const { return width; }
        virtual int GetHeight() const { return height; }
        virtual bool haveSeparateContext() const { return false; }
protected:
    int      width;
    int      height;
    GLenum    textureTarget;
    unsigned int textureID;
    unsigned int depthID;
    unsigned int framebufferID;
    int      oldFramebufferID;
    bool      initialized;
};
}
}
#endif // __OpenCSG__frame_buffer_object_ext_h__

```

9. Classe OcclusionQuery.h

```

#ifndef __OpenCSG__occlusion_query_h__
#define __OpenCSG__occlusion_query_h__
#include "opencsgConfig.h"
namespace OpenCSG {
    namespace OpenGL {
        class OcclusionQuery {
        public:
            virtual ~OcclusionQuery() {};
            virtual void beginQuery() = 0;
            virtual void endQuery() = 0;
            virtual unsigned int getQueryResult() = 0;
        protected:
            OcclusionQuery() {};
        };
        OcclusionQuery* getOcclusionQuery(bool exactNumberNeeded);
    }
}
#endif // __OpenCSG__occlusion_query_h__

```

10. Classe OffscreenBuffer.h

```

#ifndef __OpenCSG__offscreen_buffer_h__
#define __OpenCSG__offscreen_buffer_h__
#include "opencsgConfig.h"
#include <opencsg.h>
namespace OpenCSG {
    namespace OpenGL {
        class OffscreenBuffer {
        public:
            virtual ~OffscreenBuffer() {};
            virtual bool Initialize(int width, int height,
                                    bool shareObjects=true,
                                    bool copyContext=false) = 0;
            virtual bool IsInitialized() const = 0;
            virtual bool Reset() = 0;
            virtual bool Resize(int width, int height) = 0;
            virtual bool BeginCapture() = 0;
            virtual bool EndCapture() = 0;
            virtual void Bind() const = 0;
            virtual void EnableTextureTarget() const = 0;
            virtual void DisableTextureTarget() const = 0;
        };
    }
}

```

```

        virtual unsigned int GetTextureTarget() const = 0;
        virtual int GetWidth() const = 0;
        virtual int GetHeight() const = 0;
        virtual bool haveSeparateContext() const = 0;
    };
    OffscreenBuffer* getOffscreenBuffer(OffscreenType type);
    void freeResources();
}
}
#endif // __OpenCSG__offscreen_buffer_h__

```

11. Classe OpenCSGConfig.h

```

#ifndef __OpenCSG__opencsg_config_h__
#define __OpenCSG__opencsg_config_h__
#ifdef WIN32
#ifdef __INTEL_COMPILER
# pragma warning(disable: 985)
#elif _MSC_VER
# if _MSC_VER < 1300
# pragma warning(disable: 4786)
#endif
#endif
#ifdef _MSC_VER
# if _MSC_VER < 1300
# ifdef min
# undef min
# endif
# ifdef max
# undef max
# endif
namespace std {
    template <class T>
    inline const T& min(const T& a, const T& b) {
        return b < a ? b : a;
    }
    template <class T>
    inline const T& max(const T& a, const T& b) {
        return a < b ? b : a;
    }
    template <class T, class Compare>
    inline const T& min(const T& a, const T& b, Compare comp) {
        return comp(b, a) ? b : a;
    }
    template <class T, class Compare>
    inline const T& max(const T& a, const T& b, Compare comp) {
        return comp(a, b) ? b : a;
    }
}
#endif
#endif
#endif
#ifdef __APPLE__
#define OPENCSG_HAVE_PBUFFER 1
#endif
#endif // __OpenCSG__opencsg_config_h__

```

12. Classe OpenCSGRender.h

```

#ifndef __OpenCSG__opencsg_render_h__

```

```

#define __OpenCSG__opencsg_render_h__
#include "opencsgConfig.h"
#include <opencsg.h>
#include <vector>
namespace OpenCSG {
    void render(const std::vector<Primitive*>& primitives,
                Algorithm,
                DepthComplexityAlgorithm);
    void renderSCS(const std::vector<Primitive*>& primitives, DepthComplexityAlgorithm);
    void renderGoldfeather(const std::vector<Primitive*>& primitives, DepthComplexityAlgorithm);
}
#endif // __OpenCSG__opencsg_render_h__

```

13. Classe OpenGLHelper.h

```

#ifndef __OpenCSG__opengl_helper_h__
#define __OpenCSG__opengl_helper_h__
#include "opencsgConfig.h"
#include <glew/GL/glew.h>
#include "area.h"
namespace OpenCSG {
    namespace OpenGL {
        extern GLfloat projection[16];
        extern GLfloat modelview[16];
        extern int canvasPos[4];
        extern int stencilBits;
        extern int stencilMax;
        extern int stencilMask;
        extern int scissorPos[4];
        void scissor(const PCArea& area);
        void drawQuad();
    }
}
#endif // __OpenCSG__opengl_helper_h__

```

14. Classe PBufferTexture.h

```

#ifndef __OpenCSG__pbuffer_texture_h__
#define __OpenCSG__pbuffer_texture_h__
#include "opencsgConfig.h"
#include "offscreenBuffer.h"
#ifdef OPENCSG_HAVE_PBUFFER
class RenderTexture;
namespace OpenCSG {
    namespace OpenGL {
        class PBufferTexture : public OffscreenBuffer {
        public:
            PBufferTexture();
            virtual ~PBufferTexture();
            virtual bool Initialize(int width, int height, bool shareObjects=true, bool copyContext=false);
            virtual bool IsInitialized() const;
            virtual bool Reset();
            virtual bool Resize(int width, int height);
            virtual bool BeginCapture();
            virtual bool EndCapture();
            virtual void Bind() const;
            virtual void EnableTextureTarget() const;
            virtual void DisableTextureTarget() const;
            virtual unsigned int GetTextureTarget() const;
            virtual int GetWidth() const;

```

```

        virtual int GetHeight() const;
        virtual bool haveSeparateContext() const { return true; }
    protected:
        RenderTexture* r;
        const char* s;
    };
}
}
#endif
#endif // __OpenCSG__pbuffer_texture_h__

```

15. Classe PrimitiveHelper.h

```

#ifndef __OpenCSG__primitive_helper_h__
#define __OpenCSG__primitive_helper_h__
#include "opencsgConfig.h"
#include "area.h"
#include <vector>
namespace OpenCSG {
    class Primitive;
    namespace Algo {
        m bool intersectXY(const Primitive* a, const Primitive* b);
        bool intersectXYZ(const Primitive* a, const Primitive* b);
        bool containsXY(const Primitive* a, const Primitive* b);
        unsigned int getConvexity(const std::vector<Primitive*>& primitives);
    } // namespace Algo
    namespace OpenGL {
        unsigned int calcMaxDepthComplexity(const std::vector<Primitive*>& primitives,
                                             const PCArea& area);
        void renderLayer(unsigned int layer, const std::vector<Primitive*>& primitives);
    }
} #endif // __OpenCSG__primitive_helper_h__

```

16. Classe ScissorMemo.h

```

#ifndef __OpenCSG__scissor_memo_h__
#define __OpenCSG__scissor_memo_h__
#include "opencsgConfig.h"
#include "area.h"
#include "channelManager.h"
#include <vector>
namespace OpenCSG {
    class ScissorMemo {
    public:

        ScissorMemo();
        void store(Channel);
        void recall(Channel);
        void enableScissor() const;
        void disableScissor() const;
        void enableDepthBounds() const;
        void enableDepthBoundsBack() const;
        void disableDepthBounds() const;
        void setIntersected(const std::vector<Primitive*>& primitives);
        void setCurrent(const std::vector<Primitive*>& primitives);
        const NDCVolume& getIntersectedArea() const;
        const NDCVolume& getCurrentArea() const;
    protected:
        void calculateArea();
    private:

```



```

        NDCVolume mIntersection;
        NDCVolume mCurrent;
        NDCVolume mArea;
        std::vector<NDCVolume> mScissor;
        bool mUseDepthBoundsTest;
    };
}
#endif // __OpenCSG__scissor_memo_h__

```

17. Classe Settings.h

```

#ifndef __OpenCSG__settings_h__
#define __OpenCSG__settings_h__
#include "opencsgConfig.h"
#include <opencsg.h>
namespace OpenCSG {
    void initOptions();
    void setOption(OptionType option, int newSetting);
    int getOption(OptionType option);
}
#endif // __OpenCSG__settings_h__

```

18. Classe StencilManager.h

```

#ifndef __OpenCSG__stencil_manager_h__
#define __OpenCSG__stencil_manager_h__
#include "opencsgConfig.h"
#include "area.h"
namespace OpenCSG {
    namespace OpenGL {
        class StencilManager {
        public:
            StencilManager(const PCArea&);
            virtual ~StencilManager();
            const PCArea& getArea() const;
            void clear();
            bool alreadySaved() const;
            virtual void save();
            virtual void restore();
        private:
            const PCArea mArea;
            bool mSaved;
        };
        StencilManager* getStencilManager(const PCArea&);
    }
}
#endif // __OpenCSG__stencil_manager_h__

```

ANEXO B

Neste anexo são detalhadas as geometrias dos principais recursos de fabricação e transporte importados no formato OBJ.

1. Torno

2. Fresa

3. Furadeira

4. Retificadora

5. Mandriladora

6. Brochadeira

7. Plana

8. Alargador

9. Operário

10. Esteira 1 metros

11. Esteira 2 metros

12. Esteira 3 metros

13. Esteira 4 metros

14. Empilhadeira

15. AVG

16. Paleta