

Handbook de TI para Concursos

O Guia Definitivo



Camatta • Gobira • Melotti

Sumário

I Fundamentos de Computação	16
1 Arquitetura e Organização de Computadores	17
1.1 Conceitos Básicos	17
1.2 Estrutura e Funcionamento da CPU	18
1.2.1 Pipelines	20
1.3 Conjunto de Instruções	22
1.4 Unidade de Controle	24
1.5 Modos de Endereçamento	25
1.6 Organização de Memória	26
1.7 Desempenho do computador	28
1.7.1 Tempo de execução de um programa	29
1.7.2 Desempenho da CPU	29
1.7.3 Programas para medir desempenho	30
1.7.4 Comparando desempenho	30
1.7.5 Lei de Amdahl	31
2 Componentes de um Computador	33
2.1 Principais componentes de Hardware	34
2.1.1 Discos Rígidos	34
2.1.2 Teclado	36
2.1.3 Mouse	37
2.1.4 Placa de rede	37
2.1.5 Impressora	38
2.1.6 Monitor	38
3 Aritmética Computacional	39
3.1 Números Com Sinal e Números Sem Sinal	39
3.1.1 Sinal e amplitude/magnitude	39
3.1.2 Complemento de 1	39
3.1.3 Complemento de 2	39
3.1.4 Notação em excesso	40
3.2 Adição e Subtração	41
3.3 Operações Lógicas	41
3.4 Construção de uma Unidade Lógica Aritmética	42
3.5 Ponto Flutuante	43

4 Sistemas Operacionais	44
4.1 Introdução	44
4.2 Conceitos Básicos	46
4.2.1 Multiprogramação	46
4.2.2 Processo	46
4.2.3 Interrupções	47
4.2.4 Threads	48
4.3 Escalonamento de Processos	49
4.4 Entrada e Saída	50
4.4.1 Camadas do subsistema de Entrada e Saída	51
4.5 Gerência de Memória	52
4.6 Sistemas de Arquivos	54
4.6.1 Conceitos básicos sobre arquivos	54
4.6.2 Implementação de arquivos	56
4.6.3 Cache de Sistema de Arquivos	57
4.6.4 Gerenciamento do espaço livre	58
4.6.5 Diretórios	59
4.6.6 Implementação de diretórios	61
4.7 Sistemas Operacionais Distribuídos	61
4.7.1 Estruturação de Sistemas Distribuídos	63
5 Principais Processadores de Mercado	65
5.1 Processadores Intel	65
5.1.1 Família Pentium	65
5.1.2 Família Celeron	68
5.1.3 Família Core	69
5.1.4 Xeon	71
5.1.5 Itanium	74
5.2 AMD	75
5.2.1 Sempron	75
5.2.2 Athlon 64	76
5.2.3 Turion 64	79
5.2.4 Opteron	81
II Lógica de Programação	83
6 Orientação a Objetos	84
6.1 Introdução	84
6.2 Conceitos fundamentais	84
6.3 Princípios de programação orientada a objetos	90
6.4 Tratamento de exceções	90
III Metodologia de Desenvolvimento	92
7 Ciclo de Vida	93
7.1 Modelo seqüencial linear	95
7.2 Modelo em V	95
7.3 Modelo de prototipagem	96

7.4	Modelo RAD	96
7.5	Modelos de processo de software evolucionários	97
7.5.1	Modelo incremental	97
7.5.2	Modelo espiral	98
7.5.3	Modelo espiral ganha-ganha	99
7.5.4	Modelo de desenvolvimento concorrente	100
7.6	Desenvolvimento baseado em componentes	100
7.7	Modelo de métodos formais	100
7.8	Técnicas de quarta geração	100
8	Análise Comparativa de Processos de Desenvolvimento	102
8.1	RUP - Rational Unified Process	102
8.2	XP - Extreme Programming	105
8.3	Scrum	105
8.4	Crystal	106
8.5	Feature Driven Development (FDD)	106
8.6	Dynamic Systems Development Method (DSDM)	107
8.7	Adaptive Software Development (ASD)	107
9	Engenharia de Requisitos	109
9.1	O Processo de Engenharia de Requisitos	109
9.2	Técnicas de Levantamento de Requisitos	110
9.2.1	Observação	110
9.2.2	Entrevista	110
9.2.3	Análise de Protocolo	111
9.2.4	JAD	111
9.2.5	PD	111
9.2.6	QFD	111
9.2.7	CRC	112
9.2.8	Prototipação	112
9.2.9	Cenários	112
9.2.10	FAST	112
9.3	Análise de Requisitos	113
9.3.1	Métodos de análise	114
9.3.2	Modelagem da análise	114
9.4	Gerenciamento de Requisitos	116
10	Métricas	118
10.1	Métricas de processo e aperfeiçoamento de processo de software .	118
10.2	Métricas de projeto	119
10.3	Medição de software	119
10.3.1	Métricas orientadas a tamanho	120
10.3.2	Métricas orientadas a função	120
10.3.3	Métricas de pontos por função estendidas	122
10.4	Métricas de qualidade de software	122
10.4.1	Fatores de qualidade de McCall	123
10.4.2	FURPS	123
10.4.3	ISO 9126	124
10.5	Estimativas	124
10.5.1	COCOMO (Constructive Cost Model)	125

11 Testes	127
11.1 Teste de caminho básico	127
11.2 Teste de estrutura de controle	129
11.2.1 Teste de condição	129
11.2.2 Teste de fluxo de dados	130
11.2.3 Teste de ciclo	130
11.3 Teste caixa-preta	131
11.3.1 Métodos de teste baseados em grafo	131
11.3.2 Particionamento de equivalência	131
11.3.3 Análise de valor limite	132
11.3.4 Teste de comparação	132
11.3.5 Teste de matriz ortogonal	132
11.4 Teste de ambientes, arquiteturas e aplicações especializadas	133
11.5 Estratégia de teste de software	134
12 UML	136
12.1 Diagrama de caso de uso	136
12.1.1 Ator	136
12.1.2 Descrição do caso de uso	137
12.2 Diagrama de classe	137
12.2.1 Associações de classe	138
12.3 Diagramas de seqüência	140
12.4 Diagramas de colaboração	140
12.5 Diagramas de estado	141
12.6 Diagramas de atividade	143
12.7 Elementos auxiliares	144
12.8 Diagramas de componente	144
12.9 Diagramas de distribuição	144
13 Gerência de Configuração e Mudanças	145
13.1 As Atividades	146
13.2 Artefatos	147
13.3 Papéis e Responsabilidades	147
14 CMM - Capability Maturity Model	149
14.1 Os níveis de maturidade no CMM	150
14.1.1 Nível 1 - Inicial	150
14.1.2 Nível 2 - Repetitivo	150
14.1.3 Nível 3 - Definido	151
14.1.4 Nível 4 - Gerenciado	152
14.1.5 Nível 5 - Otimizado	152
14.2 Um pouco mais sobre KPA's	152
14.3 Efeitos da evolução do nível de maturidade	153
IV Linguagem de Programação Java	155
15 Conceitos Básicos de Java	156
15.1 Pacotes	156
15.2 Modificadores de Acesso	157

15.3 Variáveis	157
15.4 Operadores	158
15.5 Expressões, Sentenças e Blocos	160
15.6 Comandos de Controle de Fluxo	161
15.7 Classes Aninhadas	166
15.8 Tipos Enumerados	167
15.9 Anotações	168
15.10 Genéricos	169
15.11 Reflexão	171
16 Classes Essenciais	173
16.1 Exception e Controle de Exceções	173
16.1.1 Exceções típicas	173
16.1.2 Capturando Exceções	175
16.2 Threads e Concorrência	176
16.2.1 Definindo e Iniciando uma Thread	176
16.2.2 Pausando a execução com sleep	177
16.2.3 Interrupções	178
16.2.4 Joins	178
16.2.5 Sincronização	179
16.2.6 Executores e Thread Pools	180
16.3 Streams e Serialização	181
16.3.1 I/O Streams	181
16.3.2 Serialização - Streams de Objetos	183
16.4 Classes e Operações de I/O	185
16.5 Classes para manipulação de propriedades	185
17 Coleções	188
17.1 Interface Collection	189
17.2 Interface Set	190
17.3 Interface List	193
17.4 Interface Map	195
17.5 Interface Queue	197
18 JDBC - Java Database Connectivity	199
18.1 Conceitos Básicos	199
18.2 Carregamento de drivers	200
18.3 Conexão	200
18.4 Statements	201
18.5 Prepared Statements	203
18.6 Transação	203
18.7 Informações Complementares	204
18.8 Exemplo Extra	205
19 A plataforma J2EE	207
19.1 Containers J2EE	208
19.2 Clientes J2EE	209
19.3 Um pouco mais sobre Servlets	209
19.3.1 Ciclo de Vida dos Servlets	211
19.3.2 Mantendo o estado do cliente	212

19.4 Um pouco mais sobre páginas JSP	213
19.4.1 JSP vs. Servlets	215
19.5 Um pouco mais sobre EJB's	216
19.5.1 Ciclo de Vida dos EJB's	217
V Desenvolvimento Web	220
20 Usabilidade	221
20.1 Definição	221
20.2 Princípios da usabilidade	222
20.3 Técnicas de avaliação de usabilidade	223
21 Acessibilidade	224
21.1 Definição	224
21.2 Princípios da acessibilidade	224
21.3 Técnicas de avaliação de acessibilidade	227
22 Padrões Web W3C	229
23 XML	233
23.1 O que é XML?	233
23.2 Características do XML	234
23.3 Comparação entre XML e HTML	234
23.4 Sintaxe básica do XML	234
23.5 Conjunto de tags	236
23.6 NameSpaces	238
23.7 Gramática de um documento XML	239
23.8 Tecnologias XML	244
23.9 Benefícios da linguagem XML	245
23.10 Ferramentas de desenvolvimento	246
24 XSLT	247
24.1 O que é uma folha de estilo?	247
24.2 Comparação entre o CSS e XSL	247
24.3 O que é o XSL?	248
24.4 O que é o XSLT?	248
24.5 Características do XSLT	249
24.6 Declarando um documento XSL	249
24.7 Elemento <xsl:template>	250
24.8 Elemento <xsl:value-of>	251
24.9 Elemento <xsl:for-each>	252
24.10 Elemento <xsl:sort>	252
24.11 Elemento <xsl:if>	252
24.12 Elemento <xsl:choose>	253
24.13 Elemento <xsl:apply-templates>	253
24.14 XSL no lado Cliente	254
24.15 XSL no lado Servidor	254
24.16 Processadores XSLT	255

25 Gerenciador de Conteúdo Web Zone/Plone	256
25.1 Gestão de Conteúdo	256
25.2 Sistema de Gestão de Conteúdo	257
25.3 Zope	258
25.4 Plone	260
26 Web Services	263
26.1 O que é Web Services?	263
26.2 SOAP	266
26.3 WSDL	267
26.4 UDDI	269
26.5 Segurança	269
VI Redes de Comunicação	270
27 Técnicas Básicas de Comunicação	271
27.1 Base Teórica da Comunicação de Dados	271
27.2 Taxa Máxima de Dados em um Canal	272
27.3 Sinais Digitais Binários	272
27.4 Transmissão em Banda Base	273
27.5 Classificação dos Sinais	273
27.6 Técnicas de Codificação de Linha	274
27.6.1 Codificação NRZ	274
27.6.2 Codificação RZ	275
27.6.3 Codificação AMI (Alternate Mark Inversion)	275
27.6.4 Codificação HDB-3 (High Density Bipolar with 3 Zero Maximum Tolerance)	275
27.6.5 Codificação Manchester	276
27.7 Modulação	276
27.7.1 Modulação de Onda Contínua	277
27.7.2 Modulação de Pulso	279
27.8 Técnicas de Multiplexação	280
27.8.1 FDM - Frequency Division Multiplexing	281
27.8.2 TDM - Time Division Multiplexing	281
27.8.3 OFDM	281
27.8.4 WDM - Wavelength Division Multiplexing	282
27.9 Protocolos de Acesso Múltiplo	283
28 Topologias de Redes	284
29 Arquitetura de Redes	286
29.1 Organização em Camadas	286
30 Protocolos de Rede	287
30.1 ARP - Address Resolution Protocol	287
30.2 DHCP - Dynamic Host Configuration Protocol	287
30.3 DNS - Domain Name System	289
30.4 TCP - Transmission Control Protocol	291
30.5 UDP - User Datagram Protocol	293

30.6 HTTP - Hyper Text Transfer Protocol	294
30.7 SMTP - Simple Mail Transfer Protocol	299
30.8 POP3 - Post Office Protocol Version 3	301
30.9 IMAP - Internet Mail Access Protocol	303
30.10LDAP - LightWeight Directory Access Protocol	305
30.11SNMP - Simple Network Management Protocol	305
30.12FTP - File Transfer Protocol	306
30.13IP - Internet Protocol	310
30.14TELNET - TELeType NETwork	311
31 O Modelo de Referência OSI	314
32 Roteamento	316
32.1 Link State e Distance Vector	317
32.1.1 Vetor de Distâncias vs. Estado do Link	319
32.2 Protocolos de Roteamento	320
32.2.1 RIP - Routing Information Protocol	320
32.2.2 OSPF - Open Shortest Path First	320
32.2.3 IGRP e EIGRP	321
33 Redes Ethernet	322
33.1 Protocolo CSMA/CD	322
33.2 Fast Ethernet	323
33.3 Gigabit Ethernet	324
34 Cabecimento Estruturado	326
34.1 Par Trançado	326
34.1.1 Interferências nos Cabos de Par Trançado	326
34.2 Categorias 5e	327
34.3 Categoria 6	328
34.4 Categoria 5e vs. Categoria 6	328
34.5 Cabecão Estruturada – Norma EIA/TIA 568	329
34.5.1 Sistemas de Cabecamento Estruturado	329
34.6 Desempenho do Hardware e Meios de Transmissão	333
34.6.1 Cabecamento UTP	334
34.6.2 Fibra Óptica	335
34.7 Código de Cores para Sistemas de Cabecão UTP	336
35 Redes sem fio	337
35.1 O padrão IEEE 802.11	337
35.1.1 CSMA/CA	338
35.1.2 Formato do Quadro 802.11	339
36 Elementos de Interconexão de Redes de Computadores	340
36.1 Repetidores	340
36.2 Hubs	340
36.3 Switches	341
36.4 Bridges	342
36.5 Roteadores	342
36.6 Gateways	343

37 Redes Multimídia	344
37.1 Qualidade de Serviço	344
37.2 Serviços Integrados - <i>IntServ</i>	346
37.3 Serviços Diferenciados - <i>DiffServ</i>	347
38 Redes X.25 e Frame Relay	348
38.1 X.25	348
38.2 Frame Relay	348
38.2.1 Estrutura do Frame	349
38.2.2 Envio de um datagrama IP de Ethernet para Frame Relay e Ethernet	350
38.3 Interligação de Redes LAN	351
38.3.1 Voz sobre Frame Relay (VoFR)	351
38.3.2 Interação entre Frame Relay e ATM	352
38.3.3 CIR (Taxa de Informação Comprometida)	352
39 Redes Virtuais Locais	354
39.1 VLANs	354
39.1.1 Definição	354
39.1.2 Protocolo 802.1q	354
40 Redes de Circuito Virtuais	356
40.1 Redes ATM	356
40.2 MPLS - Multiprotocol Label Switching	358
41 Arquitetura TCP/IP	360
41.1 Visão geral	360
41.2 Comparação entre a arquitetura OSI e TCP/IP	360
41.3 Camada Física (host/rede)	361
41.4 Camada de Inter-Rede	361
41.5 Camada de Transporte	362
41.6 Camada de Aplicação	362
42 Camada de Aplicação	364
42.1 Proxy Cache	364
VII Gerência de Redes	366
43 O protocolo SNMP	367
43.1 Management Information Base	368
VIII Segurança da Informação	370
44 Políticas de Segurança de Informação	371
44.1 Políticas de Segurança	371
44.2 Projeto de Segurança	372
44.3 Plano de Segurança	372
44.4 Normas de Segurança	373
44.4.1 ISO/IEC 17799	373

44.4.2 Família ISO 27000	375
44.4.3 Diferenças entre a ISO/IEC 17799 e a ISO 27001	376
44.5 Procedimentos de Segurança	376
44.6 Arquitetura de Segurança	377
44.7 Classificação de Informações	377
45 Segurança Física e Lógica	379
45.1 Segurança Física	379
45.2 Segurança Lógica	379
45.2.1 Matrizes de acesso, listas de controle de acesso e capabilities	379
45.2.2 Modelos de Controle de Acesso	380
46 Backup de Dados	384
46.1 Meios de Armazenamento	384
47 Vírus e Ataques	386
47.1 Estratégias de combate à pragas eletrônicas	388
47.1.1 Antivírus	388
48 Princípios de Criptografia	390
48.1 Tipos de Criptografia	391
48.2 Algoritmos de Criptografia Simétricos	392
48.3 Algoritmos de Criptografia Assimétricos	393
48.4 Técnicas de Quebra de Criptografia	394
49 Autenticação	395
49.1 Autenticação de Mensagens	395
49.2 Protocolos de Autenticação	396
49.2.1 Métodos de Autenticação	396
49.2.2 Autenticação baseada em uma chave secreta compartilhada	396
49.3 Certificado Digital	397
50 Segurança em diversas camadas	399
50.1 <i>Secure Sockets Layer</i>	399
50.2 IPSec	400
50.3 Virtual Private Network (VPN)	401
50.4 Filtragem de Pacotes e Firewalls	403
50.4.1 Regras iptables - Exemplo 1	405
50.4.2 Regras iptables - Exemplo 2	405
50.4.3 Firewall Stateful	406
50.4.4 Application Gateway	407
50.4.5 Arquitetura de firewall e DMZ	407
50.5 Sistemas de Detecção de Intrusão (IDS)	409
50.6 Segurança em Redes Wireless 802.11	409
50.6.1 WEP	409
50.7 802.11i	410

IX Alta Disponibilidade	411
51 Soluções de Armazenamento RAID, SAN e NAS	412
51.1 RAID	412
51.1.1 RAID 0	412
51.1.2 RAID 1	414
51.1.3 RAID 2	415
51.1.4 RAID 3	416
51.1.5 RAID 4	417
51.1.6 RAID 5	418
51.1.7 RAID 6 (Redundância de P+Q)	419
51.1.8 Tipos Híbridos	419
51.1.9 Comparativo de Desempenho entre as diversas configurações RAID	420
51.2 SAN - Storage Area Network	420
51.2.1 Hardware para SAN	421
51.2.2 Topologias de SAN	422
51.3 NAS - Network Attached Stotage	423
51.4 Comparativo entre SAN e NAS	424
52 Clusters de servidores	427
52.0.1 Princípios de um Cluster	427
52.0.2 Abstrações em um Cluster	428
52.0.3 Arquitetura de um Cluster	429
52.0.4 Cluster X Sistemas Distribuídos	430
52.0.5 Cluster de Alta Disponibilidade	431
52.0.6 Cluster de Alto Desempenho	433
53 Balanceamento de Carga	436
53.1 Balanceamento de armazenamento (storage)	436
53.2 Balanceamento de rede	436
53.2.1 NAT	437
53.2.2 IP Tunneling	437
53.2.3 Direct Routing	438
53.3 Algoritmos de balanceamento	438
53.4 Balanceamento de CPU	439
53.4.1 Sistema de processamento distribuído	439
X Sistemas Operacionais	442
54 Ambiente Microsoft Windows 2000/2003	443
54.1 DHCP - Dynamic Host Configuration Protocol	443
54.1.1 Processo de Instalação/Configuração	443
54.1.2 Integração do DHCP com o DNS	445
54.1.3 APIPA - Automatic Private IP Addressing	445
54.1.4 Comandos ipconfig Relacionados ao DHCP	446
54.1.5 Regra "80/20"	446
54.2 DNS - Domain Name System	446
54.2.1 Processo de Instalação/Configuração	447

54.2.2 Segurança de Acesso	449
54.2.3 Integração do DNS com o Active Directory	449
54.2.4 Servidor DNS somente Cache	451
54.2.5 Arquivo Hosts	451
54.2.6 Distribuição de Carga	451
54.2.7 Comando ipconfig/dnscmd Relacionadas ao DNS	451
54.3 Active Directory	452
54.3.1 Tipos de Servidores	453
54.3.2 Definições de Floresta, Domínio, Site e Unidade Organizacional	453
54.3.3 Recursos do Active Directory	454
54.3.4 Segurança com o Active Directory	455
54.3.5 Ferramentas de Controle	456
54.4 IIS - Internet Information Services	456
54.4.1 IIS versus Apache HTTP Server	456
54.4.2 Principais Componentes do IIS	459
54.4.3 Principais Recursos do IIS	460
54.4.4 Principais Diferenças entre IIS4, IIS5 e IIS6	461
54.5 Terminal Services	461
54.5.1 Principais Benefícios	462
54.5.2 Protocolos de Comunicação	463
54.5.3 Licenças	464
XI Banco de Dados	465
55 Conceitos Básicos	466
56 Abordagem Relacional	468
56.1 Conceitos	468
56.2 Esquemas e Restrições de Integridade	468
57 Modelagem Entidade Relacionamento	470
57.1 Conceitos	470
57.2 Cardinalidade	471
57.3 Representação Gráfica	471
57.4 Recursos do Modelo Entidade Relacionamento	471
58 Normalização	473
58.1 Aspectos desejáveis em um bom projeto	473
58.2 Forma normal de Boyce-Codd	473
58.3 Terceira forma normal	474
59 Transformação do Modelo Conceitual	475
60 Linguagem SQL	476
60.1 Criação de tabela	476
60.2 Consultas	476
60.3 Funções de agregação	477
60.4 Atualizações e exclusões	478
60.5 Visões	479

60.6 Chaves estrangeiras	479
61 Conceitos de Datawarehousing e Bussiness Intelligence	480
61.1 Banco de Dados Multidimensionais	480
61.1.1 Modelagem Multidimensional	481
61.2 Datawarehousing	483
61.3 OLTP, OLAP, MOLAP, ROLAP e HOLAP	485
61.4 Outros conceitos importantes	487
XII Administração de Bancos de Dados Relacionais	489
62 Gerência de Transações	490
63 Controle de Concorrência	492
64 Gerência de Desempenho	494
XIII Oracle e Microsoft SQL Server	497
65 Administração de Bancos de Dados Oracle	498
65.1 Arquitetura de um Servidor Oracle	498
65.1.1 Estruturas em memória	498
65.1.2 Processos server	499
65.1.3 Processos user	499
65.1.4 Processos em Background	499
65.1.5 Arquivos	500
65.2 Arquitetura Oracle de Armazenamento de Dados	501
65.3 Tratamento de Transações no Oracle	501
65.3.1 Gerenciamento do <i>Redo Log</i>	501
65.3.2 Checkpoints	502
65.3.3 Segmentos de rollback	502
65.3.4 Consistência de leitura	503
65.4 Configuração do Servidor	503
65.5 Tipos de Usuários Oracle	504
65.5.1 Administradores de banco de dados	504
65.5.2 Outros papéis	505
66 Administração de Bancos de Dados SQL Server	506
66.1 Arquitetura de um Servidor SQL Server	506
66.1.1 Catálogos de sistema	506
66.1.2 Processos em <i>background</i>	507
66.2 Arquitetura SQL Server de Armazenamento de Dados	507
66.3 Tratamento de Transações no SQL Server	507
XIV ITIL	509
67 Suporte a Serviços	510
67.1 Service Desk	510

67.1.1	Objetivos	510
67.1.2	Responsabilidades	510
67.1.3	Vários Tipos de Central	511
67.2	Gerenciamento de Incidentes	511
67.2.1	Objetivos	511
67.2.2	Atividades do Processo	511
67.2.3	Papéis e Responsabilidades	512
67.3	Gerenciamento de Problemas	512
67.3.1	Objetivos	512
67.3.2	Definições Importantes	513
67.3.3	Atividades do Processo	513
67.3.4	Papéis e Responsabilidades	513
67.4	Gerenciamento de Configuração	514
67.4.1	Objetivos	514
67.4.2	Atividades	514
67.4.3	Papéis e Responsabilidades	515
67.5	Gerenciamento de Mudanças	515
67.5.1	Objetivos	515
67.5.2	Responsabilidades	515
67.5.3	Definições Importantes	515
67.6	Gerenciamento de Liberação	516
67.6.1	Objetivo	516
67.6.2	Atividades do Processo	516
67.6.3	Definições Importantes	517
67.6.4	Papéis e Responsabilidades	517
68	Entrega de Serviços	518
68.1	Gerenciamento do Nível de Serviço	518
68.1.1	Objetivos	518
68.2	Gerenciamento Financeiro	519
68.2.1	Objetivos	519
68.2.2	Responsabilidades	519
68.2.3	Atividades do Processo	519
68.2.4	Elementos de Custo	520
68.3	Gerenciamento da Capacidade	521
68.3.1	Objetivos	521
68.3.2	Atividades	521
68.4	Gerenciamento de Disponibilidade	522
68.4.1	Objetivos	522
68.4.2	Ciclo de vida do incidente	522
68.5	Gerenciamento de Continuidade	523
68.5.1	Objetivos	523
68.5.2	Estágios	523
68.5.3	Tipos de Continuidade	524
XV	Gerência de Projetos segundo PMBOK	525
69	Gerenciamento de Escopo	526
69.1	WBS e Definição do Escopo	526

70 Gerenciamento de Recursos Humanos	528
70.1 Estruturas Organizacionais	528
70.1.1 Organização Funcional	528
70.1.2 Organização por Projeto	529
70.1.3 Organização Matricial	530
70.2 Planejamento Organizacional	531
70.3 Desenvolvimento da Equipe	531
71 Gerenciamento do Tempo	534
71.1 Técnicas de Desenvolvimento do Cronograma	534
71.1.1 Análise Matemática	535
71.1.2 Compressão do Cronograma	535
71.1.3 Simulação	536
71.1.4 Heurística do nivelamento de recursos	536
71.1.5 Estrutura de Codificação	536
72 Gerenciamento de Custo	537
72.1 Técnicas de Estimativas de Custos	537
72.1.1 Estimativas Análogas	537
72.1.2 Modelagem Paramétrica	538
72.1.3 Estimativa bottom-up	538
73 Gerenciamento de Riscos	539
73.1 Análise Qualitativa	539
73.2 Análise Quantitativa de Riscos	540
74 Gerenciamento de Qualidade	542
74.1 Técnicas de Planejamento da Qualidade	542
74.1.1 Análise Custo/Benefício	542
74.1.2 Benchmarking	543
74.1.3 Fluxograma	543
74.1.4 Elaboração de Experimentos	543
74.1.5 Custo da Qualidade	544
74.2 Técnicas de Controle da Qualidade	544
74.2.1 Gráficos de Controle	545
74.2.2 Diagramas de Pareto	545
74.2.3 Diagramas de Dispersão	546
75 Gerenciamento da Comunicação	547
75.1 Um mais sobre Planejamento da Comunicação	547
76 Gerenciamento das Aquisições	548
76.1 Um pouco mais sobre Planejamento de Aquisições	548
77 Gerenciamento da Integração	550
77.1 Ferramentas de Apoio à Integração	550
78 Sobre os Ciclos do Projeto e Processos de Gerenciamento	551

Parte I

Fundamentos de Computação

Capítulo 1

Arquitetura e Organização de Computadores

1.1 Conceitos Básicos

Dois conceitos fundamentais no estudo dos sistemas de computação são o de Arquitetura e Organização de computadores. O termo arquitetura refere-se aos atributos do ponto de vista do programador, e portanto, têm impacto direto sobre sobre a execução lógica de um programa. O termo organização, refere-se às unidades operacionais e suas interconexões. Desta forma, uma mesma arquitetura pode ser implementadas por meio de diferentes organizações.

As funções básicas de um computador são o processamento de dados, armazenamento de dados, transferência de dados e controle. Para desempenhar essas funções o computador precisa executar um conjunto de instruções (programa). Os computadores que conhecemos são baseados no conceito de programa armazenado, introduzido por *Von-Neuman*. As instruções do programa e os dados são armazenados em uma memória, de forma que a alteração de um programa consiste na alteração de um endereço de memória.

O ciclo de execução de cada uma das instruções de um programa é dividido nos seguintes estados: (i)Calculo do Endereço de Instrução; (ii)Busca da Instrução (Instruction Fetch); (iii) Decodificação da Instrução; (iv)Cálculo do Endereço do Operando; (v)Busca do Operando (Operand Fetch); (vi)Execução da Operação; (vii)Armazenamento do Resultado. No entanto, os computadores modernos utilizam o conceito de interrupção para diminuir o tempo de ociosidade dos processadores, o ciclo de execução das instruções ganham mais alguns estados. As classes de interrupções mais comuns são interrupções de software, de relógio, de E/S e de falha de hardware.

A estrutura básica de um computador é composta pelos seguintes componentes:

- (i)Unidade Central de Processamento(CPU);
- (ii)Memória Principal;

- (iii) Dispositivos de E/S;
- (iv) Sistemas de Interconexão.

Esses componentes também possuem suas subdivisões. A CPU por exemplo se subdivide em: Unidade de Controle, Unidade Lógica e Aritmética (ALU), Registradores e por fim as Interconexões da CPU. Cada um desses componentes será melhor descrito posteriormente. Para interconectar dois ou mais dispositivos em um sistema são utilizados os chamados barramentos. Os barramentos são compostos por linhas que podem ser de Dados, Endereço ou Controle. Os barramentos de controle podem ser utilizados por exemplo para controlar direito de leitura ou escrita em memória ou E/S, interrupções, confirmações, relógio e reset. O projeto dos barramentos que compõe um sistema são de grande importância no desempenho do sistema. Questões importantes no projeto de barramentos são:

- (i) Tipo - dedicado ou multiplexado;
- (ii) Método de Arbitragão - Centralizado ou Distribuído;
- (iii) Temporização - Síncrona ou Assíncrona;
- (iv) Largura - número de linhas;
- (v) Tipo de Transferência - leitura, escrita, leitura/modificação/escrita, escrita/leitura, em bloco.

Para aumentar o desempenho do sistema, os barramentos são organizados de forma hierárquica, de forma isolar o tráfego de dados entre CPU e memória do tráfego proveniente de operações de E/S. Os chamados barramentos de expansão proporcionam maior flexibilidade ao sistema (ex: SCSI), enquanto os barramentos de alta velocidade são utilizados para dispositivos de alta capacidade (ex: FireWire).

1.2 Estrutura e Funcionamento da CPU

Os principais elementos da CPU são a Unidade de Controle , a Unidade Lógica e Aritmética (ULA) e os Registradores. Esses elementos se conectam internamente através do barramento interno da CPU.

A CPU se comunica com o mundo externo através dos barramentos do sistema. Ao longo da execução de um programa, os barramentos constituem os chamados *caminho dos dados*. No topo da organização hierárquica de memória em um sistema se encontram os registradores. Esses se dividem em dois tipos: Registradores visíveis ao Usuário e Registradores de Controle e de Estado.

Os registradores visíveis ao usuário são aqueles que podem ser referenciados pela linguagem de montagem. Eles podem ser registradores de dados, endereço ou então de propósito geral. Os registradores de Controle e de Estado são utilizados para controlar a operação da CPU. Na maioria das vezes não são visíveis aos usuários. Exemplos de registradores de Controle e de Estado são o *Program*

Counter (PC), Instruction Register (IR), Memory Address Register (MAR), Memory Buffer Register (MBR), Program Status Word (PSW), Stack Pointer (SI), Page Table Base Register (PTBR), Page Table Base Limit (PTBL).

A seqüência de eventos ao longo de um ciclo de instrução depende do projeto da CPU, no entanto, em termos gerais, pode-se indicar o que acontece em nos subciclos de *busca*, *indireto* e *interrupção*. O ciclo de *execução* depende do código da operação que será executada. A figura 1.1 mostra o diagrama de transição de estados do ciclo de instrução.

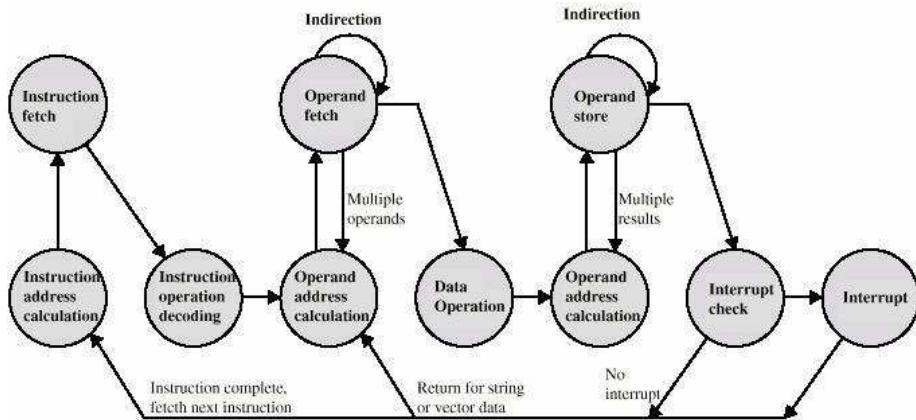


Figura 1.1: Transição de Estados do Ciclo de Instrução

Durante o ciclo de busca, o contador de programa contém o endereço da próxima instrução a ser buscada na memória. Esse endereço é movido para o registrador MAR e a unidade de controle requisita uma leitura na memória. O resultado da leitura é colocado no registrador MBR, que em seguida é copiado para o registrador IR. Enquanto isso o PC é incrementado de 1 para preparar a busca da próxima instrução. O fluxo de dados do ciclo de busca é mostrado na figura 1.2.

Ao fim do ciclo de busca, o unidade de controle examina se a instrução especifica algum operando com endereçamento indireto. Os n bits mais a direita de MBR são colocados em MAR, e então a unidade de controle requisita uma leitura a memória para carregar o valor do operando para MBR. O fluxo de dados do ciclo de indireto é mostrado na figura 1.3.

No ciclo de interrupção, o conteúdo do registrador PC dever ser salvo, para que mais tarde a CPU possa retornar sua atividade normal depois de processar a interrupção. O conteúdo do PC é transferido para MBR. A endereço de memória reservado para guardar o valor de PC (ex: topo da pilha) é carregado para MAR, e então a unidade de controle solicita uma escrita na memória. Por fim o PC é carregado com o endereço da rotina de interrupção, para que o no próximo ciclo de instrução seja feita a busca da instrução apropriada. A figura 1.4 mostra o fluxo de dados do ciclo de interrupção.

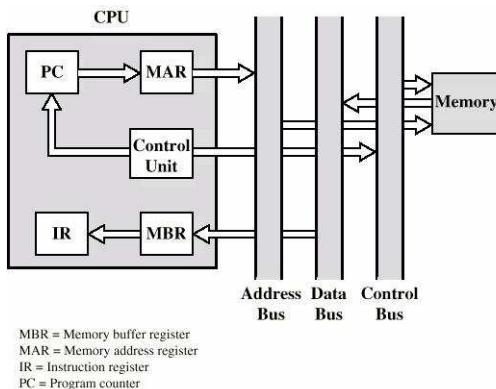


Figura 1.2: Fluxo de Dados do Ciclo de Busca

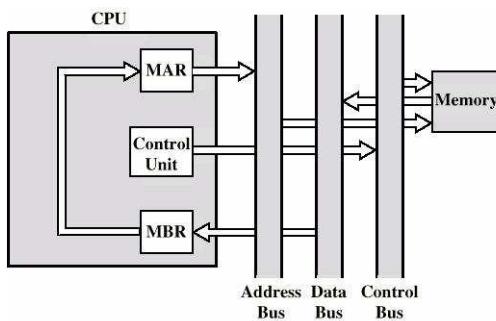


Figura 1.3: Fluxo de Dados do Ciclo de Indireto

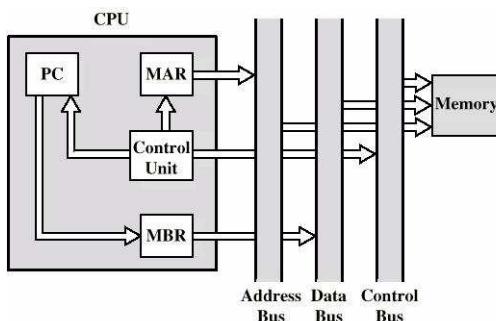


Figura 1.4: Fluxo de Dados do Ciclo de Interrupção

1.2.1 Pipelines

Como pudemos ver, um ciclo de instrução pode subdividido em etapas menores. Uma divisão comum é a baseada nos ciclos de busca, indireto, execução e interrupção. A idéia da técnica de *pipeline* é trabalhar as diversas etapas do ciclo de instrução de forma paralela, e não de forma serial, de forma aumentar o

desempenho da CPU.

A técnica de *pipeline* pode ser descrita genericamente como uma estratégia de aceitar novas entradas em uma extremidade sem que as entradas prévias tenha aparecido como saídas na outra extremidade. Imaginemos um esquema em que o ciclo de instrução é subdividido em 2 etapas que são a busca da instrução e a execução. Enquanto uma instrução está sendo executada, a próxima instrução pode estar sendo buscada. Este esquema configura um *pipeline* de 2 estágios, e a técnica utilizada é a de busca antecipada de instrução (*Instruction Prefetch* ou *Fetch Overlap*). Embora possa ser obtido um ganho de desempenho, o *pipeline* de 2 estágios ainda é muito fraco para lidar com instruções de desvio, nas quais a busca da próxima instrução depende do resultado da execução da instrução corrente.

Portanto, uma subdivisão do ciclo de instrução em um número maior de estágios pode proporcionar maior desempenho. Imaginemos o ciclo de instrução como sendo composto dos seguintes estágios:

- (1)BI - Busca da Instrução;
- (2)DI - Decodificação da Instrução;
- (3)CO - Cálculo dos Endereços dos Operandos;
- (4)BO - Busca dos Operandos;
- (5)EI - Execução da instrução;
- (6)EO - Escrita do Operando.

Essa subdivisão busca fazer com que os tempos gastos em cada um dos 6 estágios seja parecido. Se os 6 estágios pudessem ser executados em paralelo, o desempenho seria aproximadamente 6 vezes maior.

No entanto, podem existir conflitos de acesso à memória por parte dos estágios e nem todas as instruções possuem os seis estágios. Somam-se à esses dois problemas, a existência de instruções de desvio, principais inimigas da técnica de *pipeline*. Para lidar com os problemas introduzidos pelas instruções de desvio são utilizadas técnicas como:

- Múltiplos Fluxos (duplicação de estágios iniciais);
- Busca Antecipada de Instrução-Alvo de Desvio;
- Memória de Laço de Repetição (*loop buffer*);
- Previsão de Desvio baseadas no *opcode*;
- Previsão de Desvio baseadas em histórico (BTB - *Branch Target Buffer*, também conhecida como BHT - *Branch History Table*).

As figuras 1.5 e 1.6 mostram os diagramas de tempo para o pipeline de instruções com e sem instruções de desvio.

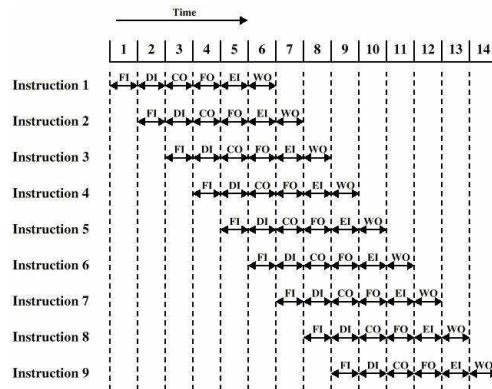


Figura 1.5: Diagrama do Tempo para *Pipeline* de Instruções

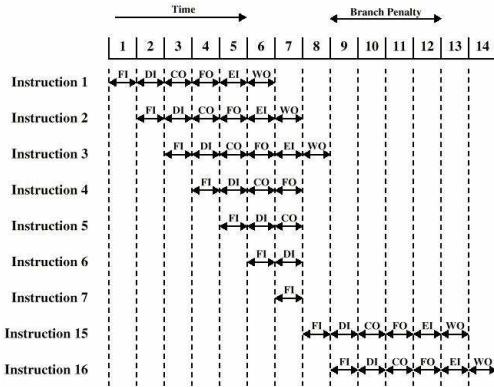


Figura 1.6: Efeito do desvio condicional no *Pipeline* de Instruções

1.3 Conjunto de Instruções

A operação da CPU é determinada pelo conjunto de instruções que ela executa. São as chamadas instruções de máquina. A coleção de instruções que uma CPU pode executar é chamada Conjunto de Instruções. O conjunto de instruções deve ser suficiente para traduzir programas escritos em uma linguagem de alto nível para a linguagem de máquina. Os principais elementos de uma instrução são:

- (i) Código da operação (opcode): especifica a operação a ser efetuada;
- (ii) Referência à Operando Fonte: indica as entradas para a operação;
- (iii) Referência ao operando Destino: A operação pode produzir um resultado;
- (iv) Endereço da próxima instrução: Pode ser indicado implicitamente (ex: registrador PC).

Os operandos fonte e destino podem estar localizados em memória (principal ou virtual), em algum registrador, ou em algum dispositivo de E/S.

Cada instrução de um computador é representada internamente como um conjunto de bits. A instrução é dividida em campos correspondentes aos elementos da instrução. O papel da CPU é ler a instrução, extrair informação de cada um dos campos e efetuar a operação. Devido às dificuldades de se lidar com a representação binária, é utilizada uma espécie de notação simbólica para representar as instruções de máquina. Os códigos das operações são representados por mnemônicos. (ex: ADD, SUB, MPY, DIV, LOAD, STOR). O mesmo ocorre para os operandos. Um conjunto de instruções pode apresentar mais de um formato de instrução.

As instruções podem ser classificadas em:

- (i)Processamento de Dados: instruções lógicas e aritméticas;
- (ii)Armazenamento de dados: instruções de memória;
- (iii)Movimentação: instruções de E/S;
- (iv)Controle: instruções de teste e desvio.

No projeto do conjunto de instruções as questões mais relevantes são o repertório de operações, os tipos de dados, o formato e tamanho das instruções, os registradores acessíveis, e os modos de endereçamento. As classes de dados sobre as quais as instruções de máquina operam são endereços, números (ex: ponto fixo, ponto flutuante, decimal), caracteres (ex: ASCII e EBCDIC) e dados lógicos.

Os tipos de operações mais comuns são:

- (i)Transferência de Dados: mov,push/pop,xlat,in/out;
- (ii)Aritméticas: add,sub,mul,idiv;
- (iii)Lógicas: and,or,shl/shr;
- (iv)Conversão de Tipos: jmp,call,loop,int/into;
- (vi)Controle do Sistema: hlt,wait;
- (vii)Transferência de Controle: blt,bgt,beq,call,jmp.

Nas operações de salto e desvio, é importante conhecer cada um dos códigos de condição envolvidos. (ex: Vai-Um, Zero, Paridade, Sinal, Overflow) Na implementação das chamadas de procedimento é importante ressaltar a utilização de pilhas para permitir chamadas reentrantes (uma chamada dentro da outra).

1.4 Unidade de Controle

A unidade de controle coordena os diversos elementos do processador para que este possa realizar todas as suas funções. A execução de um programa consiste de uma seqüência de ciclos de instrução. Um ciclo de instrução pode ser subdividido em quatro subciclos que são *busca*, *indireto*, *execução* e *interrupção*. Somente os ciclos de busca e execução estão presentes em todos os ciclos de instrução. Cada subciclo, por sua vez, é composto por *microoperações*.

Os quatro registradores básicos da unidade de controle são:

- PC (*Program Counter*): Mantém o endereço da próxima instrução a ser buscada na memória;
- MAR (*Memory Address Register*): Especifica endereço de memória para uma operação de leitura ou escrita;
- MBR (*Memory Buffer Register*): Conectado ao barramento do sistema. Contém um valor a ser armazenado na memória ou o último valor dela lido;
- IR (*Instruction Register*): Mantém a última instrução buscada na memória.

O ciclo de busca ocorre no início de cada ciclo de instrução, fazendo com que a instrução seja obtida na memória no endereço indicado pelo registrador PC, e armazenada no registrador IR. Uma vez completada essa etapa, pode ser necessário que se busquem operandos para a instrução. Isso é realizado no ciclo de indireto. Após o término do ciclo de execução, é feita uma checagem para determinar se ocorreu alguma interrupção, e neste caso o conteúdo de PC é salvo em memória e PC é carregado com um endereço da rotina de interrupção apropriada. Os ciclos de busca, indireto e de interrupção envolvem um número pequeno e fixo de microoperações. Isso não ocorre nos ciclos de execução. Em uma máquina com N códigos de instrução podem existir N diferentes seqüências de microoperações para o ciclo de execução.

Todas as microoperações caem em uma das seguintes categorias:

- (i) Transferência de dados entre registradores;
- (ii) Transferência de dados entre registrador e interface externa (barramento);
- (iii) Transferência de dados de interface externa para registrador;
- (iv) Execução de operações lógicas e aritméticas, usando registradores como entrada e saída.

Portanto, a unidade de controle desempenha duas tarefas básicas que são o *seqüenciamento* e a *execução* das microoperações. A base para o funcionamento da unidade de controle são os *sinais de controle*, que constituem as entradas e saídas.

As unidades de controle podem ser implementadas tanto em hardware quanto em software (*Microprogramação*). A implementação baseada em microprogramação é mais simples e também mais barata do que as implementações em hardware. As implementações em hardware envolvem a criação de uma lógica complexa para sequenciamento das microoperações, o que a torna mais cara. No entanto, é mais eficiente dos que as implementações basadas em microprogramação. As arquiteturas CISC geralmente utilizam unidades de controle microprogramadas, devido ao grande número de instruções e sua complexidade. Já as arquiteturas RISC, geralmente utilizam implementações baseadas em hardware uma vez que o número de instruções é reduzido e de baixa complexidade.

As técnicas de microprogramação também podem ser aplicadas em *emulações*, permitindo que máquinas rodem programas escritos originalmente para outras máquinas. Microprogramas podem conferir maior desempenho ao sistema operacional se forem utilizados na implementação de certas primitivas. Além disso, as técnicas de microprogramação podem ser utilizadas na implementação de dispositivos de propósito especial, por exemplo uma placa de rede (*firmware*).

1.5 Modos de Endereçamento

Os modos de endereçamento estão relacionados com a forma utilizada para especificar o valor ou endereço de um operando de uma instrução. Questões importantes na escolha do modo de endereçamento são a quantidade de posições de memória endereçáveis, flexibilidade de endereçamento, número de referências a memória feitas pela instrução e complexidade do cálculo do endereço. Em geral, as arquitetura não oferecem só um modo de endereçamento, mas sim um conjunto de modos. Por este motivo, é necessário que exista uma forma de se identificar qual o modo de endereçamento que se está utilizando. Isso é feito através do *campo de modo de endereçamento*, que consiste em um subconjunto dos bits de uma instrução.

- *Imediato*: o valor do operando é especificado diretamente na instrução. Sua principal vantagem é não requer acesso a memória para obter o operando. A desvantagem é que esse modo impõe uma limitação no tamanho do operando;
- *Direto*: o campo de endereço contém o endereço efetivo do operando na memória. Requer portanto apenas um acesso para determinar o valor do operando. Sua limitação é fornecer um espaço de endereçamento limitado;
- *Indireto*: o campo de endereço aponta para uma posição de memória que contém o endereço de memória do operando. Sua principal desvantagem é a necessidade de dois acessos à memória. A vantagem em relação ao modo de endereçamento direto é o aumento do espaço de endereçamento, que passa a ser igual 2^n onde n é o tamanho da palavra;
- *Registrador*: é semelhante ao modo direto, no entanto o modo de endereço se refere a um registrador e não à uma posição de memória. Geralmente é

composto por 3 ou 4 bits, o que permite referenciar de 8 a 16 registradores de propósito geral. Suas vantagens são o tamanho pequeno do campo de endereço e não necessidade de se acessar a memória. Sua desvantagem é o espaço de endereçamento limitado pelo número de registradores;

- *Indireto via Registrador*: semelhante ao modo de endereçamento indireto. O campo de endereço aponta para o registrado que contém a posição de memória do operando. Sua vantagem é a necessidade de um único acesso a memória, um a menos que no modo indireto;
- *Deslocamento*: requer que uma instrução tenha dois campos de endereço, com pelo menos um explícito. O valor de um dos campos é usado diretamente (valor = A). O outro campo é baseado no código da operação, e especifica um registrador cujo conteúdo é adicionado a A, para produzir o endereço efetivo. Os três modos de endereçamento por deslocamento são *relativo*, *via registrador-base* e *indexado*;
- *Pilha*: A pilha é um bloco reservado de posições em memória. Elementos podem ser colocados e removidos do topo da pilha. O apontador do topo da pilha (*stack-pointer*) é mantido em um registrador. Portanto, de fato, referências a pilha são feitas por endereçamento indireto via registrador.

1.6 Organização de Memória

Um sistema de computador típico é equipado com uma hierarquia de subsistemas de memória, algumas internas (diretamente acessíveis pelo processador, como registradores, cache e memória principal) e outras externas (acessíveis pelo processador por meio de um módulo de E/S, como disco e cdrom). As características fundamentais de um sistema de memória são:

1. Localização - processador, interna (principal), externa (secundária);
2. Capacidade - tamanho da palavra e número da palavra;
3. Unidade de Transferência - palavra, bloco;
4. Método de Acesso - seqüencial, direto, aleatório, associativo;
5. Desempenho - tempo de acesso, tempo de ciclo e taxa de transferência;
6. Tecnologia - semicondutores, magnética, óptica, magneto-óptico;
7. Características Físicas - volátil/não volátil, apagável/não apagável.

Quanto ao método de acesso das memórias internas, vale a pena destacar os acessos aleatório e associativo. No acesso aleatório, cada unidade endereçável possui um mecanismo de endereçamento único e fisicamente conectado a ela. É o método utilizado na memória principal. O esquema associativo consiste em um tipo de memória de acesso aleatório que possibilita comparar simultaneamente um certo número de bits de uma palavra com todas palavras da memória (totalmente associativo) ou com um conjunto de palavras de memória (associativo por conjunto). O método associativo é empregado pelas memórias cache.

As restrições de um projeto de memória podem ser resumidos por 3 questões: Capacidade, Velocidade e Custo. Neste cenário, valem as seguintes relações:

1. Menor tempo de acesso, maior custo por bit;
2. Maior capacidade, menor custo por bit;
3. Maior capacidade, menor tempo de acesso.

A organização hierárquica dos sistemas de memória visa lidar com o dilema imposto pelas relações apresentadas. A hierarquia é elaborada de forma que as a medida que nela desemos as seguintes relações são também válidas:

- 1 O custo por bit diminui;
- 2 A capacidade aumenta;
- 3 O tempo de acesso aumenta;
- 4 A freqüência de acesso pelo processador diminui.

Desse modo, as memórias menores, mais caras e mais rápidas são combinadas com memória de maior capacidade, mais lentas e baratas.

A chave do sucesso dessa organização baseia-se principalmente na relação 4, que resume o princípio da Localidade das Referências. Este princípio diz que ao longo da execução de um programa, as referências feitas à memória pelo processador, tanto no caso de instruções como dados, tendem a formar grupos no qual estão próximas umas das outras. Desse modo é possível organizar os dados ao longo de uma hierarquia de forma que a porcentagem de acessos à um certo nível seja sucessivamente bem inferior do que a porcentagem de acessos à um nível imediatamente superior.

Registradores, memória cache e memória principal são as três formas de memória interna que empregam tecnologias de semicondutores. O uso de três níveis explora as diferenças de velocidade e custo dessas memórias. Além delas, alguns sistemas utilizam tecnologias e técnicas adicionais na hierarquia de memória.

A Memória Expandida emprega uma tecnologia mais lenta que a memórias principais. Ela funciona como um ramo lateral a memória principal, não se comunicando com a memória externa. Já a técnica de Memória Virtual permite que os discos funcionem como uma extensão da memória principal, aumentando o desempenho do sistema.

A utilização de memórias cache tem por objetivo proporcionar uma velocidade de acesso próxima a velocidade de acesso aos registradores, no entanto oferecendo uma capacidade maior do que o conjunto de registradores, e custo não muito superior ao da memória principal. Os principais elementos de projeto de memórias cache são:

- i Tamanho - deve ser projetado para conjugar bem velocidade, capacidade e custo;

- ii Função de Mapeamento - direto, associativo, associativo por conjuntos;
- iii Algoritmo de Substituição - LRU, FIFO, LFU, Aleatório;
- iv Política de Escrita - direta (write-through), de volta (write-back) e única (write-once);
- v Tamanho da Linha;
- vi Número de Memórias Cache - um ou dois níveis, unificada/separada.

Entre os elementos de projeto de memória cache vale destacar três. O primeiro é a Função de Mapeamento, que diz respeito a determinar onde um bloco da memória principal pode ser encontrado na memória cache. Para realizar este mapeamento são necessários o endereço do bloco na memória principal e a função de mapeamento. No esquema de mapeamento direto, um determinado conjunto de blocos da memória principal só pode ser encontrado em uma linha específica da memória cache. É de fácil implementação, porém pode utilizar de forma inefficiente o espaço da cache. No mapeamento associativo um bloco da memória principal pode ser colocado em qualquer linha da cache. Maximiza a ocupação da cache, porém exige uma lógica de controle que realize comparação do rótulo com todas as linhas do cache simultaneamente. No esquema associativo por conjuntos, um bloco da memória principal pode se encontrar em um conjunto de linhas da cache, e não nela toda. Visa conjugar vantagens dos métodos direto e associativo.

O segundo elemento é Política de Escrita, que visa garantir a coerência das informações nos diferentes memórias acessíveis pelo processador e dispositivos de E/S. Na técnica é a de escrita direta (write-through), todas as operações de escrita são feitas na memória principal e no cache. Esta garante a coerência em todas as memórias do sistema, no entanto é de baixo desempenho. Na técnica de escrita de volta (write-back), as escritas são feitas apenas na cache. Minimiza as operações de escrita em memória principal, porém impõe que operações de E/S accessem o cache.

O terceiro elemento é o número de memórias cache do sistema. Atualmente, a organização mais comum é baseada em 2 níveis, um interno ao processador (L1) e outro externo (L2). Originalmente, a maioria dos projetos de cache inclui uma única memória cache, que armazenava tanto instruções como dados. Recentemente, tornou-se comum a utilização de memórias separadas para instruções e dados. Em processadores modernos que se valem de técnicas de busca antecipada de instrução (Pipeline), técnicas de Cache Separadas são mais eficientes que as de Cache Unificada.

1.7 Desempenho do computador

O desempenho de um computador pode ser definido como:

$$\text{Desempenho} = \frac{1}{\text{Tempo de Execução}} \quad (1.1)$$

O tempo é a medida de desempenho de um sistema computacional. Em geral, ele é medido em segundos e pode ser definido de diferentes maneiras. O tempo de resposta ou tempo decorrido (*elapsed time*) define o tempo total para se completar uma tarefa computacional, incluindo os acessos à memória e ao disco, as atividades de entrada e saída e o *overhead* do sistema operacional.

O tempo do processador (*CPU time*) define o tempo gasto pelo processador para executar um programa em particular, não considerando o tempo de execução de outros programas, tempo de espera por I/O, etc. Este tempo é dividido em tempo do usuário e tempo do sistema. O tempo do usuário é o tempo gasto na execução das instruções do programa do usuário. Já o tempo do sistema é o tempo gasto pelo sistema operacional para executar tarefas em benefício do programa do usuário. A medida de tempo que mais interessa é o tempo de usuário.

Os projetistas medem a velocidade do hardware na execução de suas funções básicas com o clock. O clock possui uma taxa constante e determina o momento da ocorrência de eventos do próprio hardware. O tamanho de um período de clock é referenciado tanto como o tempo necessário para completar um ciclo de clock quanto como a freqüência do clock (inverso do ciclo de clock). Por exemplo, um ciclo de clock igual a 2 ns corresponde a uma freqüência de 500MHz , que é o inverso do ciclo de clock.

1.7.1 Tempo de execução de um programa

Fórmulas bastante simples relacionam a medida do tempo de execução gasto no processador com a métrica básica baseada nos ciclos de clock e tempo do ciclo de clock:

$$\begin{aligned}\text{Tempo de CPU do programa} &= N \text{ de ciclos} \times \text{Período de clock} \\ &= N \text{ de ciclos} / \text{Freqüência do clock}\end{aligned}$$

Essas fórmulas não incluem qualquer referência ao número de instruções necessárias à execução de um programa. O tempo de execução também depende do número de instruções do programa. O número de ciclos de clock necessários à execução de uma instrução é dado por:

$$\text{Nº de ciclos de clock} = \text{Nº instruções do programa} \times \text{CPI} \quad (1.2)$$

A CPI é a média do número de ciclos por instrução. Este parâmetro permite a comparação entre diferentes implementações de uma mesma arquitetura do conjunto de instruções, uma vez que o número de instruções para a execução do programa nas diferentes implementações é o mesmo.

1.7.2 Desempenho da CPU

O desempenho da CPU na execução de um programa pode ser medido em termos quantidade de instruções, do CPI e do período do clock:

$$\text{Tempo de CPU} = \text{Nº de instruções} \times \text{CPI} \times \text{Período do clock} \quad (1.3)$$

O tempo de CPU é medido executando o programa, o período do clock é divulgado pelo fabricante e o número de instruções é obtido por meio de softwares conhecidos como *execution profilers* ou por simuladores de arquitetura.

Em uma primeira aproximação, o número de instruções, a CPI e o período do clock são afetados respectivamente pela capacidade de otimização do compilador, pela arquitetura do processador e de seu conjunto de instruções; e pela tecnologia empregada na implementação da máquina.

1.7.3 Programas para medir desempenho

Existem quatro níveis de programas que podem ser usados para avaliação de desempenho, eles estão listados em ordem decrescente de precisão de previsão:

- Programas reais;
- Núcleos ou kernels (pedaços de programas reais);
- Toy Benchmarks (programas com 10 a 100 linhas de código que produzem um resultado conhecido a priori);
- Benchmarks sintéticos (similar em filosofia aos núcleos, tentam casar a freqüência média de operações de um grande conjunto de programas).

Os benchmarks são conjuntos de aplicações que representam cargas de trabalho cujo objetivo é estimar o desempenho das cargas de trabalho reais. Os benchmarks podem conter aplicações típicas de processamento científico, compiladores, processadores de texto entre outras.

Um Benchmark Suite é um conjunto de programas de avaliação. A Standard Performance Evaluation Corporation (SPEC) tem lançado vários benchmark suites: SPEC89, SPEC92, SPEC95 e SPEC2000. Estas benchmark suites são compostas por programas reais, escolhidos para serem representativos de programas que tipicamente demandam muita CPU e pouco I/O.

1.7.4 Comparando desempenho

Uma vez selecionados os programas adequados para usar como benchmarks e decidida a métrica de avaliação, tempo de resposta ou throughput (número de tarefas executadas por unidade de tempo), é necessário decidir como comparar os dados de desempenho obtidos a partir de diferentes benchmarks.

A maneira mais simples de considerar o desempenho relativo é usar o tempo total de execução de dois programas em máquinas diferentes. Por exemplo, os tempos de execução de dois programas conforme a tabela 1.1.

Outra maneira de sumarizar os tempos é utilizando as médias aritmética, harmônica ou geométrica. A média geométrica é inadequada, pois não prediz o tempo de execução.

	Computador A	Computador B
Programa 1 (s)	1	10
Programa 2 (s)	1000	100
Total (s)	1001	110

Tabela 1.1: Tempo total de execução de 2 programas

$$\text{Média Aritmética} = \text{Tempo } (i,n) = \frac{1}{n} \quad (1.4)$$

$$\text{Média Harmônica} = \text{Taxa } (i,n) = \frac{n}{\sum_{i=1}^n \frac{1}{\text{Taxa}_i}} \quad (1.5)$$

$$\text{Média Geométrica} = \text{Tempo Normalizado } (i,n) = \sqrt[n]{\prod_{i=1}^n T_{normalizado_i}} \quad (1.6)$$

Além das médias, existem ainda outras medidas de desempenho. Uma das alternativas é a métrica MIPS (million instruction per second), que é dada por umas das seguintes expressões:

$$\begin{aligned} \text{MIPS} &= \frac{\text{Número de instruções}}{\text{Tempo de execução} \times 10^6} \\ &= \frac{\text{Freqüência de Clock}}{\text{CPI} \times 10^6} \end{aligned}$$

Existem problemas com o uso da métrica MIPS. Ela especifica a taxa de execução de instruções, mas não considera a capacidade de executar mais ou menos trabalho. Portanto, não podemos comparar máquinas com conjuntos de instruções diferentes. Outro problema é que os resultados obtidos variam entre programas no mesmo computador, o que impede que determinada máquina tenha um MIPS característico.

Outra alternativa é a métrica denominada MFLOPS (million floating-point operations per second), que é dada pela seguinte expressão:

$$\text{MFLOPS} = \frac{\text{Número de op. de ponto flutuante}}{\text{Tempo de execução} \times 10^6} \quad (1.7)$$

Uma operação de ponto flutuante pode ser uma operação de adição, subtração, multiplicação ou divisão aplicada a operandos expressos em precisão simples ou dupla.

1.7.5 Lei de Amdahl

A lei de Amdahl pode ser utilizada para demonstrar o ganho de desempenho de uma máquina. Este ganho é dito aceleração ou speedup. Entende-se por aceleração a medida de como a máquina se comporta após a implementação

de uma melhora em relação ao seu comportamento anterior. Podemos definir speedup como:

$$\text{speedup} = \frac{\text{Desempenho após a melhora}}{\text{Desempenho antes da melhora}} \quad (1.8)$$

A lei de Amdhal demonstra que é errado esperar que a melhora em um dos aspectos que influenciam no desempenho da máquina resulte numa melhora no desempenho total proporcional ao tamanho do ganho inicial da fração.

Capítulo 2

Componentes de um Computador

O computador está organizado em dois componentes que são:

- Hardware: corresponde a parte física que está dividida em: unidade de entrada e saída, processador, memória principal e memória secundária;
- Software: é o conjunto de programas que dá vida à máquina. É classificado em software aplicativo (jogos, planilha, etc.) e software básico (sistema operacional, compiladores, editores de texto, etc.).

Para interligar os componentes do hardware existe uma placa de suporte especial, chamada placa-mãe. A placa-mãe é responsável por gerenciar toda a transação entre o processador e os periféricos. Os componentes principais da placa-mãe são: chipset, BIOS, barramentos, e slots.

Chipset é o chip responsável pelo controle de diversos dispositivos de entrada e saída como o barramento, o acesso à memória, o acesso ao HD, periféricos on-board e off-board, comunicação do processador com a memória RAM e entre outros componentes da placa-mãe. Uma placa-mãe possui dois chipsets: o chipset sul e o chipset norte. O chipset sul (*south bridge*) é responsável pelo controle de dispositivos de entrada ou saída enquanto o chipset norte (*north bridge*) faz a comunicação entre o processador e a memória principal.

O BIOS (Basic Input/Output System) é o primeiro programa executado pelo computador ao ser ligado. Sua função primária é preparar a máquina para que o sistema operacional, que pode estar armazenado em diversos tipos de dispositivos (discos rígidos, disquetes, CDs, etc) possa ser executado. O BIOS é armazenado num chip ROM (Read-Only Memory) localizado na placa-mãe, chamado ROM BIOS. O BIOS também é responsável por controlar o uso dos dispositivos e manter informações de data e hora. O BIOS trabalha junto com o *post*, um software que testa os componentes do micro em busca de eventuais erros. Pode-se alterar as configurações de hardware através do *setup*.

Os barramentos permitem a interligação entre os dispositivos da placa-mãe. São divididos em três conjuntos: via de dados, via de endereços e via de controle. O desempenho do barramento pode ser medido pela sua largura de banda (32, 64 bits, etc.) e pela sua velocidade de transmissão (100 Mbps, 1G bps, etc.).

Os slots são responsáveis por ligar os periféricos aos barramentos e suas velocidades e largura de banda são correspondentes as dos seus respectivos barramentos. Na placa-mãe são encontrados vários slots para encaixe de placas (vídeo, som, rede, modem, etc.). Alguns exemplos de slots: ISA, PCI, AGP, PCI Express, etc.

2.1 Principais componentes de Hardware

2.1.1 Discos Rígidos

Os discos rígidos são dispositivos de armazenamento destinados a grandes quantidades de dados. Fisicamente, um disco rígido pode ser visto como um composto por dois grandes blocos, como na figura 2.2. O primeiro bloco é um conjunto de discos magnéticos superpostos em alturas diferentes com auxílio de um eixo central. No momento de acesso ao disco, essa estrutura é mantida em uma rotação constante. O segundo bloco é uma estrutura mecânica que suporta um conjunto de cabeçotes, um para cada superfície de disco. Essa estrutura é capaz de realizar movimentos de vai-e-vem de maneira que os cabeçotes possam ser deslocados desde a borda do disco até o centro.

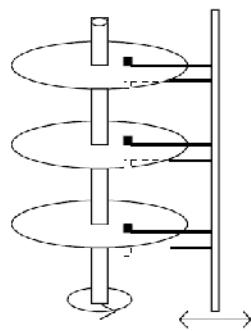


Figura 2.1: Organização física do disco

Do ponto de vista da organização lógica, cada superfície de um disco é dividida em circunferências concêntricas denominadas trilhas. Cada trilha, por sua vez, é subdividida radialmente em unidades chamadas setores. Em geral, os setores têm o mesmo tamanho. O setor possui a unidade mínima de leitura e gravação em um disco. O conjunto de todas as superfícies do disco que ficam exatamente à mesma distância do eixo central forma o cilindro, conforme a figura ???. As abstrações cilindro, trilha e setor são utilizadas para designar a organização lógica da unidade de disco. A definição de trilhas e de setores em

um disco chama-se formatação física e é um procedimento realizado pelo fabricante. A capacidade total do disco é obtida multiplicando-se cabeças x cilindros x setores x tamanho do setor.

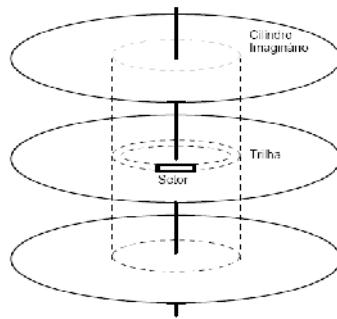


Figura 2.2: Organização lógica da unidade de disco

Para acessar dados no disco, é necessário informar à controladora o cilindro, a superfície e o setor a ser acessado. Esse método é denominado de CHS (Cylinder, Head, Sector). Outra maneira é acessar o disco é enxergá-lo como um conjunto de blocos, no qual cada bloco é um ou mais setores. O número de blocos é então convertido em cilindros, superfície e setores por um procedimento que se denomina de LBA (Linear Block Addressing).

Outros termos bastante comuns associados a disco rígidos são formatação lógica e partições. A formatação lógica consiste em gravar informações no disco de forma que arquivos possam ser escritos, lidos e localizados pelo sistema operacional. O conceito de partição está associado à capacidade de dividir logicamente um disco em vários outros discos.

Para realizar um acesso a um disco, é necessário posicionar o cabeçote de leitura e escrita sob um determinado setor e trilha onde dado será lido ou escrito. O tempo total de acesso aos disco, seja para leitura ou para escrita, é dado pela seguinte fórmula:

$$T_{acesso} = T_{seek} + T_{latencia} + T_{transferencia} \quad (2.1)$$

A descrição de cada um dos termos da soma é a seguinte:

- Tempo de Seek: tempo necessário para deslocar o cabeçote de leitura e escrita até o cilindro correspondente à trilha a ser acessada;
- Tempo de Latência: tempo necessário, uma vez o cabeçote posicionado já na trilha correta, para o setor a ser lido, ou escrito, se posicionar sob o cabeçote de leitura e escrita no início do setor a ser lido (ou escrito);
- Tempo de Transferência: corresponde ao tempo necessário à transferência dos dados, isso é, à leitura ou a escrita dos dados.

Outro fator relacionado com a redução do tempo de acesso a um disco é o entrelaçamento (interleaving). Essa técnica numera os setores não mais de forma contígua, mas sim com um espaço entre eles. O disco 2 da Ilustração 14 possui um fator de entrelaçamento igual a 2. Isso significa que entre o setor k e o setor $k+1$, existem dois outros setores. O melhor fator de entrelaçamento para uma determinada unidade de disco depende da velocidade do processador, do barramento, do controlador e da velocidade de rotação do disco.

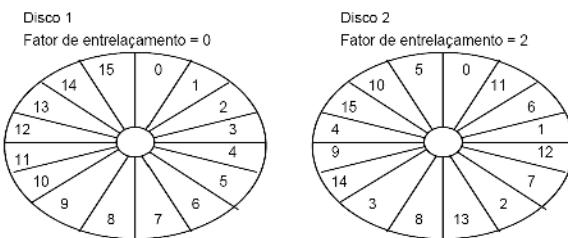


Figura 2.3: Trilha com 16 setores e diferentes fatores de entrelaçamento

2.1.2 Teclado

O teclado é o principal periférico de entrada de dados utilizado na integração direta de usuários com o computador. O princípio de operação do teclado é bastante simples: gerar um símbolo para cada tecla pressionada. Mecanicamente, um teclado pode ser visto como uma matriz de i linhas e j colunas as quais entram em contato quando uma tecla é pressionada. A cada elemento i,j da matriz correspondente um caractere.

Quando uma tecla é pressionada, o teclado identifica a linha e a coluna associadas a essa tecla e gera um código que é denominado de scan mode (código de varredura). O pressionar da tecla gera ainda uma interrupção de hardware, e por consequência, a execução de um tratador de interrupções específico para o teclado. Com base no scan mode, o tratador de interrupções consulta uma tabela interna, substituindo o scan mode pelo código ASCII correspondente à tecla pressionada. O código ASCII da tecla, em seguida, é armazenado em uma região especial da memória (buffer do teclado) de onde é recuperado por chamadas do sistema operacional.

Um teclado brasileiro difere de um teclado inglês na posição dos acentos e da cedilha, por exemplo. A solução empregada é associar a certos programas aplicativos mapas de códigos. Através desses mapas de códigos, os programas aplicativos são capazes de consumir caracteres do buffer de teclado e convertê-los de forma apropriada.

O procedimento de ler os dados do teclado e escrevê-los na tela denomina-se ecoamento. Quando se tem várias janelas abertas, os caracteres digitados devem ser direcionados à janela correta. Dois métodos são normalmente empregados: centralizado e dedicado.

No método centralizado o driver de teclado disponibiliza um conjunto de mini-buffers os quais podem ser encadeados para formar um buffer maior. Nesse caso, para cada janela aberta o sistema operacional atribui uma estrutura de dados na qual um dos seus elementos é um ponteiro utilizado para referenciar e lista encadeada de mini-buffers. No método dedicado, a bufferização é feita diretamente em uma área de memória provida pela estrutura de dados associada ao terminal. Nesse caso, o número de entradas para o terminal é limitado pelo tamanho do buffer dessa estrutura.

2.1.3 Mouse

O mouse é um periférico de entrada que historicamente se juntou ao teclado como auxiliar no processo de entrada de dados, especialmente em programas com interface gráfica. O mouse tem como função movimentar o cursor (apontador) pela tela do computador. O formato mais comum do cursor é uma seta, contudo, existem opções no sistema operacional e softwares que permitem personalizarmos o cursor do mouse.

O mouse funciona como um apontador sobre a tela do monitor, e disponibiliza, normalmente, quatro tipos de operações: movimento, click (clique), duplo click e drag and drop (arrastar e largar).

Existem modelos com um, dois, três ou mais botões cuja funcionalidade depende do ambiente de trabalho e do programa que está a ser utilizado. Claramente, o botão esquerdo é o mais utilizado.

O mouse é normalmente ligado ao computador através de portas: serial, PS2 ou, mais recentemente, USB (Universal Serial Bus). Também existem conexões sem fio, as mais antigas em infra-vermelho, as atuais em Bluetooth. Outros dispositivos de entrada competem com o mouse: Touchpads (usados basicamente em notebooks) e Trackballs.

O mouse original possuía dois discos que rolavam nos eixos X e Y e tocavam diretamente na superfície. O modelo mais conhecido de rato é provavelmente o mouse baseado em uma esfera, que roda livremente, mas que na prática gira dois discos que ficam em seu interior. O movimento dos discos pode ser detectado tanto mecanicamente quanto por meio ótico.

Os modelos mais modernos de mouse são totalmente óticos, não tendo peças móveis. De modo muito simplificado, eles tiram fotografias que são comparadas e que permitem deduzir o movimento que foi feito.

2.1.4 Placa de rede

Uma placa de rede é um dispositivo de hardware responsável pela comunicação entre os computadores em uma rede. A placa de rede é o hardware que permite aos computadores conversarem entre si através da rede. Sua função é controlar todo o envio e recebimento de dados através da rede. Cada arquitetura de rede exige um tipo específico de placa de rede, sendo as arquiteturas mais comuns as

redes Ethernet e Token Ring.

Além da arquitetura usada, as placas de rede à venda no mercado diferenciam-se também pela taxa de transmissão, cabos de rede suportados e barramento utilizado (On-Board, PCI, ISA ou Externa via USB). As placas de rede para Notebooks podem ser on-board ou por uma placa PCMCIA.

Quanto à taxa de transmissão, as placas mais comuns são Ethernet de 10, 100 ou 1000 Mbps e placas Token Ring de 4 Mbps e 16 Mbps. Usando placas Ethernet de 10 Mbps, por exemplo, devemos utilizar cabos de par trançado de categoria 3 ou 5, ou então cabos coaxiais. Usando uma placas de 100 Mbps o requisito mínimo a nível de cabeamento são cabos de par trançado nível 5. No caso de redes Token Ring, os requisitos são cabos de par trançado categoria 2 (recomendável o uso de cabos categoria 3) para placas de rede de 4 Mbps, e cabos de par trançado blindado categoria 4 para placas de 16 Mbps. Devido às exigências de uma topologia em estrela das redes Token Ring, nenhuma placa de rede Token Ring suporta o uso de cabos coaxiais.

2.1.5 Impressora

As impressoras são dispositivos de saída que tem por finalidade imprimir em papel ou filme plástico os resultados do processamento. Da mesma forma que os monitores, a imagem impressa é resultado de muitos pontos impressos individualmente que no conjunto formam o texto ou a imagem desejados. Também de forma semelhante aos monitores, as impressoras evoluíram a partir de dispositivos que imprimiam apenas caracteres em uma única cor para as modernas impressoras capazes de reproduzir imagens sofisticadas, de alta resolução gráfica, em milhares de cores. As impressoras são classificadas em:

- Impressoras alfanuméricas: Esses equipamentos recebem do computador códigos que representam caracteres alfanuméricos e portanto tem capacidade de imprimir apenas esses caracteres. Geralmente é possível usar apenas uma fonte gráfica, característica do equipamento. Algumas impressoras permitem trocar o dispositivo de impressão, viabilizando a utilização de um pequeno número de fontes gráficas;
- Impressoras gráficas: Esses equipamentos recebem do computador a informação sobre os pontos a serem impressos. Dessa forma, podem imprimir gráficos. Na impressão de textos, os caracteres são impressos como pontos, que em determinada configuração formam a imagem gráfica do caractere a ser impresso. Quando se utiliza uma impressora gráfica para imprimir texto, existe a possibilidade de utilizar um grande número de diferentes fontes gráficas, definidas por software.

2.1.6 Monitor

Capítulo 3

Aritmética Computacional

3.1 Números Com Sinal e Números Sem Sinal

Existe uma grande variedade de opções para representar os números inteiros com ou sem sinal. Apenas quatro se destacam: sinal e amplitude/magnitude ($S + M$), complemento de 1, complemento de 2 e notação em excesso (*biased*).

3.1.1 Sinal e amplitude/magnitude

Neste sistema de representação o bit mais a esquerda representa o sinal: 0 (zero) para indicar o valor positivo e 1 (um) para indicar o valor negativo. Os bits restantes representam o módulo. A amplitude ou faixa de representação para n bits é $[-(2^{n-1} - 1); 2^{n-1} - 1]$. Por exemplo, para $n=8$ a faixa de representação é $[-127; 127]$. Neste sistema o zero possui duas representações, por exemplo, $n = 4$ (0000 e 1000).

3.1.2 Complemento de 1

Neste sistema de representação o bit mais a esquerda representa o sinal: 0 (zero) para indicar o valor positivo e 1 (um) para indicar o valor negativo. Para os números positivos, os $n-1$ bits representam o módulo. O simétrico de um número positivo é obtido pelo complemento de todos os seus dígitos (trocando 0 por 1 e vice-versa) incluindo o bit de sinal. A amplitude ou faixa de representação para n bits é $[-(2^{n-1} - 1); 2^{n-1} - 1]$. Por exemplo, para $n = 8$ a faixa de representação é $[-127; 127]$. Neste sistema o zero possui duas representações, por exemplo, $n = 4$ (0000 e 1000).

3.1.3 Complemento de 2

Neste sistema de representação o bit mais a esquerda representa o sinal: 0 (zero) para indicar o valor positivo e 1 (um) para indicar o valor negativo. Para os números positivos, os bits restantes representam o módulo. A amplitude ou faixa de representação para n bits é $[-(2^n - 1); 2^n - 1]$. Por exemplo, para $n=8$ a faixa de representação é $[-128; 127]$. Este sistema possui representação assimétrica, o número de representações negativas é maior do que a positiva. Este sistema o zero possui uma única representação, por exemplo, $n=4$ (0000).

O complemento de 2 de um número é obtido em dois passos: primeiro obtém-se o complemento de todos os bits (inclusive o de sinal); e ao resultado obtido soma-se 1 (um) em binário, desprezando o último transporte, se houver.

Uma maneira prática de se obter o complemento de 2 de um número é copiar da direita para a esquerda todos os bits até encontrar o primeiro bit 1 (copiar inclusive este bit), e inverter todos os demais bits. Um exemplo para n=4:

$$\begin{array}{r}
 0110 \text{ (6 na base 10)} & \text{usando a regra:} \\
 1001 \text{ (número binário invertido)} & 0110 < - > 1001 \\
 + 0001 \text{ (soma binária com 1)} \\
 \hline
 1010 \text{ (complemento de 2)}
 \end{array}$$

Os computadores manipulam tanto número inteiros positivos quanto negativos, que são representados em complemento de 2.

3.1.4 Notação em excesso

Neste sistema de representação o bit mais à esquerda representa o sinal: 1 (um) para indicar o valor positivo e 0 (zero) para indicar o valor negativo. A amplitude ou faixa de representação para n bits é $[-(2^{n-1}); 2^{n-1} - 1]$. Por exemplo, para n=8 a faixa de representação é $[-128; 127]$. Neste sistema cada número inteiro corresponde ao valor desejado acrescido de um excesso de 2^{n-1} , onde n pode ser 4, 5, 7, 8, 16, etc.

Este sistema tem uma vantagem em relação aos outros sistemas apresentados anteriormente: o valor em binário com todos os bits a 0 (zero) representa o menor valor inteiro, que este tenha sinal ou não, e o mesmo se aplica ao maior valor em binário, i.e., com todos os bits 1: representa o maior inteiro, com ou sem sinal.

Binário (4 bits)	S + M	C 1	C 2	Excesso de 8 (n=4)
0000	+0	+0	0	-8 (0 - 8)
0001	1	1	1	-7 (1 - 8)
0010	2	2	2	-6 (2 - 8)
0011	3	3	3	-5 (3 - 8)
0100	4	4	4	-4 (4 - 8)
0101	5	5	5	-3 (5 - 8)
0110	6	6	6	-2 (6 - 8)
0111	7	7	7	-1 (7 - 8)
1000	-0	-7	-8	0 (8 - 8)
1001	-1	-6	-7	1 (9 - 8)
1010	-2	-5	-6	2 (10 - 8)
1011	-3	-4	-5	3 (11 - 8)
1100	-4	-3	-4	4 (12 - 8)
1101	-5	-2	-3	5 (13 - 8)
1110	-6	-1	-2	6 (14 - 8)
1111	-7	-0	-1	7 (15 - 8)

3.2 Adição e Subtração

Numa soma os bits são somados um a um da direita para a esquerda, com os *carries* sendo passados para o próximo bit à esquerda. A operação de subtração usa a adição. O subtraendo é simplesmente negado antes de ser somado ao minuendo. Lembre-se que a máquina trata com números representados em complemento a 2. O exemplo a seguir mostra as operações de soma ($6 + 7$) e subtração ($7 - 6$) (complemento 2) bit a bit entre dois números representados com 4 dígitos binários.

```
7 = 0111  
6 = 0110  
13 = 1101 (soma)  
1 = 0001 (subtração)
```

Tanto a soma como a subtração podem gerar *overflow* ou *underflow*, se o resultado obtido não puder ser representado pela quantidade de bits que formam uma palavra. Se somarmos ou subtrairmos dois números com sinais contrários, nunca ocorrerá *overflow* ou *underflow*. Isto porque operandos com sinais contrários nunca podem ser maior do que qualquer dos operandos.

O *overflow* ocorre quando somamos dois operandos positivos e obtemos um resultado negativo, ou vice-versa. Isto significa que utilizamos o bit de sinal, gerando um carry, para armazenar um valor pertencente ao resultado da operação. Raciocínio semelhante é realizado para detectar a ocorrência do *underflow* numa subtração. Neste caso, o bit de sinal também é usado para armazenar um valor pertencente ao resultado da operação.

Os projetistas de um sistema devem decidir onde tratar a ocorrência de *overflow* ou de *underflow* em operações aritméticas. Elas podem ser tratadas tanto por hardware quanto por software. Pode existir a detecção por hardware que gera uma exceção, e que depois é tratada por software.

3.3 Operações Lógicas

Os computadores manipulam palavras, mas é muito útil, também, manipular campos de bits dentro de uma palavra ou mesmo bits individuais. O exame de caracteres individuais (8 bits) dentro de uma palavra é um bom exemplo dessa necessidade. Assim, as arquiteturas de conjuntos de instruções incluem instruções para manipulação de bits.

Um dos tipos de instrução utilizado é a de deslocamento de bits. As instruções podem deslocar bits tanto à direita quanto à esquerda. Todos os bits são movidos para o lado determinado e os bits que ficam vazios são preenchidos com 0s. Outras instruções lógicas muito úteis são implementadas na unidade lógica e aritmética de um processador são as operações NOT, AND, OR e XOR. O exemplo abaixo mostra as operações lógicas, bit a bit, de deslocamento à direita, à esquerda, NOT, AND, OR e XOR.

3.4 Construção de uma Unidade Lógica Aritmética

A ALU (Arithmetic and Logic Control) é a parte central do hardware de uma CPU. Ela é responsável por realizar operações aritméticas e lógicas básicas, e pode ser implementada com quatro componentes: AND, OR, INVERTER e MULTIPLEXOR (3.4).

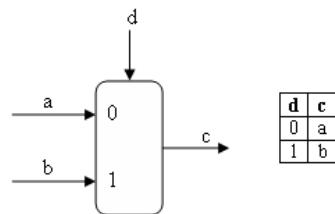


Figura 3.1: Multiplexador

Para implementar um somador de um bit, precisa-se de duas entradas para os operandos, uma saída para o resultado, uma entrada relativa ao *carry in* e um saída relativa ao *carry out*. A tabela abaixo mostra a *tabela verdade* para o somador de um bit.

Entrada			Saída	
a	b	carry in	soma	carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

As saídas soma e carry out são:

$$\text{soma} = (a.b.\text{carryin} + a.b.\text{carryin} + a.b.\text{carryin} + a.b.\text{carryin})$$

$$\text{carryout} = (b.\text{carryin} + a.\text{carryin} + a.b)$$

Para completar o projeto de uma ALU de n bits, podemos conectar n somadores de um bit. Os carry outs gerados pelos bits menos significativos são propagados por toda extensão do somador.

A operação de subtração pode ser realizada somando-se o minuendo com a negação do subtraendo. Este efeito é realizado acrescentando uma entrada complementada de b ao somador e ativando o carry in do bit menos significativo para um. O somador então calcula $a + b + 1$.

Outras funções de interesse, com o XOR podem ser incluídas na ALU. Algumas operações não são, em geral, implementadas na ALU (3.4). Exemplos: shift, multiplicação, divisão e operações de ponto flutuante.

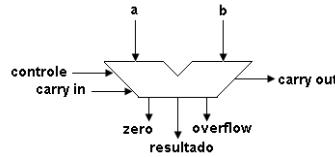


Figura 3.2: ALU

3.5 Ponto Flutuante

Para somar dois números na base 10 em notação científica temos que:

- Alinhar os pontos;
- Somar os números;
- Normalizar o resultado;
- Arredondar o resultado para o número máximo de casas permitidas.

Um hardware para somar números de ponto flutuante tem que fazer as mesmas operações com números binários. Vejamos um exemplo:

Exemplo: Somar os números 0.5 e -0.4375 em binário e apresente com 4 bits de precisão; trunque o resultado na fase de arredondamento.

$$0.5 = 0.1 \text{ (binário)} = 0.1 \times 2^0 = 1.000 \times 2^{-1}$$

$$-0.4375 = -0.0111 \text{ (binário)} = -0.0111 \times 2^0 = -1.110 \times 2^{-2}$$

1. Igualar os expoentes (modificar o número menor)
 $-1.110 \times 2^{-2} = -0.111 \times 2^{-1}$
2. Somar as partes significantes (mantissas)
 $1.000 \times 2^{-1} + (-0.111 \times 2^{-1}) = 1.000 \times 2^{-1} + (1.001 \times 2^{-1}) = 0.001 \times 2^{-1}$
3. Normalizar o resultado
 $0.001 \times 2^{-1} = 1.0 \times 2^{-4}$
4. Arredondamento
O número final está OK.

Capítulo 4

Sistemas Operacionais

4.1 Introdução

O sistema operacional é uma camada de software colocada entre o hardware e os programas que executam tarefas para os usuários. O sistema operacional é responsável pelo acesso aos periféricos, ou seja, sempre que um programa necessita de algum tipo de operação de entrada e saída, ele solicita ao sistema operacional.

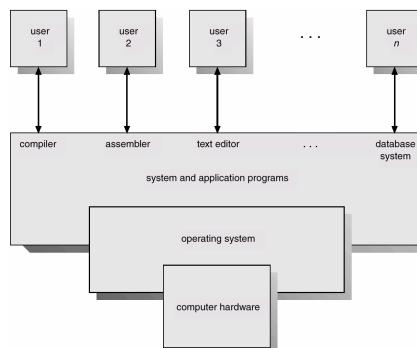


Figura 4.1: Sistema Operacional

O objetivo do sistema operacional é tornar a utilização do computador mais eficiente e mais conveniente. Uma utilização mais eficiente do computador é obtida por meio da distribuição de seus recursos (memória principal, tempo de processador, impressora, espaço em disco, etc.) entre os programas. Uma utilização mais conveniente é obtida escondendo-se do programador detalhes do hardware, em especial os periféricos. Para atingir esses objetivos, o sistema operacional oferece uma série de serviços como: gerência de memória, gerência do processador, memória virtual, sistema de arquivos, sistema de entrada e saída.

A arquitetura de um sistema operacional corresponde à imagem que o usuário tem do sistema, a forma como ele percebe o sistema. Esta imagem é definida

pela interface por meio da qual o usuário acessa os serviços do sistema operacional. Essa interface é formada pelas chamadas de sistema e pelos programas de sistema.

Os programas solicitam serviços ao sistema operacional por meio das chamadas de sistema que transferem a execução dos programas para o sistema operacional. A parte do sistema operacional responsável para implementar as chamadas de sistema é normalmente chamadas de núcleo ou kernel. Os principais componentes do kernel de qualquer sistema operacional são a gerência de processador, gerência de memória, sistema de arquivos e gerência de entrada e saída.

Os programas de sistemas, algumas vezes chamados de utilitários, são programas normais executados fora do kernel do sistema operacional. Eles implementam tarefas básicas para utilização do sistema e muitas vezes são confundidos com próprio sistema operacional. Exemplos de programas de sistemas são os utilitários para manipulação de arquivos, listar arquivos, imprimir arquivos, trocar nome de arquivos, data, hora e etc. O programa de sistema mais importante é o interpretador de comando. Sua tarefa é receber comandos e executá-los. Esses comandos podem ser enviados de forma textual ou por meio de uma interface gráfica de usuário (GUI).

O sistema operacional não resolve problemas do usuário final. Somente quando ocorre algum evento especial que o sistema operacional é ativado. Dois tipos de eventos ativam o sistema operacional: uma chamada de sistema ou uma interrupção de periférico.

Os sistemas operacionais podem ser classificados segundo inúmeros critérios, dentre os quais os mais comuns são:

- Número de usuários
 - Monousuários: projetados para suportar um único usuário. Exemplos desse tipo de sistema são o MS DOS, Windows 3x, Windows 9x, etc;
 - Multiusuários: projetados para suportar várias sessões de usuários. Exemplos desse sistema são Windows NT, UNIX, etc;
- Número de tarefas
 - Monotarefa: capazes de executar apenas uma tarefa de cada vez. Por exemplo, o MS DOS, etc;
 - Multitarefa: capazes de executar várias tarefas simultaneamente, como uma compilação e um processamento de texto. Por exemplo, Windows, Unix, etc;
- Tipo de serviço oferecido ao usuário
 - Sistemas de processamento em lote (batch);
 - Sistemas de tempo compartilhado (time sharing);
 - Sistemas de tempo real (real time);

- Sistemas Mainframes;
- Sistemas desktop;
- Sistemas distribuídos;
- Sistemas handheld;
- Sistemas paralelos.

4.2 Conceitos Básicos

4.2.1 Multiprogramação

A multiprogramação torna mais eficiente o aproveitamento dos recursos do computador. Isso é conseguido por meio da execução simultânea de vários programas. Em um sistema multiprogramado diversos programas são mantidos na memória ao mesmo tempo.

A idéia da multiprogramação é aproveitar o tempo ocioso do processador durante as operações de entrada e saída, ou seja, enquanto o periférico executa o comando enviado, o sistema operacional inicia a execução de outro programa. Isso faz com que exista uma maximização do uso do processador e da memória. Em um ambiente monoprogramado, o processador ficaria parado durante a realização do acesso a um periférico.

4.2.2 Processo

Um processo pode ser definido como um programa em execução. O processo é considerado um elemento ativo, pois altera o seu estado à medida que executa um programa. É o processo que realiza as chamadas de sistema.

Em muitos sistemas operacionais os processos são criados por outros processos, por meio de chamada de sistema. Nesse caso, é possível definir uma hierarquia de processos. O processo que faz a chamada de sistema é chamado de processo pai e o processo criado é chamado de processo filho. Um mesmo processo pai pode estar associado a vários processos filhos. Os processos filhos, por sua vez, podem criar outros processos.

Durante a sua execução, um processo passa por diversos estados, refletindo o seu comportamento dinâmico, isto é, sua evolução no tempo. Os possíveis estados para um processo são: criação (new), apto (ready), executando (running), bloqueado (blocked) e terminado (exit). A figura 4.2 mostra esses cinco estados e as transições entre eles.

No estado New um novo processo é criado. Após ser criado, o processo entra em um ciclo de processador. Ele precisa do processador para executar, mas o processador poderá estar ocupado com outro processo, ele deverá esperar. Diversos processos podem estar no mesmo estado. Nesse caso, é necessário manter uma fila com os processos aptos a ganhar o processador. Essa fila é chamada de fila de aptos (ready queue). No estado New, o sistema operacional

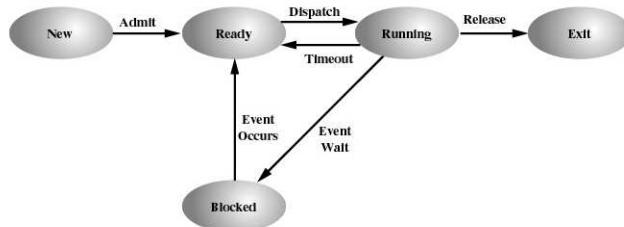


Figura 4.2: Ciclo de vida dos processos

aloca recursos para o processo, mas não existe uma garantia de que o processo será executado.

Os processos na fila do processador estão no estado Ready. Um único processo ocupa o processador a cada instante. O processo que ocupa o processador está no estado Running. Neste estado processo pode realizar as chamadas de sistema. Enquanto o sistema espera pelo término da chamada de sistema, o processo está no estado Blocked.

O processo fica no estado Blocked até ser atendido. Com isso, o processador fica livre. O sistema operacional seleciona outro do processo da fila de aptos para receber o processador. O estado Exit indica que o processo terminou sua execução ou foi abortado.

A mudança de estado de qualquer processo é iniciada por um evento. Esse evento aciona o sistema operacional, que então altera o estado de um ou mais processos. O evento pode ser uma chamada de sistema ou uma interrupção de hardware.

Alguns outros caminhos são possíveis no grafo de estado. Por exemplo, pode ocorrer de nenhum processo na memória principal está no estado Ready, pois todos estão aguardando uma operação de entrada e saída. Nesse caso, o sistema operacional realiza o swap (mover todo ou parte de um processo da memória para o disco) de um dos processos bloqueados para disco, e coloca numa fila de processo suspensos. O estado para essa situação é o Suspend. Quando a operação de entrada e saída de um dos processos é finalizada, o sistema operacional trás do disco o processo da fila de suspenso colocando no estado de Ready.

4.2.3 Interrupções

O mecanismo de interrupção constitui a base de operação de um sistema multiprogramação. O mecanismo de interrupção permite que o controlador de periférico chame a atenção do processador. A função básica do controlador de periférico é conectar o dispositivo em questão ao processador.

Uma interrupção sempre sinaliza a ocorrência de algum evento. Quando ela acontece, desvia a execução da posição atual de programa para uma rotina

específica. Essa rotina, responsável por atender a interrupção é chamada de tratador de interrupção. O tratador realiza as ações necessárias em função da ocorrência da interrupção.

Em computador podem existir diversos controladores capazes de gerar interrupções. A forma mais simples de identificar a origem da interrupção é associar a cada controlador um tipo diferente de interrupção. Por exemplo, cada tipo de interrupção é identificado por um número.

Existem momentos que um programa não pode ser interrompido enquanto realiza uma tarefa crítica. Para isso, o processador possui instruções para habilitar e desabilitar as interrupções. Enquanto as interrupções estiverem desabilitadas, elas serão ignoradas pelo processador. Elas não são perdidas, apenas ficam pendentes. Quando o programa tornar a habilitar as interrupções, elas serão imediatamente atendidas pelo processador.

Interrupções de software, também chamadas de *traps*, são causadas pela execução de uma instrução específica para isso. O efeito é semelhante a uma chamada de sub-rotina, pois o próprio programa interrompido é quem gera a interrupção. O maior uso para interrupções de software é a implementação de chamadas de sistemas, por meio das quais os programas solicitam serviços ao sistema operacional. Não é possível desabilitar as interrupções de software, mesmo porque não é necessário.

Existe uma terceira classe de interrupções geradas pelo próprio processador. São as interrupções por erro, muitas vezes chamadas de interrupções de exceção. Elas acontecem quando o processador detecta algum tipo de erro na execução do programa. Por exemplo, uma divisão por zero, um acesso a posição de memória inválido, etc.

4.2.4 Threads

Uma thread nada mais é que um fluxo de execução. Na maior parte das vezes, cada processo é formado por um conjunto de recursos mais uma única thread.

A idéia do multithreading é associar vários fluxos de execução (várias threads) a um único processo. Em determinadas aplicações, é conveniente disparar várias threads dentro de um mesmo processo (programação concorrente). É importante notar que as threads existem no interior de um processo, compartilhando entre elas os recursos de processo, como espaço de endereçamento, código e dados. Devido a essa característica, a gerência de threads (criação, destruição, troca de contexto, sincronização) é mais leve quando comparada com processos. O chaveamento entre duas threads de um mesmo processo é muito mais rápido que o chaveamento entre dois processos. Em função disso, threads são muitas vezes chamadas de processos leves.

4.3 Escalonamento de Processos

Os mecanismos de gerenciamento de processador visam permitir que vários processos compartilham o processador de forma aumentar a Utilização, aumentar o Throughput, diminuir o Tempo de Resposta e o Tempo Total de Execução. Estas são as principais métricas de desempenho de um sistema operacional no que diz respeito à gerencia de processador.

Para alcançar bons resultados, os sistemas operacionais empregam várias políticas de escalonamento para determinar qual processo tomará posse do processador em um determinado instante. Essas políticas são os chamados Algoritmos de Escalonamento. Exemplo de algoritmos de escalonamento são: *First In First Out* (FIFO), Prioridades, *Round-Robin*, *Shortest Job First* (SJF), Múltiplas Filas com Realimentação, entre outras.

A parte do sistema operacional responsável por escolher qual o processo tomará posse do processador em um determinado instante é o *Scheduller*, enquanto o responsável por entregar o processador de fato a um processo é o *Dispatcher*.

É o *Dispatcher* quem realiza o chaveamento de contexto, que consiste em salvar o estado dos registradores do processo que deixará o processador e carregar os registradores para o novo processo.

O algoritmo FIFO é o mais simples e de implementação mais fácil. No entanto, não é dos mais eficiente no quesito tempo de resposta. Caso um processo muito grande tome posse do processador, outros processos podem ter que esperar um tempo muito longo até que possam executar.

O algoritmo *Shortest Job First* dá posse do processador ao processo que gastará menos tempo executando. O tempo de resposta médio é otimizado em relação ao FIFO, no entanto, essa política não pode ser implementada de forma perfeita uma vez que não é possível determinar quanto tempo um processo gastar á no processador na próxima vez em que tomar posse. Implementações do SJF estimam esse tempo utilizando informações passadas do processo.

Os algorítimos *Round-Robin* consistem em dividir o tempo de processamento em fatias de tempo chamadas *time slots*. Cada processo na fila de prontos tem direito de executar durante um período de tempo fixo antes de perder a posse do processador. Um problema neste algoritmo é a necessidade de determinar a fatia de tempo ideal, de forma que a impressão de paralelismo na execução não seja perdida. Uma fatia de tempo muito pequena,por exemplo, pode fazer com que o tempo gasto com chaveamento de contexto diminua a performance.

Filas de Prioridades são a base das políticas de escalonamento nos sistemas operacionais modernos. A cada processo é associada uma prioridade. O processo de maior prioridade é quem toma a posse do processador no momento oportuno. Uma variante dos algoritmos de prioridades pura, são os os algoritmos de Prioridade com Preempção. Neste esquema, um processo da fila de prontos que tenha prioridade maior que o processo em execução pode tomar

posse do processador antes do processo em execução terminar de executar.

Na verdade, nos sistemas operacionais, não só um algoritmo de escalonamento é utilizado na gerência de processador. Usualmente, esses algoritmos são combinados de forma melhorar o desempenho do sistema. Algoritmos de múltiplas filas com realimentação são exemplos de combinação de várias políticas. Esses algoritmos permitem que seja dado um tratamento adequado à um determinado processo de acordo com o seu comportamento.

Em gerência de processador existem também os conceitos de execução em *Background* e *Foreground*. Processos em *background* são geralmente aqueles que estão rodando com uma prioridade muito baixa, necessitam de pouco input e geram também um mínimo de output. Processos em *foreground* funcionam da forma oposta.

O escalonador responsável por determinar qual processo receberá o direito de executar é chamado Escalonador de Curto Prazo. Existem também os escalonadores de Longo e Médio Prazo. O escalonador de Médio Prazo é parte por exemplo do processo *swapper* e está intimamente ligado a gerência de memória, enquanto o escalonador de longo prazo determina quando um processo novo é de fato admitido no sistema para disputa dos recursos. É o escalonador de longo prazo quem determina o grau de multiprogramação do sistema e também é conhecido como *Job Scheduler*.

4.4 Entrada e Saída

Uma das atribuições dos sistemas operacionais é realizar a gerência de periféricos, também conhecida como gerência de entrada e saída. Os periféricos são dispositivos que permitem que o computador se comunique com o mundo externo. A primeira preocupação dos sistemas operacionais no que diz respeito a gerência de E/S é a forma de comunicação que será utilizada, que pode ser serial ou paralela. As três formas básicas de se implementar o controle de periféricos são:

- E/S programada: O controle dos estados da operação de E/S é feito através de loops de status. A responsabilidade é do programador; A técnica utilizar o estado da operação de E/S utilizada E/S programada é conhecida como polling;
- Interrupções: Os periféricos chamam a atenção do processador através de um sinal de hardware. Cabe ao processador identificar, priorizar e mascarar as interrupções geradas pelos periféricos;
- Acesso Direto à Memória (DMA): Em situações em que o volume de dados é muito grande, utiliza-se esta técnica para permitir que periféricos tenham acesso direto a memória sem a necessidade da intermediação por parte do processador.

O localização dos periféricos do ponto de vista da arquitetura do sistema pode ser feita de basicamente de duas maneiras: Mapeamento em Memória e

Espaço de E/S. Os *drivers* de dispositivos consistem em uma camada superior ao hardware e têm por objetivo esconder as diferenças entre dispositivos de mesmo tipo.

Existe também a necessidade de se empregar técnicas de escalonamento de E/S de forma otimizar o atendimento das requisições feitas aos periféricos. Nos discos magnéticos, por exemplo, são utilizados algoritmos de escalonamento como:

- FCFS: *First Come First Served*. Atende as requisições na ordem de chegada;
- SSTF: *Shortest Seek Time First*. Atende primeiro as requisições que necessitam de menor tempo de seek (seek time é o tempo necessário para mover o cabeçote para a trilha adequada);
- SLTF: *Shortest Latency Time First*. Atende primeiro as requisições de menor latência (latência é o tempo necessário para localizar um setor dentro de uma trilha do disco. Diretamente relacionado com a velocidade de rotação do disco.);
- Scan: Varre o disco na direção radial atendendo requisições. Só atende requisições em um sentido;
- CScan: Similar ao Scan, porém atende requisições na subida e na descida.

Além de políticas de escalonamento de E/S, também são utilizadas técnicas de *Buffer* e *Cache* para aumentar o desempenho. A técnica empregada para realizar a alocação e liberação de recursos é conhecida como *Spooling*. A gerência de periféricos também se responsabiliza pelo controle de acesso aos periféricos e tratamentos de erros.

4.4.1 Camadas do subsistema de Entrada e Saída

O objetivo do subsistema de entrada e saída é padronizar ao máximo as rotinas de acesso aos periféricos de forma a reduzir o número de rotinas de entrada e saída. Para isso, o subsistema de entrada e saída é organizado em uma estrutura de quatro camadas: hardware dos dispositivos de entrada e saída, os drivers, a E/S independente de dispositivo e E/S nível de usuário. A figura 4.3 mostra essas camadas.

A camada inferior de software (drivers) é composta por um conjunto de módulos de software implementados para fornecer os mecanismos de acesso a um dispositivo de entrada e saída específico. A camada de software de E/S independente do dispositivo implementa procedimentos e funções gerais a todos os dispositivos de entrada e saída como: escalonamento de E/S, denominação, bufferização, cache de dados, alocação e liberação, direitos de acesso e tratamentos de erro. A E/S nível de usuário é uma interface de programação associada às bibliotecas de entrada e saída, ou aplicativos de desenvolvimento. É importante notar que as bibliotecas de entrada e saída não fazem parte do sistema operacional.

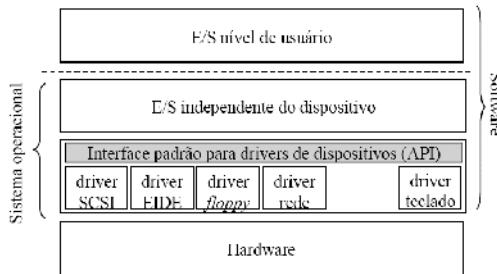


Figura 4.3: Camadas do Subsistema de E/S

4.5 Gerência de Memória

Umas das funções fundamentais de um sistema operacional moderno é realizar a gerencia da memória principal do computador de forma permitir que os diversos processos executem de forma eficiente e protegida. Quando o assunto é gerência de memória, dois conceitos básicos são os de memória lógica e memória física. Os programas fazem referência à endereços lógicos que no momento da execução são traduzidos em um endereço real chamado endereço físico. As primeiras técnicas de tradução de endereços eram baseadas em registradores de base e limites.

Nos sistemas multiprogramados, é necessário também implementar técnicas para que os diversos programas possam utilizar a memória ao mesmo tempo. Inicialmente, a memória era dividida em partições de tamanho fixo. Dois problemas decorrentes desta técnica são conhecidos como Fragmentação Interna e a Fragmentação Externa. A fragmentação interna ocorre quando um programa aloca uma partição de memória que excede a quantidade necessária. O espaço excedente naquela partição é desperdiçado. A fragmentação externa ocorre quando apesar da quantidade total de memória ser suficiente, não existe uma partição contígua capaz de atender as necessidades de um programa.

Para solucionar o problema da fragmentação interna, foi criado o mecanismo de particionamento dinâmico, no qual um programa aloca somente a quantidade exata de memória. No entanto, esse método aumenta a fragmentação externa uma vez que permite o aparecimento de lacunas pequenas demais para serem utilizadas por algum programa. Neste método de particionamento são utilizadas várias técnicas de preenchimento de lacunas. Exemplos são: *First-Fit*, *Best-Fit*, *Worst-Fit* e *Circular-Fit*. Para evitar o aparecimento de lacunas muito pequenas, foi criada uma técnica chamada Parágrafo, que consiste em determinar a menor unidade de alocação de memória.

Todas as técnicas apresentadas até aqui levam em consideração o fato de que os programas devem ocupar áreas contíguas de memória. Os sistemas operacionais modernos supriram esta necessidade através da técnica de Paginação.

Aqui aparecem os conceitos de páginas lógicas e páginas físicas, semelhantes aos de endereços lógicos e físicos. Neste contexto, o endereço lógico é formado

por duas partes que são o número da página mais o deslocamento dentro dela. Existe também a necessidade de traduzir uma página lógica em uma página física. A Tabela de Páginas é a responsável por essa tarefa. Existem várias formas de implementá-la.

A primeira preocupação é onde armazenar a tabela de páginas. Em sistemas com tabelas muito pequenas, as tabelas de páginas podem ser armazenadas em registradores, no entanto, em geral a tabela de páginas é armazenada na própria memória principal utilizando registradores (PTBR e PTBL) para indicar a posição da tabela na memória. Um problema inerente a esta técnica é a necessidade de se acessar a memória duas vezes quando se deseja ler um dado. Um acesso a tabela de páginas e outra ao dado em si. Para minimizar este problema é utilizado um mecanismo chamado *Translation LookAside Buffer* (TLB). O TLB consiste em uma memória de rápido acesso que armazena partes da tabela de páginas. Quando a tradução de uma pagina lógica em uma página física é possível utilizando apenas o TLB, é dito que ocorreu um HIT, caso contrário dizemos que ocorreu um MISS.

A técnica de Segmentação é utilizada para implementar a proteção de endereços de memória utilizados por um processo. Usualmente isso é feito através de registradores de limite. Quando um processo tenta acessar uma região de memória protegida ocorre uma falha de segmentação (*Segmentation Fault*). Esta técnica não deve ser confundida com a técnica de segmentação de memória presente em algumas arquiteturas, onde a memória é dividida em partes como Segmento de Dados, Segmento de Código, Pilha etc.

O *Swapping* é uma outra técnica utilizada para gerenciamento de memória. Nesta técnica, um processo é suspenso e todas suas páginas de memória são descarregadas para o disco (swap-out), liberando a memória para que algum outro processo possa executar. Quando processo suspenso pode voltar para memória, as páginas do processo são novamente carregadas para a memória (swap-in). O *swapper* é responsável pelo swap-in e swap-out.

Na técnica de paginação pura, o processo não precisa mais ocupar um trecho contíguo de memória, no entanto, ainda é necessário que todas as páginas de um processo estejam carregadas na memória no momento da execução. Uma evolução do esquema de paginação é a técnica de Memória Virtual. Memória Virtual consiste em um esquema de paginação sob demanda, no qual somente as páginas necessárias para a execução de um processo são carregadas para a memória.

Além de aumentar o desempenho do sistema, esta técnica permite que existam programas com espaço de endereçamento lógico maiores. Para indicar se uma página se encontra ou não carregada na memória em um determinado instante, é utilizado um bit adicional para cada entrada da tabela. Quando um processo tenta acessar uma página que não está carregada na memória é dito que ocorreu um *page-fault*. O processo é então suspenso e até que esteja pronto para executar novamente ocorre a seguinte sequência de eventos:

1. Alocação de uma página física;
2. Localização da Página Física no Disco;
3. Leitura da Página no Disco;
4. Atualização da Tabela de Páginas;
5. Processo vai para fila de pronto.

O responsável por carregar a página solicitada é o *pager*. Para carregar uma nova página lógica para memória muitas vezes é necessário descarregar uma página para o disco. Os algoritmos de substituição de páginas são os responsáveis por decidir qual página será escolhida para deixar a memória. Exemplos de algoritmos de substituição de página são:

- FCFS: Escolhe a página que está a mais tempo na memória;
- Ótimo: Escolhe a página que vai ser acessada mais remotamente no futuro (não é implementável);
- LRU: Escolhe a que a mais tempo não é acessada (obs: algoritmo implementado por histórico de bits);
- *Second Chance*: Organiza páginas em forma de um fila circular e utiliza bits de controle.

O *Trashing* é a situação em que um processo possui poucas páginas físicas e o tempo gasto em *page-faults* é muito alto, predominando no tempo total de processamento. Uma forma de solucionar este problema é utilizar a técnica de swap para permitir que o processo possa executar de forma satisfatória.

4.6 Sistemas de Arquivos

O sistema de arquivos é a parte do sistema operacional mais visível para os usuários. Durante todo tempo, os usuários manipulam arquivos contendo textos, planilhas, desenhos, figuras, jogos e etc. Este fato exige que o sistema operacional apresente uma interface coerente e simples. Para isso, o sistema de arquivos implementa o conceito de arquivo e diretório.

4.6.1 Conceitos básicos sobre arquivos

Um arquivo é um recipiente no qual dados são armazenados. Em geral, cada arquivo contém um conjunto de dados que possui algum significado prático para o usuário ou para o sistema. Um arquivo pode conter um programa executável, um módulo de um programa fonte, um texto, uma figura, etc. Cada arquivo é identificado por um nome, o qual permite que o usuário faça referências a ele. Além do nome, cada arquivo possui uma série de outros atributos que são mantidos pelo sistema operacional como tipo de conteúdo, tamanho, data e hora do último acesso, data e hora da última alteração, lista de usuários que podem

acessar o arquivo, etc.

Em geral, o sistema operacional suporta diversas operações sobre os arquivos, como criação e destruição do arquivo, leitura e alteração do conteúdo, troca de nome do arquivo, etc. Essas operações correspondem a chamadas de sistema que os programas de usuário podem usar para manipular os arquivos.

Em sistemas multiusuários, é importante controlar o acesso aos arquivos. Sistemas operacionais multiusuários implementam mecanismos que permitem controlar quais os usuários podem fazer o que em quais arquivos.

O controle de acesso inicia com a identificação dos usuários por meio de um código de usuário e uma senha. A partir do momento que a identificação do usuário é aceita, todos os processos disparados a partir do terminal em questão passam a ter os direitos de acesso associados com aquele usuário. É possível associar a cada arquivo uma lista de usuários e direitos de acesso. A forma usual é permitir que apenas o usuário que criou o arquivo possa alterar a lista contendo os direitos de acesso do arquivo.

A forma como os dados são dispostos dentro de um arquivo determina sua estrutura interna. Cada tipo de arquivo possui uma estrutura interna apropriada para a sua finalidade. Por exemplo, arquivos de texto são organizados em linha ou parágrafos. Arquivos que contêm programas executáveis são organizados em termos de segmento de código e segmentos de dados.

Em geral, os sistemas operacionais ignoram a estrutura interna dos arquivos. Para o sistema operacional, cada arquivo corresponde a uma seqüência de bytes, cujo significado é conhecido pelo usuário que o criou. A única exceção são os arquivos que contêm programas executáveis. Nesse caso, a estrutura interna é definida pelo próprio sistema operacional. Como o conceito de tipo de arquivo é útil para os usuários, muitos sistemas operacionais suportam nomes de arquivos onde o tipo é indicado. A forma usual é acrescentar extensão de nome que identifique o tipo de arquivo em questão.

O método de acesso diz respeito à forma como o conteúdo de um arquivo é acessado. O método de acesso mais simples é o seqüencial. Este método é usado pelos compiladores, para a impressão de um arquivo, etc. Outro método de acesso é o acesso relativo. Neste método, o programa inclui na chamada de sistema qual posição do arquivo a ser lida. As posições do arquivo são numeradas a partir de 0 (ou a partir de 1 em alguns sistemas), sendo que cada posição corresponde a um byte.

Em muitos sistemas operacionais, existe o conceito de posição corrente no arquivo. Nesse caso, a chamada de sistema para leitura ou escrita não informa uma posição. Essa sempre acontece a partir da posição corrente. O sistema operacional também permite que o programa altere a posição corrente do arquivo por meio de uma chamada de sistema.

Existem outros métodos de acesso mais sofisticados, tais como seqüencial indexado, indexado, direto, etc. Tais métodos de acesso são implementados a

partir dos métodos seqüencial e relativo.

4.6.2 Implementação de arquivos

A forma básica de implementar arquivos é criar, para cada arquivo no sistema, um descritor de arquivo. O descritor de arquivo é um registro no qual são mantidas as informações a respeito do arquivo. Essas informações incluem os seus atributos, além de outros dados que não são visíveis aos usuários, mas são necessários para o que o sistema operacional implemente as operações sobre os arquivos.

Um descritor de arquivo contém as seguintes informações: nome do arquivo, extensão do nome do arquivo, tamanho em byte, data e hora do último acesso, data e hora da última alteração, identificação do usuário que criou o arquivo, local no disco onde o conteúdo do arquivo foi alocado, etc. A forma usual é manter o descritor de um arquivo na mesma partição onde está o seu conteúdo. Dessa forma, esse disco poderá ser até mesmo fisicamente removido de um computador e conectado a outro. Os arquivos nele poderão ser acessados normalmente no novo computador.

O descritor é acessado a cada operação de escrita ou leitura para determinar a localização no disco dos dados a serem escritos ou lidos. Para tornar mais rápido o acesso aos arquivos, o sistema de arquivos mantém na memória uma tabela contendo todos os descritores dos arquivos em uso. Quando um arquivo entra em uso, o seu descritor é copiado do disco para a memória. Quando o arquivo deixa de ser usado, o seu descritor em memória pode ter sido alterado em relação à cópia do descritor em disco. Nesse caso, o sistema de arquivos escreve o descritor atualizado que está na memória sobre a cópia original do descritor em disco. A maioria dos sistemas operacionais exige que os próprios programas informem quando um arquivo entra em uso e quando ele não é mais necessário. Para tanto, existem as chamadas de sistema open e close. Também é útil passar como parâmetro o tipo de acesso que será feito, isto é, leitura (READONLY ou RD) ou leitura e escrita (READWRITE ou RW).

O sistema de arquivos utiliza uma Tabela de Descritores de Arquivos Abertos (TDAA) para manter em memória os descritores dos arquivos abertos. A TDAA mantém informações relativas aos arquivos abertos por todos os processos no sistema. Isso é necessário porque normalmente é permitido que vários processos abram um mesmo arquivo simultaneamente. Cada entrada armazena uma cópia do descritor do arquivo mantido em disco, assim como algumas informações adicionais, necessárias apenas enquanto o arquivo está aberto. Por exemplo, número de processos utilizando o arquivo no momento.

As entradas da TDAA armazenam informações que não variam conforme o processo que está acessando o arquivo. Entretanto, existem informações diretamente associadas com o processo que acessa o arquivo. Essas informações não podem ser mantidas na TDAA, pois como vários processos podem acessar o mesmo arquivo, elas possuirão um valor diferente para cada processo. Um exemplo de informação que depende do processo é a posição corrente no arquivo.

Uma solução típica é criar, para cada processo, uma Tabela de Arquivos Abertos por Processo (TAAP). Cada entrada ocupada na TAAP corresponde a um arquivo aberto pelo processo correspondente. No mínimo, a TAAP contém em cada entrada as seguintes informações: posição corrente no arquivo, tipo de acesso (leitura ou leitura e escrita) e apontador para a entrada correspondente na TDAA. A figura 4.4 mostra as tabelas TDAA e TAAP. Toda TDAA como as TAAP devem ficar na memória do sistema operacional, fora do acesso dos processos de usuário.

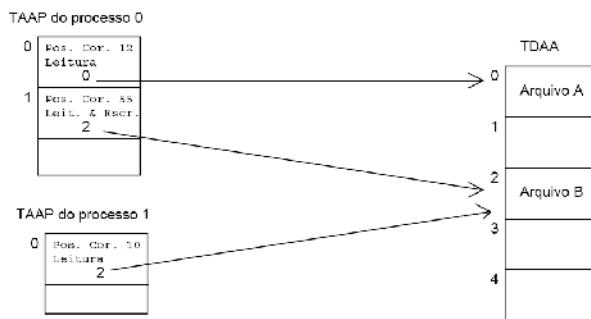


Figura 4.4: TAAP vs. TDAA

Uma vez aberto um arquivo, o processo utiliza chamadas de sistema com read e write para acessar o seu conteúdo. Não é necessário, nem conveniente, que a cada chamada de sistema, o processo forneça novamente o nome do arquivo. Como resultado de um open com sucesso, o sistema de arquivos retorna para o processo o número da entrada na TAAP associada com o arquivo aberto. Dessa forma, nas chamadas de sistemas após o open, o processo indica o arquivo através do número de sua correspondente entrada na TAAP. A partir da TAAP, o sistema de arquivos pode imediatamente localizar o descritor no arquivo TDAA. Muitos sistemas operacionais chamam esse número de *handle* do arquivo.

Existem duas funções importantes que o sistema de arquivos deve realizar na implementação das chamadas de sistema read e write. São elas a montagem e desmontagem de blocos lógicos e a localização dos blocos lógicos no disco. Essas funções são implementadas baseando-se em três formas básicas de alocação de arquivos: alocação com áreas contíguas, alocação encadeada e a alocação indexada.

4.6.3 Cache de Sistema de Arquivos

Uma importante estrutura de dados presente na implementação de um sistema de arquivos é a sua cache. A cache não oferece nenhuma funcionalidade nova, isto é, a presença ou ausência de uma cache não adiciona ou elimina nenhuma

função, chamada de sistema ou operação sobre arquivos. Entretanto, caches representam um grande aumento no desempenho de qualquer sistema de arquivos, pois o uso do disco tende a ser intenso em sistemas operacionais de propósito gerais. O objetivo do cache é manter na memória principal uma certa quantidade de blocos do disco. Dessa forma, se algum bloco for requisitado para leitura ou escrita, ele será encontrado na memória principal, evitando o acesso ao disco.

A cache do sistema de arquivos utiliza uma área da memória principal e é controlada pelo sistema operacional. Na verdade, existem diversos locais onde uma cache de disco pode ser mantida. É possível haver uma cache global, uma cache exclusiva para cada sistema de arquivos, etc.

A forma básica de operação é bem simples. Toda vez que um bloco de disco é necessário para leitura e/ou escrita, a cache é pesquisada. Se o bloco estiver na cache, essa cópia é usada. Se o bloco não estiver na cache, ele é lido do disco, colocado na cache e então utilizado.

Uma questão importante é quando atualizar o disco após um bloco presente na cache ter sido alterado. Do ponto de vista de desempenho, o ideal é postergar a atualização do disco ao máximo no sentido de minimizar as escritas em disco. Por outro lado, caso ocorra uma pane do sistema, toda a informação na cache será perdida, o disco ficará desatualizado e, possivelmente, o sistema de arquivos ficará corrompido. Existem diversas políticas que podem ser utilizadas.

Uma cache de sistema de arquivo pode possuir milhares de blocos, o que torna inviável uma pesquisa seqüencial da mesma para localizar determinado bloco. Dada a natureza dinâmica dessa estrutura e sua importância para o desempenho do sistema como um todo, uma tabela hash é utilizada. O sistema de arquivos fornece o número da partição e o número do bloco, e uma função hash é utilizada para determinar o endereço do bloco na cache, caso ele esteja na cache.

Eventualmente, a cache pode se encontrar completamente ocupada, sendo necessário liberar espaço para algum outro bloco. A solução típica é escolher um bloco da cache, atualizar o seu conteúdo no disco se necessário, declarar esse bloco da cache como livre e utilizá-lo para hospedar um novo bloco de disco. A política para escolher o bloco da cache a ser liberado geralmente é a LRU (*Least Recently Used*).

4.6.4 Gerenciamento do espaço livre

Uma das tarefas do sistema de arquivos é gerenciar o espaço livre nos discos. Em outras palavras, determinar quais setores do disco estão livres e podem ser alocados para aumentar o tamanho de um arquivo. Uma forma simples de gerenciar o espaço livre em disco é por meio de um mapa de bits. A figura 4.5 mostra esse mecanismo.

Cada bit presente no mapa representa um bloco físico do disco. Os bits são considerados numerados da direita para esquerda, isto é, o bit menos significativo do primeiro byte é o bit número zero. Bit ligado indica bloco ocupado,

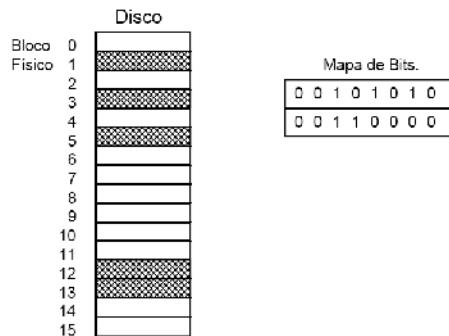


Figura 4.5: Mapa de bits para gerenciamento de espaço livre

e bit desligado indica bloco livre. O endereço do bloco representado por um determinado bit é definido pela própria posição do bit dentro do mapa.

Para alocar mais um bloco físico para um arquivo, basta percorrer o mapa de bits até encontrar um bit zero. O bit é então ligado para indicar que o respectivo bloco agora está ocupado. Quando um arquivo é destruído e seus blocos liberados, basta desligar os bits correspondentes. O mapa de bits deve ser mantido na memória principal para que a busca seja rápida.

O espaço livre em disco também pode ser gerenciado por meio de uma lista contendo os números de todos os blocos físicos livres (Lista de Blocos Livres). Como essa lista é grande no caso de um disco quase vazio, ela deve ser mantida no próprio disco. Para acelerar o processo de alocação e liberação de blocos físicos, alguns blocos da lista são copiados para a memória principal. Logo, sómente será necessário acessar o disco quando todos os endereços de blocos livres copiados para a memória principal tiverem sido alocados. Ou ainda, quando remoções de arquivos liberarem blocos físicos em tal quantidade que alguns blocos de endereços tenham que ser escritos em disco. Alguns sistemas operacionais atualizam periodicamente a lista de endereços em disco para minimizar a corrupção do sistema de arquivos em caso de falha no computador.

4.6.5 Diretórios

Os diretórios são as estruturas do sistema de arquivos que contêm a informação *quais arquivos existem no disco*. Um diretório pode ser entendido como sendo um conjunto de arquivos ou um conjunto de referências a arquivos. Existem diversas formas de estruturar os diretórios de um sistema. A mais simples é ter um único diretório para o disco inteiro. Nesse caso, o diretório corresponde a uma lista de todos os (possivelmente milhares) arquivos do disco. Essa solução, conhecida como diretório linear, é aceitável apenas para sistemas de arquivo muito pequenos. Por exemplo, pode ser utilizada para discos flexíveis de pequena capacidade.

Para sistemas multiusuários, o diretório linear é problemático, pois os arquivos de diferentes usuários ficam misturados. Esse problema pode ser resolvido com uma estrutura de diretórios organizada em dois níveis. O diretório principal contém uma entrada para cada usuário do sistema. Essa entrada não corresponde a um arquivo, mas sim a um subdiretório que, por sua vez, contém os arquivos do usuário correspondente. Também é necessário criar no diretório principal uma entrada para conter os arquivos do sistema. As entradas do diretório principal são usualmente chamadas de subdiretórios.

É possível estender o conceito de subdiretórios de tal forma que os usuários também possam criar livremente os seus próprios subdiretórios. Dessa forma, cada usuário tem a liberdade de organizar os seus arquivos de forma lhe for mais conveniente. O resultado é um sistema de diretórios organizado na forma de árvore conforme a figura 4.6.

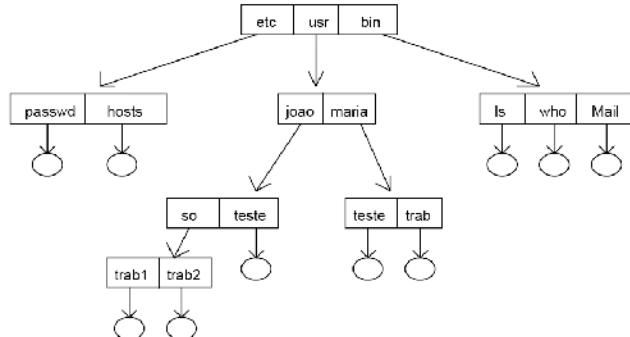


Figura 4.6: Diretório organizado em forma de árvore

Em um sistema de diretórios organizado na forma de árvore, qualquer arquivo ou subdiretório pode ser identificado de forma não ambígua por meio do caminho (pathname) para atingi-lo a partir da raiz da árvore. Facilmente, a árvore de diretório cresce até uma altura tal que passa a ser desconfortável fornecer sempre o caminho completo até cada arquivo ou diretório. O conceito de diretório corrente facilita a identificação de arquivos nesse contexto. Dessa forma, um arquivo pode ser identificado por seu caminho absoluto, que inicia a raiz da árvore, ou pelo seu caminho relativo, que inicia no diretório corrente do usuário em questão.

Uma flexibilidade adicional presente em muitos sistemas operacionais está na possibilidade de incluir o mesmo arquivo ou subdiretório em vários diretórios. Dessa forma, o mesmo arquivo passa a ter diversos nomes absolutos (mesmo na árvore, cada arquivo possui diversos nomes relativos, uma vez que o caminho relativo depende do diretório corrente em questão). Essa facilidade é denominada de link e efetivamente transforma a estrutura de diretórios em um grafo.

4.6.6 Implementação de diretórios

A forma mais simples de implementar diretórios é considerá-lo como arquivos especiais, cujo conteúdo é manipulado pelo próprio sistema operacional. Dessa forma, todo o mecanismo de alocação, liberação e localização de blocos físicos no disco, disponível para arquivos, também é usado para os diretórios. Diretórios passam a ser arquivos cujo conteúdo é definido pelo sistema operacional e cujo acesso é controlado por parte do usuário.

Como diretórios são implementados como arquivos, cada diretório possui também seu descritor. Os diretórios são implementados como conjunto de descritores de arquivos, ou conjuntos de endereços de descritores de arquivos. Quando diretórios são implementados como conjunto de descritores de arquivos, o conteúdo de um diretório corresponde aos descritores dos arquivos e dos subdiretórios contidos naquele diretório, conforme a figura 4.7. Nesse caso, o nome do arquivo ou subdiretório faz parte do seu descritor.

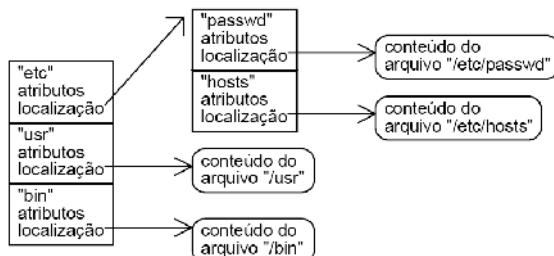


Figura 4.7: Diretórios contendo descritores de arquivos

Outra possibilidade é separar um conjunto de blocos da partição para armazenar exclusivamente os descritores de arquivos e de subdiretórios. Esse conjunto de blocos forma um vetor de descritores, no qual cada descritor pode ser identificado pelo número da partição e pela posição nesse vetor. Essa estrutura de dados forma o que é normalmente conhecido como um flat file system, pois os descritores não incluem nomes, não existe nenhuma estruturação dos arquivos em diretórios, apenas um diretório único (vetor) e arquivos identificados pela posição do vetor.

Em qualquer solução, cada diretório nada mais é que uma tabela. Existem diversas implementações possíveis para tabelas que podem ser usadas na implementação de um diretório. Entre elas, destaca-se a lista não ordenada, lista ordenada e a tabela hash.

4.7 Sistemas Operacionais Distribuídos

Um sistema distribuído é uma coleção de computadores independentes que parecem ao usuário como um único computador. Essa definição implica hardware

formado por máquinas autônomas e software fornecendo a abstração de uma máquina única. As principais vantagens são:

- Econômicas: aproveitar máquinas potencialmente ociosas; mais barato vários processadores interconectados do que um supercomputador.
- Distribuição inerente: algumas aplicações são distribuídas por natureza.
- Tolerância a falhas: em caso de falha de uma máquina, o sistema como um todo pode sobreviver, apresentando apenas uma degradação de desempenho.
- Crescimento incremental: o poder computacional pode ser aumentado através da inclusão de novos equipamentos.
- Flexibilidade: sistemas distribuídos são mais flexíveis do que máquinas isoladas, por isso muitas vezes são utilizados até mesmo que não se esteja buscando desempenho. É essa flexibilidade que permite que vários usuários compartilhem dados e periféricos.

E as desvantagens:

- Pouco software de alto nível disponível para sistemas distribuídos.
- Dificuldades para evitar acesso indevido (segurança).
- A rede de interconexão pode causar problemas ou não dar vazão a demanda.

Sistemas distribuídos consistem de várias CPUs interconectadas. No entanto, há várias formas diferentes no qual esse hardware pode estar organizado. Dentre as várias classificações existentes, Flynn propõe uma taxonomia considerando o número de fluxo de instruções e o número de fluxo de dados.

- SISD(Single Instruction Single Data): fluxo de instruções e dados único é a característica dos uniprocessadores tradicionais
- MIMD(Multiple Instructions Multiple Data): caracteriza-se por vários processadores interconectados. Tanembaum apresenta a seguinte subclassificação, onde os dois primeiros são definidos em relação a organização da memória e os dois últimos em relação a forma de interconexão:
 - Multiprocessador: máquinas MIMD com memória compartilhada (um único espaço de endereçamento virtual compartilhado por todas as CPUs).
 - Multicomputador: máquinas que não possuem memória compartilhada, isto é, cada processador possui sua memória privada.
 - Barramento: um único cabo, rede, barramento ou outro meio que conecte todas as máquinas. Analogia: TV a cabo.
 - Switch: existem cabos individuais conectando máquina a máquina, com vários padrões possíveis.

Outra classificação: Fortemente acoplado(Tightly Coupled): comunicação rápida entre os processadores (grande número de bits por segundo). Fracamente acoplado(Loosely Coupled): atraso para troca de mensagem entre máquinas é alto.

Com a criação de novas arquiteturas de computadores, surgiram novas demandas de software e, em especial, novas funções exigidas ao S.O. Pode-se considerar como uma boa classificação da evolução dos Sistemas Operacionais a tabela abaixo. A tabela 4.1 apresenta uma comparação entre as características dos S.O modernos

Geração	Sistema	Característica	Objetivo
1 ^a	SO Centralizado	Gerenciamento de Processos Gerenciamento de Memória Gerenciamento de E/S Gerenciamento de Arquivos	Gerenciamento de recursos Memória estendida <i>Virtualidade</i>
2 ^a	SO de Rede	Acesso Remoto Troca de Informações Navegação na rede	Compartilhamento de recursos <i>Interoperabilidade</i>
3 ^a	SO Distribuído	Visão global do Sistema de arquivos, Espaço de nomes Tempo, Segurança Poder Computacional	Visão de computador Único em sistema de Múltiplos Computadores Transparência
4 ^a	SO Cooperativo Autônomo	Aplicações Distribuídas Abertas e Cooperativas	Trabalho cooperativo <i>Autonomia</i>

Tabela 4.1: Sistemas Operacionais

4.7.1 Estruturação de Sistemas Distribuídos

Estruturação baseada na distribuição física

Dentre os vários modelos baseados da distribuição física, encontram-se o modelo hierárquico, o do cache de CPU, o usuário servidor e o modelo de conjunto de processadores.

No modelo hierárquico, os computadores são dispostos em uma rede sob a forma de árvore, de maneira que quanto mais próximos estiverem da raiz, mais potentes deverão ser. O computador da raiz tratará, de forma geral, do sistema como um todo, enquanto que os computadores distantes da raiz tratarão de tarefas específicas e especializadas.

O modelo de cache de CPU consiste na utilização de computadores de menor porte como elementos que interligam terminais com uma grande CPU. Nesse caso, uma parte do processamento será executada no computador ligado ao terminal, e a outra parte no computador central.

O modelo usuário-servidor(cliente/servidor) surgiu na medida em que os computadores pequenos, crescendo em potência e tendo seus preços reduzidos, diminuíram gradativamente a importância do computador central do modelo cache de CPU.

Estruturação Lógica

Modelo de processos:

Esses processos podem encapsular tanto elementos ativos com natureza, consistindo dos programas referentes às atividades do sistema e do usuário quanto elementos naturalmente passivos correspondendo aos recursos e suas respectivas operações, confinados em gerenciadores de recursos. A comunicação entre os processos, implementada pelo sistema operacional, pode ocorrer de duas formas:

- *Chamada remota de procedimento:* este tipo de comunicação entre processos é bastante dependente da linguagem usada para implementação do sistema, devendo satisfazer suas restrições com relação a chamada de procedimento e a passagem de parâmetros.
- *Troca explícita de mensagem:* este já é mais flexível do que o anterior, suas restrições estão relacionadas com a existência de ligações implícitas ou explícitas entre os processos e com a interpretação da mensagem.

Modelo de objetos:

O modelo de objetos baseia-se no encapsulamento das várias partes de um sistema em elementos denominados objetos que são estruturados de forma a apresentarem um conjunto de operações responsáveis pelo seu comportamento. Para conseguir acessar um objeto, um processo deve ter capacidade de fazê-lo, usando o conhecimento do nome do objeto e a posse da autorização para cessar algumas ou todas as suas operações. Cada objeto distribuído não opera sozinho. A princípio ele é construído para trabalhar com outros objetos e, para isso, precisa de uma espécie de “barramento”. Tais “barramentos” fornecem infra-estrutura para os objetos, adicionando novos serviços que podem ser herdados durante a construção do objeto, ou mesmo em tempo de execução para alcançar altos níveis de colaboração com outros objetos independentes. CORBA é um exemplo de “barramento”.

Capítulo 5

Principais Processadores de Mercado

5.1 Processadores Intel

5.1.1 Família Pentium

Pentium 4

O processador Pentium 4 da Intel foi lançado em novembro de 2000, usando a microarquitetura x86 de sétima geração da Intel, chamada *Netburst*, que veio com um pipeline muito longo com a intenção de permitir clocks mais altos. Os processadores Pentium 4 podem encontrados em três versões de núcleos: *Willamette*, *Northwood* e *Prescott*.

Os primeiros modelos de Pentium 4 utilizavam soquete 423, que, como o próprio nome já sugere, possuía 423 terminais. Depois foram lançados modelos de Pentium 4 com soquete 478, que, apesar de possuírem mais contatos do que os modelos anteriores (soquete 423), eram fisicamente menores. Os modelos de Pentium 4 atuais utilizam um novo tipo de soquete, chamado Soquete 775.

Willamette

Os primeiros modelos de Pentium 4 eram baseados no núcleo *Willamette*, que tinha 8 KB de cache L1 para dados; 12 KB de cache L1 para instrução; 256 KB de cache L2; trabalhava externamente a 400 MHz (100 MHz transferindo quatro dados por pulso de clock); padrão de pinagem soquete 423 e 478; clock interno de 1,30 a 2 GHz; suporte a instruções MMX (oferece um modelo de execução SIMD (*Single Instruction Multiple Data*, ou seja, fluxo único de instruções e múltiplos de dados) simples, capaz de efetuar processamentos de dados inteiros, empacotados em registros de 64 bits. As instruções MMX melhoraram a execução das assim chamadas tarefas multimídias, como codificar e decodificar vídeo), SSE (*Streaming SIMD Extensions*) e SSE2; tecnologia de construção de 180 nanômetro.

Northwood

Em seguida vieram os modelos de Pentium 4 baseados no núcleo *North-*

wood. Este núcleo é cerca de 60% menor do que o núcleo *Willamette* devido ao seu processo de fabricação de 130 nanômetros. O núcleo *Northwood* possui 8 KB de cache L1 para dados; 12 KB de cache L1 para instrução; 512 KB ou 2 MB de cache L2; barramento externo rodando a 400 MHz, 533 MHz ou 800 MHz (100 MHz, 133 MHz e 200 MHz transferindo quatro dados por pulso de clock, respectivamente); clock interno de 1,60 a 3,4 GHz; suporte a instruções MMX, SSE e SSE2. Alguns modelos possuem suporte a tecnologia *Hyper-Threading*.

Prescott

O núcleo *Prescott* é construído com tecnologia de 90 nanômetros é utilizado nos processadores Pentium 4 modernos. Ele pode ser encontrado com 16 KB de cache L1 para dados; 12 KB de cache L1 para instrução; 512 KB, 1 MB ou 2 MB de cache L2; trabalha externamente a 533 MHz ou 800 MHz (133 MHz e 200 MHz transferindo quatro dados por pulso de clock, respectivamente); com clock interno de 2,26 a 3,80 GHz; suporte às novas instruções MMX, SSE, SSE2 e SSE3. Alguns modelos possuem suporte a tecnologia *Hyper-Threading*, XD, EM64T, SpeedStep (permite que o sistema operacional ajuste o clock do processador, diminuindo-o ao executar aplicativos que exigem menos poder e economizando energia) e a VT (*Virtualization Technology*), originalmente conhecida como *Vanderpool*. A tecnologia VT permite que um processador funcione como se fosse vários processadores trabalhando em paralelo de modo a permitir que vários sistemas operacionais sejam executados ao mesmo tempo em uma mesma máquina. Embora o clock de um *Prescott* seja o mesmo de um *Northwood*, alguns softwares de teste mostraram que um *Northwood* é ligeiramente mais veloz que um *Prescott*.

Prescott 2M

O Pentium 4 Extreme Edition foi lançado em novembro de 2003 e foi o primeiro processador para desktop a possuir o cache L3 integrado, característica esta presente apenas em processadores voltados para o mercado corporativo.

Este processador possui 2 MB de cache L3 sendo acessado na mesma freqüência de operação interna do processador. Os primeiros modelos de Pentium 4 Extreme Edition eram baseados no núcleo *Gallatin*, que tinha 8 KB de cache L1 para dados; 12 KB de cache L1 para instrução; 512 KB de cache L2 e 2 MB de cache L3; trabalhava externamente a 800 MHz e 1066MHz (200 MHz ou 266 MHz transferindo quatro dados por pulso de clock, respectivamente); clock interno de 3,20 a 3,46 GHz; suporte às instruções MMX, SSE e SSE2; tecnologia *Hyper-Threading*; tecnologia de construção de 130 nanômetros.

Os modelos de Pentium 4 Extreme Edition atuais são baseados no núcleo *Prescott 2M* com tecnologia de 90 nanômetros. Possuem 16 KB de cache L1 para dados; 12 KB de cache L1 para instrução; 2 MB de cache L2; não possuem cache L3; trabalhava externamente a 1066 MHz (266 MHz transferindo quatro dados por pulso de clock); clock interno de 3,73 GHz; suporte às instruções MMX, SSE, SSE2 e SSE3; tecnologia *Hyper-Threading*, EM64T.

Pentium D e Pentium Extreme Edition

O processador Pentium D é a versão de dois núcleos do Pentium 4, e o Pentium Extreme Edition é a versão do Pentium D com tecnologia *HyperThreading* habilitada. Os processadores Pentium D e Pentium Extreme Edition podem ser encontrados em duas versões de núcleos: *Smithfield* e *Presler*.

O Pentium D e o Pentium Extreme Edition são baseados na microarquitetura x86 de sétima geração da Intel, chamada *Netburst*, ou seja, apesar do nome diferente, eles são internamente um Pentium 4 (ou melhor, dois processadores Pentium 4 em um único encapsulamento). A diferença básica entre o Pentium D e o Pentium Extreme Edition é a ausência da tecnologia *HyperThreading* nos processadores Pentium D.

Smithfield

Os processadores Pentium D e Pentium Extreme Edition da série 800 são baseados no núcleo *Smithfield*. O núcleo *Smithfield* consiste na verdade em duas pastilhas de silício do núcleo *Prescott* montadas em um único processador.

As principais características dos processadores Pentium D da série 800 são: tecnologia de núcleo duplo; 16 KB de cache L1 de dados por núcleo; 12 KB de cache L1 de instrução por núcleo; 2 MB de cache L2 (1 MB por núcleo); barramento externo de 800 MHz (200 MHz transferindo quatro dados por pulso de clock), 533 MHz no caso do Pentium D 805 (133 MHz transferindo quatro dados por pulso de clock); clock interno de 2,66 a 3,20 GHZ para o Pentium D e 3,20 GHz para o Extreme Edition; suporte às instruções SSE3; soquete 775; processo de fabricação de 90 nm; tecnologia EM64T, XD e SpeedStep (modelos 840 e 830); e tecnologia *Hyper-Threading* nos processadores Pentium Extreme Edition. Os processadores Pentium D não têm esta tecnologia.

Presler

Os processadores Pentium D e Pentium Extreme Edition da série 900 são baseados no núcleo *Presler*.

As principais características dos processadores Pentium D e Pentium Extreme Edition da série 900 são: tecnologia de núcleo duplo; 16 KB de cache L1 de dados por núcleo; 12 KB de cache L1 de instrução por núcleo; 4 MB de cache L2 (2 MB por núcleo); barramento externo de 800 MHz (200 MHz transferindo quatro dados por pulso de clock) nos processadores Pentium D ou 1.066 MHz (266 MHz transferindo quatro dados por pulso de clock) nos processadores Pentium Extreme Edition; clock interno de 2,80 a 3,60 GHZ para o Pentium D e 3,46 a 3,73 GHz para o Extreme Edition; suporte às instruções SSE3; soquete 775; processo de fabricação de 65 nm; tecnologia EM64T, XD, VT e SpeedStep (modelos 840 e 830); e tecnologia *Hyper-Threading* nos processadores Pentium Extreme Edition.

Os processadores Pentium D não têm esta tecnologia.

Pentium M

O Pentium M é o processador da Intel voltado para o mercado de notebooks e utilizado pela plataforma Centrino. A plataforma Centrino da Intel é um con-

junto de tecnologias desenvolvidas para notebooks e é formada por três componentes:

- Processador Pentium M
- Intel Chipsets 855 ou 915
- Rede Wireless Intel/PRO

Um notebook só pode ser considerado Centrino se ele possuir todos esses três componentes. O processador Pentium M da Intel foi lançado em março de 2003, usando a microarquitetura x86 de sexta geração da Intel, ou seja, a mesma arquitetura usada pelos processadores Pentium Pro, Pentium II e Pentium III. Podem ser encontrados em duas versões de núcleos: *Banias* e *Dothan*.

Banias

Os primeiros modelos de Pentium M eram baseados no núcleo *Banias*, que tinha 32 KB de cache L1 de instruções e 32 KB de cache L1 de dados; 1 MB de cache L2; trabalhava externamente a 400 MHz (100 MHz transferindo quatro dados por pulso de clock); clock interno de 1,10 a 1,50GHz; suporte as instruções SSE2; tecnologia Enhanced SpeedStep; tecnologia de construção de 0,13 μ m; padrão de pinagem soquete 478 e 479.

Dothan

O núcleo *Dothan* é construído com tecnologia de 90 nanômetros é utilizado nos processadores Pentium M modernos. Ele possui 32 KB de cache L1 de instruções e 32 KB de cache L1 de dados; 2 MB de cache L2; trabalha externamente a 400 MHz ou 533 MHz (100 MHz e 133 MHz transferindo quatro dados por pulso de clock, respectivamente); clock interno de 1,10 a 2,26 GHz; suporte as instruções SSE2; tecnologia Enhanced SpeedStep, Execute Disable (alguns modelos); padrão de pinagem soquete 478 e 479.

5.1.2 Família Celeron

O nome Celeron é utilizado pela Intel para designar sua linha de processadores de baixo custo. Na verdade, o Celeron é uma versão econômica dos processadores topo de linha da Intel. Ou seja, o Celeron é uma versão ?capada? do Pentium II, Pentium III ou do Pentium 4, com algumas características reduzidas ou removidas. O Celeron diferencia-se do Pentium II, Pentium III ou do Pentium 4 em basicamente três aspectos:

- Tamanho do cache L2
- Clock interno
- Clock do barramento externo

Essas diferenças fazem com que o Celeron seja mais barato e tenha um desempenho menor do que os processadores Pentium II, Pentium III e Pentium 4, sendo, portanto, destinado para o mercado de usuários domésticos ou para aqueles que não necessitam de grande poder computacional.

Existem vários modelos de Celeron, mas atualmente o comercializado pela Intel é o Celeron D. O sufixo D é apenas para diferenciá-lo das gerações anteriores.

O Celeron D é a versão topo de linha dos processadores Celeron. Esse processador é baseado no Pentium 4 com núcleo *Prescott* e possui tecnologia de 65 e 90 nanômetros.

O Celeron D possui 16 KB de cache L1 de dados, 256 e 512 KB de cache L2, trabalha externamente a 533 MHz (133 MHz transferindo quatro dados por pulso de clock), suporte ?s instruções multimídia SSE3, encapsulamento FC-PGA e FC-LGA, padrão de pinagem soquete 478 ou 775, e pode ser encontrado com clocks de 2,13 GHz a 3,60 GHz. Por ser uma versão ?capada? do Pentium 4 *Prescott*, que permite alcançar com maior facilidade freqüências mais elevadas, o Celeron D não suporta a tecnologia *Hyper-Threading*, que permite simular em um único processador físico dois processadores lógicos, e não possui núcleo duplo.

Alguns processadores Celeron D possuem a tecnologia EM64T (*Extended Memory 64-bit Technology*), que permite ao processador acessar mais memória RAM, e a tecnologia XD (*eXecute Disable*), que impede que determinados tipos de vírus ataquem o micro.

Antes do Celeron D, vieram alguns outros modelos: Celeron SEPP (*Convington*), Celeron A (*Medocino*), Celeron PPGA (*Mendocino*), Celeron *Coppermine*, Celeron *Tualatin*, Celeron *Willamette*, Celeron *Northwood* e o Celeron M.

5.1.3 Família Core

Core Duo

O Core Duo (conhecido anteriormente pelo nome-código *Yonah*) é o primeiro processador da Intel voltado para o mercado de notebooks a ter tecnologia de dois núcleos, isto é, dentro dele há dois processadores completos. Curiosamente este é também o primeiro processador da Intel adotado pela Apple.

Na realidade este processador é um Pentium M com dois núcleos de processamento e construído com tecnologia de 65 nm (O Pentium M é atualmente construído com tecnologia de 90 nm).

Apesar de ter dois núcleos de processamento dentro de um único processador, o tamanho do núcleo do Core Duo é praticamente o mesmo do Pentium M (núcleo *Dothan*). Isto significa que o custo para a Intel produzir um Core Duo é quase o mesmo para produzir um Pentium M, que tem apenas um único núcleo.

O cache de memória L2 do Core Duo é de 2 MB compartilhado entre os núcleos (a Intel chama esta implementação de cache L2 compartilhado de "Smart Cache", ou "cache inteligente"). No Pentium D 840, por exemplo, que é um processador de núcleo duplo, o tamanho do seu cache L2 é de 2 MB, sendo 1 MB destinado para cada núcleo. Ou seja, no Pentium D existem dois cache

L2 de 1 MB, um por núcleo. Já no Core Duo, existe apenas um cache L2 de 2 MB que é compartilhado entre os dois núcleos.

Com o cache compartilhado, a quantidade de memória cache que cada núcleo utiliza não é fixa. Com um cache L2 de 2 MB, em um dado momento um núcleo pode estar usando 1,5 MB de cache e o outro 512 KB (0,5 MB), por exemplo. Se em um processador de núcleo duplo com cache separado o cache L2 de um núcleo "acabe" (isto é, seu 1 MB está sendo totalmente usado), ele precisa ir à lenta memória RAM buscar os dados, diminuindo o desempenho do sistema. No caso do cache compartilhado, cada núcleo pode simplesmente "redimensionar" o seu cache L2.

Outra vantagem do cache L2 compartilhado é que se um núcleo buscou um dado ou uma instrução e a armazenou no cache L2, esta mesma informação pode ser aproveitada pelo outro núcleo. Em processadores de núcleo duplo com memórias cache separadas o segundo núcleo teria de acessar este dado (ou instrução) através do barramento local do processador, isto é, "pelo lado de fora" do processador, usando o clock do barramento local, que é muito inferior ao clock interno do processador, diminuindo o desempenho do sistema.

As principais características do Core Duo são as seguintes: tecnologia de núcleo duplo; Nome-código: *Yonah*; 32 KB de cache L1 de instruções e 32 KB de cache L1 de dados 2 MB de cache L2 compartilhado entre os dois núcleos; Soquete 478 ou 479; tecnologia de 65 nm; barramento externo de 667 MHz (166 MHz transferindo quatro dados por pulso de clock); tecnologia de Virtualização; tecnologia Execute Disable; tecnologia Enhanced SpeedStep; suporte às instruções SSE3.

Podemos dividir a plataforma Centrino em três famílias. A primeira família é formada pelo processador Pentium M, chipset Intel 855/915 Express e rede sem fio Intel PRO/Wireless. A segunda família é formada pelo processador Intel Core Solo (versão do processador Intel Core Duo, mas com um único núcleo de processamento ? até o momento somente um modelo de Core Solo foi lançado, T1300, rodando internamente a 1,66 GHz, externamente a 667 MHz, 2 MB de cache L2), chipset Intel 945 Express e rede sem fio Intel PRO/Wireless 3945ABG. Já a terceira família, também conhecida como Centrino Duo (antes chamada Napa), traz para os notebooks o poder computacional dos processadores de dois núcleos e é formada pelo processador Intel Core Duo com clock interno de 1,50 a 2,16 GHz, chipset 945 Express e rede Intel PRO/Wireless 3945ABG.

Core 2 Duo

O Core 2 é a geração mais recente de processadores lançada pela Intel (os primeiros modelos foram lançados oficialmente em 27 de julho de 2006). A chegada do Core 2 significa a substituição da marca Pentium, que estava sendo usada pela companhia desde 1993. Os modelos mais comuns e conhecidos dessa linha se chamam Core 2 Duo (com núcleo duplo), que substitui o Pentium 4 e o Pentium D, mas existe também um modelo Core 2 Quad (com núcleo quádruplo) e os modelos Core 2 Extreme ("high end"), que substitui o Pentium Extreme

Edition.

O Core 2 Duo é o primeiro processador para desktop a usar a nova microarquitetura Core. O lançamento do processador Core 2 Duo (nome-código *Conroe* para desktop e *Meron* para portáteis) marca o início de uma nova geração de processadores baseados na nova microarquitetura Core, e declara de uma vez por todas o fim da microarquitetura *Netburst* usada desde 2000 pelos processadores Intel de 7ª geração. Como a microarquitetura Core é baseada na arquitetura do Pentium M e do Pentium III, podemos dizer que o Core 2 Duo é um processador Intel de 6ª geração.

A diferença entre o Core 2 Duo e o Core 2 Extreme é que este último trabalha com clocks mais elevados e tem o multiplicador de clock destravado, o que permite fazer overclock alterando o multiplicador de clock do processador.

Cuidado para não confundir o processador Core 2 Duo com o Core Duo. O Core Duo (conhecido anteriormente pelo nome-código *Yonah*) é o nome comercial para um Pentium M com dois núcleos de processamento construído com tecnologia de 65 nm. Já o Core 2 Duo é o nome comercial para o processador de nome-código *Merom* (para notebooks) ou *Conroe* (para desktops), que utiliza a nova microarquitetura Core da Intel.

As principais características técnicas dos processadores da família Core 2 (Core 2 Duo e Core 2 Extreme) são as seguintes: arquitetura Core; 64 KB de cache L1 (32 KB de dados + 32 KB de instruções) por núcleo; tecnologia de dois núcleos (o Core 2 Extreme QX6700 tem tecnologia de quatro núcleos); tecnologia de 65 nm; soquete 775; barramento externo de 800 MHz (200 MHz transferindo quatro dados por pulso de clock) ou 1.066 MHz (266 MHz transferindo quatro dados por pulso de clock); 2 MB, 4 MB ou 8 MB (Extreme QX6700) de cache de memória L2 compartilhado; tecnologia de Virtualização (exceto o Core 2 Duo E4300); tecnologia Intel EM64T; instruções SSE3 e SSSE3 (atualização da SSE3); Execute Disable; Intelligent Power Capability; tecnologia Enhanced SpeedStep.

O modelos Core 2 Duo possuem clock interno de 1,80 a 2,66 GHz, o Core 2 Quad possuiu clock interno de 2,40 GHz e o Core 2 Extreme 2,66 e 2,93 GHz.

Existem os modelos Core 2 Duo para notebook com as características mencionadas anteriormente, mas com um barramento externo de 667 ou 533 MHz.

5.1.4 Xeon

Em 1998 a Intel estabeleceu uma distinção entre seus processadores voltados para o mercado de servidores e estações de trabalho dos voltados para o mercado de usuários domésticos. Desde então, a Intel passou a incluir o termo "Xeon" (pronuncia-se "zión") no nome dos processadores voltados para o mercado de servidores e estações de trabalho. Esses processadores reconhecem mais memória RAM, permitem trabalhar em ambiente multiprocessado (isto é, com placas-mãe com vários processadores instalados sobre ela) e possui um desem-

penho maior que os processadores voltados para o mercado doméstico.

A Intel lançou versões para o mercado de servidores e estações de trabalho dos seus processadores Pentium II e Pentium III, chamadas, respectivamente, de Pentium II Xeon e Pentium III Xeon. Assim, o processador Pentium II era direcionado para o mercado de usuários domésticos enquanto que o Pentium II Xeon era um processador voltado para o mercado de servidores e estações de trabalho. A mesma coisa acontece com o Pentium III e Pentium III Xeon. No caso do Pentium 4, em vez do nome escolhido ter sido Pentium 4 Xeon, optou-se pelo nome Xeon. Ou seja, o Xeon é um processador voltado para o mercado de servidores e estações de trabalho baseado no Pentium 4. Atualmente, existem os modelos Xeon Core 2 Duo e Core 2 Quad.

Pentium 4 Xeon

Este processador deveria se chamar Pentium 4 Xeon, mas a Intel optou pelo nome Xeon. Como comentamos anteriormente, o Xeon é um processador voltado para o mercado de servidores e estações de trabalho baseado no Pentium 4, sendo, portanto, um processador Intel de 7^a geração. Como vimos, os processadores anteriores da série Xeon usavam a arquitetura Intel de 6^a geração (a mesma do Pentium Pro).

A diferença entre os processadores Xeon MP e Xeon é que o primeiro permite multiprocessamento simétrico com quatro ou mais processadores, enquanto que o Xeon permite multiprocessamento com no máximo dois processadores. Antigamente, o processador Xeon era chamado "Xeon DP" (multiprocessamento simétrico com até dois processadores), sendo posteriormente renomeado para apenas "Xeon".

Os processadores Xeon possuem 8 KB de memória cache L1 para dados (16 KB nos modelos que possuem a tecnologia de 64 bits EM64T) e um cache L1 de execução de 150 KB. O cache L2 pode ser de 512 KB, 1 MB ou 2 MB, sendo que alguns modelos possuem um cache L3, que pode ser de 1 MB, 2 MB, 4 MB ou 8 MB. O barramento externo pode ser de 667, 1066 ou 1333 MHz. Alguns modelos possuem suporte às instruções SSE3, e às tecnologias XD, EM64T e *Hyper-Threading*.

Os modelos Xeon DP 53xx possuem suporte a tecnologia Quad-Core e não suportam a tecnologia *Hyper-Threading*.

Xeon MP

Como já explicamos, a diferença entre o Xeon MP e o Xeon é a quantidade de processadores suportados no modo de multiprocessamento simétrico: o Xeon suporta até dois processadores em uma mesma placa-mãe, enquanto que o Xeon MP suporta até quatro processadores por barramento.

Na realidade é possível construir servidores com mais de quatro processadores Xeon MP em uma mesma placa-mãe. Para isso, no entanto, os processadores devem ser agrupados de quatro em quatro ? já que eles só suportam até quatro processadores por barramento ? devendo haver uma conexão entre os chipsets. Por exemplo, em um servidor com oito processadores Xeon MP, os quatro primeiros processadores estarão interligados através do mesmo barramento, enquanto os outros quatro processadores estarão interligados através de um segundo barramento. A comunicação entre os barramentos locais será feita pelo chipset.

As principais características dos processadores Xeon MP são: soquete 603; cache L1 de execução de 150 KB; cache L1 de dados de 8 KB ou de 16 KB nos modelos com suporte à tecnologia EM64T; multiprocessamento simétrico diretamente com até quatro processadores; tecnologia *Hyper-Threading*. Alguns modelos possuem suporte às instruções SSE3, e às tecnologias XD, EM64T.

Xeon Núcleo Duplo (50xx e 7xxx)

A tecnologia de núcleo duplo traz dois processadores inteiros dentro de um mesmo invólucro. Como os processadores Xeon de núcleo duplo modelos 50xx e 7xxx têm a tecnologia *HyperThreading* ? que simula a existência de dois processadores em cada núcleo ? o sistema operacional reconhece cada processador Xeon de núcleo duplo como sendo quatro processadores. Assim, em um servidor com dois processadores Xeon de núcleo duplo, o sistema operacional reconhecerá oito processadores (quatro núcleos, dois por processador, e dois processadores lógicos por núcleo).

Todos os processadores Xeon de núcleo duplo possuem as seguintes características: soquete 604 (modelos 7xxx) ou 771 (modelos 50xx); Mesma arquitetura interna no Pentium 4 (*Netburst*); instruções SSE3; cache L1 de dados de 16 KB e cache de execução de 150 KB; suporte a multiprocessamento simétrico com até dois processadores por placa-mãe; barramento externo de 667, 800 ou 1066 MHz; tecnologia Execute Disable; tecnologia EM64T; tecnologia *HyperThreading*; tecnologia de Virtualização nos modelos 7xxx e 50xx; tecnologia Demand-Based Switching (DBS), exceto nos modelos 5060 e 5063; tecnologia Enhanced SpeedStep; tecnologia de construção 90 nanômetros ou 65 nanômetros.

Xeon Núcleo Duplo (51xx)

Intel lançou recentemente uma nova série de processadores Xeon (51xx) baseada na nova microarquitetura Core, a mesma usada pelos processadores Core 2 Duo. Esta nova série era conhecida anteriormente por seu nome-código, *Woodcrest*.

Tenha em mente que os processadores Xeon de núcleo duplo de outras séries (50xx e 7xxx) são baseados na microarquitetura do Pentium 4 (*Netburst*) e por isso possuem a tecnologia *HyperThreading*, que não está presente na microarquitetura Core.

Todos os processadores Xeon da série 51xx possuem as seguintes características: tecnologia de núcleo duplo; tecnologia de 65 nm; soquete 771; instruções SSE3; cache L1 dividido, sendo 32 KB para dados e 32 KB para instruções por núcleo; 4 MB de cache L2 compartilhado entre os núcleos; barramento externo de 1066 ou 1333 MHz; tecnologia EM64T; tecnologia de Virtualização; tecnologia Execute Disable; tecnologia Demand-Based Switching (DBS), nos modelos 5160, 5150, 5148 e 5140; tecnologia Enhanced SpeedStep; tecnologia Dual Independent Bus (DIB), onde cada núcleo tem seu próprio barramento externo em vez de ter apenas um barramento compartilhado entre os núcleos para a comunicação com os outros dispositivos do micro.

5.1.5 Itanium

O projeto de processadores de 64 bits da Intel já se arrasta por muitos anos. O primeiro processador lançado usando essa tecnologia foi o Itanium e recentemente a Intel lançou mais um processador IA-64, o Itanium 2.

Esses dois processadores possuem características de hardware bastante "pesadas". O Itanium tem os dois caches de memória (L1 e L2) dentro do próprio processador, como ocorre com os demais processadores atualmente, e ainda um cache extra (L3) dentro de seu cartucho, podendo esse circuito ter 2 MB ou 4 MB, dependendo da versão do processador. Ele consegue acessar até 16 EB (Exabytes, 1 EB = 2^{60}) de memória RAM e usa um barramento externo de 64 bits rodando a 266 MHz, atingindo uma taxa de transferência de 2,1 GB/s. Esse processador usa uma mistura de soquete com cartucho.

Já o Itanium 2 tem uma memória cache L1 de 32 KB, uma memória cache L2 de 256 KB e uma memória cache L3 de 1,5 MB ou de 3 MB, dependendo do modelo. Os mais atuais possuem uma cache L3 de 6, 8, 12, 18 ou 24 MB dependendo do modelo. Seu barramento externo é de 128 bits, rodando a 400 ou 533 MHz. Possuem suporte a tecnologia Dual-Core, EM64T, VT.

Esses dois processadores são voltados exclusivamente para o mercado de servidores de alto desempenho. Isso deixa a AMD em grande vantagem, já que haverá processadores de 64 bits da AMD voltados para usuários comuns (como o *Clawhammer*).

Outra vantagem da arquitetura de 64 bits da AMD sobre a arquitetura de 64 bits da Intel é que a arquitetura da Intel não roda nativamente código de 32 bits usado pelos programas atuais, ao contrário do que ocorre nos processadores da AMD, que rodam diretamente esse tipo de código. Isso significa que para rodar sistemas operacionais e aplicativos de 32 bits, os processadores IA-64 (Itanium, Itanium 2 e futuros processadores) precisam traduzir as instruções de 32 bits em instruções equivalentes de 64 bits. Isso faz com que haja uma demora na execução da instrução, pois há tempo perdido com essa conversão. O resultado disso é óbvio: em alguns casos pode ocorrer de sistemas operacionais e programas de 32 bits rodarem mais lentamente nos processadores IA-64 da Intel do que em processadores de 32 bits como o Pentium 4 e o Athlon XP.

Mas, como esses processadores da Intel foram destinados ao mercado de servidores, isso não tem muita importância, já que máquinas usando esses processadores com certeza rodarão programas e sistemas escritos diretamente com instruções de 64 bits. Mas como essa é uma nova tecnologia, não só é demorado reescrever programas antigos como escrever novos programas para essa nova plataforma. É claro que esse mesmo problema existe nos processadores da AMD, isto é, possivelmente demorará algum tempo até existirem sistemas operacionais e programas escritos usando código de 64 bits desses processadores. Mas, por outro lado, eles podem rodar diretamente código de 32 bits, facilitando a entrada desses processadores no mercado.

5.2 AMD

5.2.1 Sempron

O Sempron é o processador da AMD voltado para o mercado low-end, ou seja, ele é destinado a usuários que não precisam de grande poder computacional e que estão mais preocupados com preço do que com desempenho. O concorrente do Sempron é Celeron D da Intel.

O processador Sempron está disponível em três versões de soquete: 462 (Socket A), 754 e AM2 (940 pinos). Os processadores Sempron soquete 462 são versões mais simples do Athlon XP, enquanto que os processadores Sempron soquete 754 e soquete AM2 são versões mais simples do Athlon 64.

Como o Sempron soquete 462 usa uma arquitetura interna completamente diferente dos processadores Sempron soquete 754 e soquete AM2, uma comparação direta entre esses dois processadores não é possível.

A nomenclatura "PR" (Performance Rating) usada pelo Sempron só serve para a comparação entre modelos de Sempron usando o mesmo tipo de soquete. Não é possível comparar a nomenclatura PR do Sempron com a do Athlon XP ou com a do Athlon 64. Por exemplo, um Sempron 3000+ não é necessariamente mais rápido do que um Athlon XP 2800+ ou do que um Athlon 64 2800+.

Soquete 462

Os processadores Sempron soquete 462 são, na realidade, processadores Athlon XP com barramento externo de 333 MHz (166 MHz transferindo dois dados por pulso de clock) e 256 KB de memória cache L2 (ou 512 KB no caso do modelo 3000+).

Essa categoria de Sempron possui as demais características do Athlon XP, tais como: 64 KB de cache L1 de instruções e 64 KB de cache L1 de dados; 256 KB ou 512 KB de cache de memória L2; suporte às instruções MMX, 3DNow!, SSE e SSE2 (mas não às instruções SSE3); processo de fabricação de 130 nanômetro.

Soquete 754

Os processadores Sempron soquete 754 são na realidade processadores Athlon 64 com menos memória cache e sem as extensões de 64 bits, sendo que modelos lançados mais recentemente passaram a contar com as extensões de 64 bits.

As principais características dos processadores Sempron soquete 754 são: 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados; 128 KB ou 256 KB de cache de memória L2; barramento *HyperTransport* (barramento externo para acesso ao chipset *Northbridge*). Esta tecnologia permite o processador comunicar-se com a memória RAM e com os demais circuitos do micro ao mesmo tempo, pois existe outro barramento externo para acesso à memória RAM. Antigamente, era apenas um barramento de acesso externo) trabalhando a 800 MHz (este clock pode também ser referenciado como "1.600 MHz") para os modelos atuais; configuração de memória single channel; suporte a instruções SSE3 nos modelos que têm as extensões de 64 bits habilitadas; processo de fabricação de 90 nanômetros.

Soquete AM2

Ao contrário dos processadores Sempron soquete 754, que trabalham apenas na configuração de um único canal (single channel), os processadores Sempron soquete AM2 podem utilizar a configuração de dois canais (dual channel), dobrando a taxa de transferência no acesso à memória desde que você use dois ou quatro módulos de memória em seu micro. Lembre-se que os processadores Sempron soquete 754 aceitam somente memórias DDR, enquanto que os processadores Sempron soquete AM2 aceitam somente memórias DDR2.

As principais características técnicas do Sempron AM2 são as seguintes: 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados; 128 KB ou 256 KB de cache de memória L2; barramento *HyperTransport* trabalhando a 800 MHz (3.2 GB/s). Este clock pode também ser referenciado como "1.600 MHz"; o controlador de memória integrado suporta memórias DDR2-400, DDR2-533 e DDR2-667 na configuração de dois canais (dual channel), o que significa que o processador acessa a memória a 128 bits, se dois ou quatro módulos forem usados; conjunto de instruções SSE3; extensões de 64 bits habilitadas; processo de fabricação de 90 nanômetros.

Existem os modelos de Sempron chamados Mobile Sempron. É concorrente direto do Pentium M. O Mobile Sempron possui as seguintes características: 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados; 128KB ou 256KB de cache L2 incorporado ao processador; *Northbridge* integrado; proteção avançada contra vírus; suporte a instruções SSE2; processo de fabricação 90 nanômetros.

5.2.2 Athlon 64

Os processadores mais novos da AMD encontrados no mercado atualmente são baseados na arquitetura do Athlon 64, também conhecida como x86-64 ou "ham-

mer”. Os modelos de Athlon 64 são o Athlon 64, Athlon 64 FX e o Athlon 64 X2.

Esses três processadores mais o Sempron são voltados para o mercado de desktops. O Athlon 64 é voltado para o mercado mid-range (usuários entusiastas ou aqueles que necessitam de um poder computacional maior do que o proporcionado pelo Sempron) e o Athlon 64 FX e o Athlon 64 X2 são voltados para o mercado high-end (alto desempenho). Existem três outros processadores baseados na arquitetura do Athlon 64: Athlon 64 Mobile e Turion 64, que são voltados para o mercado de notebooks, e o Opteron, que é voltado para o mercado de servidores.

A principal característica da arquitetura do Athlon 64 é a presença do controlador de memória dentro do próprio processador e não no chipset, como acontece com outros processadores. Por causa desta arquitetura a comunicação entre o processador e os módulos de memória é feita através de um barramento dedicado, enquanto que a comunicação entre o processador e o chipset é feita através de um barramento independente, chamado *HyperTransport*.

Processadores da AMD baseados na arquitetura do Athlon 64 podem ser encontrados com os seguintes padrões de pinagem:

Soquete 754

Usado pelas primeiras versões de Athlon 64, alguns modelos de Sempron e Turion 64. Seu controlador de memória usa somente um canal (single channel), o que significa que o processador acessa a memória a 64 bits;

Soquete 939

Usado pelos processadores Athlon 64, Athlon 64 FX, Athlon 64 X2 e o Opteron. Seu controlador de memória usa dois canais (dual channel), o que significa que o processador acessa à memória a 128 bits, se dois módulos de memória forem usados.

Soquete 940

Usado pelos primeiros processadores Athlon 64 FX e pelo Opteron. Seu controlador de memória usa dois canais (dual channel), o que significa que o processador acessa a memória a 128 bits, se dois módulos forem usados (ou um número par de módulos de memória for usado). É necessária a utilização de memórias do tipo ECC;

Soquete AM2

Usado pelos novos processadores Athlon 64, Athlon 64 FX e Athlon 64 X2. Nesses processadores o controlador de memória integrado suporta memórias DDR2-533, DDR2-667 e DDR2-800 na configuração de dois canais (dual channel), o que significa que o processador acessa a memória a 128 bits, se dois módulos forem usados. Lembre-se que o controlador de memória dos processadores soquete 754, 939 e 940 suportam apenas memórias DDR;

Soquete F

Este soquete de 1.207 pinos criado para os novos modelos do Opteron também é usado pelos processadores Athlon 64 FX utilizados na plataforma

Quad FX da AMD (Athlon 64 FX modelos 7x). Os processadores que utilizam este soquete trabalham no modo SMP (multiprocessamento simétrico), podendo trabalhar com mais de um processador em paralelo. Assim como os processadores soquete AM2, nesses processadores o controlador de memória integrado também suporta memórias DDR2-533, DDR2-667 e DDR2-800 na configuração de dois canais (dual channel), o que significa que o processador acessa a memória a 128 bits, se um número par de módulos de memória for usado.

O controlador de memória integrado nos novos processadores Athlon 64 soquete AM2 e Athlon 64 FX soquete F suporta memórias DDR2-533, DDR2-667 e DDR2-800. O problema, no entanto, é como o clock do barramento de memória é obtido. Em vez de ser gerado através do clock base do processador (clock HTT, que é de 200 MHz), é usada uma divisão do clock interno do processador. O valor desta divisão é metade do valor do multiplicador do processador.

Por exemplo, um processador AMD64 com um multiplicador de clock 12x terá um divisor do barramento de memória de 6. Este processador trabalhará a 2,4 GHz (200 MHz x 12) e sua memória funcionará a 400 MHz (DDR2-800, 2.400 MHz / 6). Tenha em mente que as memórias DDR e DDR2 são rotuladas com o dobro dos seus clocks reais.

O problema é quando o multiplicador de clock do processador é um número ímpar. Para um processador AM2 com um multiplicador de clock 13x teoricamente o divisor do seu barramento de memória seria de 6,5. Como o barramento de memória do AMD64 não trabalha com divisores "quebrados" este valor é arredondado para o próximo número inteiro, sete neste caso. Enquanto este processador funcionará a 2,6 GHz (200 MHz x 13) seu barramento de memória funcionará a 371 MHz (742 MHz DDR) e não a 400 MHz (800 MHz DDR), fazendo com que o processador não alcance a largura de banda máxima que as memórias DDR2 podem fornecer.

Outras características encontradas nos processadores baseados na arquitetura do Athlon 64 são as seguintes: O processador não é vendido com base em seu clock de operação, mas sim através de um indicativo de desempenho chamado "performance rating"(PR); podem acessar até 1 TB de memória RAM (barramento de endereços de 40 bits, $2^{40} = 1$ TB); suporte às instruções MMX, 3Dnow!, SSE e SSE2 (SSE3 apenas nos modelos mais novos); tecnologia EVP (Enhanced Vírus Protection, também conhecida como "NX Bit Disable"); tecnologia Cool'n'Quiet (tecnologia que permite reduzir o barulho, o calor e o consumo de energia).

Athlon 64

O Athlon 64 pode ser encontrado em versões para o soquete 754, soquete 939 e o novo soquete AM2. E pode ser encontrado com diferentes versões de núcleo (90 nanômetros ou 130 nanômetros). O concorrente direto do Athlon 64 é o Pentium 4.

As principais características técnicas do Athlon 64 são as seguintes: 64 KB

de cache de L1 de instruções e 64 KB de cache L1 de dados; 512 KB ou 1 MB de cache de memória L2; barramento *HyperTransport* trabalhando a 800 MHz (3,2 GB/s) ou a 1 GHz (4 GB/s). Eses clocks podem também ser referenciados como "1.600 MHz" ou "2.000 MHz", respectivamente; configuração de memória DDR dual channel nos modelos soquete 939 e AM2 (precisa de dois módulos de módulos de memória para usar este recurso); conjunto de instruções SSE3 em alguns modelos.

Athlon 64 X2

O Athlon 64 X2 é um Athlon 64 com tecnologia de núcleo duplo, ou seja, ele possui dois processadores dentro de um só. Todos os processadores Athlon 64 X2 são encontrados para soquete 939 e AM2. O principal corrente do Athlon 64 X2 é o Pentium D Dual Core.

As principais características técnicas do Athlon 64 X2 são as seguintes: 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados por núcleo; 512 KB ou 1 MB de cache de memória L2 por núcleo; barramento *HyperTransport* trabalhando a 1 GHz (4 GB/s). Esse clock pode também ser referenciado como "2.000 MHz"; configuração de memória DDR dual channel em todos os modelos; conjunto de instruções SSE3 em alguns modelos.

Athlon 64 FX

Originalmente a diferença entre o Athlon 64 e o Athlon 64 FX era a quantidade de memória cache L2 (512 KB no Athlon 64 vs. 1 MB no Athlon 64 FX) e maior flexibilidade para overclock, já que ele vinha com o multiplicador de clock destravado.

As principais características técnicas do Athlon 64 FX são as seguintes: 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados por núcleo; 1 MB de cache de L2; barramento *HyperTransport* trabalhando a 800 MHz (3,2 GB/s) ou 1 GHz (4 GB/s); configuração de memória DDR dual channel em todos os modelos; conjunto de instruções SSE3 em alguns modelos; núcleo duplo nos modelos terminados em um número par. O Athlon 64 FX-60 foi o primeiro Athlon 64 FX de núcleo duplo lançado. Esta tecnologia faz com que o processador possua dois processadores completos em seu interior; O Athlon 64 FX-62 é baseado no novo soquete AM2 e, portanto, o seu controlador de memória suporta as memórias DDR2.

5.2.3 Turion 64

Lançado para ser o principal concorrente do Pentium M da Intel, o Turion 64 da AMD é um processador de baixo consumo voltado para o mercado de notebooks.

O Turion 64 é baseado na arquitetura do Athlon 64 e a principal diferença entre o Turion 64 e o Athlon 64 Mobile é o consumo de energia. Uma outra

diferença entre eles é a quantidade de memória cache L2, que é de 1 MB nos processadores Athlon 64 Mobile, enquanto que os processadores Turion 64 podem ter memória cache L2 de 512 KB ou 1 MB, dependendo do modelo.

Tanto o Turion 64 quanto o Athlon 64 Mobile possuem a tecnologia PowerNow! da AMD, que é similar a tecnologia Cool'n'Quiet usado pelos processadores desktop. Esta tecnologia altera o clock e a tensão de alimentação do processador de acordo com a carga de trabalho que esteja sendo realizada, de modo a economizar bateria.

As principais características do Turion 64 são as seguintes: o processador não é vendido com base em seu clock de operação, mas através de um "número de modelo"; 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados; 512 KB ou 1 MB de cache de memória L2; barramento *HyperTransport* trabalhando a 800 MHz (3,2 GB/s); configuração de memória DDR single channel; soquete 754; podem acessar até 1 TB (terabyte) de memória RAM; suporte às instruções MMX, 3Dnow!, SSE e SSE2 e SSE3; tecnologia PowerNow!; tecnologia EVP (Enhanced Virus Protection), também conhecida como "NX Bit Disable"; tecnologia de 90 nanômetros.

Os processadores Turion 64 estão disponíveis em duas séries: ML, que tem consumo máximo (TDP, Thermal Design Power) de 35 W, e MT, que tem consumo máximo de 25 W.

Turion 64 X2

O Turion 64 X2 é um Turion 64 com tecnologia de dois núcleos e com suporte as memórias DDR2 na configuração de dois canais (dual channel), o que significa que o processador pode acessar a memória a 128 bits.

O Turion 64 X2 é o primeiro processador para notebook da AMD com tecnologia de dois núcleos e a usar o novo padrão de pinagem soquete S1, que utiliza 638 pinos em vez dos 754 pinos usados no soquete do Turion 64. Portanto, fisicamente um Turion 64 X2 é menor do que um Turion 64 por ter menos pinos.

Enquanto que o concorrente do Turion 64 é o Pentium M, os concorrentes do Turion 64 X2 são os processadores da Intel Core Duo e Core 2 Duo (nome-código Merom).

As principais características do Turion 64 X2 são as seguintes: 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados por núcleo; 256 ou 512 KB de cache de memória L2 por núcleo; barramento *HyperTransport* trabalhando a 800 MHz (3,2 GB/s); configuração de memória DDR2 dual channel; soquete S1; podem acessar até 1 TB (terabyte) de memória RAM; suporte às instruções MMX, 3Dnow!, SSE e SSE2 e SSE3; tecnologia PowerNow!; tecnologia de Virtualização; tecnologia EVP (Enhanced Vírus Protection); tecnologia de 90 nanômetros.

5.2.4 Opteron

O Opteron é o processador da AMD voltado para o mercado de servidores de rede. Ele é baseado na arquitetura do Athlon 64. Os principais concorrentes são o Intel Xeon e Intel Itanium.

Existem duas diferenças principais entre o Opteron e os outros processadores baseados na arquitetura do Athlon 64. Primeiro, vários modelos de Opteron permitem o multiprocessamento simétrico (SMP), ou seja, permitem trabalhar com mais de um processador na placa-mãe, enquanto que os outros processadores não.

Os processadores Opteron são identificados através de um "número de modelo" e o primeiro dígito deste número indica qual é o grau de processamento simétrico que o processador aceita: os modelos de Opteron começando com "1" não permitem multiprocessamento simétrico, enquanto que os modelos começando com "2" permitem multiprocessamento simétrico com até 2 processadores e os modelos começando com "8" permitem multiprocessamento simétrico com até 8 processadores.

Processadores Opteron com suporte a memórias DDR usam um número de modelo de três dígitos, enquanto que processadores Opteron com suporte a memórias DDR2 usam um número de modelo de quatro dígitos.

A segunda diferença principal é no número de barramentos *HyperTransport* suportados. Todos os processadores baseados na arquitetura do Athlon 64 e os processadores Opteron iniciados com "1" têm apenas um barramento *HyperTransport*. Processadores Opteron iniciados com "2" têm dois barramentos *HyperTransport* (ou três, no caso dos processadores Opteron de quatro dígitos), enquanto que processadores Opteron iniciados com "8" têm três barramentos *HyperTransport*.

Os processadores Opteron podem ser encontrados para vários tipos de soquete:

Soquete 939

Existem alguns modelos de Opteron da série 1 para este soquete. Eles não passam de modelos de Athlon 64 ou de Athlon 64 X2 (se tiver dois núcleos) com outro nome. Estes modelos trabalham com memórias DDR comuns;

Soquete 940

Estes modelos necessitam memórias DDR "registradas" (isto é, com buffer), que é um tipo "especial" de memória para servidores;

Soquete AM2

Existem alguns modelos de Opteron da série 1 para este soquete. Eles não passam de modelos de Athlon 64 X2 com outro nome (porém com a tecnologia de virtualização, não presente no Athlon 64 X2). Estes modelos trabalham com memórias DDR2 comuns;

Soquete F

Estes modelos trabalham com memórias DDR2 "registradas" (isto é, com buffer), que é um tipo "especial" de memória para servidores.

Processadores Opteron que trabalham com memórias DDR2 também são chamados de "Opteron de Segunda Geração".

Em todos os modelos de Opteron o controlador de memória usa dois canais (dual channel), ou seja, o processador acessa a memória a 128 bits, se dois módulos forem usados (ou se um número par de módulos de memória forem usados).

As principais características técnicas do Opteron são as seguintes: suporte a multiprocessamento Simétrico; 64 KB de cache de L1 de instruções e 64 KB de cache L1 de dados; 1 MB de cache de memória L2 por núcleo; barramento *HyperTransport* trabalhando a 800 MHz (3,2 GB/s) ou 1 GHz (4 GB/s); configuração de memória em dois canais (você precisa instalar dois ou um número par de módulos de memória para usar este recurso); podem acessar até 1 TB (terabyte) de memória RAM; suporte às instruções MMX, 3Dnow!, SSE e SSE2 (SSE3 apenas nos modelos mais novos); tecnologia EVP (Enhanced Vírus Protection); tecnologia de virtualização AMD-V nos modelos de quatro dígitos.

Os seguintes modelos de Opteron são encontrados: Opteron Modelos 1xx (DDR) (soquete 940); Opteron Modelos 2xx (DDR) (soquete 940); Opteron Modelos 8xx (DDR) (soquete 940); Opteron Modelos 1xxx (DDR2) (soquete AM2); Opteron Modelos 2xxx (DDR2) (soquete F); Opteron Modelos 8xxx (DDR2) (soquete F).

Parte II

Lógica de Programação

Capítulo 6

Orientação a Objetos

6.1 Introdução

A análise e o projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de software. O funcionamento deste sistema se dá por meio do relacionamento e troca de mensagens entre os objetos.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e os estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

As linguagens de programação que dão suporte a orientação a objetos são: Smalltalk, Perl, Python, Ruby, PHP, ColdFusion, C++, Object Pascal, Java, JavaScript, ActionScript, Delphi e C#.

6.2 Conceitos fundamentais

1. Objetos

Os objetos do mundo real compartilham duas características: possuem um estado e tem um comportamento. Por exemplo, cachorro tem um estado (nome, cor, raça) e um comportamento (latindo, comendo, lambendo). Analogamente, objetos de software são modelados de acordo com os objetos do mundo real, ou seja, possuem estado e comportamento (objeto é uma abstração do mundo real). Um objeto de software armazena seu estado em variáveis e implementa seu comportamento com métodos. Estas variáveis e métodos são formalmente chamados de variáveis de instância e métodos de instância a fim de distingui-los de variáveis e métodos de classe.

As variáveis de um objeto fazem parte do seu núcleo (centro). Os métodos envolvem e escondem o núcleo do objeto de outros componentes da aplicação, 1. Empacotar as variáveis de um objeto sobre proteção de seus métodos é chamado de encapsulamento. O encapsulamento é utilizado para esconder detalhes de implementação pouco importante. Este é um dos princípios fundamentais da programação orientada a objetos.

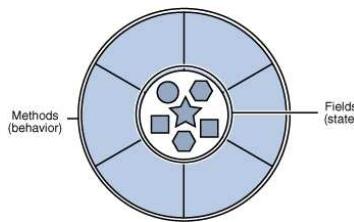


Figura 6.1: Representação de um objeto

Às vezes, por razões de eficiência ou implementação, um objeto deseja expor algumas de suas variáveis ou esconder alguns de seus métodos. Esta capacidade de controlar quais componentes podem acessar seus métodos e suas variáveis é chamada de Controle de Acesso.

Cada objeto tem sua identidade que significa que dois objetos são distintos mesmo que eles apresentem exatamente as mesmas características (atributos iguais). Esta identificação de objeto deve ser única, uniforme e independente do conteúdo do objeto.

A estrutura de dados de um objeto é representada em termos de atributos (também chamados de campos (fields)), ou seja, os dados ou informações do objeto.

A 1 mostra um exemplo da representação de objeto do mundo real para o objeto de software. Os atributos Raio, x e y são as variáveis de instância. Em Java, para declarar uma variável de classe basta acrescentar a palavra static. O mesmo vale para o método.

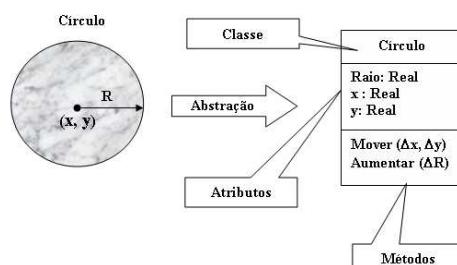


Figura 6.2: Representação do objeto do mundo real para o objeto de software

2. Métodos

É uma rotina que é executada por um objeto ao receber uma mensagem. Os métodos determinam o comportamento dos objetos de uma classe e são análogos às funções ou procedimentos da programação estruturada. O envio de mensagens (chamado de métodos) pode alterar o estado de um objeto.

3. Atributos

Os atributos são os elementos que definem a estrutura de uma classe. Os

atributos também são conhecidos como variáveis de classe, e podem ser divididos em dois tipos básicos: atributos de instância e de classe. Os valores dos atributos de instância determinam o estado de cada objeto. Um atributo de classe possui um estado que é compartilhado por todos os objetos de uma classe. Atributos de classe podem ser chamados também de atributos estáticos ou constantes.

As mensagens enviadas a um objeto (chamada de método) podem mudar o valor de um ou mais atributos, alterando o estado de um objeto.

4. Classes

Uma classe é um modelo (protótipo) que define as variáveis (estado) e os métodos (comportamento) comuns a todos os objetos do mesmo tipo. Na classe são definidas as variáveis e implementados os métodos. Os objetos são criados a partir de suas classes.

As classes não são diretamente suportadas em todas as linguagens de objetos, e não são necessárias para que a linguagem seja orientada a objetos.

Além dos métodos e atributos de uma classe, outros possíveis membros são:

Construtor: define o comportamento no momento da criação de um objeto de uma classe; Destruitor: define o comportamento no momento da destruição do objeto de uma classe. Normalmente utilizado para liberar recurso (memória) do sistema; Propriedade: define o acesso a um estado do objeto; Evento: define um ponto em que o objeto pode chamar outros procedimentos de acordo com seu comportamento e estado interno;

Obs.: Lembrar sempre que a classe define as características comuns e os objetos são instâncias dessa classe, com estado própria.

5. Mensagens

Objetos se comunicam entre si por meio do envio de mensagens. Quando um objeto A deseja que o objeto B realize um dos seus métodos (de B), o objeto A envia uma mensagem para o objeto B. Algumas vezes, o objeto que recebe a mensagem precisa de mais informações para saber exatamente o que fazer. Por exemplo, quando você quer trocar as marchas de uma bicicleta, deve-se indicar qual a marcha desejada. Esta informação acompanha a mensagem como um parâmetro.

Assim, três componentes fazem parte de uma mensagem:

o objeto para onde a mensagem é endereçada (bicicleta); o nome do método a realizar (mudar marcha); parâmetro(s) necessários para realizar o método (segunda marcha);

A mensagem, também, pode ser direcionada diretamente a uma classe por meio da invocação de um método dinâmico.

6. Herança

A herança é um mecanismo que permite criar novas classes a partir de classes já existentes, aproveitando-se das características existentes na classe a ser extendida. Com a herança é possível criar classes derivadas (sub-classes) a partir de classes bases (superclasses).

As subclasses herdam todas as características de suas superclasses, como suas variáveis (estado) e seus métodos (comportamento). Entretanto, as subclasses não estão limitadas ao comportamento herdado de sua superclasse. As subclasses podem adicionar variáveis e métodos a aqueles herdados.

As subclasses, também, podem redefinir (override) métodos herdados e oferecer implementações especializadas para estes métodos.

O conceito de herança pode ser aplicado para mais de um nível. A árvore de herança, ou hierarquia de classe, pode ser tão *profunda* quanto necessário. Os métodos e variáveis são herdados por meio dos níveis. Em geral, quanto mais baixa na hierarquia for a posição da classe, mais específico é o seu comportamento.

Como benefícios do conceito de herança, podemos citar:

Subclasses oferecem comportamentos especializados a partir de elementos básicos comuns, oferecidos pela superclasse. A utilização do conceito de herança é muito interessante, pois promove um grande reuso e reaproveitamento de código existente; Programadores podem definir classes abstratas que determinam comportamentos *genéricos*. A superclasse abstrata define e pode implementar parcialmente o comportamento de uma classe, mas a maioria das informações da classe fica indefinida ou não é implementada.

A herança múltipla ocorre quando uma classe pode estender características de várias outras classes ao mesmo tempo, ou seja, quando a subclass possuir mais de uma superclasse. Algumas linguagens evitam utilizar este mecanismo, pois pode levar uma codificação confusa o que dificulta a manutenção do código. A linguagem Java não suporta herança múltipla, apenas herança simples. Já a linguagem C++ oferece suporte à herança múltipla.

Uma linguagem ao utilizar herança múltipla está sujeita a dois problemas: colisão de nomes (nomes idênticos nas classes superiores) e herança repetida (classe herda de uma classe superior que herda de uma classe comum);

7. Polimorfismo

Segundo a terminologia de orientação a objetos, polimorfismo significa que uma mesma mensagem enviada a diferentes objetos resulta em um comportamento que é dependente da natureza do objeto que está recebendo a mensagem. Ou seja, duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma assinatura (nome, lista de parâmetros e retorno), mas comportamentos distintos, especializado para cada classe derivada, usando para tanto uma referência a um objeto do tipo superclasse.

A decisão sobre qual método deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em tempo de execução por meio do mecanismo de ligação tardia. No caso do polimorfismo, é necessário que os métodos tenham exatamente a mesma identificação, sendo utilizado o mecanismo de redefinição de métodos. A seguir um exemplo de polimorfismo.

```
public abstract class OperacaoMatematica
{
    public abstract double calcular(double x, double y);
}

public class Soma extends OperacaoMatematica
{
    public double calcular(double x, double y)
    {
        return x+y;
    }
}

public class Subtracao extends OperacaoMatematica
{
    public double calcular(double x, double y)
    {
        return x-y;
    }
}

public class Contas
{
    public static void mostrarCalculo(OperacaoMatematica operacao, double x, double y)
    {
        System.out.println("O resultado é: " + operacao.calcular(x, y));
    }

    public static void main( String args[] )
    {
        //Primeiro calculamos uma soma
        Contas.mostrarCalculo(new Soma(), 5, 5); //Imprime o resultado é: 10
        Contas.mostrarCalculo(new Subtracao(), 5, 5); //Imprime o resultado é: 0
    }
}
```

Note que, embora o método calcular tenha sido chamado duas vezes no interior de mostrarCalculo, o comportamento apresentado variou de acordo com a classe ao qual ele representava no momento.

Os benefícios do polimorfismo são: a clareza e manutenção do código, a divisão da complexidade e aplicações flexíveis.

Algumas linguagens promovem o polimorfismo principalmente por meio do uso de classes abstratas e interfaces, como é o caso da linguagem Java.

8. Sobrecarga

A sobrecarga é um tipo de polimorfismo que permite a existência de vários métodos de mesmo nome, porém com assinaturas levemente diferentes, ou seja, variando no número e tipo de argumentos e o valor de retorno. Ficará a cargo do compilador escolher de acordo com as listas de argumento os procedimentos ou métodos a serem executados. Por exemplo:

```
public class Soma
{
    public int Soma (int x, int y)
    {
        return x+y;
    }

    public String Soma (String x, String y)
    {
        return x+y;
    }

    public double Soma (double x, double y)
    {
        return x+y;
    }
}
```

9. Interfaces

Uma interface é uma coleção de definições de métodos abstratos (sem implementação). É utilizada para os objetos interagirem entre si. A classe que implementa a interface deve implementar todos os métodos definidos na interface. Uma interface, também, pode incluir declarações de constantes.

Apesar de uma classe abstrata implementar métodos abstratos, não se pode confundir interface com classe abstrata. As classes abstratas não podem ser instanciadas, e estas só são utilizadas como superclasses. Por exemplo, comida no mundo real. Não existe uma instância (objeto) do tipo comida. O que existe são instâncias de cenouras, maçãs, bananas, etc. Comida representa o conceito abstrato e não é capaz de criar uma instância própria. As classes abstratas podem possuir métodos implementados.

Em Java, para definir uma interface utiliza-se a palavra interface. Embora, uma classe não possa conter mais de uma superclasse, a classe pode implementar mais de uma interface. Por este motivo, muitas das vezes as interfaces são apresentadas como uma alternativa para herança múltipla de classes.

10. Pacotes

Para tornar as classes mais fáceis de serem encontradas e utilizadas, para evitar conflitos de nomes e para controlar o acesso, pode-se agrupar classes relacionadas em pacotes (packages). Os pacotes organizam as classes e as interfaces relacionadas. Cada pacote criado corresponde a um diretório, ou seja, as classes e as interfaces de um mesmo pacote devem estar em um mesmo diretório.

A utilização de pacotes proporciona as seguintes vantagens:

Fica mais fácil para outros programadores determinar quais classes e interfaces estão relacionadas; Os nomes das classes de um pacote não irão

conflitar com nomes e classes de outros pacotes; Pode-se permitir que classes dentro de um mesmo pacote tenham acesso irrestrito entre si, e restringir o acesso por parte de classes definidas fora do pacote.

Em Java, para criar um pacote coloca-se a palavra chave package no início do arquivo onde a classe ou interface é definida.

Obs.: Apenas os membros (classe, variáveis e métodos) públicos são acessíveis fora do pacote onde foram definidos.

Obs.: Para uma linguagem ser considerada verdadeiramente orientada a objetos, a linguagem de programação deve oferecer, no mínimo, as características de: encapsulamento, herança e polimorfismo.

6.3 Princípios de programação orientada a objetos

Basicamente, os princípios de programação orientada a objetos são:

Classes, objetos e instâncias; Campos, métodos e propriedades; Sobrecarga; Herança e classes hierárquicas; Polimorfismo; Interfaces.

6.4 Tratamento de exceções

O termo exceção é usado para designar um evento que ocorre durante a execução de um programa que desvia o fluxo normal de instruções. Em outras palavras, uma exceção é uma condição provocada por uma situação excepcional que requer uma ação específica imediata. Muitos tipos de erros podem causar exceções: problemas que variam desde erros sérios de hardware, tal como uma falha no disco rígido, a erros simples de programação, tal como tentar acessar um índice inexistente de um vetor, divisão por zero, um objeto não inicializado, etc.

Erros causam exceções, mas podem existir exceções que não são erros. Por exemplo, numa função que lê dados de um arquivo, a chegada ao final do arquivo é uma condição excepcional que pode ser considerada como exceção.

Linguagens como ADA, C++, JAVA e EIFFEL possuem mecanismos próprios de tratamento de exceções que facilitam a vida dos programadores tornando ainda o código mais legível uma vez que separam o código principal do código de tratamento do erro.

Quando uma exceção ocorre, ela necessita ser tratada. A unidade ou bloco de código que manipula a exceção é denominada tratador de exceção e a ação de indicar a ocorrência de exceção e transferir o controle para o tratador de exceção é denominada sinalização da exceção.

Tratadores de exceção se comportam como procedimentos chamados implicitamente pela ocorrência de uma exceção. Como tratadores de exceção não são chamados diretamente, eles não possuem nome. Em sua definição costumam conter variáveis locais e bloco de código de tratamento. Já as exceções devem possuir nome bem definido para que possam ser identificadas.

Caso o tratador que capturou a exceção não tenha conhecimento suficiente para tratar a exceção ele pode lançar a exceção para um nível superior (propagação da exceção). Assim pode-se gerar uma cadeia de propagação até que um tratador capture a exceção ou até atingir-se o nível mais alto da hierarquia dos

tratadores onde normalmente um tratador padrão será executado na saída do programa.

Outra questão importante é o ponto de retorno após a execução do bloco de código do tratador. Isto dependerá do modelo de continuação adotado pelo mecanismo de tratamento de exceção. Em geral, adota-se um dos seguintes modelos de continuação:

Terminação: assume que o erro é crítico e que não existem condições de retornar ao ponto onde a exceção foi gerada, retornando o controle para o nível superior. Nesta alternativa a unidade onde ocorreu a exceção e todas as unidades na pilha de chamada de subprogramas até onde o tratador de exceção foi executado são encerradas, continuando-se a execução do programa na unidade onde o tratador foi encontrado, após a região de tratamento; Retomada: assume-se que o erro pode ser corrigido e a execução pode retornar para o bloco onde ocorreu a exceção. Portanto, o retorno é feito para o bloco gerador do erro. Embora, em princípio, essa solução pareça ser a mais apropriada, a experiência tem mostrado que essa alternativa raramente é efetiva.

Normalmente, as linguagens têm adotado o modelo de terminação. Em Java, a exceção é um tipo especial de objeto que é criado quando algo sai de errado em um programa. Após criar uma exceção, o ambiente de execução envia este objeto ao seu programa, numa ação denominada *levantar uma exceção*. É responsabilidade do programa capturar esta exceção. Para capturar uma exceção utilizam-se as cláusulas try (blocos de códigos protegidos) e catch (tratadores de exceções). Para *passar para frente* a exceção, utiliza-se a palavra throws. As exceções em Java são instâncias da classe Exception (superclasse) ou de suas herdeiras.

Parte III

Metodologia de Desenvolvimento

Capítulo 7

Ciclo de Vida

Um modelo de ciclo de vida ou modelo de processo pode ser visto como uma representação abstrata de um esqueleto de processo, incluindo tipicamente algumas atividades principais, a ordem de precedência entre elas e, opcionalmente, artefatos requeridos e produzidos. De maneira geral, um modelo de processo descreve uma filosofia de organização de atividades, estruturando as atividades do processo em fases e definindo como essas fases estão relacionadas. Entretanto, ele não descreve um curso de ações preciso, recursos, procedimentos e restrições. Ou seja, ele é um importante ponto de partida para definir como o projeto deve ser conduzido, mas a sua adoção não é o suficiente para guiar e controlar um projeto de software na prática.

Ainda que os processos tenham de ser definidos caso a caso, de maneira geral, o ciclo de vida de um software envolve, pelo menos, as seguintes fases:

Planejamento O objetivo do planejamento de projeto é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Uma vez estabelecido o escopo de software, com os requisitos esboçados, uma proposta de desenvolvimento deve ser elaborada, isto é, um plano de projeto deve ser elaborado configurando o processo a ser utilizado no desenvolvimento de software. À medida que o projeto progride, o planejamento deve ser detalhado e atualizado regularmente. Pelo menos ao final de cada uma das fases do desenvolvimento (análise e especificação de requisitos, projeto, implementação e testes), o planejamento como um todo deve ser revisto e o planejamento da etapa seguinte deve ser detalhado. O planejamento e o acompanhamento do progresso fazem parte do processo de gerência de projeto.

Análise e Especificação de Requisitos Nesta fase, o processo de levantamento de requisitos é intensificado. O escopo deve ser refinado e os requisitos mais bem definidos. Para entender a natureza do software a ser construído, o engenheiro de software tem de compreender o domínio do problema, bem como a funcionalidade e o comportamento esperados. Uma vez capturados os requisitos do sistema a ser desenvolvido, estes devem ser modelados, avaliados e documentados. Uma parte vital desta fase é a construção de um modelo descrevendo o que o software tem de fazer (e não como fazê-lo).

Projeto Esta fase é responsável por incorporar requisitos tecnológicos aos requisitos essenciais do sistema, modelados na fase anterior e, portanto, requer que a plataforma de implementação seja conhecida. Basicamente, envolve duas grandes etapas: projeto da arquitetura do sistema e projeto detalhado. O objetivo da primeira etapa é definir a arquitetura geral do software, tendo por base o modelo construído na fase de análise de requisitos. Essa arquitetura deve descrever a estrutura de nível mais alto da aplicação e identificar seus principais componentes. O propósito do projeto detalhado é detalhar o projeto do software para cada componente identificado na etapa anterior. Os componentes de software devem ser sucessivamente refinados em níveis maiores de detalhamento, até que possam ser codificados e testados.

Implementação O projeto deve ser traduzido para uma forma passível de execução pela máquina. A fase de implementação realiza esta tarefa, isto é, cada unidade de software do projeto detalhado é implementada.

Testes inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os diversos componentes devem ser integrados sucessivamente até se obter o sistema. Finalmente, o sistema como um todo deve ser testado.

Entrega e Implantação uma vez testado, o software deve ser colocado em produção. Para tal, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta fase é estabelecer que o software satisfaz os requisitos dos usuários. Isto é feito instalando o software e conduzindo testes de aceitação. Quando o software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.

Operação nesta fase, o software é utilizado pelos usuários no ambiente de produção.

Manutenção Indubitavelmente, o software sofrerá mudanças após ter sido entregue para o usuário. Alterações ocorrerão porque erros foram encontrados, porque o software precisa ser adaptado para acomodar mudanças em seu ambiente externo, ou porque o cliente necessita de funcionalidade adicional ou aumento de desempenho. Muitas vezes, dependendo do tipo e porte da manutenção necessária, essa fase pode requerer a definição de um novo processo, onde cada uma das fases precedentes é re-aplicada no contexto de um software existente ao invés de um novo.

Os modelos de processo, de maneira geral, contemplam as fases Análise e Especificação de Requisitos, Projeto, Implementação, Testes e Entrega e Implantação. A escolha de um modelo de processo é fortemente dependente das características do projeto. Assim, é importante conhecer alguns modelos de ciclo de vida e em que situações são aplicáveis.

7.1 Modelo seqüencial linear

Algumas vezes chamado de *ciclo de vida clássico* ou *modelo em cascata*, o *modelo seqüencial linear* requer uma abordagem sistemática seqüencial para o desempenho do software. Abrange as seguintes atividades:

Modelagem e engenharia do sistema. Estabelecimento de todos os requisitos do sistema, alocação de algum desses requisitos para o software.

Análise de requisitos de software. O processo de definição de requisitos é intensificado e focalizado especificamente no software. Os requisitos do sistema do software são documentados e revistos com o cliente.

Projeto. Traduz os requisitos para uma representação do software, que pode ser avaliada quanto à qualidade, antes que a codificação tenha início. Enfoca quatro atributos distintos: estrutura de dados, arquitetura do software, representações da interface e detalhes procedimentais (algorítmicos).

Geração de código. O projeto é traduzido para linguagem de máquina.

Teste. São conduzidos para descobrir erros e garantir que entradas definidas produzirão resultados reais, que concordam com os resultados exigidos.

Manutenção. O software irá inevitavelmente sofrer modificações. A manutenção replica cada uma das fases precedentes a um programa existente.

O modelo seqüencial linear é o mais amplamente usado. Entre os problemas encontrados quando esse modelo é aplicado são:

1. Modificações podem causar confusão à medida que o projeto prossegue, pois o modelo acomoda interação apenas indiretamente.
2. Exige que o cliente estabeleça todos os requisitos explicitamente.
3. Uma versão executável do programa só ficará disponível quando o projeto terminar.

7.2 Modelo em V

O modelo em V é uma variação do modelo em cascata que procura enfatizar a estreita relação entre as atividades de teste (teste de unidade, teste de integração, teste de sistema e teste de aceitação) e as demais fases do processo, como mostra a figura 7.1.

O modelo em V sugere que os testes de unidade são utilizados basicamente para verificar a implementação e o projeto detalhado. Uma vez que os testes de integração estão focados na integração das unidades que compõem o software, eles também são usados para avaliar o projeto detalhado. Assim, testes de unidade e integração devem garantir que todos os aspectos do projeto do sistema foram implementados corretamente no código. Quando os testes de integração atingem o nível do sistema como um todo (teste de sistema), o projeto da arquitetura passa a ser o foco. Neste momento, busca-se verificar se o sistema atende aos requisitos definidos na especificação. Finalmente, os testes de aceitação, conduzidos tipicamente pelos usuários e clientes, validam os requisitos,

confirmando que os requisitos corretos foram implementados no sistema (teste de validação). A conexão entre os lados direito e esquerdo do modelo em V implica que, caso sejam encontrados problemas em uma atividade de teste, a correspondente fase do lado esquerdo e suas fases subsequentes podem ter de ser executadas novamente para corrigir ou melhorar esses problemas.

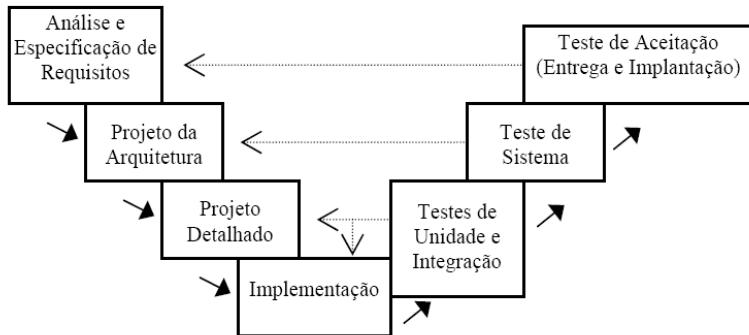


Figura 7.1: O modelo em V

Os modelos seqüenciais pressupõem que o sistema é entregue completo, após a realização de todas as atividades do desenvolvimento. Entretanto, nos dias de hoje, os clientes não estão mais dispostos a esperar o tempo necessário para tal, sobretudo, quando se trata de grandes sistemas. Dependendo do porte do sistema, podem se passar anos até que o sistema fique pronto, sendo inviável esperar. Assim, outros modelos foram propostos visando a, dentre outros, reduzir o tempo de desenvolvimento. A entrega por partes, possibilitando ao usuário dispor de algumas funcionalidades do sistema enquanto outras estão sendo ainda desenvolvidas, é um dos principais mecanismos utilizados por esses modelos, como veremos a seguir.

7.3 Modelo de prototipagem

O *paradigma de prototipagem* começa com a definição dos requisitos. Um projeto rápido é realizado e concentra-se na representação daqueles aspectos que ficarão visíveis pelo cliente. O protótipo é criado e avaliado e é ajustado para satisfazer as necessidades do cliente.

Ideialmente, o protótipo serve como um mecanismo para identificação dos requisitos do software. A prototipagem pode ser problemática, pois o cliente vê o que parece ser uma versão executável do software, ignorando que o protótipo apenas consegue funcionar precariamente.

7.4 Modelo RAD

O *desenvolvimento rápido de aplicação* (*rapid application development*, RAD) é um modelo incremental que enfatiza o ciclo de desenvolvimento extremamente curto. Abrange as seguintes fases:

Modelagem do negócio. O fluxo de informação entre as funções do negócio é modelado.

Modelagem dos dados. O fluxo de informação é refinado num conjunto de objetos de dados.

Modelagem do processo. Os objetos de dados são transformados para conseguir o fluxo de informação necessário para implementar uma função do negócio. Descrições do processamento são criadas.

Geração da aplicação. O RAD considera o uso de técnicas de quarta geração.

O processo RAD trabalha para reusar componentes de programas existentes ou criar componentes reusáveis.

Teste e entrega. Os componentes novos devem ser testados e todas as interfaces devem ser exaustivamente exercitadas.

As desvantagens do RAD são:

1. O RAD exige desenvolvedores e clientes compromissados com atividades continuamente rápidas.
2. Nem todos tipo de aplicação são apropriados para o RAD.
3. Quando riscos técnicos são elevados o RAD não é adequado.

7.5 Modelos de processo de software evolucionários

7.5.1 Modelo incremental

Este modelo é uma extensão do modelo espiral sendo porém mais formal e rigoroso. O desenvolvimento de um produto comercial de software é uma grande tarefa que pode ser estendida por vários meses, possivelmente um ano ou mais. Por isso, é mais prático dividir o trabalho em partes menores ou iterações. Cada iteração resultará num incremento. Iterações são passos em fluxo de trabalho e incrementos são crescimentos do produto.

O princípio subjacente ao processo incremental e iterativo é que a equipe envolvida possa refinar e alargar paulatinamente a qualidade, detalhe e âmbito do sistema envolvido.

Por exemplo, numa primeira iteração deve-se identificar a visão global e determinar a viabilidade econômica do sistema, efetuar a maior parte da análise e um pouco de desenho e implementação. Numa segunda geração, deve-se concluir a análise, fazer uma parte significativa do desenho e um pouco mais de implementação. Numa terceira iteração, deve-se concluir o desenho, fazer-se parte substancial da implementação, testar e integrar um pouco, etc. Ou seja, a principal consequência da aproximação iterativa é que os produtos finais de todo o processo vão sendo amadurecidos e completados ao longo do tempo, mas cada iteração produz sempre um conjunto de produtos finais.

A cada iteração são realizadas as seguintes tarefas:

Análise. Refinamento de requisitos, refinamento do modelo conceitual.

Projeto. Refinamento do projeto arquitetural, projeto de baixo nível.

Implementação. Codificação e testes.

Transição para produto. Documentação, instalação,

Vantagens do processo incremental e iterativo:

- Possibilidade de avaliar mais cedo os riscos e pontos críticos do projeto, e identificar medidas para os eliminar ou controlar.
- Redução dos riscos envolvendo custos a um único incremento. Se a equipe que desenvolve o software precisar repetir a iteração, a organização perde somente o esforço mal direcionado de uma iteração, não o valor de um produto inteiro.
- Definição de uma arquitetura que melhor possa orientar todo o desenvolvimento.
- Disponibilização natural de um conjunto de regras para melhor controlar os inevitáveis pedidos de alterações futuras.
- Permite que os vários intervenientes possam trabalhar mais efetivamente pela interação e partilha de comunicação daí resultante.

7.5.2 Modelo espiral

O modelo original em espiral organiza o desenvolvimento como um processo iterativo em que vários conjuntos de fases se sucedem até se obter o sistema final. Um ciclo se inicia com a determinação de objetivos, alternativas e restrições (primeira tarefa) onde ocorre o comprometimento dos envolvidos e o estabelecimento de uma estratégia para alcançar os objetivos. Na segunda tarefa, análise e avaliação de alternativas, identificação e solução de riscos, executa-se uma análise de risco. Prototipação é uma boa ferramenta para tratar riscos. Se o risco for considerado inaceitável, pode parar o projeto. Na terceira tarefa ocorre o desenvolvimento do produto. Neste quadrante pode-se considerar o modelo cascata. Na quarta tarefa o produto é avaliado e se prepara para iniciar um novo ciclo.

Variações do modelo espiral consideram entre três e seis tarefas ou setores da espiral, que podem ser:

- comunicação com o cliente;
- planejamento;
- análise de risco;
- engenharia;
- construção e liberação;
- avaliação do cliente.

O modelo espiral é, atualmente a abordagem mais realística para desenvolvimento de software em grande escala, e usa uma abordagem que capacita a empresa que presta o serviço, e o cliente a entender e reagir aos riscos em cada etapa evolutiva. Este tipo de modelo exige considerável experiência na determinação de riscos e depende dessa experiência para ter sucesso.

Vantagens deste modelo:

- O modelo em espiral permite que ao longo de cada iteração se obtenham versões do sistema cada vez mais completas, recorrendo à prototipagem para reduzir os riscos.
- Este tipo de modelo permite a abordagem do refinamento seguido pelo modelo em cascata, mas que incorpora um enquadramento iterativo que reflete, de uma forma bastante realística, o processo de desenvolvimento.

Desvantagens:

- Pode ser difícil convencer grandes clientes (particularmente em situações de contrato) de que a abordagem evolutiva é controlável.
- A abordagem deste tipo de modelo exige considerável experiência na avaliação dos riscos e baseia-se nessa experiência para o sucesso. Se um grande risco não for descoberto, poderão ocorrer problemas.
- Este tipo de modelo é relativamente novo e não tem sido amplamente usado.
- É importante ter em conta que podem existir diferenças entre o protótipo e o sistema final. O protótipo pode não cumprir os requisitos de desempenho, pode ser incompleto, e pode refletir somente alguns aspectos do sistema a ser desenvolvido.
- O modelo em espiral pode levar ao desenvolvimento em paralelo de múltiplas partes do projeto, cada uma sendo abordada de modo diferenciado, por isso é necessário o uso de técnicas específicas para estimar e sincronizar cronogramas, bem como para determinar os indicadores de custo e progresso mais adequados.

7.5.3 Modelo espiral ganha-ganha

As melhores noegociações buscam um resultado ganha-ganha. Isto é, o cliente ganha obtendo um produto ou sistema que satisfaz à maior parte das necessidades do cliente, e o desenvolvedor ganha trabalhando com orçamentos e prazos de entrega realísticos e possíveis de serem cumpridos.

O modelo espiral ganha-ganha define um conjunto de atividades de negociação no começo de cada passagem, em torno da espiral. Ao invés de uma única atividade de comunicação com o cliente, as seguintes atividades são definidas:

1. Identificação dos principais interessados do sistema ou do subsistema.
2. Determinação das condições de lucro do interessado.
3. Negociação das condições de ganho do interessado para reconciliá-las no âmbito das condições ganha-ganha para todos os envolvidos.

Além da ênfase na negociação inicial, o modelo espiral ganha-ganha introduz três marcos de processo, chamados *pontos de ancoragem*, que ajudam a estabelecer quando um ciclo é completado em torno da espiral e fornecem marcos de decisão antes do projeto de software presseguir. Esses pontos de ancoragem são:

Objetivos de ciclo de vida (life cycle objectives, LCO). Define um conjunto de objetivos para cada atividade principal.

Arquitetura do ciclo de vida (life cycle architecture, LCA). Estabelece objetivos que precisam ser satisfeitos, à medida que a arquitetura do sistema, e do software, é definida.

Capacidade operacional inicial (initial operational capability, IOC). Representa um conjunto de objetivos associado com a preparação do software para instalação e distribuição.

7.5.4 Modelo de desenvolvimento concorrente

O modelo de desenvolvimento concorrente é representado como uma série de grandes atividades técnicas, tarefas e seus estados associados. Ele define uma série de eventos que podem disparar transições de um estado para outro, para cada uma das atividades da engenharia de software. É freqüentemente usado como um paradigma para o desenvolvimento de aplicações Cliente/Servidor. Pode ser aplicado a todo tipo de desenvolvimento de software e fornece uma visão exata de como está o estado do projeto.

7.6 Desenvolvimento baseado em componentes

O desenvolvimento baseado em componentes incorpora características de tecnologias orientadas a objetos no modelo espiral. A atividade de engenharia começa com a identificação de classes candidatas. Se a classe existe, ela será reutilizada. Se a classe não existe, ela será desenvolvida nos moldes do paradigma de orientação a objetos.

7.7 Modelo de métodos formais

O modelo de métodos formais compreende um conjunto de atividades que determinam uma especificação matemática para o software. Uma variante dessa abordagem é denominada *engenharia de software cleanroom (Sala Simpa)*. Utilizando métodos formais eliminam-se muitos problemas encontrados nos outros modelos, como p.ex., ambigüidade, incompletitude e inconsistência, que podem ser corrigidas mais facilmente de forma não ad hoc, mas através de análise matemática.

7.8 Técnicas de quarta geração

As técnicas de quarta geração concentram-se na capacidade de se especificar o software a uma máquina em um nível que esteja próximo à linguagem natural.

Engloba um conjunto de ferramentas de software que possibilitam que:

- O sistema seja especificado em uma linguagem de alto nível.
- O código fonte seja gerado automaticamente a partir dessas especificações.

Atividades das técnicas de quarta geração englobam:

Obtenção dos Requisitos. O cliente descreve os requisitos os quais são traduzidos para um protótipo operacional. Os principais problemas são:

- O cliente pode estar inseguro quanto aos requisitos.
- O cliente pode ser incapaz de especificar as informações de um modo que uma ferramenta 4GL possa realizar.
- As 4GLs atuais não são sofisticadas suficientemente para acomodar a verdadeira linguagem natural.

Estratégia de projeto. : Para pequenas aplicações é possível mover-se do passo de obtenção dos requisitos para o passo de implementação usando uma Linguagem de 4G. Para grandes projetos é necessário desenvolver uma estratégia de projeto. De outro modo ocorrerão os mesmos problemas encontrados quando se usa abordagem convencional (baixa qualidade).

Implementação usando 4GL. Os resultados desejados são representados de modo que haja geração automática de código. Deve existir uma estrutura de dados com informações relevantes e que seja acessível pela 4GL .

Teste. O desenvolvedor deve efetuar testes e desenvolver uma documentação significativa. O software desenvolvido deve ser construído de maneira que a manutenção possa ser efetuada prontamente.

Capítulo 8

Análise Comparativa de Processos de Desenvolvimento

Um processo de desenvolvimento de software pode ser visto como um guia para o desenvolvimento e deve estabelecer: as atividades a serem realizadas durante o desenvolvimento de software, sua estrutura e organização (decomposição e precedência de atividades), incluindo a definição de um modelo de ciclo de vida; os artefatos requeridos e produzidos por cada uma das atividades do processo; os procedimentos (métodos, técnicas e normas) a serem adotados na realização das atividades; os recursos necessários para a realização das atividades, dentre eles recursos humanos, recursos de hardware e recursos de software, incluindo as ferramentas a serem utilizadas para apoiar a aplicação dos procedimentos na realização das atividades; e roteiros para a elaboração dos principais documentos (artefatos) gerados no desenvolvimento. Entre as principais metodologias em uso no mercado de software atualmente estão *RUP* e *XP*.

De maneira geral, o ciclo de vida de um software envolve, pelo menos, as atividades de planejamento, levantamento de requisitos, análise, projeto, implementação, testes, implantação, operação e manutenção.

Uma vez que o software é sempre parte de um sistema (ou negócio) maior, o trabalho começa pelo estabelecimento dos requisitos para todos os elementos do sistema e, na seqüência, procede-se a alocação para software de algum subconjunto destes requisitos. Esta etapa é a Engenharia de Sistemas e antecede a todas as demais relacionadas.

8.1 RUP - Rational Unified Process

O Processo Unificado proposto pela Rational (Rational Unified Process – RUP) foi criado para apoiar o desenvolvimento orientado a objetos, fornecendo uma forma sistemática para se obter reais vantagens no uso da Linguagem de Modelagem Unificada (Unified Modeling Language – UML). De fato, ele não é exatamente um processo: é uma infraestrutura genérica de processo que pode ser

especializada para uma ampla classe de sistemas de software, para diferentes áreas de aplicação, tipos de organização, níveis de competência e tamanhos de projetos.

O RUP está fundamentado em três princípios básicos: orientação a casos de uso, arquitetura e iteração. Ele é dito dirigido a casos de uso, pois são os casos de uso que orientam todo o processo de desenvolvimento. Com base no modelo de casos de uso, são criados uma série de modelos de análise, projeto e implementação, que realizam estes casos de uso. É centrado em arquitetura, pois defende a definição de um esqueleto para a aplicação (a arquitetura), a ganhar corpo gradualmente ao longo do desenvolvimento. Finalmente, o RUP é iterativo e incremental, oferecendo uma abordagem para particionar o trabalho em porções menores ou mini-projetos. Esses três conceitos são igualmente importantes. A arquitetura provê a estrutura para guiar o desenvolvimento do sistema em iterações, enquanto os casos de uso definem as metas e conduzem o trabalho de cada iteração.

O ciclo de vida adotado no RUP é tipicamente evolutivo. Contudo, uma forma de organização em fases é adotada para comportar os ciclos de desenvolvimento, permitindo uma gerência mais efetiva de projetos complexos. Ao contrário do tradicionalmente definido como fases na maioria dos modelos de ciclo de vida – planejamento, levantamento de requisitos, análise, projeto, implementação e testes, são definidas fases ortogonais a estas, a saber:

Concepção nesta fase, é estabelecido o escopo do projeto e suas fronteiras, determinando os principais casos de uso do sistema. Esses casos de uso devem ser elaborados com a precisão necessária para se proceder estimativas de prazos e custos. As estimativas devem ser globais para o projeto como um todo e detalhadas para a fase seguinte. Assim, a ênfase nesta etapa recai sobre o planejamento e, por conseguinte, é necessário levantar requisitos do sistema e preliminarmente analisá-los. Ao término dessa fase, são examinados os objetivos do projeto para se decidir sobre a continuidade do desenvolvimento;

Elaboração o propósito desta fase é analisar mais refinadamente o domínio do problema, estabelecer uma arquitetura de fundação sólida, desenvolver um plano de projeto para o sistema a ser construído e eliminar os elementos de projeto que oferecem maior risco. Embora o processo deva sempre acomodar alterações, as atividades da fase de elaboração asseguram que os requisitos, a arquitetura e os planos estão suficientemente estáveis e que os riscos estão suficientemente mitigados, de modo a se poder prever com precisão os custos e prazos para a conclusão do desenvolvimento.

Construção durante esta fase, um produto completo é desenvolvido de maneira iterativa e incremental, para que esteja pronto para a transição à comunidade usuária.

Transição nesta fase, o software é disponibilizado à comunidade usuária. Após o produto ter sido colocado em uso, naturalmente surgem novas considerações que vão demandar a construção de novas versões para permitir ajustes do sistema, corrigir problemas ou concluir algumas características que foram postergadas.

É importante realçar que dentro de cada fase, um conjunto de iterações, envolvendo planejamento, levantamento de requisitos, análise, projeto e implementação e testes, é realizado. Contudo, de uma iteração para outra e de uma fase para a próxima, a ênfase sobre as várias atividades muda, como mostra a figura 8.1, em que a cor preta indica grande ênfase, enquanto a cor branca indica muito pouca ênfase. Na fase de concepção, o foco principal recai sobre o entendimento dos requisitos e a determinação do escopo do projeto (planejamento e levantamento de requisitos). Na fase de elaboração, o enfoque está na captura e modelagem dos requisitos (levantamento de requisitos e análise), ainda que algum trabalho de projeto e implementação seja realizado para prototipar a arquitetura, evitando certos riscos técnicos. Na fase de construção, o enfoque concentra-se no projeto e na implementação, visando evoluir e recheiar o protótipo inicial, até obter o primeiro produto operacional. Finalmente, a fase de transição concentra-se nos testes, visando garantir que o sistema possui o nível adequado de qualidade. Além disso, usuários devem ser treinados, características ajustadas e elementos esquecidos adicionados.

	Levantamento de Requisitos	Análise	Projeto	Implementação	Testes
Concepção					
Elaboração					
Construção					
Transição					

Figura 8.1: A ênfase principal de cada uma das fases

Além dos conjuntos de iterações em cada fase, as fases em si podem ocorrer de forma iterativa. Como mostra a figura 8.2, elas não necessariamente têm a mesma duração. O esforço realizado em cada uma varia fortemente em função das circunstâncias específicas do projeto.

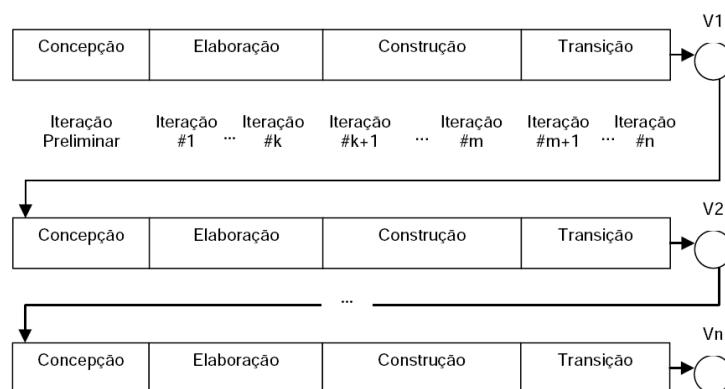


Figura 8.2: O modelo de ciclo de vida proposto no RUP

8.2 XP - Extreme Programming

A metodologia XP (*Extreme Programming*) é destinada a grupos pequenos de desenvolvimento, e em projetos de duração de até 36 meses. Os principais papéis nesta metodologia são: *Programador* - foco da metodologia; *Coach* - responsável por questões técnicas do projeto, maior conhecimento do processo, valores e práticas XP. Verifica o comportamento da equipe; *Tracker* - Coletar sinais vitais do projeto uma ou 2 vezes por semana e mantém todos informados; *Gerente* - responsável pelos assuntos administrativos, mantém um forte relacionamento com o cliente.

Entre as principais características da metodologia XP estão: (i) *Metáforas* - Utilização de comparações e analogias para facilitar entendimento; (ii) *Design Simples do Sistema*; (iii) *Testes Automatizados* - Testes de Unidade e de Aceitação; (iv) *Refactoring* - Todo desenvolvedor tem o dever de melhorar um código que esteja funcionando porém mal escrito; (v) *Programação de Dupla* - Todo código deve ser implementado em dupla; (vi) *Propriedade Coletiva do Código*; (vii) *Semana de 40 horas* - Evitar trabalhar a mais. Inicialmente se obtém resultados, mas depois há o desgaste; (viii) *Integração contínua* - Eliminar erros graves de integração; (ix) *Releases Curtos* - Release é um conjunto de funcionalidades bem definidas e que representam algum valor para o cliente. Um projeto XP pode ter um ou mais releases no seu ciclo; (x) *Story Cards* - Cartões escritos pelos usuários onde são descritas funcionalidades do sistema; (xi) *CRC* - Linguagem para Modelagem de Classes do XP. Utiliza os story cards.

8.3 Scrum

Scrum é um processo para construir software incrementalmente em ambientes complexos, onde os requisitos não são claros ou mudam com muita freqüência. Em Rugby, Scrum é um time de oito integrantes que trabalham em conjunto para levar a bola adiante no campo. Ou seja: times trabalhando como uma unidade altamente integrada com cada membro desempenhando um papel bem definido e o time inteiro focando num único objetivo.

O objetivo do Scrum é fornecer um processo conveniente para projetos e desenvolvimento orientado a objetos. A metodologia é baseada em princípios semelhantes aos de XP: equipes pequenas, requisitos pouco estáveis ou desconhecidos, e iterações curtas para promover visibilidade para o desenvolvimento. No entanto, as dimensões em Scrum diferem de XP.

Scrum divide o desenvolvimento em *sprints* de 30 dias. Equipes pequenas, de até 7 pessoas, são formadas por projetistas, programadores, engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidade (os requisitos, em outras palavras) definidas no início de cada *sprint*. A equipe toda é responsável pelo desenvolvimento desta funcionalidade.

Todo dia, é feita uma reunião de 15 minutos onde o time expõe à gerência o que será feito no próximo dia, e nestas reuniões os gerentes podem levantar os fatores de impedimento, e o progresso geral do desenvolvimento.

Scrum é interessante porque fornece um mecanismo de informação de status que é atualizado continuamente, e porque utiliza a divisão de tarefas dentro da equipe de forma explícita. Scrum e XP são complementares pois Scrum

provê práticas ágeis de gerenciamento enquanto XP provê práticas integradas de engenharia de software.

8.4 Crystal

Crystal/Clear faz parte, na realidade, de um conjunto de metodologias criado por Cockburn. As premissas apresentadas para a existência deste conjunto são:

- Todo projeto tem necessidades, convenções e uma metodologia diferentes.
- O funcionamento do projeto é influenciado por fatores humanos, e há melhora neste quando os indivíduos produzem melhor.
- Comunicação melhor e lançamentos freqüentes reduzem a necessidade de construir produtos intermediários do processo.

Crystal/Clear é uma metodologia direcionada a projetos pequenos, com equipes de até 6 desenvolvedores. Assim como com SCRUM, os membros da equipe tem especialidades distintas. Existe uma forte ênfase na comunicação entre os membros do grupo. Existem outras metodologias Crystal para grupos maiores.

Toda a especificação e projeto são feitos informalmente, utilizando quadros publicamente visíveis. Os requisitos são elaborados utilizando casos de uso, um conceito similar às estórias de usuário em XP, onde são enunciados os requisitos como tarefas e um processo para sua execução.

As entregas das novas versões de software são feitos em incrementos regulares de um mês, e existem alguns subprodutos do processo que são responsabilidade de membros específicos do projeto. Grande parte da metodologia é pouco definida, e segundo o autor, isto é proposital; a idéia de Crystal/Clear é permitir que cada organização implemente as atividades que lhe parecem adequadas, fornecendo um mínimo de suporte útil do ponto de vista de comunicação e documentos.

8.5 Feature Driven Development (FDD)

- Método ágil e adaptativo
- Foco nas fases de desenho e construção.
- Interage com outras metodologias.
- Não exige nenhum processo específico de modelagem.
- Possui desenvolvimento iterativo.
- Enfatiza aspectos de qualidade durante o processo e inclui entregas freqüentes e tangíveis.
- Superta desenvolvimento ágil com rápidas adaptações às mudanças de requisitos e necessidades do mercado.

O FDD consiste de 5 processos principais:

- Desenvolver um modelo comprehensível (Develop an overall model).
- Construir uma lista de funcionalidades (Build a features list).
- Planejar por funcionalidade (Plan By Feature).
- Projetar por funcionalidade (Design by feature).
- Construir por funcionalidade (Build by feature).

8.6 Dynamic Systems Development Method (DSDM)

Progenitor do XP. É um arcabouço para desenvolvimento rápido de aplicações (RAD). Fixa tempo e recursos ajustando a quantia de funcionalidades. Funciona com pequenas equipes. Suporta mudanças nos requisitos durante o ciclo de vida. Consiste em cinco fases:

- Estudo das possibilidades (Feasibility study).
- Estudo dos negócios (Business study).
- Iteração do modelo funcional (Functional model iteration).
- Iteração de projeto e construção (Design and build iteration).
- Implementação final (Final implementation).

Práticas:

- Usuário sempre envolvido.
- Equipe do DSDM autorizada a tomar decisões.
- Foco na frequente entrega de produtos.
- Adaptação ao negócio é o critério para entregas.
- "Construa o produto certo antes de você construí-lo corretamente".
- Desenvolvimento iterativo e incremental.
- Mudanças são reversíveis utilizando pequenas iterações.
- Requisitos são acompanhados em alto nível.
- Testes integrados ao ciclo de vida.

8.7 Adaptive Software Development (ASD)

É iterativo e incremental. Funciona bem em sistemas grandes e complexos. É Arcabouço para evitar o caos. O cliente deve estar sempre presente. É uma metodologia voltada para o desenvolvimento de aplicações em conjunto (Joint Application development - JAD).

Possui três ciclos de fase:

Especular (Speculate). Fixa prazos e objetivos e define um plano baseado em componentes.

Colaborar (Collaborate). Construção concorrente de vários componentes.

Aprender (Learn). Repetitivas revisões de qualidade e foco na demonstração das funcionalidades desenvolvidas (Learning loop). Há presença do cliente e de especialistas do domínio.

Os ciclos duram de 4 a 8 semanas. É orientado a missões (Mission-driven), pois atividades são justificadas através de uma missão, que pode mudar ao longo do projeto. É baseado em componentes e interativo. Os prazos são pré-fixados (time-boxed), pois força os participantes do projeto a definir severamente decisões do projeto. Uma propriedade interessante é a tolerância a mudanças (Change-tolerant). As mudanças são freqüentes, então é sempre melhor estar pronto a adaptá-las do que controlá-las. Isso é feito através de constante avaliação de quais componentes podem mudar.

Capítulo 9

Engenharia de Requisitos

9.1 O Processo de Engenharia de Requisitos

As descrições das funções que um sistema deve incorporar e das restrições que devem ser satisfeitas são os requisitos para o sistema. Isto é, os requisitos de um sistema definem o que o sistema deve fazer e as circunstâncias sob as quais deve operar. Em outras palavras, os requisitos definem os serviços que o sistema deve fornecer e dispõem sobre as restrições à operação do mesmo.

Requisitos são, normalmente, classificados em requisitos funcionais e não funcionais. Requisitos funcionais, como o próprio nome indica, apontam as funções que o sistema deve fornecer e como o sistema deve se comportar em determinadas situações. Já os requisitos não funcionais descrevem restrições sobre as funções oferecidas, tais como restrições de tempo, de uso de recursos etc. Alguns requisitos não funcionais dizem respeito ao sistema como um todo e não a funcionalidade específica. Dependendo da natureza, os requisitos não funcionais podem ser classificados de diferentes maneiras, tais como requisitos de desempenho, requisitos de portabilidade, requisitos legais, requisitos de conformidade etc.

Os processos de engenharia de requisitos variam muito de uma organização para outra, mas, de maneira geral, a maioria dos processos de Engenharia de Requisitos é composta das seguintes atividades:

Levantamento (ou Descoberta ou Elicitação) de Requisitos Nesta fase, os usuários, clientes e especialistas de domínio são identificados e trabalham junto com os engenheiros de requisitos para descobrir, articular e entender a organização como um todo, o domínio da aplicação, os processos de negócio específicos, as necessidades que o software deve atender e os problemas e deficiências dos sistemas atuais.

Análise de Requisitos visa a estabelecer um conjunto acordado de requisitos consistentes e sem ambigüidades, que possa ser usado como base para o desenvolvimento do software. Para tal, diversos tipos de modelos são construídos.

Documentação de Requisitos é a atividade de representar os resultados da Engenharia de Requisitos em um documento (ou conjunto de documentos), contendo os requisitos do software.

Verificação e Validação de Requisitos A verificação de requisitos avalia se o documento de Especificação de Requisitos está sendo construído de forma correta, de acordo com padrões previamente definidos, sem conter requisitos ambíguos, incompletos ou, ainda, requisitos incoerentes ou impossíveis de serem testados. Já a validação diz respeito a avaliar se esse documento está correto, ou seja, se os requisitos e modelos documentados atendem às reais necessidades e requisitos dos usuários / cliente.

Gerência de Requisitos se preocupa em gerenciar as mudanças nos requisitos já acordados, manter uma trilha dessas mudanças, gerenciar os relacionamentos entre os requisitos e as dependências entre o documento de requisitos e os demais artefatos produzidos no processo de software, de forma a garantir que mudanças nos requisitos sejam feitas de maneira controlada e documentada.

9.2 Técnicas de Levantamento de Requisitos

9.2.1 Observação

A observação possibilita um contato pessoal e estreito do pesquisador com o fenômeno pesquisado, o que apresenta uma série de vantagens. As técnicas de observação são extremamente úteis para "descobrir" aspectos novos de um problema. Isto se torna crucial nas situações em que não existe uma base teórica sólida que oriente a coleta de dados. Ao mesmo tempo em que o contato direto e prolongado do pesquisador com a situação pesquisada apresenta as vantagens mencionadas, envolve também uma série de problemas. Algumas críticas são feitas ao método de observação, primeiramente por provocar alterações no ambiente ou no comportamento das pessoas observadas. Outra crítica é a de que este método se baseia muito na interpretação pessoal. Além disso há críticas no sentido de que o grande envolvimento do pesquisador leve a uma visão distorcida do fenômeno ou a uma representação parcial da realidade.

9.2.2 Entrevista

Entrevista é uma técnica de elicitação de requisitos muito usada. O engenheiro de requisitos ou analista discute o sistema com diferentes usuários , a partir da qual elabora um entendimento de seus requisitos. Há, basicamente, 2 tipos de entrevista: a) entrevistas fechadas onde o engenheiro de requisitos procura as perguntas para um conjunto um pré-definido de questões; b) entrevistas abertas onde não há agenda prédefinida e o engenheiro de requisitos discute, de modo aberto, o que os usuários querem do sistema. Entrevistas podem ser efetivas para desenvolver um entendimento do problema e para eliciar muitos requisitos gerais do sistema. Usuários finais são usualmente felizes para descreverem seus trabalhos e as dificuldades que eles enfrentam de forma relativamente natural, entretanto eles podem ter expectativas não realistas sobre o suporte que o computador dará. Portanto, entrevistas são muito menos efetivas para entendimento do domínio da aplicação e para o entendimento das questões organizacionais as quais afetam os requisitos.

9.2.3 Análise de Protocolo

A análise de protocolo pede à pessoa se engajar em alguma tarefa e correntemente falar sobre esta tarefa, explicando o seu pensamento do processo. Usuários alegam que esse tipo de linguagem pode ser considerada uma "verbalização direta do processo cognitivo específico". A análise de protocolo não é um guia confiável sobre o que as pessoas estão pensando, ela está sujeita a problemas de interpretações pelos analistas. A restrição em estudar protocolos é que as pessoas podem produzir linguagens que oferece um perfil de atividade cognitiva autônoma. Segundo (Goguen, 1997), mesmo se fosse possível conseguir um perfil de atividade cognitiva autônoma da pessoa, tal objeto seria inapropriado para o processo de requisitos, porque o cliente não tem qualquer modelo mental pré-existente do sistema desejado. Os clientes têm conhecimento sobre negócios e necessidades organizacionais, enquanto o time de requisitos tem conhecimento sobre as possibilidade técnicas.

9.2.4 JAD

JAD é uma marca registrada da IBM. O tema principal do JAD é colocar autoridades representativas e gerenciais juntas dentro de um workshop estruturado para promover decisões. Segundo (Damian, 1997) JAD consiste de 5 fases: definição do projeto, pesquisa, preparação para a sessão JAD, a sessão JAD, o documento final. As fases de definição de projeto e pesquisa no processo JAD lidam com a coleta de informações. A sessão JAD é então usada para validar as informações recolhidas nas fases anteriores. O processo JAD concentrase na sessão JAD, e deste modo JAD contribui para a elicitação de requisitos como significado para validar as informações já colhidas. Na sessão JAD, as pessoas certas têm que estar envolvidas, e a presença de um facilitador pode ajudar a manter a sessão focalizada e minimizar ataques e defesas emocionais improdutivas. JAD promove cooperação, entendimento, e trabalho em grupo entre os vários grupos de usuários e o pessoal de sistemas de informação.

9.2.5 PD

Tradicionalmente valores democráticos, os quais tem sido levados em conta no processo de projeto, tem sido somente aqueles concentrados em fatores técnicos e econômicos. Mas com o uso do PD fatores técnicos-sociais tem sido levados em conta. O projeto deveria ser feito com o usuário. Aprendizado mútuo deveria ser uma parte importante do trabalho em um grupo de projeto, não sendo meramente uma visita aos laboratórios de pesquisa. Trabalhadores e clientes são inteligentes e criativos, contribuidores produtivos para as organizações, desde que sejam encorajados a expressarem seus desejos, aplicarem sua esperteza e exercitarem suas capacidades de tomar decisões, assumindo responsabilidades do impacto de suas ações.

9.2.6 QFD

O termo qualidade é definido como "um conjunto de meios para produzir economicamente produtos ou serviços, os quais satisfaçam os requisitos do cliente". QFD é "um conceito que provê meios de interpretar requisitos do cliente em requisitos técnicos, apropriados para cada estágio do desenvolvimento e produção

do produto” (Damian, 1997). As fases iniciais do QFD podem ser descritas como sendo ”simplesmente um sistema de identificação e priorização das necessidades do cliente obtidas de cada recurso avaliado”. QFD é um conceito que aplica-se bem para a elicitação de requisitos, especialmente num modelo de elicitação onde a voz do cliente é o guia para a criação de requisitos.

9.2.7 CRC

Como definido em (Zhu, 1998), CRC é uma sessão de grupo, que é similar ao JAD, onde os papéis dos participantes e o papel do facilitador são claramente definidos. Em CRC, participantes consistem não somente de usuários e facilitador, mas também de outras pessoas envolvidas indiretamente no sistema. CRC é diferente de JAD e QFD pois ele foca-se no usuário operativo.

9.2.8 Prototipação

Este método para elicitação de requisitos utiliza-se do uso de uma técnica de prototipação para a avaliação do usuário. O conjunto inicial de requisitos é usado como base para criar o Protótipo de Interface do Usuário com o sistema inicial (simplificado). Para essa criação, o projetista precisa manter o protótipo tão simples quanto possível. O ponto forte desta atividade é apresentar muitas alternativas para o usuário antes de se gastar muito esforço para qualquer protótipo em particular. Após a aceitação do protótipo pelos usuários, os desenvolvedores precisam criar um documento de especificação dos requisitos paralelo ao protótipo de interface (McConnel, 1998).

9.2.9 Cenários

Usuários finais e outros agentes do sistema acham a utilização de cenários mais fáceis para relacionar-se aos exemplos da vida real do que descrições abstratas das funções realizadas pelo sistema. Por essa razão, é freqüentemente útil desenvolver um conjunto de interação dos cenários, e usar estes para eliciar eclarear requisitos de sistema. Cenários são exemplos de sessões de interação as quais são concentradas com um tipo único de interação entre um usuário final e o sistema. Usuários finais simulam suas interações usando cenários. Eles explicam para o time de engenheiros de requisito o que eles estão fazendo e a informação da qual eles precisam do sistema para descrever a tarefa descrita no cenário.

9.2.10 FAST

Técnicas facilitadas de especificação de aplicações (facilitated application specification techniques). Consiste nos seguintes passos:

1. Reuniões iniciais entre o cliente e o desenvolvedor (sessão de perguntas e respostas). Produzem como resultado a *solicitação de produto*.
2. São selecionados local, horário e data para o FAST e é escolhido um facilitador.
3. A solicitação de produto é distribuída para todos os participantes antes da data da reunião.

4. É solicitado a cada participante que faça uma lista dos objetos que são parte do ambiente, uma lista de serviços (processos ou funções), uma lista de restrições (custo, tamanho, regras de negócio) e critérios de desempenho.
5. Quando a reunião a FAST começa, o primeiro tópico de discussão é a necessidade e a justificativa do produto.
6. Cada um apresenta suas listas, críticas e debates são proibidos nesta etapa.
7. Uma lista combinada que elimina objetos redundantes é produzida, mas não apaga nada.
8. A lista é modificada com uma discussão dirigida pelo facilitador. O produto é a lista de consenso.
9. A equipe é subdividida em equipes menores; cada uma trabalha para produzir *miniespecificações*, ou seja, detalham cada objeto da lista de consenso.
10. As equipes apresentam suas miniespecificações e a lista é modificada novamente através de discussões.
11. Cada participante da reunião cria os critérios de validação para os elementos da lista.
12. Uma lista de consenso de critérios de validação é criada.
13. Alguém fica responsável em redigir o rascunho completo das especificação.

9.3 Análise de Requisitos

A análise de requisitos é uma atividade de engenharia de software que vence o espeço entre a engenharia de requisitos, no nível de sistema, e o projeto de software. As atividades de engenharia de requisitos resultam na especificação das características operacionais do software (função, dados e comportamento). A análise de requisitos constrói modelos de domínio que fornecem ao projetista uma visão do software.

A *especificação de requisitos* é produzida no auge da tarefa de análise. Alguns padrões sugerem que uma especificação de requisitos deve conter:

Introdução. Declara as metas e objetivos do software.

Descrição da informação. Fornece uma descrição detalhada do problema que o software deve resolver. O conteúdo, fluxo e estrutura da informação são documentados.

Descrição funcional. Uma narrativa de processamento é fornecida para cada função.

Descrição comportamental. Examina a operação do software como consequência de eventos externos e características de controle geradas internamente.

Critérios de validação. Informa como devemos reconhecer uma implementação bem-sucedida e quais classes de testes devem ser conduzidas para validar a função, o desempenho e as restrições.

Bibliografia e apêndice. A bibliografia contém todos os documentos que se relacionam ao software. O apêndice contém informações que complementa as especificações.

9.3.1 Métodos de análise

Entre os mais importantes métodos de análise, três se destacam: A análise estruturada, a análise essencial e a análise orientada a objetos. A seguir discutiremos suas principais características.

Análise estruturada

A Análise Estruturada adota uma visão de desenvolvimento baseada em um modelo entrada-processamento-saída. No paradigma estruturado, os dados são considerados separadamente das funções que os transformam e a decomposição funcional é usada intensamente.

Análise essencial

Os métodos orientados a funções conduzem o desenvolvimento de software estruturando as aplicações segundo a ótica das funções (ações) que o sistema deverá realizar. Os métodos orientados a dados, por sua vez, enfatizam a identificação e estruturação dos dados, subjugando a análise das funções para um segundo plano. Esses métodos têm origem no projeto de bancos de dados e, geralmente, têm no modelo de Entidades e Relacionamentos (ER) sua principal ferramenta.

A *análise essencial* é uma metodologia estruturada, sendo uma evolução da análise estruturada. Essa metodologia procurou conciliar as abordagens orientadas a dados e a funções em um único método, utilizando modelos para dados, funções e controles (DFDs, DERs e Diagramas de Transição de Estados, respectivamente) como ferramentas para a modelagem de sistemas.

Análise orientada o objeto

A orientação a objetos pressupõe que o mundo é composto por objetos, onde um objeto é uma entidade que combina estrutura de dados e comportamento funcional. No paradigma orientado a objetos, os sistemas são estruturados a partir dos objetos que existem no domínio do problema, isto é, os sistemas são modelados como um número de objetos que interagem.

9.3.2 Modelagem da análise

DFD's

Ferramenta de modelagem utilizada para descrever a transformação de entradas em saídas. Representa um sistema como uma rede de processos interligados entre si por fluxos de dados e depósitos de dados. É a ferramenta de modelagem mais importante da análise estruturada. Junto com o DER, é a ferramenta mais importante da análise essencial. Seus componentes são:

Processo. Também chamado de função, bolha ou transformação, o processo representa as transformações realizadas sobre os dados em um sistema e correspondem a funções ou procedimentos que um sistema tem que prover. É representado graficamente por círculos e deve ter um nome composto por um verbo e um objeto.

Fluxo de dados. Representa uma coleção de itens de dados em movimento. É representado graficamente através de uma seta entrando ou saindo de um processo, com a ponta indicando o sentido do fluxo. Deve ter associado um nome o mais significativo possível, de modo a facilitar a validação do diagrama com os usuários. Esse nome deve ser um substantivo que facilite a identificação da coleção de dados transportada.

Depósito de dados. O depósito de dados é utilizado para modelar um repositório de uma coleção de pacotes de dados em repouso. É representado graficamente por duas linhas paralelas com um nome que represente seu conteúdo e que normalmente é o plural do nome dos pacotes transportados pelos fluxos para dentro e para fora do depósito.

Terminador ou entidade externa. Entidades externas ou terminadores são fontes ou destinos de dados do sistema. Representam os elementos do ambiente com os quais o sistema se comunica. Tipicamente, uma entidade externa é uma pessoa (p.ex. um cliente), um grupo de pessoas (p. ex. um departamento da empresa ou outras instituições) ou um outro sistema que interaja com o sistema em questão. Uma entidade externa deve ser identificada por um nome e representada por um retângulo. Assim como no caso dos depósitos de dados, em diagramas complexos, podemos desenhar um mesmo terminador mais de uma vez, para se evitar o cruzamento de linhas de fluxos de dados ou para impedir que longas linhas de fluxos de dados saiam de um lado a outro do diagrama.

Quando o software é complexo, devemos organizar o DFD em níveis. Dois DFDs especiais estão presentes em qualquer modelo funcional:

Contexto. Mostra o sistema como uma caixa-preta, trocando informações (fluxos de dados) com entidades externas ao sistema.

Nível 0 - Geral ou de Sistema. É uma decomposição do diagrama de contexto, mostrando o funcionamento do sistema em questão, isto é, as grandes funções do sistema e as interfaces entre elas. Os processos nesse diagrama recebem os números 1, 2, 3, etc... É necessário assegurar a coerência entre os diagramas de contexto e de nível 0, isto é, assegurar que os fluxos de dados entrando ou saindo do diagrama efetivamente reproduzem as entradas e saídas do diagrama de contexto. Neste diagrama, devem aparecer os depósitos de dados necessários para a sincronização dos processos.

Dicionário de Dados

O *dicionário de dados* é uma listagem organizada de todos os elementos de dados pertinentes ao sistema, com definições precisas para que os usuários e desenvolvedores possam conhecer o significado de todos os itens de dados. Exemplificando para uma estrutura de dados "nome": nome = título-cortesia +

primeiro-nome + (nome-intermediário) + último-nome, título-cortesia = [Sr. — Sra. — Srta. — Dr. — Professor] etc...

Aspectos de análise essencial

O modelo comportamental determina o comportamento interno do sistema para que este possa interagir corretamente com o ambiente. Após a construção do modelo comportamental, teremos os seguintes artefatos:

Diagrama de Entidades e Relacionamentos.

Diagramas de Fluxos de Dados Preliminar (DFD Particionado por Eventos).

Para cada evento do sistema, deve ser construído um DFD.

Diagramas de Fluxos de Dados Organizados em Níveis Hierárquicos.

Representa os processos em níveis hierárquicos, a partir do diagrama zero.

Diagramas de Transição de Estados. Representa o comportamento das entidades e relacionamentos com atributos ao longo do tempo. Será construído um DTE para cada entidade ou relacionamento com atributo do DER que possuir comportamento significativo, isto é, possuir mais de um estado ao longo de seu ciclo de vida.

Dicionário de Dados. Descreve os dados representados no DER, nos DFDs e nos DTEs.

Especificação da Lógica dos Processos (Mini-especificações). Descreve a lógica dos processos do DFD que não foram detalhados em diagramas de nível inferior (lógica dos processos primitivos).

Como podemos perceber, a análise essencial faz uso praticamente das mesmas técnicas de modelagem da análise estruturada, a saber a modelagem de dados (utilizando modelos de entidades e relacionamentos), a modelagem funcional (utilizando Diagramas de Fluxo de Dados - DFDs) e a modelagem de controle (utilizando diagramas de transição de estados). Isso é bastante natural, já que a análise essencial é, de fato, uma extensão da análise estruturada. Na realidade, a principal diferença entre a análise essencial e a análise estruturada está na estratégia para atacar o problema: a primeira defende uma abordagem baseada em eventos, onde a análise de eventos passa a ser um passo fundamental, a segunda é baseada apenas na decomposição top-down da funcionalidade do sistema.

9.4 Gerenciamento de Requisitos

Gerenciar requisitos consiste de procedimentos para coletar, verificar e avaliar mudanças, de forma melhor administrá-las. É o considerado uma das primeiras áreas chave para melhoria de qualidade. No gerenciamento de requisitos um conceito importante é o de *rastreabilidade*. A rastreabilidade pode ser vista como a habilidade de acompanhar e descrever a vida de um requisito em ambas as direções. Rastrear envolve identificar links entre requisitos, fontes dos requisitos e design do sistema. As ferramentas mais utilizadas para rastreamento de

requisitos são as *Matrizes de Rastreabilidade (Traceability Matrix)*, que usualmente correlacionam requisitos com casos de testes, e técnicas de *Referências Cruzadas*. Entre os produtos de mercado utilizadas para realizar gerenciamento e rastreamento de requisitos podemos citar o *Rational RequisitePro*, *Caliber-RM*, *RTS* e *RDT*.

Capítulo 10

Métricas

Métricas devem ser coletadas de modo que os indicadores de processo e de produto possam ser determinados. *Indicadores de processo* permitem à organização de engenharia de software ter idéia da eficácia de um processo existente (o paradigma, as tarefas de engenharia de software, os produtos de trabalho e os marcos de tempo). Já os *indicadores de projeto* permitem ao gerente de projeto de software:

- Avaliar o *status* de um projeto em andamento.
- Acompanhar riscos potenciais.
- Descobrir áreas-problemas antes que elas se tornem críticas.
- Ajustar fluxo de trabalho ou tarefas.
- Avaliar a capacidade da equipe de projeto de controlar a qualidade dos produtos do trabalho de software.

Em alguns casos, as mesmas métricas podem ser usadas para determinar indicadores de projeto e de processo.

10.1 Métricas de processo e aperfeiçoamento de processo de software

Nós medimos a eficácia de um processo de software indiretamente. Há usos públicos e privados de diferentes tipos de dados do processo. As métricas coletadas com base individual devem ser privadas e servir como indicador apenas para o indivíduo. Exemplo de métricas privadas incluem proporção de defeitos (por indivíduo e por módulo) e erros encontrados durante o desenvolvimento. Métricas públicas geralmente assimilam informações, que eram originalmente privadas. Porporções de defeitos de projeto (absolutamente não atribuíveis a um certo indivíduo), esforço, tempo transcorrido e dados relacionados são coletados e avaliados numa tentativa de descobrir indicadores, que possam aperfeiçoar o desempenho do processo organizacional.

Há uma sutil diferença entre erros e defeitos. Um defeito ocorre quando atividades de garantia de qualidade deixam de descobrir um erro num produto produzido durante o processo de software.

À medida que uma organização sente-se mais confortável, coletanto e usando métricas de processo de software, a derivação de indicadores simples dá lugar a uma abordagem mais rigorosa, chamada *melhoria estatística do processo de software (statistical software process improvement, SPPI)*. Essencialmente, a SSPI usa a análise de falhas de software para coletar informação sobre todos erros e defeitos. A análise de falhas funciona da seguinte maneira:

1. Todos os erros e defeitos são categorizados por origem (falha de especificação, falha de lógica, não atendimento a padrões).
2. O custo para corrigir cada erro e defeito é registrado.
3. A quantidade de erros e defeitos de cada categoria é contada e ordenada de forma decrescente.
4. O custo total de erros e defeitos de cada categoria é calculado.
5. Os dados resultantes são analisados, para descobrir as categorias que produzem um maior custo para a organização.
6. São desenvolvidos planos para modificar o processo, com o objetivo de eliminar (ou reduzir a freqüência das) classes de erros e defeitos que são mais dispendiosas.

Para a organização dessas métricas são utilizados o gráfico de pizza que associa os erros e defeitos às suas origens e o *diagrama espinha de peixe*.

10.2 Métricas de projeto

As métricas de projeto são usadas para evitar atrasos, avaliar a qualidade e não ultrapassar os custos permitidos. A primeira aplicação das métricas de projeto, na maioria dos projetos de software, ocorre durante a estimativa. Métricas coletadas de projetos anteriores são usadas como base e à medida que o projeto prossegue, medidas de esforço e de tempo são comparadas com as estimativas originais. Dessa maneira, o gerente do projeto monitora e controla o progresso.

À medida que o trabalho técnico se inicia, outras métricas de projeto começam a ter importância. A taxa de produção, representada em termos de páginas de documentação, horas de revisão, pontos por função e linhas de código fonte entregue, é medida. À medida que o software evolui, métricas técnicas são coletadas para avaliar a qualidade do projeto.

10.3 Medição de software

As medições são categorizadas em medidas diretas e indiretas. As *medidas diretas* do processo de engenharia de software incluem custo, linhas de código (*lines of code*, LOC) produzidas, velocidade de execução, tamanho de memória e defeitos relatados durante um certo período. *Medidas indiretas* do produto incluem funcionalidade, qualidade, complexidade, eficiência, confiabilidade, manutenibilidade e muitas outras habilidades.

10.3.1 Métricas orientadas a tamanho

São métricas baseadas em linhas de código. Métricas orientadas a tamanho não são universalmente aceitas como o melhor modo de medir o processo de desenvolvimento de software. Os opositores argumentam que essas métricas penalizam programas curtos e bem feitos e que medidas de LOC são dependentes da linguagem de programação. Por outro lado, existem muitos modelos de estimativas que usam LOC ou KLOC como entrada chave e essas métricas podem ser facilmente coletadas. As métricas orientadas a tamanho mais utilizadas incluem:

- Erros por KLOC (milhares de linha código).
- Defeitos por KLOC.
- \$ por LOC.
- Páginas de documentação por KLOC.
- Erros por pessoa-mês.
- LOC por pessoa-mês.
- \$ por página de documentação.

10.3.2 Métricas orientadas a função

Métricas de software orientadas a função usam uma medida da funcionalidade entregue pela aplicação como valor de normalização.

Pontos por função são calculados completando a tabela 10.1. Cinco características do domínio da informação são determinadas e as contagens são registradas na tabela. Essas características são definidas da seguinte maneira:

Quantidade de entradas do usuário. São contadas as entradas do usuário que fornecem dados distintos.

Quantidade de saídas do usuário. São contados telas, relatórios, mensagens de erros etc.

Número de consultas do usuário. São contadas consultas distintas. Uma consulta é definida como um entrada que resulta na geração de alguma resposta imediata.

Quantidade de arquivos. Cada grupo de dados lógico (que pode ser parte de uma base de dados maior ou um arquivo separado) é contado. Esses grupos de dados também são conhecidos como arquivos mestres lógicos.

Quantidade de interfaces externas. Todas as interfaces em linguagem de máquina (p. ex., arquivos de dados em um meio de armazenamento), que são usadas para transmitir informação a outro sistema são contadas.

Uma vez coletados esses dados, um valor de complexidade é associado com cada contagem. Para contar pontos por função (*function points*, FP), é usada a seguinte relação:

$$FP = \text{total da contagem} \times [0,65 + 0,01 \times \sum (F_i)]$$

Os F_i são valores de ajuste de complexidade, baseados nas respostas das seguintes perguntas:

Parâmetro de medição	Contagem	Simples	Médio	Complexo	Total
Quantidade de entradas do usuário	x	3	4	6	
Quantidade de saídas do usuário	x	4	5	7	
Quantidade de consultas do usuário	x	3	4	6	
Número de arquivos	x	7	10	15	
Quantidade de interfaces externas	x	5	7	10	

Tabela 10.1: Tabelas das características dos pontos de função

1. O sistema requer salvamento (*backup*) e recuperação (*recovery*)?
2. Comunicações de dados são necessárias?
3. Há funções de processamento distribuído?
4. O desempenho é crítico?
5. O sistema vai ser executado em um ambiente operacional existente, intensamente utilizado?
6. O sistema requer entradas de dados on-line?
7. A entrada de dados on-line exige que a transação de entrada seja construída através de várias telas ou operações?
8. Os arquivos mestre são atualizados on-line?
9. As entradas, saídas, arquivos ou consultas são complexas?
10. O processamento interno é complexo?
11. O código é projetado para ser reusado?
12. A conversão e a instalação estão incluídas no projeto?
13. O sistema está projetado para instalações múltiplas em diferentes organizações?
14. A aplicação está projetada para facilitar modificações e para facilidade de uso pelo usuário?

Cada uma dessas questões é respondida usando uma escala que varia entre 0 a 5. Os valores constantes e os fatores de peso podem ser ajustado empiricamente para a equação dos pontos de função. São exemplos importantes de medidas de produtividade, qualidade e outros atributos de software as métricas:

- Erros por FP.
- Defeitos por FP.
- \$ por FP.
- Páginas de documentação por FP.
- FP por pessoa-mês.

10.3.3 Métricas de pontos por função estendidas

A medida de pontos por função foi originalmente projetada para ser usada em aplicações de sistemas de informação comerciais. A medida de pontos por função ficou inadeguada para muitos sistemas embutidos e de engenharia (que enfatizam função e controle).

Uma extensão de pontos por função, chamada *pontos por característica* utilizada em aplicações de software de sistemas. Para calcular pontos por característica, valores de domínio de informação são novamente contados e ponderados. Além disso, a métrica pontos por característica trata uma nova característica do software: *algoritmos*.

Outra extensão de pontos por função, para sistemas em tempo real e produtos de engenharia, foi desenvolvida pela Boeing. Essa abordagem integra a dimensão de dados do software com as dimensões funcional e de controle para fornecer uma medida orientada a função, adequada a aplicações que enfatizam as capacidades de função e controle. Essa extensão é chamada de *pontos por função 3D*. As três dimensões são:

Dimensão de dados. Semelhante a contagem de pontos por função tradicional.

Dimensão funcional. Considera a quantidade de operações internas necessárias para transformar dados de entrada para transformar dados de entrada em saída.

Dimensão de controle. É medida contando a quantidade de transições entre estados.

10.4 Métricas de qualidade de software

A medição da qualidade de software é uma atividade de natureza subjetiva. Qualidade de software é uma mistura complexa de fatores que vão variar com cada aplicação diferente e com os clientes que as encomendam. As principais medidas são mostradas a seguir:

Correção. A medida mais comum é defeitos por KLOC. Para fins de avaliação de qualidade, defeitos são contados durante um período padrão, tipicamente um ano.

Manutenibilidade. Uma métrica utilizada é o *tempo médio para modicação* (*mean-time-to-change, MTTC*), que é o tempo despendido para analisar o pedido de modificação, projetar uma modificação adequada, implementar a modificação, testá-la e distribuí-la para todos os usuários. Outra métrica é chamada *prejuízo* - custo para corrigir defeitos encontrados depois que o software foi entregue a seus usuários finais.

Integridade. É a capacidade do sistema resistir à ataques (tanto acidentais quanto intencionais). *Ameaça* é a probabilidade de um ataque ocorrer dentro de um certo período. *Segurança*. É a probabilidade de um ataque ser repelido. A integridade do sistema pode ser então definida como *integridade* = $\sum (1 - \text{ameaca}) \times (1 - \text{segurança})$.

Utilização. Se é amigável ao uso. Pode ser medida em quatro tópicos: (1) aptidão física ou intelectual necessária para aprender a lidar com o sistema, (2) o tempo necessário para se tornar moderadamente eficiente no uso do sistema, (3) o aumento líquido de produtividade e (4) uma avaliação subjetiva das atitudes dos usuários com relação ao sistema.

Outra medição muito utilizada é a eficiência na remoção de defeitos (*defect removal efficiency, DRE*). A DRE é uma medida da capacidade de filtragem das atividades de controle e garantia de qualidade de software, à medida que são aplicadas. A fórmula utilizada é $DRE = E/(E + D)$. Onde E é a quantidade de erros encontrados e D a quantidade de defeitos.

As principais formas de classificar fatores de qualidade são apresentadas nas subseções a seguir.

10.4.1 Fatores de qualidade de McCall

Os fatores de qualidade de McCall concentram-se nos três aspectos importantes de um produto de software: suas características operacionais, sua habilidade de passar por modificações e sua adaptabilidade a novos ambientes. A seguir são listados esses aspectos e os seus respectivos fatores:

Operação do produto. Correção, confiabilidade, utilização, integridade e eficiência.

Revisão do produto. Mantenibilidade, flexibilidade e testabilidade.

Transição do produto. Portabilidade, reutilização e interoperabilidade.

É difícil desenvolver medidas diretas desses fatores. Assim, um conjunto de métricas é definido e usado para desenvolver expressões para cada um dos fatores de acordo com a seguinte relação: $F_q = c_1 \times m_1 + c_2 \times m_2 \dots + c_n \times m_n$, em que F_q é um fator de qualidade de software, c_n são coeficientes de regressão, m_n são as métricas que afetam o fator de qualidade. Infelizmente, muitas das métricas definidas podem ser medidas apenas sunjetivamente. As métricas podem estar em forma de *checklist* que é usada para atribuir um grau a atributos específicos do software. O peso atribuído a cada métrica é dependente de produtos e preocupações locais.

10.4.2 FURPS

A Hewlett-Packard desenvolveu um conjunto de fatores de qualidade que recebeu a sigla FURPS: funcionalidade, utilização, confiabilidade, desempenho e suportabilidade. Esses fatores podem ser definidos da seguinte maneira:

Funcionalidade. Avaliada pelo conjunto de características e capacidades do programa.

Utilização. Avaliada considerando fatores humanos como estética, consistência e documentação.

Confiabilidade. Avaliada medindo a freqüência e a severidade das falhas, a precisão dos resultados de saída, o tempo médio entre falhas (MTTF), a capacidade de recuperação de falhas e previsibilidade do programa.

Desempenho. Medido pela velocidade de processamento, tempo de resposta, consumo de recursos, *throughput* e eficiência.

Suportabilidade. Combina a capacidade de estender o programa (estensibilidade), adaptabilidade e reparabilidade.

10.4.3 ISO 9126

A norma da ISO identifica seis atributos-chave de qualidade:

Funcionalidade. Grau com que o software satisfaz as necessidades declaradas com os subatributos - adequabilidade, precisão, interoperabilidade, atendibilidade e segurança.

Confiabilidade. Período de tempo em que o software está disponível para uso com os subatributos - maturidade, tolerância a falha e recuperabilidade

Utilização. Grau em que o software é fácil de usar com os subatributos - inteligibilidade, adestrabilidade e operabilidade.

Eficiência. Grau em que o software faz uso otimizado dos recursos do sistema com os subatributos - comportamento em relação ao tempo e comportamento em relação aos recursos.

Mantenabilidade. Facilidade com a qual podem ser feitos reparos no software com os subatributos - analisabilidade, mutabilidade, estabilidade e testabilidade.

Portabilidade. Facilidade com a qual o software pode ser transportado de um ambiente para outro com os subatributos - adaptabilidade, instabilidade, conformidade e permutabilidade.

Esses fatores não necessariamente se prestam a medidas diretas. No entanto, fornecem de fato uma base valiosa para medidas indiretas e uma excelente lista de verificação para avaliar a qualidade de um sistema.

10.5 Estimativas

Muitas vezes, as estimativas são feitas usando-se a experiência passada como único guia. Mas se um novo projeto for totalmente diferente dos projetos realizados até o momento? Assim, apenas a experiência passada talvez não seja suficiente. Várias técnicas de estimativa estão disponíveis. Embora cada uma tenha suas particularidades, todas têm os seguintes atributos:

1. O escopo do projeto deve estar estabelecido.
2. Métricas de software são utilizadas e o histórico é usado como uma base a partir da qual estimativas são feitas.
3. O projeto é dividido em pequenas partes que são estimadas individualmente.

10.5.1 COCOMO (Constructive Cost Model)

O COCOMO é o modelo de estimativa empírica mais utilizado na indústria. Existem três modelos neste método:

COCOMO Básico. É um modelo estático de valor simples que computa o esforço (e custo) de desenvolvimento de software como uma função do tamanho de programa expresso em linha de código estimadas.

COCOMO Intermediário. Computa o esforço de desenvolvimento de software como uma função do tamanho do programa e de um conjunto de direcionadores de custo que incluem avaliações subjetivas do produto, do hardware, do pessoal e dos atributos do projeto.

COCOMO Avançado. Incorpora todas as características da versão intermediária, com uma avaliação do impacto dos direcionadores de custo sobre cada passo (análise, projeto, etc.) do processo de engenharia de software.

O COCOMO classifica os projetos em três tipos:

Modelo orgânico (ou convencional). Projetos de software simples, relativamente pequenos, nos quais pequenas equipes com boa experiência em aplicações trabalham num conjunto de requisitos não tão rígidos. Outras características: ambiente estável de desenvolvimento, algoritmos simples, prêmio relativamente baixo para término antes do prazo, tamanho relativamente pequeno, projetos na faixa de 50.000 linhas de código.

Modelo semidestacado (ou difuso). Projeto de software intermediário (em tamanho e complexidade) onde a equipe mescla grande e pouca experiência com aplicações, grande e pouca experiência com a tecnologia, o tamanho dos software varia até 300.000 linhas de código.

Modelo embutido (ou restrito). Um projeto que deve ser desenvolvido dentro de um conjunto rígido de restrições operacionais, de hardware e de software.

O COCOMO II, assim como o seu predecessor é na verdade uma hierarquia de modelos que trata das seguintes áreas:

Modelo de composição de aplicação. Usado durante os primeiros estágios do processo.

Modelo do primeiro estágio de projeto. Usado após os requisitos terem sido estabilizados e a arquitetura básica do software ter sido estabelecida.

Modelo para o estágio após a arquitetura. Usado durante a construção do software.

O COCOMO II usa pontos por objeto. Como pontos por função, o *pontos por objeto* é uma medida indireta de software que é calculada usando-se a contagem da quantidade de telas (na interface com o usuário, relatórios e componentes. Cada instância de objeto em um dos três níveis de complexidade (simples, médio, difícil). Essencialmente, a complexidade é função da quantidade e fonte das tabelas de dados do cliente e servidores, que são necessárias para gerar a tela ou

relatório, e da quantidade de vistas ou seções apresentadas como parte da tela ou relatório.

Deve-se notar que outros modelos de estimativa mais sofisticados (usando FP e KLOC) também estão disponíveis como parte do COCOMO II.

Capítulo 11

Testes

Teste é um processo de execução de um programa com a finalidade de encontrar um erro. Os testes podem ser planejados e projetados antes que qualquer código tenha sido gerado. Os primeiros testes planejados e executados geralmente concentram-se nos componentes individuais. À medida que o teste progride, o foco se desloca numa tentativa de encontrar erros em conjuntos integrados de componente e finalmente em todo o sistema.

A *testabilidade* de software é a facilidade com que o programa pode ser testado. Existem métricas que podem medir a *testabilidade* do software.

Quando o software para computador é considerado, *teste caixa-preta* refere-se a testes que são conduzidos na interface do software.

Um *teste de caixa-branca* é baseado num exame rigoroso do detalhe procedural. Caminhos lógicos internos do software são testados, definindo casos de testes que exercitam conjuntos específicos de condições ou ciclos. Teste caixa-branca completo é impossível para grandes sistemas de software. Um teste caixa-branca não deve, no entanto, ser descartado como não prático. Um número limitado de caminhos lógicos importantes pode ser selecionado e exercitado. Há três motivos importantes pelos quais se deve usar testes caixa-branca:

- Os erros tendem a ocorrer em condições ou controle que estão fora da função principal.
- Freqüentemente acreditamos que um caminho lógico não é provável de ser executado quando, na realidade, ele pode ser executado em base regular.
- É provável que um erro tipográfico exista tanto num caminho lógico obscuro quanto num caminho principal.

11.1 Teste de caminho básico

Teste de caminho básico é uma técnica de teste caixa-branca proposta inicialmente por Tom McCabe. O método de caminho básico permite ao projetista de casos de teste originar uma medida da complexidade lógica de um projeto procedural e usar essa medida como guia para definir um conjunto básico de caminhos de execução. Casos de testes derivados para exercitar o conjunto básico executam garantidamente cada comando programa pelo menos uma vez durante o teste.

O grafo de fluxo mostra o fluxo de controle lógico. Cada nó do grafo de fluxo representa um ou mais comando procedimentais. As arestas representam o fluxo de controle. As áreas delimitadas por arestas e nós são chamadas *regiões*. Cada nó que contém uma condição é chamado de *nó predicado* e é caracterizada por duas ou mais arestas saindo dele.

Complexidade ciclomática é a métrica de software que fornece uma medida quantitativa da complexidade lógica do programa. O valor calculado para a complexidade ciclomática define o número de caminhos independentes no conjunto base de um programa e nos fornece um limite superior para a quantidade de testes que deve ser conduzida para garantir que todos os comandos tenham sido executados pelo menos uma vez.

Um *caminho independente* é qualquer caminho ao longo do programa que introduz pelo menos um novo conjunto de comandos de processamento ou uma nova condição. Um *conjunto-base* é um conjunto de caminhos em que todo comando do programa terá sido garantidamente executado pelo menos uma vez e cada condição terá sido executada no seu lado verdadeiro e no seu lado falso.

A complexidade ciclomática é calculada de três maneiras:

1. O número de regiões do grafo de fluxo.
2. $V(G) = E - N + 2$ em que E é o número de arestas e N o número de nós.
3. $V(G) = P + 1$ em que P é o número de nós predicados.

Os seguintes passos podem ser aplicados para derivação de casos de teste:

1. Usando o projeto ou código como base, desenhe o grafo de fluxo correspondente.
2. Determine a complexidade ciclomática do grafo de fluxo resultante.
3. Determine um conjunto-base de caminhos linearmente independentes.
4. Prepare casos de teste que vão forçar a execução de cada caminho do conjunto-base.

Uma *matriz de grafo* é uma matriz quadrada, cujo número de colunas e linhas é igual ao número de nós do grafo de fluxo. Adicionando um *peso de ligação* a cada entrada da matriz, a matriz de grafo pode tornar-se uma possante ferramenta para avaliar a estrutura de controle do programa durante o teste. Em sua forma mais simples, o peso da ligação é 1 (existe uma conexão) ou 0 (não existe uma conexão). Outros pesos podem ser atribuídos com outras propriedades:

- A probabilidade de que uma ligação (aresta) será executada.
- O tempo de processamento gasto durante o percurso de uma ligação.
- A memória necessária durante o percurso de uma ligação.
- Os recursos necessários durante o percurso de uma ligação.

Existem algoritmos que podem ser aplicados às matrizes de grafo. Usando essas técnicas, a análise necessária para projetar casos de teste pode ser parcial ou totalmente automatizada.

11.2 Teste de estrutura de controle

A técnica de teste de estrutura de controle ampliam a cobertura da técnica de teste de caminho básico e melhoram a qualidade do teste caixa-branca.

11.2.1 Teste de condição

Teste de condição é um método de projeto de caso de teste que exercita as condições lógicas contidas em um módulo de programa. Uma *condição simples* é uma variável booleana ou uma expressão relacional, possivelmente precedida por um operador de negação. Uma *condição composta* é composta de duas ou mais condições simples, operadores booleanos e parênteses.

Diversas estratégias para teste de condição têm sido propostas. O *teste de desvio* é provavelmente a estratégia de teste de condição mais simples. Para uma condição composta C , os ramos verdadeiro e falso de C e cada condição simples de C precisam ser executadas pelo menos uma vez.

O *teste de domínio* requer que três ou quatro testes sejam derivados para uma expressão relacional. Para uma expressão relacional da forma

$$E_1 < \text{operador-relacional} > E_2$$

três testes são necessários para tornar o valor de E_1 maior que, igual a, ou menor que o de E_2 .

Para uma expressão booleana com n variáveis, todos os 2^n possíveis testes são necessários ($n > 0$), mas é prática somente quando n é pequeno.

Caso uma expressão booleana seja singular (na qual cada variável ocorre apenas uma vez), há como gerar um conjunto de testes com menos do que 2^n testes tal que esse conjunto de testes garante a detecção de erros de operadores booleanos múltiplos e é também efetivo para detectar outros erros. Uma delas é a técnica de teste de desvio e operador relacional (*branch and relational operator*, BRO). Ela usa restrições de condição para uma condição C . Uma restrição de condição para C com n condições simples é definida como (D_1, D_2, \dots, D_n) , em que $D_i (0 < i \leq n)$ é um símbolo que especifica uma restrição sobre o resultado da i -ésima condição simples da condição C . Diz-se que uma restrição de condição D para uma condição C é coberta por uma execução de C se, durante a execução de C , o resultado de cada condição simples de C satisfaz a correspondente restrição em D . Não entendeu nada? Como exemplo, considere a condição:

$$C_1 : B_1 \& B_2$$

em que B_1 e B_2 são variáveis booleanas. A restrição de condição para C_1 é da forma (D_1, D_2) , em que cada um de D_1 e D_2 é t ou f . O valor (t, f) é uma restrição de condição para C_1 . A estratégia de teste BRO exige que o conjunto de restrições $\{(t, t), (t, f), (f, t)\}$ seja coberto pelas execuções de C_1 . Se C_1 está incorreto, pelo menos uma das condições do conjunto vai fazer C_1 falhar. Outro exemplo: $C_2 : B_1 \& (E_3 = E_4)$ possui o conjunto de restrições $\{(t, =), (f, =), (t, <), (t, >)\}$, sendo E_3 e E_4 expressões aritméticas.

11.2.2 Teste de fluxo de dados

O método de *teste de fluxo de dados* seleciona caminhos de teste de um programa de acordo com a localização das definições e do uso das variáveis no programa. Para ilustrar a abordagem de teste de fluxo de dados, considere que cada comando de um programa é atribuído um número de comando único e que cada função não modifica seus parâmetros ou variáveis globais. Para um comando com S como seu número de comando,

$$DEF(S) = \{X / \text{comando } S \text{ contém uma definição de } X\}$$

$$USO(S) = \{X / \text{comando } S \text{ contém um uso de } X\}$$

Se o comando S é um comando *se* ou de *ciclo*, seu conjunto DEF é vazio e seu conjunto USO é baseado na condição do comando S . A definição da variável X no comando S é considerada como estando *viva* no comando S' se existe um caminho do comando S para o comando S' que não contém qualquer outra definição de X .

Uma cadeia *definição-uso* (DU) da variável X é da forma $[X, S, S']$ em que S e S' são números de comandos, X pertence a $DEF(S)$ e $USO(S')$, e a definição de X no comando S está viva no comando S' .

A *estratégia de teste DU* exige que cada cadeia DU seja coberta pelo menos uma vez. Foi mostrado que o teste DU não cobre todos os ramos em situações raras. A abordagem de teste de fluxo de dados é efetiva para detecção de erros. No entanto, os problemas de medida da cobertura e de seleção dos caminhos de teste são mais difíceis do que os problemas correspondentes para o teste de condição.

11.2.3 Teste de ciclo

Teste de ciclo é uma técnica de teste caixa-branca que focaliza exclusivamente a validade de construções de ciclo. Quatro diferentes classes de ciclos podem ser definidas:

Ciclos simples. O seguinte conjunto de teste pode ser aplicado a ciclos simples em que n é o número máximo de passagens permitidas no ciclo.

1. Pule o ciclo completamente.
2. Apenas uma passagem pelo ciclo.
3. Duas passagens pelo ciclo.
4. m passagens pelo ciclo em que $m < n$.
5. $n - 1, n, n + 1$ passagens pelo ciclo.

Ciclos aninhados. Seria muito complicado adotar as mesmas técnicas de ciclos simples neste caso. Uma abordagem é sugerida:

1. Comece no ciclo mais interno. Ajuste todos os outros ciclos para os valores mínimos.
2. Conduze testes de ciclo simples para o ciclo mais interno enquanto mantém os ciclos externos nos seus valores mínimos de iteração.
3. Trabalhe em direção ao exterior, conduzindo testes para o ciclo seguinte.
4. Continue até que todos os ciclos tenham sido testados.

Ciclos concatenados. A mesma abordagem para o ciclo simples a menos que o mesmo contador é utilizado nos ciclos concatenados. Neste caso, é recomendada a abordagem a ciclos aninhados.

Ciclos desestruturados. Sempre que possível essa classe de ciclos deve ser reprojetada.

11.3 Teste caixa-preta

Um *teste caixa-preta*, também chamado de *teste comportamental*, focaliza os requisitos funcionais do software. O teste caixa-preta tende a ser aplicado durante os últimos estágios do teste.

11.3.1 Métodos de teste baseados em grafo

Os objetos que estão modelados no software e as relações que conectam esses objetos são utilizadas para criar um grafo. Os nós representam os objetos, as arestas representam as ligações, os pesos de nó descrevem as propriedade de um objeto, assim como os pesos das arestas representam propriedades das ligações. Uma *ligação direcionada* (representada por uma seta) indica que a relação se move em apenas uma direção. *Ligações paralelas* são usadas quando diversas relações são estabelecidas entre os nós.

Os métodos de teste de comportamento que podem fazer usos de grafos são:

Modelagem de fluxo de transação. Os nós representam passos em alguma transação e as arestas representam conexões lógicas.

Modelagem de estado finito. Os nós representam diferentes estados do software observáveis pelo usuário e as aresta representam as transições.

Modelagem do fluxo de dados. Os nós são objetos de daods e as ligações são transformações que ocorrem para traduzir um objeto de dados em outro.

11.3.2 Particionamento de equivalência

O *particionamento de equivalência* é um método de teste caixa-preta que divide o domínio de entrada de um programa em classes de dados, das quais casos de teste podem ser derivados. Uma *classe de equivalência* representa um conjunto de estados válidos ou inválidos para condições de entrada. Classes de equivalência podem ser definidas de acordo com as seguintes diretrizes:

1. Se uma condição de entrada especifica um *intervalo*, uma classe de equivalência válida e duas inválidas são definidas.
2. Se uma condição de entrada exige um *valor* específico, uma classe de equivalência válida e duas inválidas são definidas.
3. Se uma condição de entrada especifica um membro de um *conjunto*, uma classe de equivalência válida e uma inválida são definidas.
4. Se uma condição de entrada é *booleana*, uma classe de equivalência válida e uma inválida são definidas.

Um elemento pode ter mais de uma condição de entrada. Por exemplo, considere uma entrada de senha , há condição de entrada booleana (pode estar ou não presente) e condição de entrada de valor (cadeia de seis caracteres).

11.3.3 Análise de valor limite

Um grande número de erros tende a ocorrer nas fronteiras do domínio de entrada. É por essa razão que a *análise de valor-limite* (*boundary value analysis*, BVA) foi desenvolvida como técnica de teste.

A BVA leva à seleção de casos de teste nas bordas da classe. Em vez de focalizar somente as condições de entrada, a BVA deriva casos de teste também para o domínio de saída.

Se uma condição de entrada especifica um intervalo ou um valor limitado pelos valores a e b , casos de teste devem ser projetados com os valores a e b e imediatamente acima e imediatamente abaixo de a e b . O mesmo vale para as condições de saída. Se as estruturas de dados têm limites prescritos, certifique-se de projetar um caso de teste para exercitar a estrutura de dados no seu limite.

11.3.4 Teste de comparação

Quando um software redundante é desenvolvido (para aplicações críticas), equipes de engenheiros de software separadas desenvolvem versões independentes de uma aplicação usando a mesma especificação. Em tais situações, cada versão pode ser testada com os mesmos dados de teste para garantir que todas fornecem saída idêntica. Depois, todas as versões são executadas em paralelo com comparação em tempo real para garantir consistência. Essas versões independentes formam a base da técnica de teste caixa-preta chamada *teste de comparação* ou *teste de emparelhamento*.

Quando múltiplas implementações da mesma especificação tiverem sido produzidas, casos de teste projetados usando outras técnicas de caixa-preta fornecem entradas a cada versão do software. Se a saída de cada versão é a mesma, é considerado que todas as implementações estão corretas. Se a saída é diferente, cada uma das aplicações é investigada para determinar se um defeito em uma ou mais versões é responsável pela diferença.

O teste de comparação não é a toda prova, se a especificação estiver errada, todas as versões irão provavelmente refletir o erro. Além disso, se cada uma das versões independentes produzir resultados idênticos mas incorretos, o teste de comparação vai falhar na detecção do erro.

11.3.5 Teste de matriz ortogonal

Teste de matriz ortogonal pode ser aplicado a problemas nos quais o domínio de entrada é relativamente pequeno, mas grande demais para acomodar teste exaustivo. Quando o teste de matriz ortogonal ocorre, é criada uma *matriz ortogonal L9* de casos de teste. A matriz ortogonal L9 tem uma propriedade de balanceamento (estão distribuídos uniformemente pelo domínio de teste). A abordagem de teste de matriz ortogonal nos possibilita obter boa cobertura de teste com muito menos casos de teste que a estratégia exaustiva. Outras definições são importantes:

- Se todos os casos de teste com um determinado argumento igual a 1, por exemplo, falharem, trata-se de uma *falha de modo singular*.
- Se existe um problema consistente quando níveis específicos de dois parâmetros ocorrem simultaneamente, ele é chamado de *falha de modo duplo*. É a interação danosa entre dois parâmetros de teste.
- Matrizes ortogonais só podem garantir a detecção de falhas de modo singular e duplo. No entanto, muitas *falhas de multimodo* são também detectadas por esses testes.

11.4 Teste de ambientes, arquiteturas e aplicações especializadas

Podemos citar as técnicas especializadas de testes:

Teste de GUI. Como muitas GUI modernas têm a mesma aparência e funcionamento, uma série de testes padrão pode ser derivada. Devido ao grande número de operações GUI, o teste deve ser conduzido usando ferramentas automatizadas.

Teste de arquiteturas cliente/servidor. A natureza distribuída dificulta muito, os teste são consideravelmente mais difícil que em aplicações isoladas.

Teste da documentação e dispositivos de ajuda. Pode ser abordado em duas fases. A primeira fase, *revisão e inspeção*, examina o documento quanto à clareza editorial. A segunda fase, *teste ao vivo*, usa a documentação em conjunto com o uso do programa real. Nesta fase, os vários métodos de caixa-preta podem ser utilizados.

Teste de sistemas de tempo real. O projetista de casos de teste não tem apenas que considerar casos de teste caixa-branca e caixa-preta, mas também manipulação de eventos (processamento de interrupções), a temporalidade dos dados e o paralelismo das tarefas que manipulam os dados. Uma estratégia global de quatro passos pode ser proposta:

Teste de tarefa. Testes de caixa-branca e de caixa-preta são projetados e executados para cada tarefa.

Teste comportamental. É simulado o comportamento de um sistema de tempo real e examinado seu comportamento como consequência de eventos externos.

Testes intertarefas. Tarefas assíncronas que se comunicam são testadas com diferentes taxas de dados e carga de processamento para detectar se erros de sincronização entre tarefas vão ocorrer.

Teste de sistema. O software e o hardware são integrados e todo um conjunto de testes de sistema é conduzido numa tentativa de descobrir erros na interface software-hardware.

11.5 Estratégia de teste de software

O projeto efetivo de caso de testes é importante, mas não suficiente para o sucesso da atividade de testes. A estratégia, isto é, a série planejada de realização de testes, é também crucial. Basicamente, há três grandes fases de teste:

Teste de unidade. Tem por objetivo testar a menor unidade do projeto (um componente de software que não pode ser subdividido), procurando identificar erros de lógica e de implementação em cada módulo separadamente.

Teste de integração. Visa descobrir erros associados às interfaces entre os módulos quando esses são integrados para formar a estrutura do produto de software.

Teste de sistema. Tem por objetivo identificar erros de funções (requisitos funcionais) e características de desempenho (requisito não funcional) que não estejam de acordo com as especificações.

Tipicamente, os primeiros testes focalizam componentes individuais e aplicam testes caixa-branca e caixa-preta. Na integração, o foco é o projeto e a arquitetura do sistema. Finalmente, uma série de testes de alto nível é executada quando o sistema estiver operacional, visando descobrir erros nos requisitos.

No teste de unidade, faz-se necessário construir pequenos componentes para permitir testar módulos individualmente, os ditos *drivers* e *stubs*. Um *driver* é um programa responsável pela ativação e coordenação do teste de uma unidade. Ele é responsável por receber dados de teste fornecidos pelo testador, passar esses dados para a unidade que está sendo testada, obter os resultados produzidos e apresentá-los ao testador. Um *stub* é uma unidade que substitui, na hora do teste, uma outra unidade chamada pela unidade que está sendo testada. Em geral, um *stub* simula o comportamento de uma unidade chamada com o mínimo de computação ou manipulação de dados.

A abordagem de integração de módulos pode ter impacto na quantidade de *drivers* e *stubs* a ser construída. Sejam as seguintes abordagens:

Integração ascendente (*bottom-up*). Cada módulo no nível inferior da hierarquia é testado individualmente. A seguir, são testados módulos que chamam os previamente testados. Neste caso, apenas *drivers* são necessários.

Integração descendente (*top-down*). Começa de cima para baixo. Apenas *stubs* são necessários.

Sanduíche. Considera uma camada alvo no meio da hierarquia e utiliza abordagens ascendente e descendente.

Big-bang. Testar individualmente cada módulo e depois integrá-los de uma só vez. Neste caso, tanto *drivers* quanto *stubs* são necessários para cada módulo. Trabalhoso e suicida.

Os testes de sistema incluem diversos tipos de testes, realizados na seguinte ordem:

Teste funcional. Verifica se o sistema integrado realiza as funções especificadas nos requisitos.

Teste de desempenho. Verifica se o sistema integrado atende os requisitos não funcionais do sistema (eficiência, segurança e confiabilidade).

Teste de aceitação. Realizado pelos clientes. Assegura que o sistema solicitado é o que foi construído.

Teste de instalação. Necessário quando os testes de aceitação não são feitos onde o software será realmente instalado.

Capítulo 12

UML

12.1 Diagrama de caso de uso

Diagramas de caso de uso descrevem relacionamentos e dependências entre um grupo de caso de uso e os atores participantes no processo. Um Caso de uso descreve do ponto de vista dos atores um grupo de atividades num sistema que produz um resultado concreto e tangível. Casos de uso são descrições de interações típicas entre os usuários de um sistema e o sistema propriamente dito. Eles representam a interface externa do sistema e especificam um conjunto de exigências do que o sistema deve fazer.

Quando trabalhar com Casos de Uso, é importante lembrar-se de algumas regras simples:

- Cada Caso de Uso está relacionado com no mínimo um ator
- Cada Caso de Uso possui um iniciador (isto é um ator)
- Cada Caso de Uso liga-se a um resultado relevante (um resultado com valor de negócio)

Casos de uso também podem ter relacionamentos com outros casos de uso. Os três tipos mais comuns de relacionamento entre casos de uso são:

inclui-se que especifica que um Caso de Uso toma lugar dentro de outro Caso de Uso

estende que especifica que em determinadas situações, ou em algum ponto (chamado um ponto de extensão) um caso de uso será estendido por outro.

Generalização especifica que um caso de uso herda as características do super caso de uso, e pode sobrepor algumas delas ou adicionar novas de maneira semelhante a herança entre classes.

12.1.1 Ator

Um *ator* é uma entidade externa (fora do sistema) que interage com o sistema participando (e freqüentemente iniciando) um caso de uso. Atores podem ser pessoas reais (por exemplo usuários do sistema), outro sistema de computador ou eventos externos.

Atores não representam as pessoa física ou sistemas, mas sua regra. Isto significa que quando uma pessoa interage com o sistema de diferentes maneiras (assumindo diferentes regras) ela será representada por diversos atores. Por exemplo, uma pessoa que fornece suporte ao cliente por telefone e recebe ordens do cliente para o sistema pode ser representado por um ator da equipe de suporte.

12.1.2 Descrição do caso de uso

Descrição do caso de uso são narrativas de texto do caso de uso. Elas usualmente tomam a forma de uma nota ou um documento que é de alguma maneira ligado ao caso de uso, e explana o processo ou atividades que tomarão lugar no caso de uso.

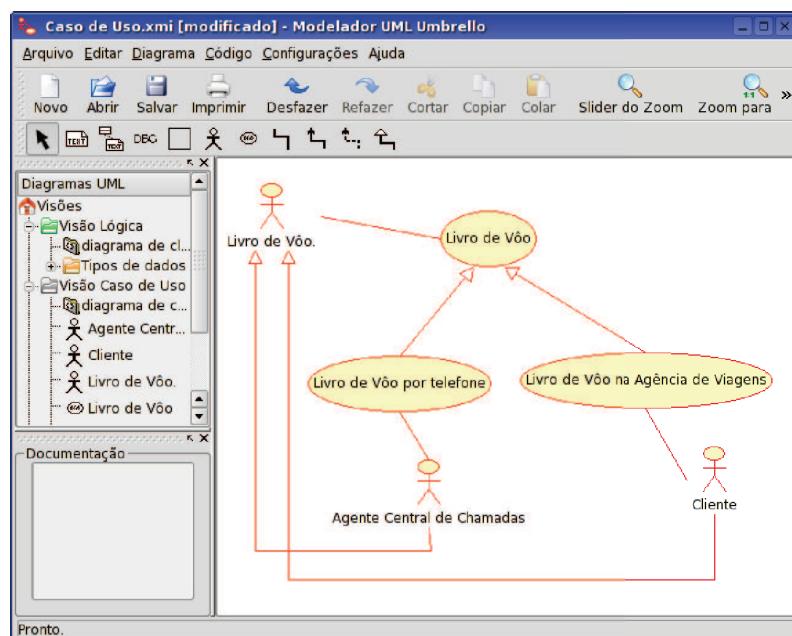


Figura 12.1: Exemplo de um diagrama de caso de uso

12.2 Diagrama de classe

Diagramas de classe mostram as diferentes classes que fazem um sistema e como elas se relacionam. Os diagramas de classe são chamados diagramas estáticos porque mostram as classes, com seus métodos e atributos bem como os relacionamentos estáticos entre elas: quais classes conhecem quais classes ou quais classes são parte de outras classes, mas não mostram a troca de mensagens entre elas.

Na UML, atributos são mostrados com pelo menos seu nome, e podem também mostrar seu tipo, valor inicial e outras propriedades. Atributos podem também ser exibidos com sua visibilidade:

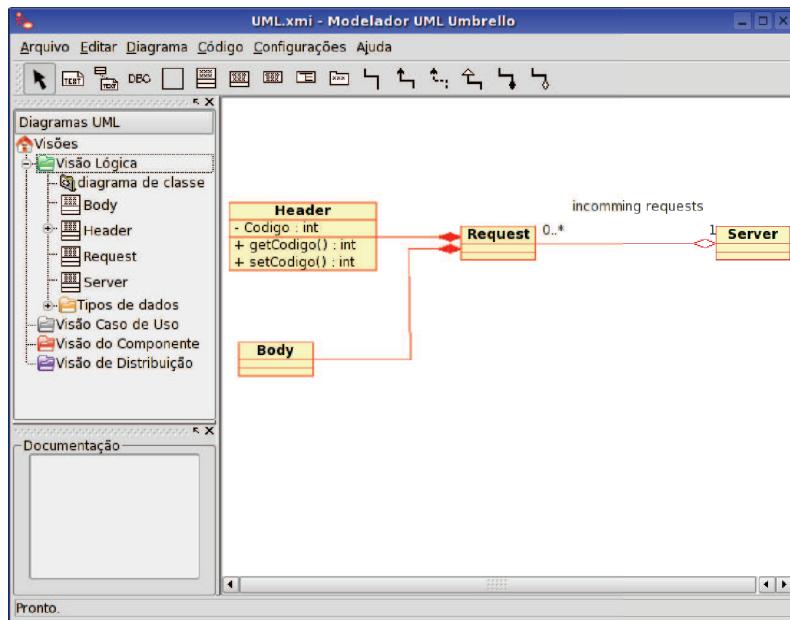


Figura 12.2: Exemplo de um diagrama de classe

- + indica atributos públicos.
- # indica atributos protegidos.
- - indica atributos privados

As operações também são exibidas com pelo menos seu nome, e podem também mostrar seus parâmetros e valores de retorno.

Classes podem ter *modelos*, um valor que é usado para uma classe ou tipo não especificado. O tipo de modelo é especificado quando uma classe é iniciada (isto é um objeto é criado). Modelos existem no C++ moderno e foram introduzidos no Java 1.5 onde eles são chamados de *genéricos*.

12.2.1 Associações de classe

Classes podem relacionar-se (ser associada com) com outras de diferentes maneiras:

Generalização

EM UML, uma *generalização* entre duas classes coloca-as numa hierarquia representando o conceito de herança de uma classe derivada de uma classe base. Em UML, Generalizações são representadas por uma linha conectando duas classes, com uma seta no lado da classe base.

Associações

São o mecanismo que permite objetos comunicarem-se entre si. Elas descrevem a conexão entre diferentes classes (a conexão entre os objetos atuais é chamada



Figura 12.3: Exemplo de generalização

conexão do objeto, ou link. Associações podem ter um regra que especifica o propósito da associação e pode ser uni ou bidirecional (indicando se os dois objetos participantes do relacionamento podem mandar mensagens para o outro, ou se apenas um deles sabe sobre o outro). Cada ponta da associação também possui uma valor de multiplicidade, que dita como muitos objetos neste lado da associação pode relacionar-se com o outro lado.

Em UML, associações são representadas como linhas conectando as classes participantes do relacionamento, e podem também mostrar a regra e a multiplicidade de cada um dos participantes. A multiplicidade é exibida como um intervalo $[min \dots max]$ de valores não negativos, com um asterisco no lado máximo representando infinito.



Figura 12.4: Exemplo de associação

Agregação

Agregações são um tipo especial de associação no qual as duas classes participantes não possuem em nível igual, mas fazem um relacionamento todo-parte. Uma agregação descreve como a classe que possui a regra do todo, é composta (tem) de outras classes, que possuem a regra das partes. Para agregações, a classe que age como o todo sempre tem uma multiplicidade de um.

Em UML, agregações são representadas por uma associação que mostra um rombóide no lado do todo. Representação visual de um relacionamento Agregação em UML.



Figura 12.5: Exemplo de agregação

Composição

Composições são associações que representam agregações muito fortes. Isto significa que composições formam relacionamentos todo-parte também, mas o relacionamento é tão forte que as partes não pode existir independentes.

Em UML, composições são representadas por um rombóide sólido no lado do todo.



Figura 12.6: Exemplo de composição

12.3 Diagramas de seqüênci a

Diagramas de seqüênci a mostram a troca de mensagens entre diversos objetos, numa situação específica e delimitada no tempo. Diagramas de seqüênci a colo cam ênfase especial na ordem e nos momentos nos quais mensagens para os objetos são enviadas.

Em diagramas de seqüênci a, objetos são representados através de linhas verticais tracejadas, com o nome do objeto no topo. O eixo do tempo é também vertical, aumentando para baixo, de modo que as mensagens são enviadas de um objeto para outro na forma de setas com a operação e os nomes dos parâmetros.

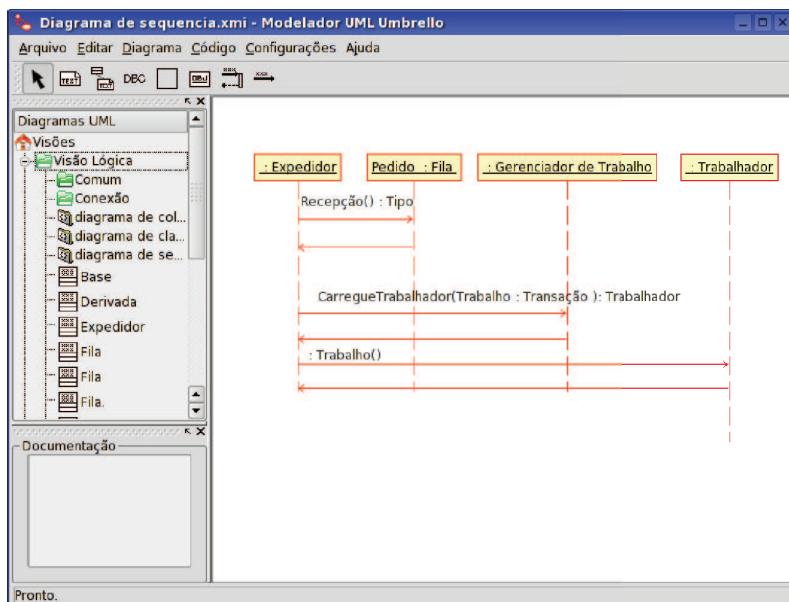


Figura 12.7: Exemplo de diagrama de seqüênci a

Mensagens podem ser síncronas, o tipo normal de mensagem de chamada em que o controle é passado para o objeto chamado até o método ter terminado sua execu ção, ou assíncronas, em que o controle é passado diretamente para o objeto chamado. Mensagens síncronas possuem uma caixa vertical no lado do objeto chamado para mostrar o controle do fluxo do programa.

12.4 Diagramas de colabora çao

Diagramas de colabora çao mostram as interações que ocorrem entre os objetos participantes numa situação específica. A informa ção é parecida com a mostrada

pelos diagramas de seqüência, mas neste, a ênfase é colocada em como as interações ocorrem no tempo, enquanto os diagramas de colaboração colocam os relacionamentos entre os objetos e sua topologia em destaque.

Em diagramas de colaboração, as mensagens enviadas de um objeto para outro são representadas por setas, mostrando o nome da mensagem, parâmetros, e a seqüência da mensagem. Diagramas de colaboração são especialmente indicados para mostrar um fluxo ou situação específica do programa e podem, rapidamente, demonstrar ou explanar um processo na lógica do programa.

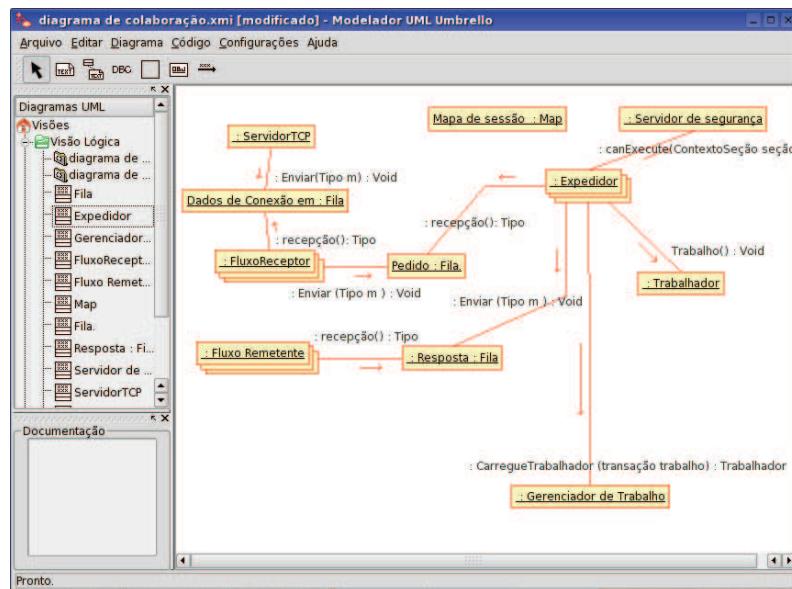


Figura 12.8: Exemplo de diagrama de colaboração

12.5 Diagramas de estado

Diagramas de Estado mostram os diferentes estados de um objeto durante sua vida, e o estímulo que faz com que o objeto mude seu estado. Diagramas de estado vêem objetos como máquinas de estado.

Estados são os blocos construídos dos Diagramas de estado. Um estado pertence a exatamente uma classe e representa um resumo dos valores dos atributos que uma classe pode tomar. Um estado UML descreve o estado interno de um objeto para uma classe em particular

Observe que nem toda mudança em um dos atributos de um objeto pode ser representada por um estado, mas somente aquelas mudanças que podem afetar significativamente o trabalho do objeto.

Existem dois tipos especiais de estados: inicial e final. Eles são especiais porque nenhum evento pode fazer com que um objeto retorne para seu estado inicial, e da mesma maneira nenhum evento pode tirar um objeto de seu estado final, uma vez que ele já o tenha alcançado.

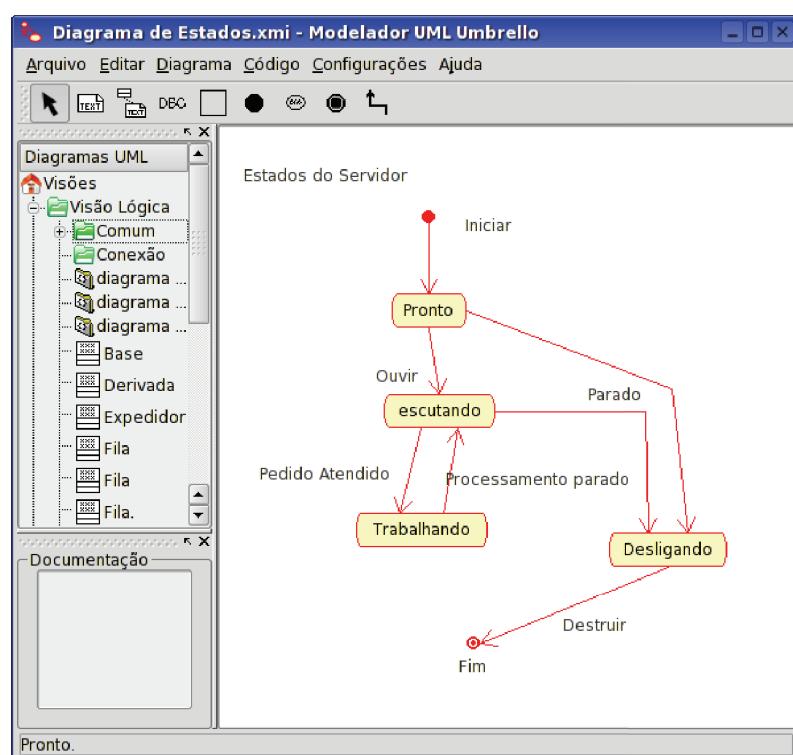


Figura 12.9: Exemplo de diagrama de estado

12.6 Diagramas de atividade

Uma *atividade* é um passo simples num processo. Uma atividade é um estado no sistema com atividade interna e, pelo menos, uma transição de saída. Atividades podem também ter mais de uma transição de saída se elas possuem condições diferentes.

O *diagrama de atividade* descreve a seqüência de atividades num sistema com a ajuda das atividades. Diagramas de atividade são uma forma especial de diagramas de estado, que somente (ou principalmente) contém atividades.

Diagramas de atividade são similares aos diagramas de fluxo de procedimentos, com a diferença de que todas as atividades são claramente anexas aos objetos.

Diagramas de atividade são sempre associados a uma classe, uma operação ou um caso de uso.

Diagramas de atividade suportam atividades seqüenciais bem como paralelas. A execução paralela é representada pelos ícones Forquilha/Esperar, e para as atividades executadas em paralelo, não é importante a ordem na qual elas se executam (elas podem ser executadas ao mesmo tempo ou uma após a outra).

Atividades podem formar hierarquias, isto significa que uma atividade pode ser composta por diversas atividades em detalhe, na qual as transições de entrada e saída devem corresponder às transições de entrada e saída do diagrama de detalhe.

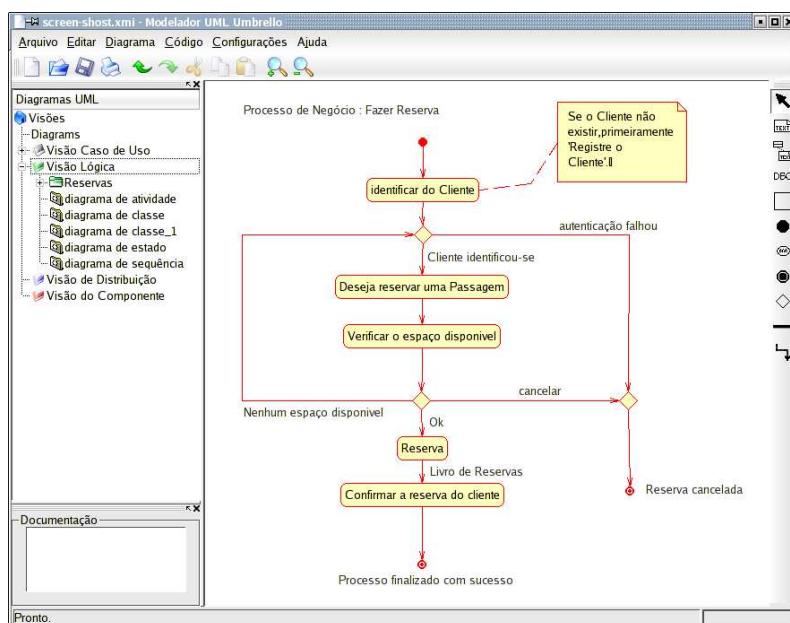


Figura 12.10: Exemplo de diagrama de atividade

12.7 Elementos auxiliares

Existem dois elementos em UML que não possuem nenhum valor real semântico para o modelo, mas auxiliam a elucidar partes do diagrama. Estes elementos são:

Linhas de texto. São úteis para adicionar informações curtas de texto ao diagrama. São textos livres.

Notas. São úteis para adicionar informações mais detalhadas sobre um objeto ou situação específica. Elas possuem a grande vantagem de poderem ser ancoradas a elementos UML para mostrar que a nota pertence a um objeto específico ou situação.

Caixas. São retângulos de forma livre que podem ser usados para agrupar itens, tornando os diagramas mais legíveis.

12.8 Diagramas de componente

Diagramas de componente mostram os componentes do software (sejam componentes de tecnologias como KParts, componentes CORBA ou Java Beans ou apenas seções do sistema que são claramente distintas) e os artefatos de que eles são feitos como arquivos de código fonte, bibliotecas de programação ou tabelas de bancos de dados relacionais.

Componentes podem possuir interfaces (isto é classes abstratas com operações) que permitem associações entre componentes.

12.9 Diagramas de distribuição

Diagramas de distribuição mostram as instâncias dos componentes de tempo de execução e suas associações. Eles incluem nós que são recursos físicos, tipicamente um computador simples. Eles também mostram interfaces e objetos (instâncias da classe).

Capítulo 13

Gerência de Configuração e Mudanças

A gestão da configuração do software é uma atividade guarda-chuva e é aplicada ao longo de todo o processo de software. O gerenciamento de configuração e de solicitações de mudança envolve os seguintes itens:

- Identificação dos itens de configuração;
- Restrição das mudanças nesses itens;
- Auditoria das mudanças nesses itens;
- Definição e o gerenciamento das configurações desses itens.

Os métodos e ferramentas utilizadas para o controle das mudanças é considerado com o Sistema de Gerenciamento de Mudanças de uma organização. Ele é contem informações chaves sobre os processos de desenvolvimento, promoção, implantação e manutenção de produtos da organização e armazenam a base de ativos e de artefatos potencialmente reutilizáveis resultantes da execução desses processos. Sendo assim, ele é parte integrante dos processos gerais de desenvolvimento.

Sem o controle dos inúmeros artefatos produzidos pelas muitas pessoas que trabalham em um projeto, são criados artefatos conflitantes, gerando um grande desperdício para a organização. Os principais problemas são a atualização simultânea, a notificação limitada e a existência de várias versões. A seguir, cada uma delas é descrita em detalhes.

- Atualização simultânea:

Quando dois ou mais membros da equipe trabalham separadamente no mesmo artefato, o ultimo membro a fazer mudanças desfaz o trabalho realizado pelo anterior. O problema básico é que se um sistema não permite a atualização simultânea, isso leva as mudanças em série e diminui o ritmo do processo de desenvolvimento. Entretanto, com a atualização simultânea, o desafio é detectar se ocorreram atualizações simultaneamente e resolver

quaisquer problemas de integração quando essas mudanças forem incorporadas.

- Notificação limitada:

Quando um problema é corrigido nos artefatos compartilhados por vários desenvolvedores e alguns deles não são notificados da mudança.

- Várias versões:

A maioria dos programas de grande porte é desenvolvida em várias versões evolutivas. Uma versão pode estar sendo usada pelo cliente, enquanto outra está em teste e uma terceira ainda está em desenvolvimento. Se forem encontrados problema em qualquer uma das versões, as correções devem ser propagadas entre elas. Isso pode levar a confusões que levam a confusões dispendiosas. Um sistema de Gerencia de Configuração é útil para gerenciar as diversas variantes de sistemas de software em desenvolvimento pois controla as versões que são utilizadas em determinadas builds do software ao compilar builds de programas individuais ou de uma nova versão do software de acordo com a especificação definida pelo usuário e ao impor políticas de desenvolvimento do sistema. Os principais benefícios do gerenciamento da mudança são:

- Suporte a diversos métodos de desenvolvimento;
- Preservação da integridade do produto;
- Garantia da abrangência e precisão do produto configurado;
- Ambiente estável no qual o produto será desenvolvido;
- Restrição das mudanças feitas nos artefatos com base nas políticas do projeto;
- Trilha de auditoria indicando por que, quando e por quem um artefato foi alterado.

Além disso, ele armazena dados detalhados sobre as alterações, tais como, quem criou uma versão específica, quais versões dos código fonte foram utilizadas em determinado build, alem de outras informações relevantes.

13.1 As Atividades

Na disciplina de Gerencia de Configuração as principais atividades são:

- Configurar ambiente;
- Estabelecer políticas;
- Escrever plano;
- Criar unidade de implantação;
- Relatar status de configuração;
- Realizar auditorias de configuração;

- Estabelecer processo de controle de mudanças;
- Revisar solicitação de mudança;
- Confirmar, duplicar ou recusar requisição de mudança;
- Criar espaços de trabalho de integração;
- Criar baselines;
- Promover baselines (de desenvolvimento para testes, de testes para homologação, etc.);
- Criar espaço de trabalho de desenvolvimento;
- Fazer mudanças;
- Liderar mudanças;
- Atualizar espaço de trabalho;
- Enviar solicitação de mudança;
- Atualizar solicitação de mudança.

13.2 Artefatos

Na disciplina de Gerencia de Configuração os principais artefatos são:

- Repositório do projeto;
- Plano de gerenciamento da configuração;
- Unidade de implantação;
- Métricas do projeto;
- Registro da auditoria de configuração.

13.3 Papéis e Responsabilidades

Na disciplina de Gerencia de Configuração os principais papéis e responsabilidades são:

- Gerente de configuração
 - Configurar ambiente;
 - Estabelecer políticas;
 - Escrever plano;
 - Criar unidades de implantação;
 - Relatar status de configuração;
 - Realizar auditoria de configuração.

- Gerente de controle de mudança
 - Estabelecer processo de controle de mudanças;
 - Revisar solicitação de mudança;
 - Confirmar, duplicar ou recusar solicitação de mudança.
- Integrador
 - Criar espaços de trabalho de integração;
 - Criar baselines;
 - Promover baselines.
- Outros papéis
 - Criar espaço de trabalho de desenvolvimento;
 - Fazer mudanças;
 - Liderar mudanças;
 - Atualizar espaço de trabalho;
 - Enviar solicitação de mudança;
 - Atualizar solicitação de mudança.

Capítulo 14

CMM - Capability Maturity Model

CMM, do acrônimo em inglês de Capability Maturity Model, é uma metodologia de diagnóstico e avaliação de maturidade do desenvolvimento de softwares em uma organização.

Ele descreve os principais elementos de um processo de desenvolvimento de software. O CMM descreve os estágios de maturidade através dos quais organizações passam enquanto evoluem o seu ciclo de desenvolvimento de software, através de avaliação contínua, identificação de problemas e ações corretivas dentro de uma estratégia de melhoria dos processos. Este caminho de melhoria é definido por cinco níveis de maturidade:

1. Inicial
2. Repetitivo
3. Definido
4. Gerenciado
5. Em Otimização

O CMM fornece às organizações orientação sobre como ganhar controle do processo de desenvolvimento de software e como evoluir para uma cultura de excelência na gestão de software. O objetivo principal nas transições desses níveis de maturidade é a realização de um processo controlado e mensurado como a fundação para melhoria contínua. Cada nível de maturidade possui um conjunto de práticas de software e gestão específicas, denominadas áreas-chave do processo (KPA). Estas devem ser implantadas para a organização atingir o nível de maturidade em questão.

O CMM identifica os níveis através dos quais uma organização deve evoluir para estabelecer uma cultura de excelência na engenharia de software. Como cada nível de maturidade do CMM forma a base necessária sobre a qual o próximo nível será construído, normalmente tentar pular níveis é improdutivo,

porque não haverá estabilidade na melhoria do processo, justamente pela falta da base que a sustentaria.



Figura 14.1: Modelo CMM

14.1 Os níveis de maturidade no CMM

14.1.1 Nível 1 - Inicial

No nível 1 de maturidade os processos são geralmente *ad hoc* e a organização geralmente não dispõe de um ambiente estável. O sucesso nestas organizações depende da competência e heroísmo dos funcionários e não no uso de processos estruturados. Devido ao imediatismo, um ambiente caótico, o nível 1 de maturidade raramente produz um produto ou serviço que funcione; assim, freqüentemente eles excedem o orçamento e o prazo em seus projetos.

14.1.2 Nível 2 - Repetitivo

No nível 2 de maturidade, o desenvolvimento do software é repetido. O processo pode não se repetir para todos os projetos da organização. A organização pode usar ferramentas de Gerência de Projetos para mapear os custos e o prazo do projeto.

A adoção de um processo de desenvolvimento ajuda a garantir que práticas existentes são utilizadas em momentos de stress. Quando estas práticas são adotadas, os projetos decorrem e são gerenciados de acordo com o planejamento inicial.

O status do projeto e os serviços entregues são visíveis ao gerenciamento (por exemplo: é possível a visualização de marcos do projeto e o término da maioria das tarefas).

Técnicas de gerenciamento de projetos são estabelecidas para mapear custos, prazos, e funcionalidades. Um mínimo de disciplina nos processos é estabelecido para que se possa repetir sucessos anteriores em projetos com escopo e aplicação similar. Ainda há um risco significante de exceder os custos e estimativas de prazo de desenvolvimento.

As áreas chaves de processo desse nível de maturidade são:

- Gerência de Requisitos (RM);
- Planejamento de Projeto de Software (SPP);
- Acompanhamento e Supervisão de Projeto de Software (SPTO);
- Gerência de Subcontratado de Software (SSM);
- Garantia da Qualidade de Software (SQA);
- Gerência da Configuração de Software (SCM).

14.1.3 Nível 3 - Definido

A organização possui um conjunto de processos padrões, os quais são a base do nível 3. Estes estão estabelecidos e são melhorados periodicamente. Estes processos padrões são usados para estabelecer uma consistência dentro da organização. Projetos estabelecem seus processos definidos pelo conjunto de padrões processuais da organização.

O gerenciamento da organização estabelece os objetivos dos processos baseado no conjunto de padrões pré-definidos e garante que estes objetivos sejam encaminhados de forma apropriada.

Uma crítica distinção entre os níveis 2 e 3 é o escopo dos padrões, descrição dos processos e procedimentos. No nível 2, os padrões, descrições de processos e procedimentos podem ser bem diferentes em cada instância específica do processo (por exemplo, em um projeto particular). No nível 3, os padrões, descrições de processo e procedimentos para o projeto são guiados pelo conjunto padrão de processos da organização.

As áreas chaves de processo desse nível de maturidade são:

- Foco no Processo da Organização (OPF);
- Definição do Processo da Organização (OPD);
- Programa de Treinamento (TP);
- Gerência Integrada de Software (ISM);
- Engenharia de Produto de Software (SPE);
- Coordenação entre Grupos (IC);
- Revisões Técnicas Formais (PR).

14.1.4 Nível 4 - Gerenciado

Utilizando métricas precisas, o gerenciamento pode efetivamente controlar os esforços para desenvolvimento de software. Em particular, o gerenciamento pode identificar caminhos para ajustar e adaptar o processo para projetos particulares sem perda de métricas de qualidade ou desvios das especificações.

Organizações neste nível conseguem metas qualitativas para o processo de desenvolvimento de software e de manutenção.

Subprocessos são selecionados conforme a importância na performance total do processo. Esses subprocessos selecionados são controlados usando técnicas estatísticas e qualitativas.

Uma crítica distinção entre o nível de maturidade 3 e 4 é a previsibilidade do desempenho do processo. No nível 4, o desempenho do processo é controlado usando técnicas estatísticas e qualitativas, e é previsível qualitativamente. No nível 3, os processos são somente previsíveis qualitativamente.

As áreas chaves de processo desse nível de maturidade são:

- Gerência Quantitativa do Processo (QPM);
- Gerência de Qualidade de Software (SQM).

14.1.5 Nível 5 - Otimizado

O nível de maturidade 5 foca no contínuo aumento do desempenho dos processos através de melhorias de inovação tecnológica e incremental. Objetivos de melhoria quantitativa dos processos para a organização são estabelecidos, continuamente revisados, refletindo os objetivos da organização, e usando critérios de gerência de processos.

Os efeitos da melhoria da revisão dos processos são medidas e acompanhadas utilizando-se de processos de melhoria de qualidade. Ambos, os processos definidos e o conjunto de processos padrões da organização são alvos de melhoria de métricas.

As áreas chaves de processo desse nível de maturidade são:

- Prevenção de Defeitos (DP);
- Gerência da Mudança Tecnológica (TCM);
- Gerência da Mudança do Processo (PCM).

14.2 Um pouco mais sobre KPA's

As áreas-chave de processo do Nível 2 estão focadas nas questões do projeto de software relacionadas ao estabelecimento de controles básicos de gerenciamento do projeto, e têm os seguintes objetivos:

- Gerência de Requisitos à Estabelecer um entendimento comum entre o cliente e os requisitos (desejos, necessidades) do cliente que serão atendidos pela solução a ser desenvolvida;
- Planejamento de Projeto de Software à Estabelecer planos coerentes para realizar a engenharia de software e o gerenciamento do projeto;
- Acompanhamento e Supervisão de Projeto de Software à Estabelecer uma adequada visibilidade do andamento (progresso) do software, de forma que o gerenciamento possa tomar ações corretivas se o planejamento original não estiver sendo seguido;
- Gerência de Subcontratado de Software à Selecionar fornecedores qualificados e gerenciá-los de forma eficiente;
- Garantia da Qualidade de Software à Fornecer ao gerenciamento visibilidade do processo em uso e dos produtos em construção;
- Gerência da Configuração de Software à Estabelecer e manter a integridade dos produtos durante todo o ciclo de vida do software.

As áreas-chave do Nível 3 estão focadas tanto nas questões do projeto, quanto da organização, conforme a organização estabelece uma infra-estrutura que efetivamente institucionaliza os processos de engenharia de software e de gerenciamento de todos os projetos.

As áreas-chave do Nível 4 estão focadas no estabelecimento quantitativo tanto do processo de software, quanto dos produtos em construção.

As áreas-chave do Nível 5 cobrem questões que tanto a organização, quanto os projetos devem considerar para implementar melhorias no processo de software que sejam contínuas e mensuráveis.

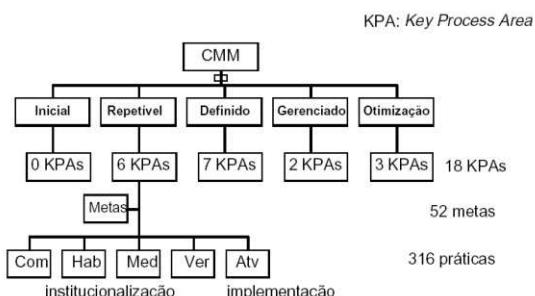


Figura 14.2: Áreas-chave de processo no modelo CMM

14.3 Efeitos da evolução do nível de maturidade

A evolução no nível de maturidade causa efeitos nas pessoas, nas tecnologias e nas práticas de medidas. A seguir são apresentados esses efeitos em cada nível

de maturidade.

- Pessoas

- Nível 1: Sucesso depende de indivíduos e heróis. Regime constante de emergência (apagar incêndio). Relacionamento entre grupos é descoordenado e muitas vezes conflitante;
- Nível 2: Sucesso ainda depende de indivíduos, mas passam a contar com apoio gerencial. Os compromissos são compreendidos e gerenciados. Existe treinamento para algumas funções;
- Nível 3: Grupos de projeto trabalham de maneira coordenada. O treinamento é planejado de acordo com as necessidades de cada papel e aplicado convenientemente;
- Nível 4: Existe um forte sentido de trabalho em equipe;
- Nível 5: Todos estão engajados em atividades de melhoria contínua.

- Tecnologia

- Nível 1: A introdução de novas tecnologias é arriscada;
- Nível 2: Atividades bem definidas facilitam a introdução de novas tecnologias;
- Nível 3: Novas tecnologias são avaliadas qualitativamente;
- Nível 4: Novas tecnologias são avaliadas quantitativamente;
- Nível 5: Novas tecnologias são planejadas e introduzidas com total controle.

- Medidas

- Nível 1: Coleta de dados é feita de maneira ad hoc;
- Nível 2: Coleta de dados de atividades de planejamento e acompanhamento é feita de maneira sistemática;
- Nível 3: Todos os processos definidos têm coleta sistemática de dados, os quais são compartilhados por todos os projetos da organização;
- Nível 4: A definição e coleta de dados são padronizadas na organização e os dados são usados para entender os processos de maneira quantitativa e estabilizá-los;
- Nível 5: Os dados coletados são usados para avaliar e selecionar possibilidades de melhoria de processos.

Parte IV

Linguagem de Programação

Java

Capítulo 15

Conceitos Básicos de Java

15.1 Pacotes

Em projetos pequenos é comum colocar todos os arquivos java em um único diretório. Essa é uma abordagem rápida, simples. No entanto, se o pequeno projeto começar a crescer, o número de arquivos aumentará e administrá-los em um único diretório pode se tornar um problema.

Pacotes não são nada mais que uma forma de organizar os arquivos que integram o projeto em diferentes diretórios de acordo com suas funcionalidades ou categoria a qual eles pertencem.

Por exemplo, os arquivos no pacote `java.io` são todos relacionados com funcionalidades de I/O, enquanto os arquivos do pacote `java.net` fornecem funcionalidades para tratar de redes. Em aplicações com interface gráfica, por exemplo, é muito comum encontrar um diretório chamado `ui` (user interface).

Além de melhorar a organização dos arquivos que compõe o projeto, a utilização de pacotes ajuda a evitar colisão de nomes entre classes. Por exemplo: Se um programador definir uma classe chamada `Vector`, esse nome iria colidir com a classe `Vector` da JDK. No entanto, isso não ocorre por que a JDK usa `java.util` como um nome de pacote para a classe `Vector`. Dessa forma, a classe implementada pelo programador pode se chamar `Vector`. A figura 15.1 mostra uma estrutura de diretórios criada pela utilização de pacotes.

Name	Size	Type
SocketImplFactory.java	2KB	JAVA File
SocketInputStream.java	5KB	JAVA File
SocketOptions.java	10KB	JAVA File
SocketOutputStream.java	3KB	JAVA File
SocketPermission.java	29KB	JAVA File
UnknownHostException.java	2KB	JAVA File
UnknownServiceException.java	2KB	JAVA File
URL.java	36KB	JAVA File
URLClassLoader.java	18KB	JAVA File
URLConnection.java	48KB	JAVA File
URLDecoder.java	3KB	JAVA File
URLEncoder.java	4KB	JAVA File
URLStreamHandler.java	7KB	JAVA File
URLStreamHandlerFactory.java	2KB	JAVA File

Para indicar que uma classe pertence a um determinado pacote basta iniciar o arquivo java como mostrado no exemplo a seguir.

```
//Somente código pode vir antes dessa linha
package world;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

15.2 Modificadores de Acesso

Os modificadores de acesso são keywords adicionadas ao código para determinar se atributos, métodos ou classes poderam ser acessados a partir de outras classes. Os modificadores de acesso presentes na linguagem Java são: public, private, protected e package-private (implícito quando não é usado nenhum modificador na declaração da classe, atributo ou método).

Uma classe pública pode ser acessada por qualquer outra classe. Caso seja declarada sem o modificador public, a classe será package-private, ou seja, só poderá ser acessada por classes do mesmo pacote.

Os métodos e atributos aceitam outros modificadores além do public. O modificador private dá acesso somente dentro da classe, enquanto o modificador protected permite acesso dentro do pacote e às subclasses. Assim como nas classes, caso nenhum modificador seja declarado, fica implícito o modificador package-private.

A tabela 15.1 abaixo resumo as permissões de acesso impostas por cada um dos modificadores:

Modificador	Pacote	Subclasse	Todos
public	sim	sim	sim
private	não	não	não
protected	sim	sim	não
nenhum	sim	não	não

Tabela 15.1: Modificadores de Acesso

15.3 Variáveis

Na linguagem de programação Java são definidos os seguintes tipos de variáveis:

- Variáveis de Instância (Atributos não estáticos): Um objeto armazena seu estado individual em atributos não estáticos, ou seja, declarados sem o

modificador *static*. Atributos não estáticos são conhecidos com variáveis de instância por que seus valores são únicos para cada instância de uma classe;

- Variáveis de Classe (Atributos estáticas): Uma variável de classe é todo atributo declarado com o modificador *static*. Isso indica ao compilador que existe apenas uma cópia da variável, não importando o número de vezes que essa classe foi instanciada. Portanto, quando o valor de uma variável de classe é alterada em um dos objetos, o novo valor se torna visível para todos os outros objetos da mesma classe. Para impedir que uma variável de classe tenha seu valor alterado ela deve ser declarada com o mofificador *final*.
- Variáveis Locais: As variáveis locais são utilizadas para armazenar o estado temporário dos métodos. A sintaxe de declaração de uma variável local é similar a de declaração de uma variável de instância. O que determina se uma variável é local é a parte do código em que ela é declarada - entre as chaves que determinam o início e o fim de um método. Uma variável é visível apenas para o método no qual foi declarada; não pode ser acessada do resto da classe.

```
public class Bicicleta {  
  
    // Variável de Instância  
    public int velocidade = 0;  
  
    // Variável de Classe  
    public static int qtdRodas = 2;  
  
    // Variável de Classe com valor constante  
    public static final String marca = "Caloi";  
  
    public void aumentaVelocidade (int incremento){  
        // Variável Local  
        int novaVelocidade; = velocidade + incremento;  
        velocidade = novaVelocidade;  
    }  
}
```

15.4 Operadores

A lista abaixo sumariza os operadores suportados pela linguagem Java:

- Operador de Atribuição Simples

= Operador de Atribuição Simples

- Operadores aritméticos

- + Adição (Também utilizado para concatenação de Strings)
- Subtração
- * Multiplicação
- / Divisão
- % Resto da Divisão

- Operadores Unários

- + Operador unário mais
- Operador unário menos; Inverte o sinal de uma variável
- ++ Operador de Incremento; Incrementa um valor de 1
- Operador de Decremento; Decrementa o valor de 1
- ! Operador Lógico de complemento; Inverte o valor de um boolean

- Operadores de Comparação

- == Igual
- != Diferente
- > Maior
- >= Maior ou Igual
- < Menor
- <= Menor ou Igual
- instanceof Compara um objeto a um tipo específico

- Operadores Condicionais

- && AND Condicional
- || OR Condicional
- ? : Ternário (Forma curta para if-then-else)

- Operadores Bit a Bit e de Deslocamento (Shift)

- Complemento bit a bit unário
- << shift para esquerda
- >> Shift para direita
- >>> Shift para direita sem sinal
- & AND bit a bit
- ^ OR exclusivo bit a bit
- | OR bit a bit

15.5 Expressões, Sentenças e Blocos

Uma expressão é uma construção feita de variáveis, operadores e chamadas de métodos, descrita de acordo com a sintaxe da linguagem. A seguir, exemplos de expressões:

```
//Declaração e inicialização de uma variável do tipo int
int cadence = 0;
//Atribuição de valor a uma posição de um vetor de int
anArray[0] = 100;
//Imprimindo um valor na tela
System.out.println("Element 1 at index 0: " + anArray[0]);
//Atribuição de valor baseado no resultado de uma operação de adição
result = 1 + 2;
//Utilização de parêntesis para explicitar ordem de execução das operações
zaz = x / (y+10);
```

Em Java uma sentença (*statements*) é definida como uma unidade completa de execução. Os seguintes tipos de expressões podem se tornar sentenças quando terminadas com um ponto e vírgula: (i) Expressões de Atribuição; (ii) Expressões com ++ ou --; (iii) Chamada de métodos e (iv) Expressões de Criação de objetos.

```
// Statement de atribuição
aValue = 8933.234;
// Statement de incremento
aValue++;
// Statement de chamada de método
System.out.println("Hello World!");
// Statement de criação de objeto
Bicycle myBike = new Bicycle();
```

Essas são as chamadas statements de expressão. Existem ainda as statements de declaração de variável e de controle de fluxo.

Um bloco é um grupo de zero ou mais statements entre um par de chaves. O exemplo a seguir ilustra o uso de blocos em Java:

```
class BlockDemo {
    public static void main(String[] args) {
        boolean condition = true;
        if (condition) { // begin block 1
            System.out.println("Condition is true.");
        } // end block one
        else { // begin block 2
            System.out.println("Condition is false.");
        } // end block 2
    }
}
```

15.6 Comandos de Controle de Fluxo

As statements de um código geralmente são executadas na medida na ordem em aparecem. No entanto, os comandos de controle de fluxo podem ser utilizadas para interromper o fluxo de execução empregando decisões, looping e desvios, permitindo que ao programa executar condicionalmente blocos de código particulares. Os principais comandos de controle de fluxo da linguagem Java são mostrados a seguir.

If-Then

```
void applyBrakes(){  
    if (isMoving){      // the "if" clause: bicycle must moving  
        currentSpeed--; // the "then" clause: decrease current speed  
    }  
}
```

If-Then-Else

```
//Exemplo 1  
void applyBrakes(){  
    if (isMoving) {  
        currentSpeed--;  
    } else {  
        System.err.println("The bicycle has already stopped!");  
    }  
}  
  
//Exemplo 2  
class IfElseDemo {  
    public static void main(String[] args) {  
  
        int testscore = 76;  
        char grade;  
  
        if (testscore >= 90) {  
            grade = 'A';  
        } else if (testscore >= 80) {  
            grade = 'B';  
        } else if (testscore >= 70) {  
            grade = 'C';  
        } else if (testscore >= 60) {  
            grade = 'D';  
        } else {  
            grade = 'F';  
        }  
        //O resultado será C pois, uma vez que a condição é atendida,  
        //o restante do código If-Then-Else não é mais avaliado  
        System.out.println("Grade = " + grade);  
    }  
}
```

```
        }
    }

Switch

class SwitchDemo {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30;
                break;
            case 2:
                if ( ((year % 4 == 0) && !(year % 100 == 0))
                    || (year % 400 == 0) )
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                System.out.println("Invalid month.");
                break;
        }
        //O resultado será 29
        System.out.println("Number of Days = " + numDays);
    }
}
```

While e Do-While

```
//Exemplo While
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        //Teste realizado antes da execução do bloco de código
```

```
//Pode acontecer do código não ser executado nenhuma vez
while (count < 11) {
    System.out.println("Count is: " + count);
    count++;
}
}

//Exemplo Do-While
class DoWhileDemo {
    public static void main(String[] args){
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        //Teste ao fim da execução do bloco de código
        //Código é executado ao menos uma vez
        } while (count <= 11);
    }
}

For

//Exemplo 1
class ForDemo {
    public static void main(String[] args){
        for(int i=1; i<11; i++){
            System.out.println("Count is: " + i);
        }
    }
}

//Exemplo 2
for ( ; ; ) { //infinite loop
    // código
}

//Exemplo 3
class EnhancedForDemo {
    public static void main(String[] args){
        int[] numbers = {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
    }
}

Break

//Exemplo 1
class BreakDemo {
```

```
public static void main(String[] args) {
    int[] arrayOfInts = {32,87,3,589,12,1076,2000,8,622,127};
    int searchfor = 12;
    int i;
    boolean foundIt = false;

    for (i = 0; i < arrayOfInts.length; i++) {
        if (arrayOfInts[i] == searchfor) {
            foundIt = true;
            //Encerra o loop for
            break;
        }
    }
    if (foundIt) {
        System.out.println("Found " + searchfor + " at index " + i);
    } else {
        System.out.println(searchfor + " not in the array");
    }
}

//Exemplo 2
class BreakWithLabelDemo {
    public static void main(String[] args) {
        int[][] arrayOfInts = {{32,87,3,589},
                               {12,1076,2000,8},
                               {622,127,77,955}};

        int searchfor = 12;
        int i;
        int j = 0;
        boolean foundIt = false;

        search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length; j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    //Encerra o loop for mais interno
                    //e desvia para o label search
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Found " + searchfor + " at " + i + ", " + j);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```

```
        }
    }
}

Continue

//Exemplo 1
class ContinueDemo {
    public static void main(String[] args) {

        String searchMe = "peter piper picked a peck of pickled peppers";
        int max = searchMe.length();
        int numPs = 0;

        for (int i = 0; i < max; i++) {
            //interested only in p's
            if (searchMe.charAt(i) != 'p'){
                //Vai para o próximo loop sem executar
                //o restante do código do bloco for
                continue;
            }
            //process p's
            numPs++;
        }
        System.out.println("Found " + numPs + " p's in the string.");
    }
}

//Exemplo 2
class ContinueWithLabelDemo {
    public static void main(String[] args) {

        String searchMe = "Look for a substring in me";
        String substring = "sub";
        boolean foundIt = false;

        int max = searchMe.length() - substring.length();

        test:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++)!= substring.charAt(k++)) {
                    //Interrompe a iteração do loop while e vai
                    //para proxima iteração do loop for,
                    //marcada pelo label test
                    continue test;
                }
            }
        }
    }
}
```

```
        }
    }
    foundIt = true;
    break test;
}
System.out.println(foundIt ? "Found it" : "Didn't find it");
}
}
```

15.7 Classes Aninhadas

No Java existe a possibilidade de se definir classes dentro de outra classe, como se fossem atributos ou métodos. Algumas das vantagens de se utilizar esse recurso do Java são:

- Legibilidade: Classes podem ser agrupadas por similaridade;
- Ocultamento: Podem ser privadas ou protegidas;
- Redigibilidade: Classes internas possuem acesso aos membros privados da classe que a definiu e vice-versa. Na verdade, o recurso de classes internas surgiu no Java 1.1 com esse propósito.

Como são definidas dentro de outras classes, o código fonte fica no mesmo arquivo .java. Ao compilar, gera-se vários arquivos .class, compondo o nome da classe externa e interna. Exemplo: Externa.java contém a classe Externa, que define uma classe interna chamada Interna. Ao compilar, gera-se Externa.class e Externa\$Interna.class. É importante ressaltar que a classe interna não pode ter o mesmo nome da classe externa que a define.

No Java existem os seguintes tipos de classes internas:

- Classes Aninhadas
- Classes Instanciadas
 - Classes Membro
 - Classes Locais
 - Classes Anônimas

As classes aninhadas são os tipos mais simples de classes internas. Uma classe é definida dentro da outra como se fosse um membro static. Elas permitem definir acesso privado ou protegido e agrupa classes logicamente relacionadas. São referenciadas via Externa.Interna, como se fosse um atributo estático. A seguir um exemplo de classe aninhada e de como referenciá-la:

```
//Classe externa
class Par {

    private Chave chave;
```

```
private Valor valor;

public Par(Chave chave, Valor valor) {
    this.chave = chave;
    this.valor = valor;
}

//Classe interna
static class Chave {
    private String nome;
    public Chave(String nome) {
        this.nome = nome;
    }
}

//Classe interna com proteção de acesso
protected static class Valor {
    private int valor;
    public Valor(int valor) {
        this.valor = valor;
    }
}

public class Teste {

    public static void main(String[] args) {
        Par.Chave chave = new Par.Chave("Nota");
        Par.Valor valor = new Par.Valor(10);
        Par par = new Par(chave, valor);
    }
}
```

15.8 Tipos Enumerados

Tipos enumerados são aqueles que possuem um conjunto finitos de valores que as variáveis podem assumir. Por exemplo: estações do ano, naipes ou cartas do baralho, planetas do sistema solar etc. Originalmente, a forma padrão utilizada para representar tipos enumerados era o chamado int Enum Pattern (tipos enumerados baseados em inteiros).

```
// int Enum Pattern
public static final int SEASON_WINTER = 0;
public static final int SEASON_SPRING = 1;
public static final int SEASON_SUMMER = 2;
public static final int SEASON_FALL = 3;
```

Esse tipo de padronização apresentava os seguintes problemas:

- Não seguro quanto ao tipo: Como uma season é representada por um int, pode-se passar qualquer valor inteiro (inclusive diferente de 0,1,2,3) quando uma season for requerida;
- Sem namespace: É necessário criar um padrão de nomenclatura para evitar colisão com outros tipos enumerados. (Exemplo: SEASON_).
- Valores imprimidos não são explicativos: Como os valores são int, quando impressos não apresentam a informação que representam diretamente.

Para contornar esses problemas a partir do Java 5 foi introduzido o *Typesafe Enum Pattern*, a partir do qual os tipos enumerados podem ser expressos de forma mais simples. O código abaixo mostra o uso do novo padrão para tipos enumerados:

```
public class Weather {  
  
    //Define tipo enumerado Refrigerante  
    public enum Season = {winter, spring, summer, fall};  
  
    private printSeasons() {  
        //Loop recuperando cada um dos valores possíveis de Season  
        for (Season s : Season.values()){  
            System.out.println(s);  
        }  
    }  
}
```

15.9 Anotações

Anotações (annotations) provêm dados sobre o programa mas não são parte do programa propriamente dito. Elas não possuem efeito direto na operações do código com o qual estão relacionadas. Esse recurso da linguagem Java permite que o programador não se preocupe em escrever código auxiliar (arquivos de configuração, por exemplo). A idéia é que o programador diga o que tem que ser feito utilizando as anotações e ferramentas auxiliares façam o restante do trabalho. Alguns exemplos do uso de anotações são:

- Informações para o compilador: Anotações podem ser utilizadas pelo compilador para detectar erros ou sumprimir mensagens de warning;
- Processamento em tempo de Compilação ou Deploy: Ferramentas auxiliares podem processar as anotações para gerar código, arquivos XML etc;
- Processamento em tempo de Execução: Algumas anotações podem ser processadas em tempo de execução;

- Documentação: Anotações podem ser utilizadas para substituir comentários no código.

Os três pré-definidos de anotações utilizadas pelo compilador são:

- **@Deprecated:** Indica que o elemento marcado está deprecated e não deve ser mais utilizado. O compilador gera um warning sempre que um programa utilizar um uma classe, um método ou um atributo marcado como `@Deprecated`;
- **@Override:** Informa ao compilador que o elemento marcado deve sobreescriver o elemento de mesmo nome declarado na superclasse. Embora a anotação não seja obrigatório para sobreescrivendo um método, utilizá-la ajuda a prevenir erros;
- **@SuppressWarnings:** Indica ao compilador situações em que os warnings devem ser suprimidos;

A seguir alguns exemplos de utilização das anotações descritas.

```
//Marca o método como deprecated
@Deprecated
static void deprecatedMethod() { //código }

//Indica que o método irá sobreescrivendo o método da superclasse
@Override
int overriddenMethod() { //código }

//Indica ao compilador para suprimir warning gerados pelo uso
//de métodos deprecated
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
    //O uso de um método deprecated geraria um warning
    objectOne.deprecatedMethod();
}
```

15.10 Genéricos

Os genéricos são um recurso introduzido a partir Java 5 e permite ao programador escrever código abstraindo tipos de dados. Os genéricos são tipicamente utilizados em conjunto com interfaces ex: `List`) que herdam da interface `Collection` ou de classes que a implementam (ex: `AbstractList`, `ArrayList`, `LinkedList`, `Vector`)

Quando um elemento é retirado de uma `Collection` é necessário realizar um cast para o tipo do elemento armazenado na `Collection`. Além de incoveniente, isso é inseguro. O compilador não checa se o cast realizado é do mesmo tipo do armazenado na coleção. Dessa forma, o cast pode falhar em tempo de execução.

```
//Criação de uma collection do tipo LinkedList
List myIntList = new LinkedList();
//Adição de um elemento do tipo Integer a collection
myIntList.add(new Integer(0));
//Recuperando um elemento da collection. Cast para Integer é necessário
Integer x = (Integer) myIntList.iterator().next();
```

Os genéricos provêem uma forma de comunicar ao compilador o tipo armazenado na coleção, permitindo que a checagem seja feita em tempo de compilação. Além de evitar erros em tempo de execução, a utilização de genéricos elimina a necessidade de casts.

```
//Criação de uma collection LinkedList para armazenar objetos do tipo Integer
List<Integer> myIntList = new LinkedList<Integer>();
//Adição de um elemento do tipo Integer a collection
myIntList.add(new Integer(0));
//Recuperando um elemento da collection. Cast não é necessário
Integer x = myIntList.iterator().next();
```

Os tipos genéricos têm como grande vantagem o aproveitamento de código. A interface Collection é um exemplo disso. As operações de adicionar ou remover um novo elemento, determinar o tamanho, verificar se a collection está vazia ou recuperar o índice de um determinado elemento, por exemplo, certamente são aplicáveis a collections do tipo Integer, String ou de outra classe qualquer. Exemplos de interfaces genéricas são List e Iterator. A seguir, alguns trechos dessas duas interfaces como definidas no pacote java.util.

```
public interface Iterator<E> {
    E next();
    boolean hasNext();
    //mais código ...
}

public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
    //mais código ...
}
```

Genéricos permitem implementações mais sofisticadas como:

```
//Wildcards ou Coringas
//Método com parâmetro Casulo. Aceita Casulo de qualquer tipo
void imprimir (Casulo<?> c) {
    // código do método ...
}

//Limitando o tipo genérico aceito pelo método
//Método só aceita Casulo de Forma ou Casulos de subclasses de Forma
```

```
void desenhar (Casulo<? extends Forma> c){
    // código do método ...
}
```

15.11 Reflexão

A Reflection permite um programa Java examinar ou fazer a introspecção nele mesmo, ou seja, olhar e examinar suas propriedades e estrutura. Com isso, você pode, por exemplo obter o nome de todos os membros de uma classe, como atributos e métodos, bem como executar um método usando a Introspection. Esse recurso é utilizado, por exemplo, pelos ambientes de desenvolvimento (as IDEs) para examinar e exibir a estrutura e conteúdo das classes e beans. A seguir um exemplo do uso de refetions.

```
import java.lang.reflect.*;

public class Classe1 {

    private int funcao1( Object p, int x ) throws NullPointerException {
        if (p == null) throw new NullPointerException();
        return x;
    }

    public static void main(String args[]) {
        try {
            //Carrega a classe
            Class cls = Class.forName("Classe1");
            //Recupera a lista de métodos da classe
            Method methlist[] = cls.getDeclaredMethods();
            //Imprime informações (métodos, modificadores, exceções)
            //sobre cada um dos métodos da classe
            for (int i = 0; i < methlist.length; i++) {
                Method m = methlist[i];
                System.out.println("-----");
                System.out.println("nome = " + m.getName());
                System.out.println("-----");
                System.out.println("membro de:" + m.getDeclaringClass());
                System.out.println("modificador:" + Modifier.toString(m.getModifiers()));

                //Recupera lista de parâmetros do método
                Class pvec[] = m.getParameterTypes();
                for (int j = 0; j < pvec.length; j++)
                    System.out.println("parâmetro #" + j + " " + pvec[j]);

                //Recupera a lista de excessões do método
                Class evec[] = m.getExceptionTypes();
                for (int j = 0; j < evec.length; j++)
```

```
        System.out.println("exceção #" + j + " " + evec[j]);  
  
        //Recupera o tipo de retorno do método  
        System.out.println("tipo de retorno = " + m.getReturnType());  
    }  
}  
catch (Throwable e) {  
    System.err.println(e);  
}  
}  
}  
  
//Saída da execução da classe Classe1:  
//Lista de métodos da Classe1 com todas as suas propriedades  
  
-----  
nome = funcao1  
-----  
membro de:class Classe1  
modificador:private  
parâmetro #0 class java.lang.Object  
parâmetro #1 int  
exceção #0 class java.lang.NullPointerException  
tipo de retorno = int  
-----  
nome = main  
-----  
membro de:class Classe1  
modificador:public static  
parâmetro #0 class [Ljava.lang.String;  
tipo de retorno = void
```

Capítulo 16

Classes Essenciais

16.1 Exception e Controle de Exceções

Uma exceção é um evento que ocorre durante a execução de um programa interrompendo seu fluxo normal de execução.

Quando um erro ocorre dentro de um método, o método cria um objeto chamado exception e o entrega ao controle de execução (runtime system), que contém informações sobre o erro, incluindo seu tipo e o estado do programa quando o erro ocorreu. O ato de criar um objeto exception e a entregá-lo ao runtime system é chamado lançamento de uma exceção (throwing an exception).

O runtime system então pesquisa na pilha de chamadas por um método que contenha um bloco de código capaz de tratar a exceção. Esse bloco de código é chamado tratador de exceções (exception handler). A pesquisa começa no método onde o erro ocorreu e procede na pilha na ordem reversa em que os métodos foram chamados. Quando um tratador adequado é encontrado, o runtime system entrega a exceção. Um tratador é considerado adequado quando o tipo da exceção lançada coincide com o tipo definido no tratador (IOException, ClassNotFoundException, etc.).

Quando um tratador é escolhido diz-se que ele capturou a exceção (catches the exception). Se o runtime system pesquisa em toda a pilha de métodos e não encontrar um tratador adequado, o programa será encerrado. A figura 16.1 mostra a ordem de pesquisa por um tratador para a exceção.

16.1.1 Exceções típicas

Em Java exceções são objetos. A classe Exception é a superclasse de todas as exceções. Exemplos típicos de exceções são:

- Índice de uma array fora do intervalo permitido (ArrayIndexOutOfBoundsException);
- Problemas em operações aritméticas, como overflows e divisões por zero (ArithmeticException);

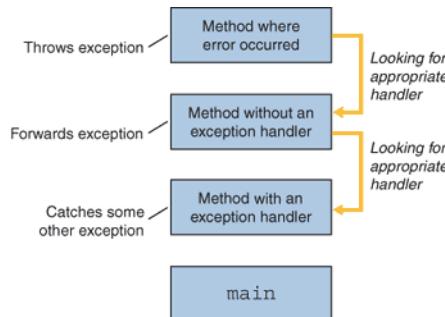


Figura 16.1: Procura pelo tratador de exceções

- Argumentos inválidos numa chamada a um método (`IllegalArgumentException`);
- Uso de uma referência que não aponta para nenhum objeto (`NullPointerException`);
- Acesso a um arquivo inexistente (`FileNotFoundException`);
- Erros de banco de dados (`SQLException`);

A classe `Exception`, por sua vez, é subclasse de `Throwable`, o que permite que as exceções sejam lançadas. Se o programador desejar que a exceção assim lançada seja tratada fora do método que a gerou, ele deve explicitar isto usando a palavra chave `throws` seguida do tipo de exceção, na declaração do método.

```

//Exemplo 1: Método indica que todas exceções serão lançadas, ou seja,
//            não serão tratadas internamente.

public void simpleMethod1 (String x) throws Exception {
    //...
    if (problema1) throw Exception;
    //...
}

//Exemplo 2: Método indica que as exceções do tipo excecao1
//            e excecao2 não serão tratadas internamente.

public void simpleMethod2 (int a, int b) throws excecao1, excecao2 {
    //...
    if (problema1) throw excecao1;
    if (problema2) throw excecao2;
    //...
}
  
```

Na verdade, existe uma classe de exceções, a `RuntimeException`, que não precisam ser lançada explicitamente. `RuntimeException` é uma superclasse das exceções que podem ocorrer durante a operação normal da máquina virtual Java.

Uma vez que a exceção foi lançada, a execução do método é interrompida e o controle volta ao objeto que chamou este método. Exemplos de classes que herdam de RuntimeException são ArithmeticException, BufferOverflowException, ClassCastException, IllegalArgumentException, IndexOutOfBoundsException, NullPointerException, SystemException, etc.

A figura 16.2 mostra como as exceções estão organizadas na hierarquia de classes da linguagem Java.

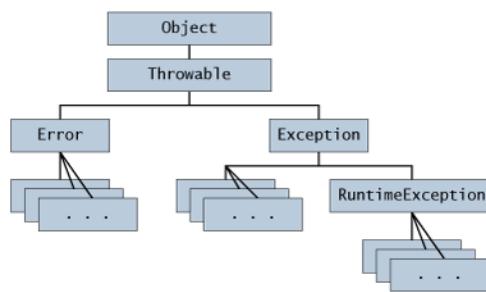


Figura 16.2: Exceptions e Errors

Além das exceções típicas, a figura 16.2 mostra também a classe `Error`. `Error` é uma subclasse de `Throwable` que indica situações excepcionais externas a aplicação, das quais a aplicação nem sempre é capaz de se recuperar, por exemplo, um problema de hardware. Uma aplicação pode decidir por capturar um erro com objetivo de notificar o usuário sobre o problema, no entanto, quase sempre faz mais sentido para o programa imprimir um stack trace e encerrar.

A classe `VirtualMachineError` é um exemplo de subclasse de `Error`, que engloba problemas como falta de memória (`OutOfMemoryError`) e estouro de pilha (`StackOverflowError`).

16.1.2 Capturando Exceções

Para capturar uma exceção é utilizada a seguinte estrutura de código:

```
try {
    // Código que pode gerar exceções dos tipos
    // ExceptionType1 e ExceptionType2

}catch(ExceptionType1 opa1){
    // Código para lidar com uma exceção do tipo
    // ExceptionType1

}catch( ExceptionType2 opa2 ){
    // Código para lidar com uma exceção do tipo
    // ExceptionType2
```

```
    }finally{
        // Código que deve ser executado em qualquer caso
    }
```

Quando ocorre uma exceção, os blocos catch são examinados sucessivamente até que o argumento corresponda ao tipo da exceção. No exemplo acima, se ExceptionType2 for uma subclasse de ExceptionType1, o segundo bloco catch nunca será alcançado. Neste caso, deve-se inverter a ordem dos blocos catch.

O bloco finally é opcional, e normalmente é usado para realizar ações que independem da ocorrência de uma exceção em um dado bloco de comandos. Por exemplo, em um sistema que escreve informações em um banco de dados, é necessário fechar a conexão com este último ao final da operação de escrita, independentemente da operação ter sido bem-sucedida ou não. Tendo isso em vista, o comando responsável por fechar a conexão deve ficar dentro de um bloco finally.

16.2 Threads e Concorrência

Em programação concorrente existem duas unidades básicas de execução: processos e threads. Em java, os recursos para programação concorrente são mais voltados para o uso de threads.

Threads são comumente chamadas de processos leves. Processos e threads provêm um ambiente de execução, mas criar uma thread requer menos recursos que criar um processo. As threads existem dentro dos processos - cada processo possui pelo menos uma. As threads compartilham os recursos do processo, incluindo memória e arquivos abertos, favorecendo a eficiência da comunicação, porém a tornando potencialmente problemática.

Cada aplicação possui pelo menos uma thread - ou várias, se considerarmos as threads de sistema. Do ponto de vista do programador, uma aplicação começa com apenas uma única thread chamada main, e essa possui a habilidade para criar novas threads.

16.2.1 Definindo e Iniciando uma Thread

Existem duas formas básicas de se definir uma thread em Java: implementar a interface Runnable ou extender a classe Thread. O código para cada uma das formas segue abaixo:

```
//Implementando a interface Runnable
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }
}
```

```
public static void main(String args[]) {
    (new Thread(new HelloRunnable())).start();
}
}

//Extendendo a classe Thread
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

A primeira forma é mais geral e flexível, porque objetos Runnable podem herdar de outras classes diferentes de Thread. No entanto, a segunda forma é mais simples e funciona bem para aplicações que pouco complexas.

Em ambos os casos, a tarefa executada pela thread deve ser implementada no método Thread.run e para iniciá-la deve ser utilizado o método Thread.start.

16.2.2 Pausando a execução com sleep

Thread.sleep faz com que a thread corrente suspenda sua execução por um período de tempo específico. Essa uma forma de liberar o processador para outras threads ou processos que estejam rodando no sistema. O parâmetro do método Thread.sleep é o tempo em milissegundos.

```
public class SleepMessages {
    public static void main(String args[]) throws InterruptedException {
        String importantInfo[] = {
            "Mares eat oats",
            "Does eat oats",
            "Little lambs eat ivy",
            "A kid will eat ivy too"
        };

        for (int i = 0; i < importantInfo.length; i++) {
            //Pausa a thread corrente (main) por 4 segundos
            Thread.sleep(4000);
            //Imprime uma mensagem
            System.out.println(importantInfo[i]);
        }
    }
}
```

16.2.3 Interrupções

Uma interrupção indica que a thread deve parar o que está fazendo e começar a fazer outra coisa. É o programador quem deve definir como a thread deve responder a uma interrupção. O uso mais comum de interrupções é finalizar a execução de uma thread.

```
for (int i = 0; i < importantInfo.length; i++) {  
    //Pause for 4 seconds  
    try {  
        Thread.sleep(4000);  
    } catch (InterruptedException e) {  
        //We've been interrupted: no more messages.  
        return;  
    }  
    //Print a message  
    System.out.println(importantInfo[i]);  
}
```

16.2.4 Joins

O método join é usado para fazer com que uma thread aguarde até que outra thread termine. Em geral, o join é utilizado para a thread esperar pelo término da execução de suas threads filhas. Imagine uma aplicação que dependa da execução de três tarefas independentes A, B e C. A aplicação só pode continuar quando cada uma das tarefas estiver sido completada. Portanto, a aplicação (thread principal) deve executar o método join para cada uma das tarefas A,B e C. O código java para essa situação teria a seguinte estrutura:

```
public static void main(String args[]) {  
    System.out.println("Thread main iniciada");  
  
    //Código inicial da thread main  
  
    //Threads auxiliares A, B e C  
    Thread a = new thread_tarefaA();  
    Thread b = new thread_tarefaB();  
    Thread c = new thread_tarefaC();  
  
    //Iniciando as threads auxiliares  
    a.start();  
    b.start();  
    c.start();  
  
    //Aguardando as threads auxiliares terminarem  
    a.join();  
    b.join();  
    c.join();  
  
    //Código restante da thread main
```

```
        System.out.println("Thread main encerrada");
    }
```

16.2.5 Sincronização

As threads se comunicam entre si essencialmente por meio de compartilhamento no acesso de atributos e referências a objetos. Essa é uma forma de comunicação eficiente, porém permite a ocorrência de dois tipos de erros: interferência entre threads e erros de consistência de memória. Para contornar esses problemas o Java oferece o recurso de sincronização.

O problema da interferência ocorre quando duas operações, rodando em threads diferentes, mas atuando sobre o mesmo dado, se intercalam.

```
//Interferência entre threads

//Três threads que atuam sobre o mesmo dado
//Não é possível garantir a ordem de execução das operações
//pois em princípio, as threads são iniciadas ao mesmo tempo.

int a = 0;
thread_increments_a.start();
thread_decrements_a.start();
thread_imprime_a.start();
```

Os erros de inconsistência de memória ocorrem quando diferentes threads têm diferentes visões daquilo que deveria ser o mesmo dado. As causas dos erros de inconsistência de memória são complexas e estão fora do escopo dessa seção.

A chave para evitar esse tipo de erro está em entender o que o Java chama de relacionamento *happens-before*. Esse relacionamento consiste em garantir que a memória escrita por uma statement específica está visível para uma outra statement. Para criar esse tipo de relacionamento entre threads pode ser utilizado o método Thread.join, como discutido anteriormente. A linguagem Java também provê outros dois idiomas para sincronização que são: métodos sincronizados e statements sincronizadas.

Com os métodos sincronizados não é possível realizar duas chamadas simultâneas do método sobre um único objeto. Quando uma thread está executando um método sincronizado sobre um objeto, todas as outras threads que invocam métodos sincronizados para o mesmo objeto suspendem sua execução até que a primeira thread libere o objeto. Além disso, os métodos sincronizados estabelecem automaticamente uma relação de *happens-before* com todas as chamadas subsequentes de métodos sincronizados. Isso garante que as mudanças no estado do objeto estarão visíveis para todas as threads.

```
public class SynchronizedCounter {
```

```
private int c = 0;

//Métodos sincronizados para manipular um contador
public synchronized void increment() {c++;}
public synchronized void decrement() {c--;}
public synchronized int value() {return c;}

}
```

Ao contrário dos métodos sincronizados, as statements sincronizadas devem especificar para qual objeto deverá ser obtido o lock. Essa técnica pode ser útil quando é implementar concorrência com uma granularidade mais fina. Um exemplo de código para criação de statements sincronizadas é mostrado abaixo:

```
public class MsLunch {

    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1() {
        //Lock somente sobre o objeto lock1
        synchronized(lock1) {c1++;}
    }

    public void inc2() {
        //Lock somente sobre o objeto lock2
        synchronized(lock2) {c2++;}
    }
}
```

16.2.6 Executores e Thread Pools

Nos exemplos mostrados, existe uma conexão direta entre a tarefa executada por uma nova thread, como definida no objeto Runnable, e a nova thread em si, definida por um objeto Thread. Isso funciona bem para aplicações pequenas. Em aplicações de grande porte é mais apropriado separar a criação e gerenciamento da threads do resto da aplicação. Os objetos que encapsulam as funções de gerenciamento de threads são chamados executores (*executors*).

O pacote java.util.concurrent define três interfaces executoras que são Executor, ExecutorService e ScheduledExecutorService. Essas interfaces definem métodos para gerenciamento de threads na aplicação. Exemplo: Se r é um objeto do tipo Runnable, e e é um objeto do tipo Executor o trecho de código

```
(new Thread(r)).start();  
poderia ser substituído por  
e.execute(r);
```

Na primeira forma, uma nova thread seria criada para realizar a tarefa definida em r, enquanto na segunda, isso dependeria da implementação da interface executora.

Geralmente, as interfaces executoras são criadas utilizando as chamadas ThreadPools, que consistem de um conjunto de threads (*worker threads*) responsáveis pela execução das tarefas. Quando o método execute é chamado sobre o objeto r, uma worker thread é alocada para rodar a tarefa definida em r. Caso não exista nenhuma worker thread disponível no momento, r é colocada em uma fila de espera até que uma worker thread seja liberada.

Usando pools de threads é possível minimizar o overhead de criação de threads e otimizar o gerenciamento de memória, uma vez que é possível definir o número máximo de threads no pool de acordo com as necessidades da aplicação e quantidade de recursos do sistema.

16.3 Streams e Serialização

16.3.1 I/O Streams

Um I/O stream pode representar diferentes tipos de fonte e destino, incluindo arquivos em discos, dispositivos, outros programas e memória. Os streams suportam diversos tipos de dados, como simples bytes, tipos de dados primitivos e até mesmo objetos complexos. Alguns streams simplesmente passam os dados, enquanto outros realizam algum tipo de manipulação e transformação conforme necessário. A seguir serão mostrados alguns exemplos da utilização de streams.

Streams de Bytes

Byte streams são utilizados para realizar input e output de bytes. Todas as classes byte stream herdam de InputStream e OutputStream. Os streams de bytes representam o nível mais baixo que pode ser alcançado, e devem evitados. Todas as outras classes de stream são construídas com base na operações definidas pelas classes de byte stream.

```
//Exemplo 1: Stream de Bytes
//Copia o arquivo xanadu.txt para outagain.txt byte a byte.

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, bytesRead);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

```
        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
        }
    } finally {
    if (in != null) {in.close();}
    if (out != null) {out.close();}
}
}
```

Streams de Caracteres

A plataforma java armazena os valores dos caracteres utilizando a conveção Unicode. As operações de com streams de caracteres automaticamente convertem do formato Unicode para o formato local.

```
//Exemplo 1: Stream de Caracteres
//Copia o arquivo xanadu.txt para characteroutput.txt caracter e caracter

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");
            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {inputStream.close();}
            if (outputStream != null) {outputStream.close();}
        }
    }
}
```

No entanto, I/O de caracteres geralmente ocorrem linha a linha. Uma linha é um conjunto de caracteres terminada por um caractere especial de fim de linha. O código a seguir executa a mesma tarefa do exemplo 1, porém copia o arquivo linha a linha e utiliza estratégias de buffering.

```
//Exemplo 2: Stream de Caracteres linha a linha
//Copia o arquivo xanadu.txt para characteroutput.txt linha a linha

import java.io.FileReader;
```

```
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;

public class CopyLines {
    public static void main(String[] args) throws IOException {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;
        try {
            inputStream = new BufferedReader(new FileReader("xanadu.txt"));
            outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));
            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) {inputStream.close();}
            if (outputStream != null) {outputStream.close();}
        }
    }
}
```

No exemplo 1 o código utiliza operações de I/O não bufferizadas. Isso significa que a cada requisição de leitura ou escrita expressa no código é prontamente tratada pelo sistema operacional, disparando, por exemplo, acesso ao disco ou à rede.

Para reduzir esse tipo de overhead a plataforma Java implementa I/O streams bufferizados. Os input streams lêem dados de uma área de buffer; a API input nativa só é chamada quando esse buffer está vazio ou não contém os dados procurados. Similarmente, os output streams escrevem dados para uma área de buffer, enquanto a API nativa de output só é chamada quando o buffer de escrita está cheio ou quando algum comando que implique no *flushing* é utilizado, por exemplo, println.

16.3.2 Serialização - Streams de Objetos

Os streams apresentados até agora são chamados data streams. Assim como os data streams suportam I/O de tipos primitivos de dados, os object streams suportam I/O para objetos. A maioria das classes padrão do java suportam a serialização de seus objetos. Para que uma objetos de uma classe possam ser serializados a classe deve implementar a interface Serializable. A interface Serializable não define métodos ou atributos, servindo apenas para indicar que uma classe é serializável.

As classes de streams de objetos são ObjectInputStream e ObjectOutputStream. Essas classes implementam as interfaces ObjectInput e ObjectOutput, que por sua vez são subinterfaces de DataInput and DataOutput. Isso significa dizer que todos os métodos para I/O de tipos de dados primitivos também são

implementados nas classes de stream para objetos. A seguir, um exemplo simples para serialização de objetos em Java.

```
import java.io.*;

public class Pessoa implements Serializable{

    public String nome = "Maria";
    public int idade = 30;
    public String sexo = "feminino";

    public Pessoa (String p_nome, int p_idade, String p_sexo){
        nome = p_nome;
        idade = p_idade;
        sexo = p_sexo;
    }

    public static void main (String [] args){

        Pessoa p = new Pessoa("Maria",10,"feminino");

        try {
            // Serializa o objeto para um arquivo
            ObjectOutputStream out = new ObjectOutputStream(
                new FileOutputStream("pessoa.ser")
            );
            out.writeObject(p);
            out.close();

            // Dessaerializa o objeto a partir do arquivo
            File file = new File("pessoa.ser");
            ObjectInputStream in = new ObjectInputStream(
                new FileInputStream(file)
            );
            Pessoa clone = (Pessoa) in.readObject();
            in.close();

            // Testa um atributo do novo objeto clone
            System.out.println("Nome de p: " + p.nome);
            System.out.println("Nome do clone de p: " + clone.nome);

        }catch (IOException e) {
            //Pode ser gerado na serialização ou desserialização
            System.out.println(e.getMessage());
        }catch (ClassNotFoundException e) {
            //Pode ser gerado desserialização
            System.out.println(e.getMessage());
        }
    }
}
```

}

Embora as classes ObjectInputStream e ObjectOutputStream sejam suficientes para serializar e desserializar a maioria dos objetos no Java, ela possui o incoveniente de utilizar arquivos binários, o que dificulta a leitura direta por um usuário ou por outras aplicações (escritas em outras linguagens, por exemplo). Muitas aplicações realizam serialização com arquivos XML, o que permite contornar os problemas da legibilidade e integração mais facilmente.

Existem inúmeros pacotes para serialização de objetos em arquivos XML. Quando se está trabalhando com java beans, por exemplo, duas classes muito utilizadas para serialização de objetos em XML são java.beans.XMLEncoder e java.beans.XMLDecoder. A seguir, um trecho de código exemplificando o uso dessas classes.

```
PessoaXML p = new PessoaXML("Maria",10,"feminino");

// Serializa o objeto para um arquivo XML
XMLEncoder e = new XMLEncoder(
    new BufferedOutputStream(
        new FileOutputStream("pessoa.xml")));
e.writeObject(p);
e.close();

// Dessorializa o objeto a partir do arquivo XML
XMLDecoder d = new XMLDecoder(
    new BufferedInputStream(
        new FileInputStream("pessoa.xml")));
PessoaXML clone = (PessoaXML) d.readObject();
d.close();
```

16.4 Classes e Operações de I/O

16.5 Classes para manipulação de propriedades

Boa parte das aplicações necessitam de algum tipo de configuração adicional que não pode ser embutida diretamente no código. Em java, uma forma tradicional de se fazer isso envolve a utilização de arquivos de propriedades, que podem ser manipulados a partir da classe java.util.Properties, que é uma extensão da classe java.util.Hashtable.

Um arquivo de propriedades guarda conjunto de pares nome/valor, que podem ser lidos ou alterados. O formato típico de um arquivo de propriedades é mostrado a seguir:

```
#Esta é uma linha de comentário
#Mon Jun 11 22:37:16 BRT 2007
app.diretorio=/zaz/traz/bin
app.usuario=camatchones
app.mododeoperacao=producao
```

E a seguir é mostrado um exemplo de código Java para manipulação de um arquivo de propriedades.

```
import java.util.Properties;
import java.io.*;

public class Exemplo {

    public Exemplo (){}

    public static void main (String [] args){
        try{
            // Cria e le os valores padrao para um objeto Propertie
            Properties defaultProps = new Properties();
            FileInputStream in = new FileInputStream("exemplo.properties");
            defaultProps.load(in);
            in.close();

            // Utilizacao as propriedades quando precisarmos
            String diretorio = defaultProps.getProperty("app.diretorio");
            String usuario = defaultProps.getProperty("app.usuario");

            // Modificamos/Criando valores das propriedades
            if (usuario.equals("joao")){
                defaultProps.setProperty("app.dataDir", "/home/joao");
                defaultProps.setProperty("app.nova_configuracao", "12345");
            }

            // Salvamos para uma proxima execussao
            FileOutputStream out = new FileOutputStream("exemplo.properties");
            defaultProps.store(out, "Isso é um comentário!");
            out.close();

        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

Uma observação importante é que o método getProperty sempre retorna uma String, de modo que se uma propriedade é de algum outro tipo, essa deve ser convertida no momento da leitura.

Atualmente o uso de arquivos XML para armazenamento de configurações

vem ganhando popularidade, no entanto, os arquivos de propriedades continuam ser uma forma simples e eficaz de gerenciar configurações simples.

Capítulo 17

Coleções

Uma coleção é simplesmente um grupo de objetos em uma única unidade. As coleções são usadas para armazenar, recuperar, manipular e comunicar dados agregados. Tipicamente, representam itens de dados que dão forma a um grupo natural, tal como uma mão do poker (uma coleção dos cartões) e uma pasta de correspondência (uma coleção das letras).

Uma framework de coleções é uma arquitetura unificada para representação e manipulação das coleções, permitindo que elas sejam manipuladas independentemente dos detalhes de sua implementação. Todos os frameworks de coleções contêm:

- **Interfaces:** Estes são os tipos de dados abstratos que representam as coleções. As interfaces permitem que as coleções sejam manipuladas independentemente dos detalhes de sua representação. Em linguagem orientadas à objeto as interfaces dão forma geralmente a uma hierarquia;
- **Implementações:** Estas são as implementações concretas das relações da coleção. Essencialmente, são estruturas de dados reusáveis;
- **Algoritmos:** Estes são os métodos que executam computações úteis, tais como procurar e ordenar objetos de uma determinada coleção. Os algoritmos são polimórficos, isto é, o mesmo método pode ser usado em diferentes implementações.

As principais vantagens do framework de coleções (collection framework) são:

- Redução do esforço de programação, fornecendo estruturas de dados e algoritmos úteis, para que não seja necessário reescrevê-los;
- Aumento da performance: fornecendo implementações de alta performance. Como as várias implementações de cada interface são substituíveis, os programas podem ser facilmente refinados trocando-se as implementações;
- Interoperabilidade entre APIs não relacionadas, estabelecendo uma linguagem comum para passagem de coleções;

- Redução do esforço de aprendizado de APIs, eliminando a necessidade de se aprender várias APIs de coleções diferentes;
- Redução do esforço de projetar e implementar APIs, eliminando a necessidade de se criar APIs de coleções próprias;
- Promover o reuso de software, fornecendo uma interface padrão para coleções e algoritmos para manipulá-los.

As coleções são necessárias, devido o uso dos arrays possuir uma série de limitações, como por exemplo:

- Um array não pode ter o tamanho modificado depois de criado;
- Somente pode conter elementos de um mesmo tipo;
- Para inserir ou retirar um elemento é necessário modificar a posição de outros elementos.

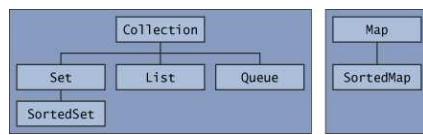


Figura 17.1: Interfaces Collection

17.1 Interface Collection

Interface base para todos os tipos de coleção. Ela define as operações mais básicas para coleções de objetos, como adição (`add`) e remoção (`remove`) abstratos (sem informações quanto à ordenação dos elementos), esvaziamento (`clear`), tamanho (`size`), conversão para array (`toArray`), objeto de iteração (`iterator`), e verificações de existência (`contains` e `isEmpty`). A seguir é mostada a Interface Collection:

```
public interface Collection<E> extends Iterable<E> {  
  
    // Basic operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(E element);           //optional  
    boolean remove(Object element); //optional  
    Iterator<E> iterator();  
  
    // Bulk operations  
    boolean containsAll(Collection<?> c);  
    boolean addAll(Collection<? extends E> c); //optional
```

```
boolean removeAll(Collection<?> c);           //optional
boolean retainAll(Collection<?> c);           //optional
void clear();                                //optional

// Array operations
Object[] toArray();
<T> T[] toArray(T[] a);

}
```

17.2 Interface Set

Interface que define uma coleção, ou conjunto, que não contém objetos duplicados. Isto é, são ignoradas as adições caso o objeto ou um objeto equivalente já exista na coleção. Por objetos equivalentes, entenda-se objetos que tenham o mesmo código hash (retornado pelo método hashCode()) e que retornem verdadeiro na comparação feita pelo método equals().

Não é garantida a ordenação dos objetos, isto é, a ordem de iteração dos objetos não necessariamente tem qualquer relação com a ordem de inserção dos objetos. Por isso, não é possível indexar os elementos por índices numéricos, como em uma List. A seguir é mostada a Interface Set:

```
public interface Set<E> extends Collection<E> {

    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);           //optional
    boolean remove(Object element);  //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);          //optional
    boolean retainAll(Collection<?> c);          //optional
    void clear();                                //optional

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);

}
```

Existem 3 implementações da interface Set:

- HashSet: Implementação de Set que utiliza uma tabela hash para guardar seus elementos. Não garante a ordem de iteração, nem que a ordem permanecerá constante com o tempo (uma modificação da coleção pode alterar a ordenação geral dos elementos). Por utilizar o algoritmo de tabela hash, o acesso é rápido, tanto para leitura quanto para modificação.
- LinkedHashSet: Implementação de Set que estende HashSet, mas adiciona previsibilidade à ordem de iteração sobre os elementos, isto é, uma iteração sobre seus elementos (utilizando o Iterator) mantém a ordem de inserção (a inserção de elementos duplicados não altera a ordem anterior). Internamente, é mantida uma lista duplamente encadeada que mantém esta ordem. Por ter que manter uma lista paralelamente à tabela hash, a modificação deste tipo de coleção acarreta em uma leve queda na performance em relação à HashSet, mas ainda é mais rápida que uma TreeSet, que utiliza comparações para determinar a ordem dos elementos.
- TreeSet: Implementação de SortedSet. SortedSet é uma interface que estende Set, adicionando a semântica de ordenação natural dos elementos. A posição dos elementos no percorrido da coleção é determinado pelo retorno do método compareTo(o), caso os elementos implementem a interface Comparable, ou do método compare(o1,o2) de um objeto auxiliar que implemente a interface Comparator. A TreeSet utiliza internamente uma TreeMap, que por sua vez utiliza o algoritmo Red-Black para a ordenação da árvore de elementos. Isto garante a ordenação ascendente da coleção, de acordo com a ordem natural dos elementos, definida pela implementação da interface Comparable ou Comparator. Use esta classe quando precisar de um conjunto (de elementos únicos) que deve estar sempre ordenado, mesmo sofrendo modificações.

A seguir são mostrados alguns exemplos:

```
import java.util.*;  
  
public class FindDups {  
  
    public static void main(String[] args) {  
        Set<String> s = new HashSet<String>();  
        for (String a : args)  
            if (!s.add(a))  
                System.out.println("Duplicate detected: " + a);  
        System.out.println(s.size() + " distinct words: " + s);  
    }  
}
```

Rodando o programa:

```
prompt#> java FindDups i came i saw i left
```

A saída produzida será:

```
Duplicate detected: i  
Duplicate detected: i  
4 distinct words: [i, left, saw, came]
```

O tipo de implementação Set no exemplo anterior é HashSet, que como dito anteriormente não faz nenhuma garantia a respeito da ordem dos elementos no Set. Caso seja necessário imprimir a lista em ordem alfabética basta mudar o tipo de implementação de HashSet para TreeSet. A nova saída do programa anterior para TreeSet seria a seguinte:

Rodando o programa:

```
prompt#> java FindDups i came i saw i left
```

A saída produzida será:

```
Duplicate detected: i  
Duplicate detected: i  
4 distinct words: [came, i, left, saw]
```

Caso fosse utilizado uma LinkedHashSet a saída seria a seguinte:

Rodando o programa:

```
prompt#> java FindDups i came i saw i left
```

A saída produzida será:

```
Duplicate detected: i  
Duplicate detected: i  
4 distinct words: [i, came, saw, left]
```

E agora um outro exemplo, modificando a classe FindDups:

```
import java.util.*;  
  
public class FindDups2 {  
  
    public static void main(String[] args) {  
        Set<String> uniques = new HashSet<String>();  
        Set<String> dups = new HashSet<String>();  
        for (String a : args)  
            if (!uniques.add(a))  
                dups.add(a);  
    }  
}
```

```
        dups.add(a);
    // Destructive set-difference
    uniques.removeAll(dups);
    System.out.println("Unique words: " + uniques);
    System.out.println("Duplicate words: " + dups);
}
}
```

Executando o código anterior com os mesmos argumentos dos programas anteriores (i came i saw i left) a saída deste programa seria:

```
Unique words: [left, saw, came]
```

```
Duplicate words: [i]
```

Há duas implementações Set com finalidades especiais: EnumSet e CopyOn-WriteArrayList.

17.3 Interface List

Interface que extende Collection, e que define coleções ordenadas (seqüências), onde se tem o controle total sobre a posição de cada elemento, identificado por um índice numérico. Na maioria dos casos, pode ser encarado como um "array de tamanho variável", pois, como os arrays primitivos, é acessível por índices, mas além disso possui métodos de inserção e remoção. A seguir é mostada a Interface List:

```
public interface List<E> extends Collection<E> {

    // Positional access
    E get(int index);
    E set(int index, E element);      //optional
    boolean add(E element);          //optional
    void add(int index, E element);  //optional
    E remove(int index);            //optional
    boolean addAll(int index,
    Collection<? extends E> c);     //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
```

}

Existem 2 implementações da interface List:

- **ArrayList:** Implementação de List que utiliza internamente um array de objetos. Em uma inserção onde o tamanho do array interno não é suficiente, um novo array é alocado (de tamanho igual a 1.5 vezes o array original), e todo o conteúdo é copiado para o novo array. Em uma inserção no meio da lista o conteúdo posterior ao índice é deslocado em uma posição. Esta implementação é a recomendada quando o tamanho da lista é previsível (evitando realocações) e as operações de inserção e remoção são feitas, em sua maioria, no fim da lista (evitando deslocamentos), ou quando a lista é mais lida do que modificada (otimizado para leitura aleatória).
- **LinkedList:** Implementação de List que utiliza internamente uma lista encadeada. A localização de um elemento na n-ésima posição é feita percorrendo-se a lista da ponta mais próxima até o índice desejado. A inserção é feita pela adição de novos nós, entre os nós adjacentes, sendo que antes é necessária a localização desta posição. Esta implementação é recomendada quando as modificações são feitas em sua maioria tanto no início quanto no final da lista, e o percorrimento é feito de forma seqüencial (via Iterator) ou nas extremidades, e não aleatória (por índices). Um exemplo de uso é como uma fila (FIFO - First-In-First-Out), onde os elementos são retirados da lista na mesma seqüência em que são adicionados.

A seguir são mostrados alguns exemplos:

```
import java.util.*;

public class ExemploArrayList {

    public static void main(String[] args) {
        List c = new ArrayList();
        c.add("Maria");
        c.add("Joao");
        c.add("Ana");
        c.add("Joao");
        c.add("Jose");
        Iterator i = c.iterator();
        while( i.hasNext() ) {
            System.out.print( i.next() + " " );
        }
        System.out.println();
        System.out.println(c.get(2));
        System.out.println(c);
    }
}
```

A saída produzida será:

```
Maria  Joao  Ana  Joao  Jose  
Ana  
[Maria, Joao, Ana, Joao, Jose]
```

ArrayList e LinkedList tem algumas diferenças, entre elas temos os métodos addFirst, addLast, getFirst, getLast, removeFirst e removeLast para LinkedList que inserem, retornam e removem um elemento no inicio ou no fim de uma lista , com estes métodos podemos simular pilhas e filas.

Há uma implementação List com finalidades especiais: CopyOnWriteArrayList.

17.4 Interface Map

Interface que define um array associativo, isto é, ao invés de números, objetos são usados como chaves para se recuperar os elementos. As chaves não podem se repetir (segundo o mesmo princípio da interface Set), mas os valores podem ser repetidos para chaves diferentes. Um Map também não possui necessariamente uma ordem definida para o percorrimento. A seguir é mostrada a Interface Map:

```
public interface Map<K,V> {  
  
    // Basic operations  
    V put(K key, V value);  
    V get(Object key);  
    V remove(Object key);  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    int size();  
    boolean isEmpty();  
  
    // Bulk operations  
    void putAll(Map<? extends K, ? extends V> m);  
    void clear();  
  
    // Collection Views  
    public Set<K> keySet();  
    public Collection<V> values();  
    public Set<Map.Entry<K,V>> entrySet();  
  
    // Interface for entrySet elements  
    public interface Entry {  
        K getKey();  
        V getValue();  
        V setValue(V value);  
    }  
}
```

}

Existem 3 implementações da interface Map:

- HashMap: Implementação de Map que utiliza uma tabela hash para armazenar seus elementos. O tempo de acesso aos elementos (leitura e modificação) é constante (muito bom) se a função de hash for bem distribuída, isto é, a chance de dois objetos diferentes retornarem o mesmo valor pelo método hashCode() é pequena.
- TreeMap: Implementação de SortedMap. SortedMap é uma Interface que estende Map, adicionando a semântica de ordenação natural dos elementos, análogo à SortedSet. Também adiciona operações de partição da coleção, com os métodos headMap(k) - que retorna um SortedMap com os elementos de chaves anteriores a k -, subMap(k1,k2) - que retorna um SortedMap com os elementos de chaves compreendidas entre k1 e k2 - e tailMap(k) - que retorna um SortedMap com os elementos de chaves posteriores a k. A TreeMap utiliza o algoritmo Red-Black para a ordenação da árvore de elementos. Isto garante a ordenação ascendente da coleção, de acordo com a ordem natural dos elementos, definida pela implementação da interface Comparable ou Comparator. Use esta classe quando precisar de um conjunto (de elementos únicos) que deve estar sempre ordenado, mesmo sofrendo modificações.
- LinkedHashMap: Implementação de Map que estende HashMap, mas adiciona previsibilidade à ordem de iteração sobre os elementos, isto é, uma iteração sobre seus elementos (utilizando o Iterator) mantém a ordem de inserção (a inserção de elementos duplicados não altera a ordem anterior). Internamente, é mantida uma lista duplamente encadeada que mantém esta ordem. Por ter que manter uma lista paralelamente à tabela hash, a modificação deste tipo de coleção acarreta em uma leve queda na performance em relação à HashMap, mas ainda é mais rápida que uma TreeMap, que utiliza comparações para determinar a ordem dos elementos.

A seguir são mostrados alguns exemplos:

```
import java.util.*;

public class Freq {

    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();
        // Initialize frequency table from command line
        for (String a : args) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println(m.size() + " distinct words:");
        System.out.println(m);
    }
}
```

}

Rodando o programa:

```
prompt#> java Freq if it is to be it is up to me to delegate
```

A saída produzida será:

8 distinct words:

```
{to=3, delegate=1, be=1, it=2, up=1, if=1, me=1, is=2}
```

Caso fosse usado uma TreeMap ao invés de HashMap a saída seria:

8 distinct words:

```
{be=1, delegate=1, if=1, is=2, it=2, me=1, to=3, up=1}
```

E caso fosse utilizado LinkedHashMap a saída seria:

8 distinct words:

```
{if=1, it=2, is=2, to=3, be=1, up=1, me=1, delegate=1}
```

Há duas implementações Map com finalidades especiais: EnumMap, WeakHashMap e IdentityMap.

17.5 Interface Queue

Uma queue é uma coleção para prender elementos antes de processarem. Além das operações básicas da coleção, as filas fornecem a inserção, a remoção e operações adicionais de inspeção. A seguir é mostrada a Interface Queue:

```
public interface Queue<E> extends Collection<E> {  
  
    E element();  
    boolean offer(E e);  
    E peek();  
    E poll();  
    E remove();  
  
}
```

Queue tipicamente, mas não necessariamente, manipulam os elementos como uma FIFO (first-in-first-out). Entre as exceções estão as priority queues (filas com prioridades), que requisitam elementos de acordo com algum de seus valores.

O primeiro elemento da fila (cabeça da fila) é o elemento que seria removido pela chamada do método remove ou poll. Em uma queue do tipo FIFO, todos os elementos novos são introduzidos no final da fila. Outros tipos de queue

podem usar regras diferentes de colocação. Cada implementação de queue deve especificar suas propriedades.

É possível que algumas implementações de queue restrinja o número de elementos que prende; tais filas são conhecidas como limitadas. Algumas implementações de queue em java.util.concurrent são limitadas, mas as implementações em java.util não são.

As implementações da queue geralmente não permitem a inserção de elementos nulos. A implementação de LinkedList, que foi adaptada para implementar uma queue, é uma exceção.

A seguir é mostrado um exemplo:

```
import java.util.*;  
  
public class Countdown {  
  
    public static void main(String[] args) throws InterruptedException {  
        int time = Integer.parseInt(args[0]);  
        Queue<Integer> queue = new LinkedList<Integer>();  
        for (int i = time; i >= 0; i--)  
            queue.add(i);  
        while (!queue.isEmpty()) {  
            System.out.println(queue.remove());  
            Thread.sleep(1000);  
        }  
    }  
}
```

A classe PriorityQueue é uma fila de prioridade baseada na estrutura de dados heap. Esta fila requisita elementos de acordo com uma ordem especificada no tempo da construção.

A interface queue é estendida pela BlockingQueue. Algumas implementações de BlockingQueue são: LinkedBlockingQueue, ArrayBlockingQueue, PriorityBlockingQueue, DelayQueue e a SynchronousQueue.

Capítulo 18

JDBC - Java Database Connectivity

18.1 Conceitos Básicos

Java Database Connectivity ou JDBC é um conjunto de classes e interfaces (API) escritas em Java que faz o envio de instruções SQL para qualquer banco de dados relacional.

- API de baixo nível e base para APIs de alto nível;
- Amplia o que você pode fazer com java;
- Possibilita o uso de banco de dados já instalados;
- Para cada banco de dados há um driver JDBC que pode cair em quatro categorias.

As 4 categorias de driver são:

- TIPO 1: PONTE JDBC-ODBC – É o tipo mais simples mas restrito à plataforma Windows. Utiliza ODBC para conectar-se com o banco de dados, convertendo métodos JDBC em chamadas às funções do ODBC. Esta ponte é normalmente usada quando não há um driver puro-java (Tipo 4) para determinado banco de dados, pois seu uso é desencorajado devido à dependência de plataforma.
- TIPO 2: DRIVER API-NATIVO – O driver API-Nativo traduz as chamadas JDBC para as chamadas da API cliente do banco de dados usado (Oracle, Sybase, Informix, DB2, ou outro SGBD). Como a ponte JDBC-ODBC, pode precisar de software extra instalado na máquina cliente.
- TIPO 3: DRIVER DE PROTOCOLO DE REDE – Traduz a chamada JDBC para um protocolo de rede independente do banco de dados utilizado, que é traduzido para o protocolo do banco de dados por um servidor. Por utilizar um protocolo independente, pode conectar as aplicações java a vários banco de dados diferente. É o modelo mais flexível.

- TIPO 4: DRIVER NATIVO – Converte as chamadas JDBC diretamente no protocolo do banco de dados. Implementado em java, normalmente é independente de plataforma e escrito pelos próprios desenvolvedores. É o tipo mais recomendado para ser usado.

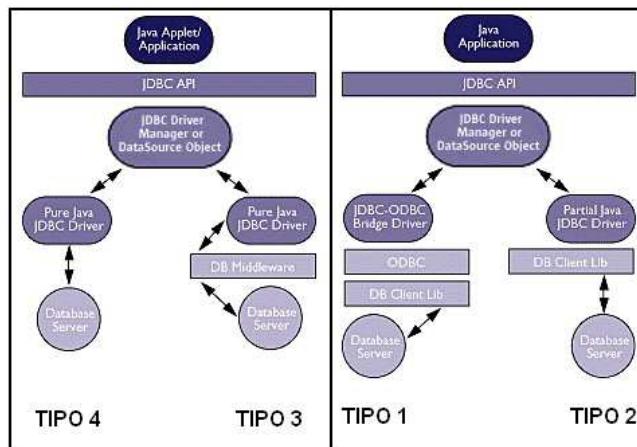


Figura 18.1: Tipos de Drivers JDBC

Como mostrado na figura 18.1 a aplicação java chama a biblioteca JDBC. Posteriormente o JDBC carrega um driver para que se comunique com a base de dados.

18.2 Carregamento de drivers

Depois de iniciar o programa e importar os pacotes necessários (`import java.sql.*`), devemos seguir certos passos para que seja possível o acesso ao SGBD. Deveremos inicialmente carregar o driver e depois criar a conexão propriamente dita. A carga do driver é muito simples. Basta invocar o método estático: `Class.forName("jdbc.XYZ")`. Exemplo:

```
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //Or any other driver
} catch(Exception x){
    System.out.println( "Unable to load the driver class!" );
}
```

A classe `Class` tem como função instanciar a classe identificada entre parênteses e registrá-la com a JVM, que irá utilizá-la para acessar o SGBD. Em nosso exemplo, utilizamos a "ponte" ODBC-JDBC. A `Class` é uma classe no pacote de `java.lang`.

18.3 Conexão

Feito o carregamento dos drivers, o próximo passo é o estabelecimento da conexão. Tipicamente uma aplicação JDBC conecta ao banco de dados usando

um dos seguintes mecanismos:

- DriverManager: Esta implementação requer que uma aplicação carregue um driver específico, usando uma URL "hardcoded". Como parte de sua inicialização, a classe DriverManager tenta carregar as classes do driver referenciado. Isto permite que você customize os drivers de JDBC usados por suas aplicações.
- DataSource: Esta interface é preferida sobre a DriverManager porque permite que os detalhes sobre a origem dos dados sejam transparentes a sua aplicação. As propriedades de um objeto de DataSource são ajustadas de modo que represente uma origem particular de dados.

Para abrirmos uma conexão, normalmente teremos que entrar em contato com o SGBD. A linha mostrada abaixo dá a ideia geral de como é feita a conexão através do mecanismo DriverManager:

```
try{  
    Connection con = DriverManager.getConnection(url, "user", "password");  
}catch( SQLException x ){  
    System.out.println( "Couldn't get connection!" );  
}
```

A parte mais complexa é montar a url. Esta parte da informação sempre se inicia com jdbc: mas o resto depende do fabricante do banco. Na seqüência normalmente temos o nome do fabricante (então temos jdbc:db2, jdbc:oracle, jdbc:pgsql e etc) e por último o nome da fonte de dados (então temos jdbc:db2:db, jdbc:oracle:server.unicamp.br:1234/meubanco e etc).

O próximo exemplo mostra como seria o estabelecimento de conexão através de um DataSource.

```
InitialContext ic = new InitialContext()  
DataSource ds = ic.lookup("java:comp/env/jdbc/myDB");  
Connection con = ds.getConnection();
```

Toda parte envolvida com a conexão propriamente dita acontece de maneira transparente. O programador não tem necessidade nenhuma de saber o que se passa e, a não ser que ele um dia resolva escrever seu próprio driver para conexão a algum SGBD, tem que se dar por satisfeito da coisa acontecer assim.

A partir de agora, o objeto con representa uma conexão ativa e aberta ao banco de dados e o programador pode utilizá-lo para enviar statements SQL para o banco.

18.4 Statements

É necessário criar um objeto da classe Statement para armazenar o pedido que será enviado ao SGBD. O programador deve simplesmente criar um objeto deste tipo, informar a cadeia de strings que representa seu comando SQL e depois executá-lo, usando métodos especiais dependendo do tipo de comando desejado. Para um SELECT, usa-se o executeQuery. Para INSERT, UPDATE e DELETE executeUpdate. Por exemplo:

```
Statement stmt = com.createStatement();
```

Neste ponto, um statement foi criado. Mas não existe nenhum comando SQL nele. Podemos executar um comando como no exemplo abaixo:

```
stmt.executeUpdate( "DELETE FROM Clientes WHERE CPF = '" + cpf + "'");
```

Note que existe uma estrutura de parênteses, sinais de concatenação e aspas. Esses sinais são importantes e muitas vezes causam confusões tremendas em programas. Deve-se ter muito cuidado com eles, principalmente quando houver variáveis do programa java misturadas ao SQL.

O método mais comum de ser utilizado é o executeQuery. Mas ele retorna apenas dados. Assim, devemos ter outro tipo de classe que serve exclusivamente para receber esses dados, de maneira que possamos fazer uso deles no processamento. Essa classe é chamada ResultSet. É necessário instanciar um objeto desta classe que receba o resultado do executeQuery. Veja o exemplo:

```
ResultSet rs = stmt.executeQuery("SELECT NOME, SOBRENOME, IDADE FROM PESSOA");
```

A partir de agora, tem-se um objeto chamado rs que armazena todos os dados recebidos. Em linhas gerais, os programadores estabelecem um loop onde fazer a leitura dos dados baseados em métodos presentes na classe ResultSet. Esses métodos permitem "andar" pelos resultados e copiar os valores para outros objetos.

```
...
String query = "SELECT NOME, IDADE FROM PESSOA";
ResultSet rs = stmt.executeQuery(query);
While(rs.next()){
    String nome = rs.getString("NOME");
    Int idade = rs.getInt("IDADE");
...
}
```

Para acessar os nomes e as idades corremos cada registro e acessamos os valores de acordo com o tipo. O método next() move o cursor para o próximo registro, tornando aquele registro o corrente. Podemos então executar outros métodos. Note que o cursor inicialmente fica posicionado acima do primeiro registro, exigindo uma primeira execução do next() para que possamos processar o primeiro registro.

Utilizamos métodos cujo nome se iniciam com get para obter os dados das colunas. Eles devem ser obtidos de acordo com o tipo (o cast pode ser efetuado, quando houver possibilidade), e esses métodos permitem que apontemos as colunas pelo nome ou pelo índice (seu número, na hora em que foi recebida e não na ordem da tabela original). Assim, o código anterior poderia ser:

```
String nome = rs.getString(1);
Int idade = rs.getInt(2);
```

Esta ultima abordagem tem uma melhor performance. Caso o programador queira atualizar o atributo nome da quinta linha do resultSet poderia fazer da seguinte maneira.

```
rs.absolute(5); //move o cursor para o quinto registro
rs.updateString("NOME", "Joao"); //altera o valor da coluna NOME
rs.updateRow(); //atualiza a coluna na tabela
```

18.5 Prepared Statements

Às vezes é mais conveniente usar um objeto PreparedStatement para enviar comandos sql para a base de dados. Este tipo especial de statement é derivado da classe mais geral, Statement.

A principal diferença é que o prepared statement contém não apenas um comando SQL, mas sim um comando SQL pré-compilado. Isso significa que quando o prepared statement é executado, o SGBD deve apenas rodar o comando SQL sem ter que fazer sua compilação primeiro.

Os objetos PreparedStatements podem ser usados sem parâmetros no comando SQL, embora seja mais freqüente o uso com parâmetros. A vantagem da utilização de parâmetros é que você pode fornecer valores diferentes ao mesmo statement a cada vez que você executá-lo.

```
PreparedStatement pstmt = conn.prepareStatement("insert into TABLE values (?,?)");
pstmt.setString(1,arquivo.getName());
pstmt.setBinaryStream(2,inputStream,(int)arquivo.length());
pstmt.executeUpdate();
```

Note que você pode possuir diversos tipos de campos diferentes, e nem por isso precisa usar aspas duplas, aspas simples ou nenhuma aspas na hora de passar os valores para serem incluídos/consultados/alterados no banco. Esta é uma das principais vantagens de se usar PreparedStatement.

Lembre-se que a ordem dos sets deverão ser dadas na mesma ordem que foi inserido os sinais de interrogação. A seguir é mostrada uma operação de UPDATE usando Statement e PreparedStatement.

```
//Usando Statement

Statement stmt = con.createStatement();
String sql = "UPDATE COFFEES SET SALES = 75 WHERE COF_NAME LIKE 'Colombian'";
stmt.executeUpdate(sql);

//Usando Prepared Statement

PreparedStatement pstmt = con.prepareStatement(
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
pstmt.executeUpdate();
```

18.6 Transação

Existem situações onde não se deseja que uma instrução tenha efeito a não ser que outras também tenham. Algumas situações necessitam que dados sejam incluídos em mais de uma tabela para que sejam considerados consistentes. Ou os dados estão presentes em todas as tabelas ou não devem estar presentes em nenhuma.

A técnica utilizada para garantir esse tipo de consistência e o uso de transações. Uma transação nada mais é do que statements que são executados juntos, como uma coisa única: ou todos falham, ou todos são executados.

A primeira providência é desabilitar o auto-commit. Quando uma conexão é criada, o padrão é que ela trate cada executeUpdate como uma transação separada, que é validada (commit) assim que é completada. Assim, devemos utilizar o código:

```
con.setAutoCommit(false);
```

A partir daqui, nenhum statement será validado até que o programador permita. Todos eles serão agrupados e validados como um só. Ao final do processo (caso tudo tenha transcorrido de acordo com o esperado), executa-se:

```
con.commit();
```

Caso algo não tenha dado certo, podemos executar o método rollback para que todo o statement seja descartado.

```
con.rollback();
```

É importante lembrar algumas coisas quando lidamos com a transação de maneira manual. Os recursos de banco normalmente ficam presos, esperando um commi() ou rollback(). Assim, se um programa tratar ou entrar em loop (ou mesmo demorar muito) o acesso ao banco fica prejudicado. Existe ainda a possibilidade de um "dirty read", onde outro programa recupera dados do disco que você está alterando, mas ainda não validou. Esse tipo de comportamento pode ser evitado aumentando o nível de isolamento entre transações. Verifique o método setTransactionIsolation() da classe Connection.

18.7 Informações Complementares

Ocasionalmente, você irá sentir necessidade de obter mais informações sobre o resultado de um SELECT ou mesmo sobre a tabela em questão. Pode-se obter esses dados utilizando a classe ResultSetMetaData, que pode ser criada segundo o exemplo:

```
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData meta = rs.getMetaData();
```

A classe ResultSetMetaData lhe fornece informações acerca do número de colunas recebidas, o nome e o tipo das colunas, se alguma coluna aceita dados nulos, campos de data etc. Além disso fornece também informações sobre a tabela com que estamos trabalhando. Abaixo segue um exemplo mais completo:

```
...
Statement stmt = conn.createStatement();

// Tabela a ser analisada
ResultSet rset = stmt.executeQuery("SELECT * from EMP ");
ResultSetMetaData rsmd = rset.getMetaData();
```

```
// retorna o numero total de colunas
int numColumns = rsmd.getColumnCount();
System.out.println("Total de Colunas = " + numColumns);

// loop para recuperar os metadados de cada coluna
for (int i=0; i<numColumns; i++) {
    System.out.println("Nome da Coluna=" + rsmd.getColumnName(i + 1));
    System.out.println(" Tipo=" + rsmd.getColumnType(i + 1) );
    System.out.println(" Nome do Tipo=" + rsmd.getColumnTypeName(i + 1));
    System.out.println(" Tamanho=" + rsmd getColumnDisplaySize(i + 1));
    System.out.println(" Casas Decimais=" + rsmd.getScale(i + 1));
}
```

18.8 Exemplo Extra

Abaixo segue um exemplo extra com os assuntos tratados neste capítulo.

```
import java.sql.*;

public class Principal{

    public static void main(String[] args) throws Exception
        //carregando o driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        //estabelecendo a conexão
        Connection con = DriverManager.getConnection("jdbc:obc:bd");

        //criando statement
        Statement stmt = con.createStatement();

        //Executando a consulta
        ResultSet rs = stmt.executeQuery("SELECT LOGIN, NOME FROM ATENDENTE");

        //Percorrendo o resultset
        while(rs.next()){
            String login = rs.getString("LOGIN");
            String nome = rs.getString("NOME");
            System.out.println("LOGIN:"+login);
            System.out.println("NOME:"+nome);
        }

        //Pegando terceiro registro do ResultSet e alterando seu LOGIN
        rs.absolute(3);
        rs.updateString("LOGIN", "jsilva");
        rs.updateRow(); //atualiza a coluna na tabela

        //desativando o auto commit
```

```
con.setAutoCommit(false);

//atualizando o segundo registro sem usar updateRow
rs.absolute(2)
stmt.executeUpdate(
    "UPDATE FROM ATENDENTE SET LOGIN= "jsouza" WHERE NOME = '"
    + rs.getString("NOME") + "'"
);

//Varias outras operações de UPDATE
...
//"commitando" todos os updates realizados de uma única vez
con.commit();
}

}
```

Capítulo 19

A plataforma J2EE

A plataforma J2EE é uma solução baseada em componentes para projeto, desenvolvimento, montagem e disponibilização de aplicações distribuídas em ambientes corporativos. O J2EE oferece, entre outras tecnologias:

- Modelo multi-camada de aplicações distribuídas (*multi-tier*);
- Componentes Reutilizáveis;
- Modelo de segurança unificado;
- Controle de transações flexíveis;
- Suporte a serviços Web (*web services*).

No modelo multi-camada distribuído da arquitetura J2EE as seguintes características são importantes:

- Os componentes que compõem a aplicação podem estar em diferentes máquinas;
- Suas localizações dependem da camada a que pertence.

Tipicamente, as aplicações J2EE são divididas em 4 camadas que são:

- Cliente (*client-tier*);
- Web (*web-tier*);
- Negócios (*bussiness-tier*);
- Sistemas de Informação (*EIS-tier*).

As características mais comuns dos componentes J2EE são o fato deles serem unidades de software auto-contidas e a capacidade de comunicação com outros componentes. O que distingue um componente J2EE de uma classe comum é o fato deles poderem ser combinados para formar uma aplicação J2EE, além de serem disponibilizados, executados e gerenciados por um servidor J2EE, comumente chamado de servidor de aplicação.

A tabela a seguir mostra exemplos de componentes J2EE e o seus respectivos tipo e local de execução.

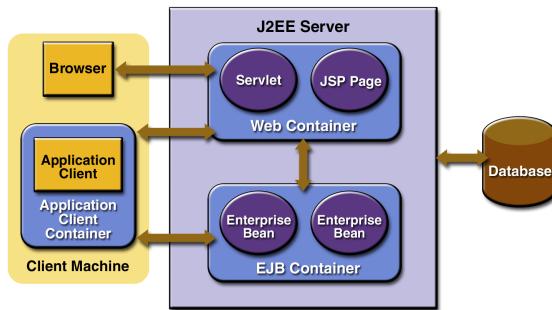


Figura 19.1: Arquitetura J2EE

Componente	Tipo	Local de Execução
Clientes da aplicação e Applets	Cliente J2EE	Cliente
Servlets e Páginas JSP	Web	Servidor
Enterprise Java Beans	Negócios	Servidor

Tabela 19.1: Componentes J2EE

19.1 Containers J2EE

Na arquitetura J2EE um container é um ambiente que fornece um conjunto de serviços para os componentes. Um componente deve ser colocado em um container apropriado para que possa ser executado. Um exemplo simples de container é o Apache Tomcat, que na arquitetura J2EE funciona como um container Web, fornecendo serviços para rodar páginas JSP e Servlets.

Algumas dos serviços prestados pelos containers podem ser configurados. Exemplo deles são:

- Segurança;
- Gerenciamento de Transações;
- Busca de Nomes e Diretórios utilizando JNDI (Java Naming and Directory Interface);
- Conectividade Remota entre clientes e EJB's;

No entanto, alguns dos serviços oferecidos pelos containers não podem ser configurados, uma vez que constituem o núcleo da arquitetura e conferem a escalabilidade da solução. Exemplo de serviços não configuráveis são:

- Criação de Destrução de EJB's e Servlets (Ciclo de Vida dos Componentes). São utilizados pools de componentes para atender as requisições dos usuários. Não é o programador instancia um componente no código do cliente, porém a criação efetiva do objeto acontece no servidor de aplicação, onde o componente existe e é executado de fato;
- Acesso às API's da plataforma;

- Gerenciamento de conexões com bancos de dados. Os servidores de aplicação implementam o conceito de pools de conexões. O programador não precisa escrever o código para abertura e encerramento da conexão, pois isso é papel do servidor de aplicação;
- Persistência do estado dos componentes.

19.2 Clientes J2EE

Na arquitetura J2EE existem basicamente dois tipos de clientes que são:

- Clientes Web: Páginas Web dinâmicas geradas por componentes Web (JSP ou Servlets) e que são exibidas no browser. Existe ainda os applets, que são pequenos clientes de aplicação escritos em Java e que rodam na JVM embutida no browser.
- Clientes de Aplicação: Permitem que o usuário interaja com o servidor de aplicação através de interfaces mais complexas, geralmente feitas utilizando as API's Swing ao AWT (*Abstract Window Toolkit*).

É importante ressaltar que toda a lógica de negócio é implementada pelos EJB's, que rodam no servidor de aplicação. Assim como os EJB's, os Servlets e páginas JSP também estão em um servidor de aplicação, porém não no mesmo container.

Os Servlets são classes Java que processam dinamicamente as requisições e controlem respostas na forma de páginas HTML. As páginas JSP permitem inserir código java diretamente em um documento HMTL, sendo apropriadas para criar conteúdo dinâmico e, em geral, para descrever a lógica de apresentação.

19.3 Um pouco mais sobre Servlets

Um servlet é uma classe java que extende as capacidades do servidor que acesam aplicações num modelo de programa requisição e resposta. Os servlets, como definidos na interface Servlet disponível no pacote javax.servlet, são independentes de protocolo, e definem os métodos que todos os servlets devem implementar. Os servlets são especialmente populares no desenvolvimento de aplicações Web, cuja comunicação é baseada no protocolo HTTP.

A classe HttpServlet, disponível no pacote javax.servlet.http, implementa a interface Servlet e provê a base para implementação de aplicações Web empregando o protocolo HTTP.

Do ponto de vista do cliente, um servlet funciona atendendo uma requisição e gerando conteúdo dinâmico em forma de páginas web, em HTML por exemplo, como resposta. O código a seguir exemplifica a criação de um servlet.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class MyNameServlet extends HttpServlet
{

    /**
     * Method to receive get requests from the web server
     * (Passes them onto the doPost method)
     * @param req The HttpServletRequest which contains the
     * information submitted via get
     * @param res A response containing the required response
     * data for this request
     **/

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        doPost(req,res);
    }

    /**
     * Method to receive and process Post requests from the web server
     * @param req The HttpServletRequest which contains the information
     * submitted via post
     * @param res A response containing the required response data for this request
     **/

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        //*****Read the value of the 'yourname' parameter*****
        String name=req.getParameterValues("yourname")[0];

        //*****Construct a response in HTML*****
        String reply="\n"+
            "<HEAD><TITLE>My Name Servlet Response</TITLE></HEAD>\n"+
            "<BODY>\n"+
            "<CENTER><BR><B>\n"+
            "Hello "+name+"\n"+
            "</B></CENTER>\n</BODY>\n</HTML>";

        //*****Send the reply*****
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println(reply);
        out.close();
    }
}
```

}

O servlet interpreta a informação dos formulário HTML para gerar uma página, também HTML, como resposta à requisição do cliente. O trecho a seguir mostra o código HTML da página que contem o formulário.

```
<HTML>
<HEAD><TITLE>My Name Servlet Demonstration</TITLE></HEAD>
<BODY>
<CENTER>
<FORM ACTION="/servlet/MyNameServlet" METHOD=POST>
Please Enter your name <INPUT TYPE=TEXT NAME="yourname">
<INPUT TYPE=SUBMIT VALUE=Submit>
</FORM>
</CENTER>
</BODY>
</HTML>
```

Enquanto o trecho de código a seguir mostra a página HTML retornada como resposta pelo servlet, supondo que o usuário tenha digitado zazaza.

```
<HTML>
<HEAD><TITLE>My Name Servlet Response</TITLE></HEAD>
<BODY>
<CENTER><BR><B>
Hello zazaza
</B></CENTER>
</BODY>
</HTML>
```

19.3.1 Ciclo de Vida dos Servlets

O inicio do clico de vida de um servlet ocorre quando o administrador do sistema inicia uma aplicação baseada em um servlet. Nesse momento, o container responsável pelos servlets no servidor de aplicação cria uma instância de um servlet. O container também passa os parâmetros de inicialização do servlet através do método **init**. Os parâmetros de inicialização persistem até o servlet ser destruído.

Se o processo de incialização ocorrer com sucesso, o servlet se torna disponível para serviço (atendimento de requisições). Se o processo falhar o container descarrega o servlet.

Após a que um servlet se torna disponível, o administrador pode optar para colocá-lo em um estado de indisponível, e assim ele permanecerá até que o administrador execute a operação contrária.

Quando um requisição de um cliente chega ao servidor de aplicação, o container cria os objetos **request** e **response**, que são passados de parâmetro para o método **service**.

O método service recupera informações sobre a requisição a partir do objeto request, processa a requisição, e em seguida utiliza o objeto response para criar uma resposta para o cliente. O método service pode ainda chamar outros métodos para processar a requisição, como doGet(), doPost(), ou métodos escritos pelo próprio programador.

Quando o administrador encerra a aplicação, o container invoca o método destroy() e descarrega o servlet. A figura 19.2 representa o diagrama de estados do ciclo de vida de um servlet.

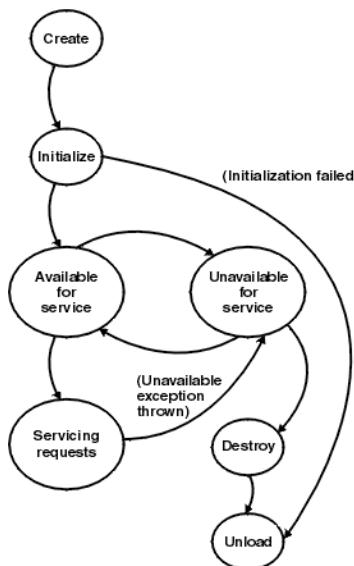


Figura 19.2: Ciclo de vida do servlet

19.3.2 Mantendo o estado do cliente

Muitas aplicações requerem que uma série de requisições de um mesmo cliente sejam associadas uma com a outra, para que uma determinada operação faça sentido. Um exemplo prático é uma aplicação de carrinho de compras, em que o cliente vai navegando durante as páginas escolhendo os produtos e ao final realiza a compra de fato. O período que comprehende a entrada no site, passando pela escolha dos produtos e chegando ao pagamento ou ao cancelamento é comumente chamada de sessão. A aplicação é responsável por manter o estado da sessão, tendo em vista que o protocolo HTTP é um protocolo stateless.

Para suportar aplicações que precisam manter um estado, os servlets oferecem API's para implementar o conceito de sessão.

As sessões são representadas pelo objeto HttpSession. Uma sessão pode ser acessada chamando o método **getSession** do objeto request. Esse método retorna a sessão corrente associada com o request, e se ela ainda não existir, uma

sessão é criada. Como o método getSession pode modificar o header da resposta (response header), esse método deve ser chamado antes de recuperados o PrintWriter ou o ServletOutputStream (objetos que servem para mandar dados para o cliente).

Atributos podem ser associados a uma sessão pelo nome. Tais atributos podem ser acessados por qualquer componente Web que pertença ao mesmo contexto esteja atendendo uma requisição que seja parte da mesma sessão. A seguir um exemplo de Servlet que mantém sessões:

```
public class CashierServlet extends HttpServlet {  
    public void doGet (HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Get the user's session and shopping cart  
        HttpSession session = request.getSession();  
        ShoppingCart cart = (ShoppingCart)session.getAttribute("cart");  
        ...  
        ...  
        // Determine the total price of the user's books  
        double total = cart.getTotal();  
    }  
}
```

Como não há nenhuma forma de um cliente HTTP sinalizar que uma sessão não é mais necessária (a não ser via aplicação), cada sessão possui um valor de timeout. O valor de timeout pode ser acessado através dos métodos **getMaxInactiveInterval** e **MaxInactiveInterval**. Geralmente o tempo máximo de inatividade permitido para uma sessão também é feita no servidor de aplicação.

19.4 Um pouco mais sobre páginas JSP

Java Server Pages (JSP) é uma tecnologia que permite embutir código java em uma página HTML para gerar conteúdo dinâmico. Um exemplo simples de uma página JSP é a seguinte:

```
<HTML>  
<BODY>  
    <% int a = 1 + 3; %>  
    Hello! O valor de a é <%= a %>  
</BODY>  
</HTML>
```

Todo código java aparece na página .jsp deve estar entre as tags `<%` e `%>`. Para imprimir algum valor na página é utilizada as tag `<%=` e `%>`, ou alternativamente os métodos `out.print()` ou `out.println()`, como mostrado no exemplo a seguir:

```
<%@ page import="java.util.*, java.lang.*" %>
```

```
<HTML>
  <BODY>
    <% Date date = new Date(); %>
    Hello! The time is now
    <%
      // This scriptlet generates HTML output
      out.println( String.valueOf( date ) );
    %>
  </BODY>
</HTML>
```

A primeira linha da página é um exemplo de diretiva JSP. Elas sempre são definidas entre as tags `<%@` e `%>`. No exemplo é utilizada uma diretiva de **import**, que tem por objetivo importar pacotes assim como em uma classe java comum. Existem outros tipos de diretivas, sendo as mais conhecidas a **include** e **taglib**.

A diretiva **include** inclue o conteúdo do arquivo especificado durante a fase de compilação da página JSP, ou seja, quando esta é convertida em um Servlet. Neste caso, os dois arquivos são combinados para gerar um só. É muito útil quando uma página é estruturada em partes bem definidas (topo, menu, rodapé, etc.). A seguir um exemplo do uso de uma diretiva **include**.

```
<%@ include file="right_menu.jsp" %>
<HTML>
<BODY>
<h1> Notícias do dia </h1>
COECA aprova mais de 30 em 2007. A soma dos salários já chega a R$ 150.000!
</BODY>
</HTML>
```

Um outro recurso das páginas JSP são as chamadas taglibs, que são um mecanismo portável para encapsular funcionalidades reusáveis em páginas da Web. As tags podem ser de dois tipos: carregadas de uma biblioteca externa de tags ou pré-definidas. As tags pré-definidas sempre se iniciam com `jsp`. Exemplo de chamadas a tag pré-definidas são mostrados abaixo:

```
// Exemplo 1:
// Inclui uma página dentro de outra no momento da compilação.

<jsp:include page="hello.jsp"/>

// Exemplo 2:
// Localiza e instancia um bean com o determinado nome e escopo

<jsp:useBean id="cart" scope="session" class="session.Carts" />
<jsp:setProperty name="cart" property="*" />
```

```
<jsp:useBean id="checking" scope="session" class="bank.Checking" >
<jsp:setProperty name="checking" property="balance" value="0.0" />
</jsp:useBean>
```

Para utilizar uma tag carregada de uma biblioteca externa antes de mais nada é preciso utilizar a diretiva taglib para especificar onde a biblioteca de tag (*tag library*) está localizada.

```
<%@ taglib prefix="blx" uri="/blx.tld" %>
<jsp:useBean id="user" class="user.UserData" scope="session"/>

<HTML>
<BODY>
<blx:getProperty name="user" property="*">
<FORM METHOD=POST ACTION="SaveName.jsp">
    What's your name? <INPUT TYPE=TEXT NAME=username SIZE=20><BR>
    What's your e-mail address? <INPUT TYPE=TEXT NAME=email SIZE=20><BR>
    What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4>
    <P><INPUT TYPE=SUBMIT>
</FORM>
</blx:getProperty>
</BODY>
</HTML>
```

19.4.1 JSP vs. Servlets

Quando uma página JSP é requisitada pela primeira vez, o container compila a página JSP gerando um servlet. Nas chamadas sucessivas essa etapa não é mais necessária, a não ser que a página JSP tenha sido modificada. Portanto, as páginas JSP são, em última instância, processadas como servlets.

Pelo fato de ser transformada em um servlet, durante o processamento de páginas JSP, o container também cria os objetos request e response. O objeto request armazena informações enviadas na requisição do cliente, por exemplo, os campos de um formulário de cadastro. O seguinte trecho de código mostra como o objeto request é utilizado em uma página JSP.

```
<HTML>
<BODY>
<%
    // Captura o valor do campo tx_name do form submetido
    String nome = request.getParameter("tx_name");
    //Captura o ip de origem da requisição
    String ip = request.getRemoteHost();
%>

Olá <b> <%=nome%> </b>, o endereço IP da sua máquina é <%=ip%>
```

```
</BODY>
</HTML>
```

19.5 Um pouco mais sobre EJB's

Os EJB's (*Enterprise Java Beans*) são componentes de software simples, reutilizáveis e portáteis, que podem ser combinados para criar aplicações corporativas. Eles encapsulam a lógica de negócio, ou seja, o código que realiza o propósito da aplicação. Tecnicamente falando, os EJB's também são classes Java, com seus próprios atributos e métodos.

As instâncias dos EJB's são executadas em containers de EJB nos servidores de aplicação. São os containers que gerenciam o ciclo de vida dos EJB's, decidindo quando eles devem ser criados ou destruídos.

Os principais benefícios da utilização dos *beans* são:

- O desenvolvedor dos clientes pode se concentrar na lógica de apresentação;
- Os clientes são menos pesados, já que os EJB's são executados nos containers do servidor de aplicação;
- Maior reutilização de componentes.

Os dois tipos mais importantes de EJB's são:

- Session Beans:

Modelam os processos da lógica de negócios, executam tarefas para os clientes e podem implementar *web services*. No momento da requisição do cliente, um session bean é alocado para atendimento, tornando exclusivo até o fim do atendimento da requisição. Os sessions beans podem ser de dois tipos: stateless ou statefull. Os beans stateless não mantêm o estado da conversação, ou seja, após o término da execução de um método todo o estado (valor dos atributos do EJB) se perdem, e o mesmo é liberado, se tornando livre para atender outra requisição. Já com os beans statefull, o estado pode ser modificados e recuperado entre sucessivas invocações de métodos. Todos os métodos invocados pelo cliente são enviados a mesma instância, que mantém o estado até que a sessão termine ou o bean seja destruído.

- Entity Beans:

São responsáveis pela persistência dos dados da aplicação, geralmente utilizando bancos de dados. Nesse contexto, bean desse tipo está associado a uma tabela em um BD, enquanto cada instância de um entity bean representa um registro de tal tabela. Esse tipo de bean pode ser compartilhado por múltiplos usuários, sendo necessário para isso o gerenciamento de transações, que é um dos serviços prestados pelo container EJB. Cada bean tem um identificador único que permite encontrá-lo. Como na tabela de um BD relacional, um entity bean pode estar relacionado a outros, podendo este relacionamento ser implementado pelo próprio bean ou pelo container. Com os entity beans, o container é responsável por todas as

chamadas ao banco de dados (Ex: Uma atualização de um entity bean faz com que o container dispare um SQL Update, enquanto a criação de um novo entity bean irá fazer o container disparar um SQL Insert.) A idéia é eliminar ao máximo a necessidade de escrever comandos SQL. Todos os comandos SQL enviados ao banco fazem uso das conexões disponíveis em um pool de conexões, que também é um serviço gerenciado pelo servidor de aplicação.

19.5.1 Ciclo de Vida dos EJB's

Um EJB passa por vários estados ao longo de seu ciclo de vida, sendo que cada tipo de EJB tem um ciclo de vida específico. No caso dos EJB's stateless, o ciclo de vida é descrito pelos seguintes estados e eventos:

- O bean inicia em estado **Does Not Exist**;
- O cliente invoca um método `create()` na interface local;
- O container instancia o bean e os métodos `setSessionContext()` e `ejbCreate()`, implementados no bean, são invocados. A partir daí, o EJB passa para o estado **Ready**;
- Ao final, o cliente invoca o método `remove()` na interface local e em seguida o container invoca o método `ejbRemove()`, implementado no bean, e o EJB volta para o estado **Does Not Exist**;

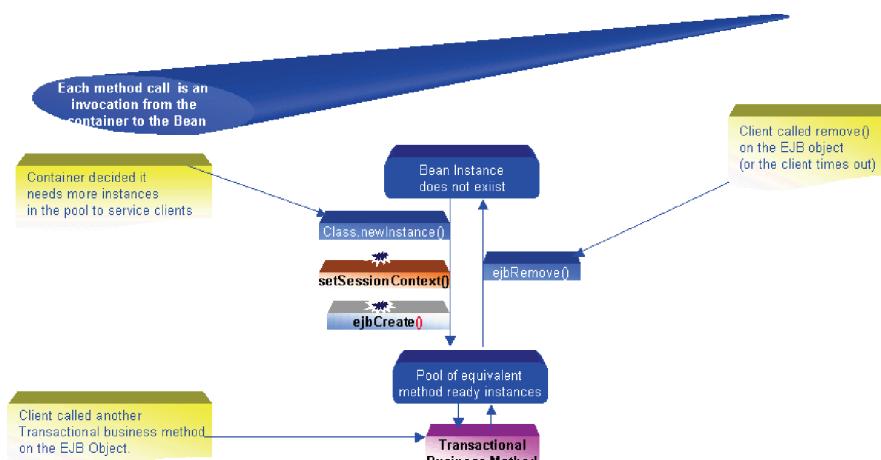


Figura 19.3: Ciclo de vida de um EJB stateless

O ciclo de vida dos EJB's statefull, por sua vez, é descrito pelo seguintes estados e eventos:

- O bean inicia em estado **Does Not Exist**;
- O cliente invoca um método `create()` na interface local;
- O container instancia o bean e os métodos `setSessionContext()` e `ejbCreate()`, implementados no bean, são invocados. A partir daí, o EJB passa para o estado **Ready**;

- Quando no estado **Ready**, o container pode optar por desativar o bean, ou seja, transferí-lo para memória secundária invocando o método ejbPassivate(). Nesse caso, o EJB vai para o estado **Pas-sive**;
- Para reativar o bean, o container deve chamar o método ejbActivate(), levando-o para o estado **Ready** novamente;
- Ao final, assim como nos EJB's stateless, o cliente invoca o método remove() na interface local e em seguida o container invoca o método ejbRemove(), implementado no bean, e o EJB volta para o estado **Does Not Exist**;

Os processos de desativação e ativação de um EJB consistem em salvar seu estado em memória secundária e restaurá-lo, respectivamente. Nesses processos os containers utilizam técnicas de serialização. Atualmente, os containers têm implementado a serialização utilizando arquivos XML.

Como foi discutido, os containers fazem uso dos chamados pools de EJB's, que são um conjunto de beans designados a atender as requisições dos usuários. Em determinados momentos, os beans de um pool podem estar completamente alocados para atendimento de usuários. Um cliente que desejasse ter sua requisição atendida teria que aguardar a desalocação de um bean por parte de um cliente.

No entanto, alguns dos beans no pool podem estar inativos, aguardando invocação de métodos por parte dos clientes. É papel do container decidir quando um bean deve ir para o estado passivo (ser serializado) para liberar recursos, que em última instância se traduz em memória no servidor de aplicação, permitindo que novas requisições sejam atendidas. Quando um cliente que teve seu bean serializado decidir realizar uma nova requisição, um bean deve ser alocado e o seu estado deve ser restaurado da memória secundária para a memória principal.

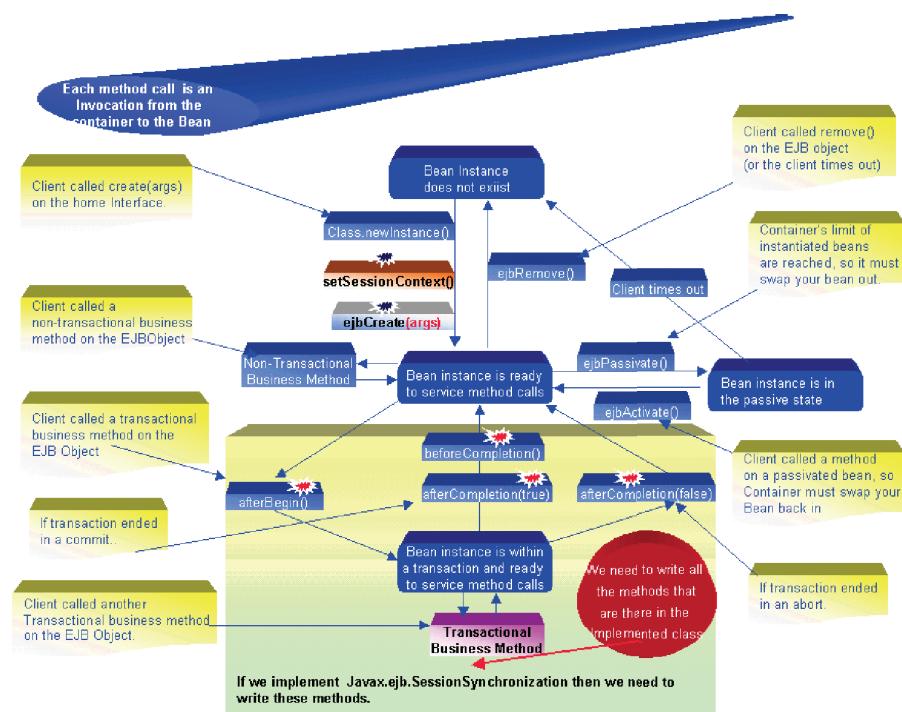


Figura 19.4: Ciclo de vida de um EJB statefull

Parte V

Desenvolvimento Web

Capítulo 20

Usabilidade

20.1 Definição

A norma ISO 9241-11 define a usabilidade como: *a extensão em que um produto pode ser usado por usuários específicos para alcançar objetivos específicos com eficácia, eficiência e satisfação num contexto específico de uso.* E, define uma estrutura para a usabilidade, figura 20.1.

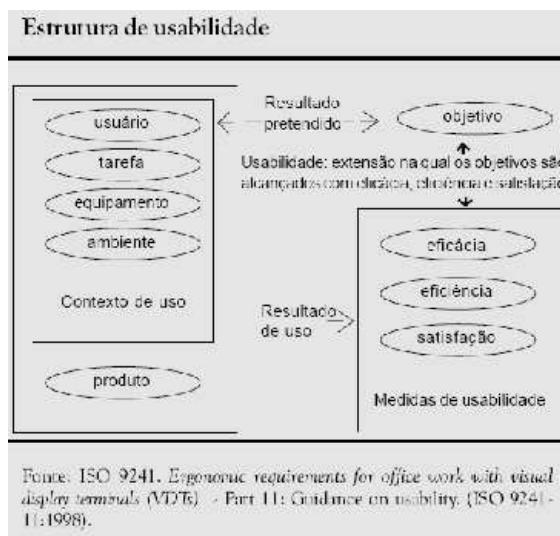


Figura 20.1: Estrutura de usabilidade segundo a ISO 9241-11

Conforme a estrutura é preciso identificar os objetivos, e decompor a usabilidade (eficácia, eficiência e satisfação) em atributos passíveis de serem verificados e mensurados, assim como o contexto de uso (usuário, tarefa, equipamento e ambiente).

Contextualizando o ambiente para as páginas de Web, a usabilidade é o termo técnico usado para referenciar a qualidade de uso de uma determinada interface. Ela refere-se ao grau com que o usuário consegue realizar uma tarefa.

Em geral, são considerados os seguintes atributos para verificação da usabilidade numa interface: funcionalidade correta; a eficiência de uso; a facilidade de aprendizagem; a facilidade de relembrar; tolerância ao erro do usuário; e satisfação do usuário.

20.2 Princípios da usabilidade

Os principais pontos que um Web site deve cumprir são:

- Usar etiquetas ALT para todas as imagens, especialmente as de navegação;
- Usar texto negro em fundo branco sempre que possível para uma legibilidade maior;
- Usar cores de fundo planas ou com texturas, mas que sejam extremamente subtis;
- Assegurar que o texto se encontra numa cor que se possa imprimir (não branco);
- Colocar o menu de navegação numa localização consistente em cada página do site;
- Usar localizações familiares para as barras de navegação; Usar um design adequado para que nunca seja necessário recorrer ao scroll horizontal;
- Usar um eixo de simetria para centrar o texto numa página; Encorajar o uso do scroll através da colocação de uma imagem que fique semi-escondida no fundo da página.

Deve-se tentar evitar:

- As etiquetas ALT sejam reduzidas (especialmente para uma pequena imagem de tamanho fixo);
- Mostrar texto estático em azul ou a sublinhado;
- Usar o negrito ou maiúsculo para texto longo;
- Deixar espaços em branco muito longo – a leitura torna-se mais difícil;
- Obrigar o utilizador a fazer scroll para encontrar informação importante, especialmente botões ou links de navegação;
- Usar travessões para separar horizontalmente diferentes tipos de conteúdos;
- Alternar frequentemente entre texto centrado e alinhado à esquerda. A maior parte do texto deve estar alinhada à esquerda;
- Fixar a resolução das páginas a mais de 800 X 600. Páginas de grandes resoluções podem obrigar o utilizador a fazer scroll horizontal.

As principais vantagens do estudo da usabilidade de um produto de software são:

- Aumentar a produtividade dos utilizadores;
- Aumentar os níveis de utilização do produto;
- Reduzir a necessidade de formação e de custos de produção de documentação;
- Reduzir os custos de suporte técnico;
- Reduzir custos e tempo de desenvolvimento;
- Minimizar o re-desenvolvimento e as alterações após a finalização.

20.3 Técnicas de avaliação de usabilidade

A usabilidade pode ser avaliada por diversas técnicas. As mais usuais são da heurística e dos testes com utilizadores.

A heurística é uma das formas de avaliar a usabilidade mais econômica e prática, permitindo detectar problemas na fase de desenvolvimento da interface. Esta técnica consiste em levantar questões heurísticas relacionadas à interface como: navegação, controle, funcionalidade, linguagem, ajuda e suporte, consistência e erros. Nunca deve ser feita individualmente, pois uma pessoa não tem capacidade de levantar todas as questões heurísticas.

A técnica dos testes com utilizadores obriga que o produto esteja pelo menos em forma de protótipo para poder ser testado. O utilizador testa a interface tendo por base um conjunto de tarefas que foram definidas como princípios heurísticos.

Portanto, o objetivo de um bom design de páginas Web é obter alta qualidade, no que diz respeito não somente a uma boa aparência visual, como também à estruturação da informação de forma a permitir aos usuários encontrá-la rápida e eficaz.

Capítulo 21

Acessibilidade

21.1 Definição

A acessibilidade descreve os problemas de usabilidade encontrados por usuários com necessidades especiais ou com limitações tecnológicas. Na prática, a acessibilidade de uma interface é indicada pela facilidade de acesso de um indivíduo, independente de suas capacidades físicas, sensoriais e cognitivas, do seu ambiente e das condições de trabalho e das barreiras tecnológicas. Ou seja, a acessibilidade significa que pessoas com necessidades especiais podem apreender, compreender, navegar e interagir com a Web.

As causas mais freqüentes de falta de acessibilidade em muitas páginas da Web para todos os usuários estão muitas vezes associadas: à falta de estrutura que desorienta o usuário dificultando a navegação e ao uso abusivo de informações gráficas (imagens, macros, scripts Java, elementos multimídia) sem proporcionar alternativas adequadas de texto ou outro tipo de comentário.

21.2 Princípios da acessibilidade

Os princípios de acessibilidade segundo W3C estão relacionados a dois principais temas: assegurar uma transformação harmoniosa e tornar o conteúdo comprehensível e navegável. Estes princípios são abordados num documento elaborado pelo W3C (Word Wide Web Consortium)-WAI (Web Accessibility Initiative), considerado referência para os princípios de acessibilidade e idéias de design, chamado Web Content Accessibility GuideLines, WCAG 1.0.

A *transformação harmoniosa* de uma página Web pode ser garantida pela observância de alguns pontos-chaves:

- Separar a estrutura da apresentação, diferenciando o conteúdo (a informação a ser transmitida), a estrutura (a forma como a informação é organizada em termos lógicos) e a apresentação (a forma como a informação é reproduzida, por exemplo, como matéria impressa, como apresentação gráfica bidimensional, sob forma exclusivamente gráfica, como discurso sintetizado, em braille, etc.);
- Criar páginas que cumpram a sua finalidade, mesmo que o usuário não possa ver e/ou ouvir, fornecendo informações que preencham a mesma fi-

nalidade ou função que o áudio ou o vídeo, de maneira a se adaptar o melhor possível a canais sensoriais alternativos e as tecnologias de apoio (software ou hardware para ajudar pessoas com incapacidade ou deficiência) atualmente disponíveis no mercado;

- Criar páginas que não dependam exclusivamente de um tipo de equipamento. As páginas devem ser acessíveis a usuários que não possuam mouse, que recebam voz ou texto, etc.

Os criadores de conteúdo para a Web necessitam tornar suas produções *compreensíveis e navegáveis*, empregando uma linguagem clara e disponibilizando meios de navegação e apropriação da informação apresentada. Disponibilizar mecanismos de orientação de página e ferramentas de navegação são fatores que potencializam a acessibilidade à Web ao garantir a perceptibilidade e naveabilidade no site, pois sem esses elementos, os usuários podem, por exemplo, não compreender tabelas, listas ou menus extensos.

O documento WCAG 1.0 descreve os 14 princípios ou diretrizes que abordam as questões relacionadas à acessibilidade à Web. Cada diretriz possui pontos de verificação os quais são classificados por níveis de prioridade. Os 14 princípios são:

1. Fornecer alternativas ao conteúdo sonoro ou visual – esta diretriz indica a necessidade de disponibilizar conteúdo que, ao ser apresentado ao usuário, transmita, em essência, as mesmas funções e finalidades do conteúdo sonoro e visual. Assim, deve-se fornecer um equivalente textual a cada elemento não textual;
2. Não recorrer apenas a cor – deve-se garantir a percepção do texto e dos elementos gráficos do documento, mesmo quando são vistos sem cores. Assim, deve-se assegurar que todas as informações veiculadas por meio de cores estejam também disponíveis sem cor, por exemplo, a partir de informações do contexto ou de marcação apropriada;
3. Utilizar corretamente anotações (proporciona efeitos de formatação, com por exemplo, B ou I em HTML) e folhas de estilo (conjunto de declarações que especificam a apresentação do documento) – esta diretriz indica a necessidade de se utilizar marcação nos documentos com os elementos estruturais adequados e controlar a apresentação por meio de folhas de estilo, em vez de elementos de apresentação e atributos;
4. Indicar claramente qual o idioma utilizado – utilizar marcações que facilitem a pronúncia e a interpretação de abreviaturas ou de texto em língua estrangeira é imprescindível. Deve-se identificar claramente quaisquer mudanças de idioma no texto de um documento, bem como nos equivalentes textuais;
5. Criar tabelas passíveis de transformação harmoniosa – assegurar que as tabelas têm as marcações necessárias para serem transformadas de forma harmoniosa por navegadores acessíveis a outros agentes do usuário. Em tabelas de dados, é preciso identificar os cabeçalhos de linha e de coluna;
6. Assegurar que as páginas dotadas de novas tecnologias sejam transformadas harmoniosamente – permitir que as páginas sejam acessíveis mesmo

quando as novas tecnologias mais recentes não forem suportadas ou tenham sido desativadas. Deve-se organizar os documentos de tal forma que possam ser lidos sem a necessidade de recorrer a folhas de estilo;

7. Assegurar o controle de usuário sobre as alterações temporais do conteúdo – assegurar a possibilidade de interrupção momentânea ou definitiva do movimento, intermitência, transcurso ou atualização automática de páginas e objetos;
8. Assegurar acessibilidade direta em interfaces integradas pelo usuário – a interface com o usuário deve obedecer a princípios de design para a acessibilidade: acesso independente de dispositivos, operacionalidade pelo teclado, emissão automática de voz (verbalização);
9. Projetar páginas considerando a independência de dispositivos – utilizando funções que permitam a ativação de páginas por meio de dispositivos de entrada e de comandos. Geralmente as páginas que permitem interação pelo teclado são também acessíveis através de interfaces de comando de voz ou de linhas de comando;
10. Utilizar soluções provisórias ou de transição – utilizar soluções de acessibilidade transitórias, para que as tecnologias de apoio e os navegadores mais antigos funcionem corretamente;
11. Utilizar tecnologias e recomendações do W3C – utilizar tecnologias do W3C (HTML, XML, CSS, SVG, SMIL e etc.) e seguir as recomendações de acessibilidade;
12. Fornecer informações de contexto e orientações – ajudar os usuários a compreenderem páginas ou elementos complexos;
13. Fornecer mecanismos de navegação claros – fornecer mecanismos de navegação coerentes e sistematizados (informações de orientação, barras de navegação, mapa do site) de modo a aumentar a probabilidade de uma pessoa encontrar o que procura em um dado site. A existência de mecanismos claros e coerente é importante para usuários com deficiência visual ou cognitiva, beneficiando a todos os usuários;
14. Assegurar clareza e simplicidade dos documentos – assegurar a produção de documentos claros e simples, para que sejam mais fáceis de compreender. Deve-se também utilizar a linguagem mais clara e simples possível, adequada ao conteúdo do site.

Conforme mencionado anteriormente, cada diretriz possui pontos de verificação que por sua vez possui os níveis de prioridade. De acordo com a classificação, existem 3 níveis de prioridade:

Prioridade 1 pontos que os criadores de conteúdo Web têm absolutamente de satisfazer para evitar que usuários fiquem impossibilitados de compreender as informações contidas na página ou site;

Prioridade 2 pontos que os criadores de conteúdo para a Web devem satisfazer para evitar que os usuários tenham dificuldade de acessar as informações contidas no documento, evitando barreiras significativas a documentos publicados na Web;

Prioridade 3 pontos que os criadores de conteúdo na Web podem satisfazer para melhorar o acesso as informações disponibilizadas nas páginas ou sites.

Na verificação da acessibilidade de um documento são estabelecidos os níveis de conformidade para as páginas ou sites na Web:

Nível de conformidade A quando satisfeitos todos os pontos de verificação de prioridade 1;

Nível de conformidade Duplo A quando satisfeitos todos os pontos de verificação de prioridade 1 e 2;

Nível de conformidade Triplo A quando satisfeitos todos os pontos de verificação de prioridade 1, 2 e 3.

21.3 Técnicas de avaliação de acessibilidade

As avaliações e validações são especificadas num documento do W3C chamado Evaluating Web Sites Accessibility, onde são apresentados dois métodos de avaliação de acessibilidade: Avaliação de Preliminar de Acessibilidade (identificação de forma rápida dos problemas gerais de acessibilidade num site) e Avaliação de Conformidade (verificação do grau de conformidade de um site com os padrões de acessibilidade, WCAG. Este método identifica problemas de acessibilidade mais específicos num site).

A avaliação e validação da acessibilidade são feitas por meio de ferramentas automáticas ou da revisão direta manual. Os métodos automáticos são geralmente rápidos, mas não são capazes de identificar todos os aspectos da acessibilidade. Esses métodos não são capazes de avaliar conteúdo gerado dinamicamente. A avaliação humana pode ajudar a garantir a clareza da linguagem e a facilidade de navegação.

Para a validação automática da acessibilidade de uma página ou de um site podemos utilizar as ferramentas ou serviços de análise da acessibilidade e compatibilidade, como Bobby, o validador par HTML 4 do W3C e o TAW. O ideal é a combinação dos métodos manuais e automáticos.

A avaliação e validação da acessibilidade de uma página ou de um site à Web devem estar presentes desde as fases iniciais do desenvolvimento do documento. O W3C-WAI aponta como método para testar uma página ou site, após a implementação dos princípios de acessibilidade, os seguintes pontos de verificação:

1. Utilizar uma ferramenta de acessibilidade automática e ferramentas de validação de navegadores;
2. Validar a sintaxe (HTML, XML, etc.);
3. Validar as folhas de estilo;
4. Utilizar um navegador só de texto ou um emulador;
5. Utilizar vários navegadores gráficos com o som e os gráficos ativos, sem gráficos, sem mouse e sem carregar frames, programas interpretáveis, folhas de estilo ou applets;

6. Utilizar vários navegadores, antigos e recentes;
7. Utilizar um navegador de emissão automática de falas, com leitores de tela, com software de aplicação, monitores monocromáticos, etc;
8. Utilizar corretores ortográficos e gramaticais;
9. Rever o documento, verificando a clareza e a simplicidade;
10. Validar páginas com usuários reais.

Acessibilidade e usabilidade são conceitos que se inter-relacionam, pois ambos buscam a eficiência e eficácia no uso de uma interface. A observação de alguns critérios ou fatores a serem ressaltados na elaboração de uma página Web pode auxiliar na concepção de bons projetos de interface e consequentemente, melhorar a qualidade da interação do usuário com a aplicação.

Capítulo 22

Padrões Web W3C

Padrões Web (Web Standards ou Tableless) é um conjunto de normas, diretrizes, recomendações, notas, artigos, tutoriais e afins de caráter técnico, produzidos pelo W3C e destinados a orientar fabricantes, desenvolvedores e projetistas para o uso de práticas que possibilitem a criação de uma Web acessível a todos, ajudados por diversos recursos que fazem uma Web mais agradável de usar.

Toda essa padronização é para atender a idéia da **Web Semântica** que consiste organizar os documentos Web de tal forma, que estes possam ser interpretados automaticamente, facilitando a pesquisa e cruzamento das informações dos documentos. Ou seja, além da informação, a máquina saberia distinguir a que se refere esta informação. Um exemplo prático disso é que, se utilizássemos o elemento address para definir os endereços em nosso site, uma ferramenta de busca poderia criar uma indexação destes endereços, e disponibilizar para o usuário. Quantas vezes você visitou o site de uma empresa e cansou de procurar pelo endereço, ou um telefone de atendimento ao cliente? A Web Semântica não é uma substituição a Web atual, mas sim uma extensão da atual. E a base da Web Semântica é o XML, o RDF e a Ontologia.

Existem diversas vantagens na aplicação dos padrões Web. Entre elas, vale destacar:

- Acessibilidade tanto para deficientes quanto para usuários de dispositivos móveis como PDA's, celulares, smartphones, etc.; O browser interpreta as informações de layout (em um arquivo CSS) de 30% a 70% mais rapidamente, o que gera uma queda considerável no download dos arquivos do site;
- Economia de banda na transmissão dos arquivos, pois com a utilização dos padrões os arquivos possuem menor tamanho; O código HTML se torna muito mais compacto ao se separar conteúdo, design e programação;
- Melhor visibilidade no Google e nos demais mecanismos de busca, pois os padrões utilizam a estrutura semântica simples do HTML;
- Maior facilidade de atualização e mudanças no layout.

Algumas tecnologias padronizadas pelo W3C:

HTML (Hyper Text Markup Language) É uma linguagem de marcação de tags utilizada para produzir páginas na Web. A última recomendação lançada pelo W3C foi a HTML 4.01;

XHTML (Extensible Hyper Text Markup Language) É uma reformulação da linguagem de marcação HTML baseada em XML. Combina as tags de marcação do HTML com as regras do XML. Esta reformulação tem em vista a exibição de páginas Web em diversos dispositivos (televisão, palm, celular). A intenção desta linguagem é melhorar a acessibilidade. A última recomendação lançada pelo W3C foi XHTML 1.0;

XML (Extensible Markup Language) É uma linguagem de marcação apropriada para representação de dados, documentos e demais entidades cuja essência fundamenta-se na capacidade de agregar informações. A última recomendação lançada pelo W3C foi XML 1.0;

CSS (Cascading Style Sheets) É uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento. A última recomendação lançada pelo W3C foi CSS 3;

XSL (Extensible Markup Language) É uma linguagem que define as folhas de estilo padrão para o XML. É constituída de três partes: XSLT (linguagem de transformação de documentos XML), XPath (linguagem para navegação em documentos XML) e XSL-FO (linguagem para formatação de documentos XML). A última recomendação lançada pelo W3C foi XSL 1.0, XSLT 2.0 e XSL-FO;

XSLT (XSL Transformations) É uma linguagem de marcação XML usada para transformar documentos XML. Esta linguagem possibilita mais transformações mais potentes do que as folhas de estilo CSS;

XPath (XML Path) É uma linguagem para encontrar informações em um documento XML. O XPath é utilizado para navegar através de atributos e elementos em um documento XML. É o principal elemento no padrão XSLT;

XQuery (XML Query) É uma linguagem utilizada para executar consultas em dados XML. A última recomendação lançada pelo W3C foi XQuery 1.0;

DOM (Document Object Model) É uma especificação do W3C, independente de linguagem e plataforma, onde se pode alterar e editar a estrutura de um documento. Com o DOM é possível acessar elementos de um documento, e poder trabalhar com cada um desses elementos separadamente, possibilitando a criação de páginas altamente dinâmicas. A última recomendação lançada pelo W3C foi DOM Level 3;

SOAP (Simple Object Access Protocol) É um protocolo de comunicação simples baseado em XML para troca de informações entre aplicações. A especificação do SOAP provê maneiras para construir mensagens que possam trafegar por meio de diversos protocolos de forma independente da

linguagem de programação e do sistema operacional. Normalmente, o protocolo utilizado para troca de informações é o HTTP, pois este não possui problema de segurança com firewalls e proxy. Ou seja, o http por ser um protocolo da Internet não é bloqueado pelos firewalls e proxy. A última recomendação lançada pelo W3C foi SOAP 1.2;

WSDL (Web Services Description Language) É uma linguagem baseada em XML para descrição de Web Services e como acessá-los. A última recomendação lançada pelo W3C foi WSDL 1.1;

RDF (Resource Description Framework) É uma linguagem para representar informações na Web. O RDF usa notação XML como sintaxe de codificação para descrição de metadados. A idéia do RDF é a descrição dos dados e dos metadados em três componentes: recurso (qualquer objeto que é identificável unicamente por um URI), propriedade (é uma característica, um atributo ou relacionamento específico de um recurso) e valor, e uma forma coerente de acesso aos padrões de metadados (namespace) publicados na Web. Um dos benefícios do RDF é permitir que aplicações possam agir de forma inteligente e automatizada sobre as informações publicadas na Web. A última recomendação lançada pelo W3C foi RDF 1.0;

OWL (Web Ontology Language) É uma linguagem para definir e instanciar ontologias na Web. Uma ontologia OWL pode incluir descrições de classes e suas respectivas propriedades e relacionamentos. A OWL foi projetada para aplicações que precisam processar o conteúdo da informação ao invés de apresentá-la aos humanos. Ela fornece um vocabulário adicional com uma semântica formal que facilita muito mais a interpretação do conteúdo da Web do o XML. A OWL foi baseada nas linguagens OIL e DAML+OIL, e possui três sublinguagens: OWL LITE, OWL DL, OWL FULL. A última recomendação lançada pelo W3C foi OWL 1.0;

SMIL (Syncronized Multimedia Integration Language) É uma linguagem baseada em XML, trabalhando com tags semelhantes ao HTML, podendo ser editado por qualquer editor de texto comum, pois os elementos multimídia não são inseridos, apenas referenciados. O SMIL permite a criação de apresentações audiovisuais interativas, e é utilizado em apresentações multimídias do tipo *rich media*, que integram áudio e vídeo streaming, texto ou qualquer outro tipo de arquivo. O SMIL permite gerenciar a transmissão de arquivos multimídia por streaming, sincronizar estes arquivos com animações flash, páginas HTML, etc., e criar uma espécie de indexação destes arquivos de áudio e vídeo. Uma grande vantagem do SMIL que ele é utilizado pelos players mutimídia Real Player QuickTime. A última recomendação lançada pelo W3C foi SMIL 2.1.

O W3C estabelece uma série de recomendações para utilizar essas tecnologias. Esses padrões são conhecidos como Padrões W3C ou Recomendações W3C.
Observações:

1. Um namespace (NS) define um vocabulário controlado que identifica um conjunto de conceitos de forma única para que não haja ambigüidade na

sua interpretação. Os namespaces XML são conjuntos de tipos de elementos e atributos possíveis para cada tipo. As triplas do RDF se baseiam em namespaces de forma que a cada recurso seja associado uma dupla de propriedade e valor. Os namespaces podem ser referenciados por meio de uma URI, que se constitui em um repositório compartilhado, e não-ambíguo, onde usuários e programas de validação de código XML podem consultar a sintaxe e propriedades semânticas dos conceitos cobertos;

2. URI (Uniform Resource Identifier) é uma cadeia de caracteres que identifica um recurso da internet. O URI mais comum é o URL que identifica um endereço de domínio, mas os URIs podem também identificar livros, elementos de uma página e pessoas individuais, por exemplo;
3. Uma Ontologia fornece uma conceituação (meta-informação) que descreve a semântica dos objetos, as propriedades dos objetos e as relações existentes entre eles num dado domínio do conhecimento. As Ontologias são desenvolvidas para fornecer um nível semântico à informação de um dado domínio de forma torná-la processável por máquinas e comunicável entre diferentes agentes (software e pessoas). As ontologias são utilizadas na Inteligência Artificial, Web Semântica, Engenharia de Software e Arquitetura da Informação como uma forma de representação de conhecimento do mundo ou alguma parte dele. São elementos da Ontologia: Indivíduos (instâncias) que incluem objetos concretos ou abstratos, Classes (conceitos) que são grupos abstratos (por exemplo, Pessoa, a classe para todas as pessoas), Atributos que armazenam informações do objeto (pelo menos um nome e um valor), e Relacionamento que são as formas como os objetos se relacionam com os outros objetos;

Capítulo 23

XML

23.1 O que é XML?

Extensible Markup Language (XML) é uma linguagem de marcação (utilização de tags) para descrever os dados de páginas Web de forma estruturada. O XML é um dos padrões adotado pelo W3C e foi projetado para descrever o conteúdo de uma página Web em vez da sua formatação. O XML é derivado da linguagem SGML (Standard Generalized Markup Language), que é um padrão internacional para definição de formatos de representação de documentos. O SGML é muito mais complexo de se implementar do que o XML. A figura 23.1 mostra um exemplo de uma página Web em XML.

Como o XML trata apenas do conteúdo do documento, existem diversos padrões para descrever o formato da apresentação dos dados. Uma das formas de apresentar tais documentos é por meio das folhas de estilo. As principais linguagens de estilo são o CSS (Cascading Style Sheets) e o XSL (Extensible Style Sheets). Então, a apresentação dos dados ficará a cargo dessas linguagens.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="book_list.xsl"?>
<book_list>
  <book>
    <title> Computer Networks, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2003 </year>
  </book>
  <book>
    <title> Modern Operating Systems, 2/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2001 </year>
  </book>
  <book>
    <title> Structured Computer Organization, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 1999 </year>
  </book>
</book_list>
```

Figura 23.1: Uma página Web simples em XML

23.2 Características do XML

Algumas características do XML:

- É uma linguagem de representação de documentos e não uma linguagem de programação;
- Facilita a geração de dados, a leitura de dados e assegura que a estrutura de dados não é ambígua;
- É extensível e independente de plataforma;
- É modular, pois a permite a definição de um novo formato de documento pela combinação e reuso de outros formatos;
- Não possui tags pré-definidas, e permite um número infinito de tags;
- Estruturas de dados podem ser agrupadas em profundidade infinita;
- Pode ser descrita por grafos com nós e rótulos, possibilitando que algoritmos conhecidos possam ser utilizados para a recuperação ou localização de dados.

23.3 Comparação entre XML e HTML

O XML e o HTML são primos. O XML não veio para substituir o HTML. Eles derivam da mesma linguagem, o SGML. A grande diferença entre o HTML e o XML é que o HTML descreve como os dados devem ser apresentados (formatação dos dados) numa página Web enquanto o XML descreve os dados (conteúdo) de uma página Web.

Ambos fazem o uso de tags (palavras delimitadas por < e >) e atributos. Enquanto o HTML especifica o que cada tag e atributo significam e frequentemente como o texto entre elas aparece num browser, o XML usa as tags somente para delimitar as porções de dados, e deixa a interpretação dos dados para a aplicação que irá ler este dado. Por exemplo, enquanto em um documento HTML uma tag `<p>` indica um parágrafo, no XML essa tag pode denotar um preço, um parâmetro, uma pessoa, etc.

Enquanto ao uso das tags nas duas linguagens, O XML as tags não são pré-definidas e pode-se usar um número infinito de tags. O HTML isso não é possível.

Os arquivos XML são arquivos textos, e não são destinados à leitura por um ser humano como o HTML é. Isso fica a cargo de outras linguagens. As especificações de XML são muito mais rígidas que a especificação do HTML. Elas estabelecem que é obrigatório rejeitar sintaticamente arquivos incorretos, mesmo que o navegador possa descobrir o que o Web designer pretendia. Os navegadores só têm permissão para indicar o erro.

23.4 Sintaxe básica do XML

O componente básico no XML é o elemento, que é composto pela uma tag inicial, pelo conteúdo e pela uma tag final. A expressão `<book>` é chamada de

tag inicial (começa com `i` e termina com `i`) e `i/booki` é chamada de tag final (começa com `i/` e termina com `i`). Estas marcações são definidas pelo usuário. A estrutura ou o texto entre as tags iniciais e finais é chamado de conteúdo. As tags no XML são case sensitive.

Os elementos podem incluir outros elementos (ditos elementos filhos) e textos. No exemplo da figura 23.1, `booklist` é o elemento raiz e o `book` é o elemento filho. Usam-se elementos repetidos com a mesma tag (`book`) para representar coleções de dados, figura 23.1.

Um elemento pode não ter qualquer conteúdo. Este tipo de elemento é designado por elemento vazio, por exemplo, o elemento `\<figura path="...">` que marca onde deve ser inserido uma figura. Os elementos vazios possuem uma sintaxe diferente, inicia-se por `<` e termina em `>`, que é a forma abreviada de escrever `<elem-ident> <elem-ident>`. Não é permitida a utilização de espaço em branco no nome dos elementos. O nome dos elementos pode começar somente com letras, underscore ou dois pontos.

A linguagem XML permite associar atributos aos elementos. Os atributos são utilizados para especificar as propriedades ou características do elemento. O atributo é definido como um par (nome, valor) separado por um sinal de igual (=) e sempre aparecem na tag inicial. Assim como nas tags, o usuário pode definir também os atributos. O valor do atributo é sempre um conjunto de caracteres que deve estar entre aspas. Por exemplo:

```
<disciplina cod="cop025"> Técnicas de programação</disciplina>
```

Neste exemplo, `cod="3cop025"` é um atributo do elemento `<disciplina>`. Cada atributo de um elemento pode ser especificado somente uma vez. O XML define apenas dois atributos como reservado: `xml:lang` e `xml:space`.

Uma observação importante a ser feita a respeito de elemento e atributo é que não existe uma fronteira entre os dois, figura 23.2. A representação (b) está correta, mas é muito mais difícil de processar.

```
<agenda>
  <entrada id="e1" tipo="pessoa">
    <nome>José Carlos Ramalho</nome>
    <email>jcr@di.uminho.pt</email>
    <telefone>253 604479</telefone>
  </entrada>
  ...
</agenda> (a)

<agenda>
  <entrada id="e1" tipo="pessoa" nome="José Carlos Ramalho"
          email="jcr@di.uminho.pt" telefone="253 604479"/>
  ...
</agenda> (b)
```

Figura 23.2: Representações diferentes de uma estrutura em XML

Algumas regras básicas devem ser lembradas na construção de um documento XML:

- O documento deve ter sempre uma declaração XML no início (<?xml version="1.0"?>);
- O documento deve incluir um ou mais elementos – o primeiro elemento, que delimitará todo o corpo do documento, é o elemento raiz e todos os outros deverão estar incluídos dentro dele;
- Todos os elementos dever ter uma tag de início e outra de fim;
- Todos os elementos dever estar aninhados corretamente;
- Todos os valores de atributos devem estar entre aspas.

23.5 Conjunto de tags

As principais tags que podem ser utilizadas em um documento XML são:

InSTRUÇÃO DE PROCESSAMENTO mecanismo que permite a inserção de informações explícitas em um documento destinadas a uma aplicação. Os interpretadores XML não interpretam essas informações, mas simplesmente as repassam para a aplicação. A instrução de processamento não faz parte do conteúdo do documento;

Sintaxe:

```
<?      ?>
```

Exemplo:

```
<?html action="hr"?>
```

Declaracão XML é um tipo de instrução de processamento. Qualquer documento XML dever começar sempre com uma declaração XML. Normalmente, se contiver algo antes da declaração ou se estiver ausente, a aplicação acusará erro. Existem três tipos de atributos para a declaração: version (Obrigatório e indica a versão do XML utilizado, "1.0"), standalone (Opcional e indica se o documento está autocontido. Aceita os valores "yes" caso o documento não tenha referências a entidade externa e "no" caso contrário.), encoding (Opcional e indica qual codificação usada para os caracteres. O valor para caractere português é "iso-8859-1");

Sintaxe:

```
<?xml      ?>
```

Exemplo:

```
<?xml version="1.0" standalone="yes" encoding="UCS-2" ?>
```

Comentários utilizado para indicar que o conteúdo a seguir não é para ser processado;

Sintaxe:

<!-- -->

Exemplo:

```
<!--Isto é um comentário -->
```

Elementos explicado anteriormente;

Atributos explicado anteriormente;

CDATA utilizado quando se deseja que os caracteres de texto não sejam confundidos com os elementos de marcação, por exemplo o uso do sinal < e >;

Sintaxe:

```
<! [CDATA[ ... ]]>
```

Exemplo:

```
<! [CDATA[Imprima a tecla <<ENTER>>]]>
```

PCDATA (Parser Character Data) utilizado para designar que o texto entre as duas marcações é um dado e não um texto. Isso é utilizado na definição das DTDs ou gramática do documento;

Referências é possível associar identificadores únicos aos elementos, por meio de um atributo de identificação, id. Com isto permite-se que elementos distintos referenciem-se entre si;

Exemplo:

```
<biblio id="bib" ano="2001">
  ...
</biblio>
```

Entidade identifica um conjunto de informação por um nome. As entidades podem ser associar a textos e ficheiros. O XML possui cinco entidades pré-definidas: < (<), > (>), & (&), &apos (') e " (");

Sintaxe:

```
<!ENTITY ...>
```

Exemplo (entidade interna):

```
<!ENTITY assinatura "Jorge Manuel Neves Coelho">
  (assinatura= Jorge Manuel Neves Coelho)
```

Exemplo (entidade externa):

```
<!ENTITY ent01 SYSTEM "ents/ent01.xml">
```

A verificação sintática de um documento XML é feita por parsers que analisam um documento e enviam mensagens de erro ao detectar erros de sintaxe. Existem dois tipos de parser: o parser validador, que verifica se um documento é bem formado, e o parser não validador, que verifica se um documento é válido. Um documento XML é bem formado quando obedece a sintaxe XML, e um documento XML é válido quando obedece a uma gramática especificada por uma DTD. Pode-se dizer que um documento válido é um documento bem formatado, mas o contrário não.

23.6 NameSpaces

Como os nomes dos elementos em XML não estão pré-definidos, pode ocorrer um conflito de nome quando um documento XML importa outro documento XML, pois esses documentos podem usar o mesmo nome para descrever dois tipos diferentes de elementos, por exemplo, o elemento `<table>` da figura 23.3.

```
<table>
  <tr>
    <td>books</td>
    <td>magazines</td>
  </tr>
</table>

(a)

<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>

(b)
```

Figura 23.3: Diferentes documentos XML com mesmo nome de elemento

O NameSpace é um padrão definido pelo W3C e basicamente é utilizado para resolver conflitos de nomes usados nos documentos XML. A resolução de conflitos pode ser feita por meio do uso de prefixos ou de namespaces.

No uso de prefixos, basta o usuário definir um nome diferente para seus elementos `<table>`, por exemplo, `<h : table>` e `<f : table>`. Usando o prefixo, criam-se dois tipos diferentes de elementos do tipo `<table>`.

Para definir/criar um namespace o usuário deve definir o atributo `xmlns` (XML NameSpace) a tag inicial de um elemento no documento XML com a sintaxe `xmlns:namespace-prefix="namespace"`. Quando o namespace é definido a tag inicial do elemento, todos os elementos filhos com o mesmo prefixo são associados com o mesmo namespace. O atributo `xmlns` pode ser utilizado em qualquer elemento e não apenas no nó raiz. A figura 23.4 mostra alguns exemplos de namespace e a Ilustração 6 mostra a utilização de namespace em vez de prefixo para o exemplo da figura 23.3.

Pode-se perceber pela figura 23.4.(a) que a string identificadora de um namespace é grande para ser manipulada diretamente. Para que sua utilização

```
xmlns="http://xml.di.uminho.pt/XMLSamples/livro.dtd"
xmlns="livro.dtd"
xmlns="meu-URL/DTDs/livro.dtd"
(a)

xmlns:catalogo="http://xml.di.uminho.pt/Samples/livro.dtd"
xmlns:encomenda="livro.dtd"
xmlns:jcr="meu-URL/DTDs/livro.dtd"
(b)
```

Figura 23.4: Declaração de namespace

```
<table xmlns="http://www.w3.org/TR/HTML4/">
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
(a)

<table xmlns="http://www.w3schools.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
(b)
```

Figura 23.5: Exemplo de utilização de namespace

seja viável, utiliza-se um prefixo, uma espécie de abreviatura para os namespaces. De acordo com a figura 23.4.(b) os prefixos seriam catalogo, encomenda e jcr.

A especificação de namespaces da W3C estabelece que o namespace deve ter um URI para garantir a unicidade dos nomes dos elementos.

Note que o endereço usado para identificar o namespace não é usado pelo parser para buscar informação. O único propósito é dar ao namespace um nome único. Entretanto, é comum usar o namespace como um ponteiro para uma página Web real contendo informações sobre o namespace. Por exemplo, a figura 23.5 mostra a declaração de namespaces distintos para especificar onde são buscados os dados da definição de elementos com o mesmo nome, no caso `<table>`. Os namespaces são também utilizados em definições de XML Schema, XSL e RDF.

23.7 Gramática de um documento XML

Algumas vezes existe a necessidade de definir os elementos que podem aparecer em um documento XML, o conteúdos destes, e os atributos que podem estar associados aos elementos. Também, existe a necessidade de definir a estrutura do documento especificando, por exemplo, quais são os subelementos de um elemento e a seqüência que eles devem aparecer. A primeira linguagem proposta

para a definição dessas regras nos documentos XML foi a DTD (Document Type Definition).

Existem quatro tipos de declarações que podem ser utilizadas em um DTD: elementos, atributos, entidades e instruções de processamento. As entidades estão relacionadas com a organização física do documento XML, as instruções de processamento foram explicadas anteriormente, os elementos são definidos pela palavra-chave ELEMENT e a lista de atributos de cada elemento é definida pela palavra-chave ATTLIST. As principais declarações que ocorrem em um DTD são de elementos e de atributos.

As declarações de elementos especificam como deve ser o conteúdo do elemento. As seguintes declarações podem ser feitas:

- Definir a seqüência em os subelementos de um elemento devem aparecer:
(a, b)

Sintaxe:

```
<!ELEMENT nome_elemento  
(seqüência de subelemento separados por vírgula na  
ordem que devem aparecer)>
```

Exemplo:

```
<!ELEMENT biblio (título, autor, tipo_pub)>
```

- Definir elementos vazios: constante EMPTY Exemplo:

```
<!ELEMENT image EMPTY>
```

- Definir a ocorrência repetida de um elemento em particular:

– Pelo menos uma vez: (elemento)+ Exemplo:

```
<!ELEMENT biblio (titulo, autor+, tipo_pub)>
```

– Zero ou mais vezes: (elemento)* Exemplo:

```
<!ELEMENT tabela_biblio (biblio*)>
```

- Definir que o elemento deve ser um entre uma lista de elementos: (a|b)
Exemplo:

```
<!ELEMENT tipo_pub (periodico | evento)>
```

Obs.: O elemento *tipo_pub* é composto por um elemento periodico ou evento.

- Definir que o elemento tem presença opcional: (elemento)? Exemplo:

```
<!ELEMENT biblio (titulo, autor, tipo_pub, resumo?)>
```

- Definir que o elemento é uma string de caracteres: #PCDATA Exemplo:

```
<!ELEMENT autor (#PCDATA)>
```

- Definir elementos livres: constante ANY. Exemplo:

```
<!ELEMENT doc (para)+>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT lugar (#PCDATA)>
<!ELEMENT data (#PCDATA)>
<!ELEMENT para ANY>
```

Obs: O elemento para está definido como sendo de conteúdo ANY, ou seja, pode ser texto com elementos nome, lugar e data pelo meio (<para>Tudo aconteceu em <lugar> Braga </lugar> ... </para>).

A figura 23.6 mostra o documento XML que descreve uma agenda e a figura 23.7 mostra as especificações do documento DTD.

```
<AGENDA>
  <ENTRADA>
    <IDENT>e1</IDENT>
    <TIPO>pessoa</TIPO>
    <NOME>José Carlos Ramalho</NOME>
    <EMAIL>jcr@di.uminho.pt</EMAIL>
    <TELEFONE>253 604479</TELEFONE>
  </ENTRADA>
  <GRUPO>
    <IDENT>ep1</IDENT>
    <ENTRADA>
      <IDENT>e2</IDENT>
      <TIPO>pessoa</TIPO>
      <NOME>Pedro Henriques</NOME>
      <EMAIL>prh@di.uminho.pt</EMAIL>
      <TELEFONE>253 604469</TELEFONE>
    </ENTRADA>
    <ENTRADA>
      <IDENT>e3</IDENT>
      <TIPO>pessoa</TIPO>
      <NOME>João Saraiva</NOME>
      <EMAIL>jas@di.uminho.pt</EMAIL>
```

Figura 23.6: Documento XML contendo a descrição de uma agenda

```
<!ELEMENT AGENDA (ENTRADA | GRUPO)+>
<!ELEMENT ENTRADA (IDENT, TIPO, NOME, EMAIL, TELEFONE) >
<!ELEMENT GRUPO (IDENT, (ENTRADA | GRUPO | REF)+)>
<!ELEMENT NOME (#PCDATA) >
<!ELEMENT EMAIL (#PCDATA) >
<!ELEMENT TELEFONE (#PCDATA) >
<!ELEMENT IDENT (#PCDATA) >
<!ELEMENT TIPO (#PCDATA) >
<!ELEMENT REF (#PCDATA) >
```

Figura 23.7: Conjunto de declarações para o XML que descreve a agenda

As declarações de atributo servem para especificar o nome, o tipo de dado associado ao atributo e opcionalmente especificar se há algum valor padrão de atributo: Sintaxe:

```
<!ATTLIST nome_elemento {nome_atributo tipo_atributo}> onde {nome_atributo tipo_atributo}
```

Obs.: Alternativamente pode-se utilizar a seguinte sintaxe:

```
<!ATTLIST nome_elemento nome_atributo1 tipo_atributo1>
<!ATTLIST nome_elemento nome_atributo2 tipo_atributo2>
```

As seguintes declarações de tipo de atributo podem ser feitas:

- Valor do atributo deve ser uma string de qualquer tamanho: CDATA
Exemplo:

```
<!ATTLIST biblio ano CDATA local CDATA >
```

- Valor do atributo pode ser um entre uma lista de valores possíveis: —
Exemplo:

```
<!ATTLIST tipo_pub abrange (regional | nacional | internacional)>
```

Valor do atributo significa um identificador do elemento ao qual está associado: ID Exemplo:

```
<!ATTLIST biblio ident ID>
```

Obs.: Cada elemento que possui o atributo ID deve ter um único atributo ID e para cada elemento o valor ID deve ser diferente.

- Valor do atributo significa um nome de identificador referenciável: IDREF
Exemplo:

```
<!ATTLIST biblio refer IDREFS>
```

Obs.: Quando da ocorrência de atributo com múltiplos valores do tipo IDREF, utiliza-se a declaração IDREFS. Usa-se o IDREF para implementar relações entre elementos.

Opcionalmente é possível estabelecer se a presença de um atributo é obrigatória, opcional ou se assume um valor padrão. As seguintes declarações são possíveis:

- Declarar que a presença do atributo é obrigatória: #REQUIRED após a declaração do tipo de atributo

Exemplo:

```
<!ATTLIST biblio ano CDATA #REQUIRED local CDATA>
```

- Declarar que a presença do atributo é opcional: #IMPLIED após a declaração do tipo de atributo

Exemplo:

```
<!ATTLIST biblio ano CDATA #REQUIRED local CDATA #IMPLIED>
```

- Especificar um valor padrão quando o valor do atributo deve ser escolhido entre uma lista de valores:

Exemplo:

```
<!ATTLIST aula tipo (teoria | exemplo | exercício) "teoria">
```

- Declarar que a presença do atributo é fixa: `#FIXED` após a declaração do tipo de atributo e seu valor será o que estiver a frente da palavra `#FIXED`. Exemplo:

```
<!ATTLIST data tipo CDATA #FIXED "data">
```

Obs.:

1. O atributo a ser definido como `#FIXED` é constante e imutável.
2. Definição de uma entidade paramétrica (somente no DTD):

```
<!ELEMENT a (c|d|e)>           <!ELEMENT a %ab;>
                               => <!ENTITY % ab "(c|d|e)"> =>
<!ELEMENT b (c|d|e)>           <!ELEMENT b %ab;>
```

Quando um documento XML atende as especificações de um esquema, este documento é chamado de documento válido. Para que o interpretador XML possa validar um documento XML quanto a sua gramática é necessário associar o DTD ao documento XML. Esta associação, que é especificada no início do documento DTD, pode ser feita tanto de forma interna quanto de forma externa.

Na forma interna, as declarações DTD são colocadas explicitamente no arquivo XML, através da sintaxe:

```
<!DOCTYPE nome_DTD [declarações da DTD]>
```

Exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE agenda [
  <!ELEMENT AGENDA (ENTRADA | GRUPO)+>
  <!ELEMENT ENTRADA (NOME, EMAIL, TELEFONE)>
  ...
]>
<AGENDA>
  ...
</AGENDA>
```

Figura 23.8: Associação entre o XML e o DTD na forma interna

Na forma externa, a associação é feita no documento XML por meio da sintaxe:

```
<!DOCTYPE nome_DTD SYSTEM "nome do arquico DTD">
```

Exemplo:

```
<!DOCTYPE agenda SYSTEM "agenda.dtd">
<!DOCTYPE agenda SYSTEM "http://xml.idi.uminho.pt/DTDs/agenda.dtd">
```

Obs.: Esta declaração indica que o elemento raiz do documento é agenda e que o DTD pode ser encontrado no sistema no arquivo agenda.dtd.

A declaração DOCTYPE não pode aparecer livremente num documento XML. Ela deve aparecer sempre após a declaração XML e antes do elemento raiz, figura 23.9.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE agenda SYSTEM "agenda.dtd">
<AGENDA>
  ...
</AGENDA>
```

Figura 23.9: Posição da declaração DOCTYPE

Pelas declarações básicas que uma DTD pode utilizar, observa-se que os tipos de dados que podem ser declarados são de certa forma limitados. Para ampliar as possibilidades de se definir tipos de dados, outro tipo de esquema foi definido pelo W3C denominado XML Schema ou XSD (XML Schema Definition).

23.8 Tecnologias XML

A seguir são apresentadas as tecnologias relacionadas ao XML:

XHTML (Extensible HTML) é uma versão explícita do HTML;

XML DOM (XML Document Object Model) define um padrão de acesso e manipulação de documentos XML;

XSL (Extensible Style Sheet Language) consiste de três partes: XSLT, XPath e XML-FO;

XSLT (XSL Transformation) é usado para transformar documentos XML em outros formatos, como XHTML;

XPath é uma linguagem para navegação em documentos XML;

XSL-FO (XSL Formatting Object) é uma linguagem para formatação de documentos XML;

XLink (XML Linking Language) é uma linguagem para criar hyperlinks em documentos XML;

XPointer (XML Pointer Language) permite os hyperlinks XLink apontar para partes específicas de um documento XML;

DTD (Document Type Description) é usado para definir a gramática de um documento XML;

XSD (XML Schema Definition) é baseado em XML alternativo ao DTD;

- XForms (XML Forms)** usa XML para definir a forma dos dados;
- XQuery (XML Query Language)** é usado para consultar dados em XML;
- SOAP (Simple Object Access Protocol)** é um protocolo em XML que permite as aplicações trocarem informações sobre o protocolo http;
- WSDL (Web Services Description Language)** é uma linguagem baseada em XML para descrever web services;
- RDF (Resource Description FrameWork)** é uma linguagem baseada em XML para descrever recursos da Web;
- RSS (Really Simple Syndication)** é um formato para indicar notícias e conteúdos novos de um site;
- WAP (Wireless Application Protocol)** foi desenvolvido para mostrar conteúdo para cliente wireless, como celulares;
- SMIL (Syncronized Multimedia Integration Language)** é uma linguagem para descrever apresentações audiovisuais;
- SVG (Scalable Vector Graphics)** define formatos gráficos em XM.

A figura 23.10 mostra como algumas dessas tecnologias interagem com a linguagem XML.

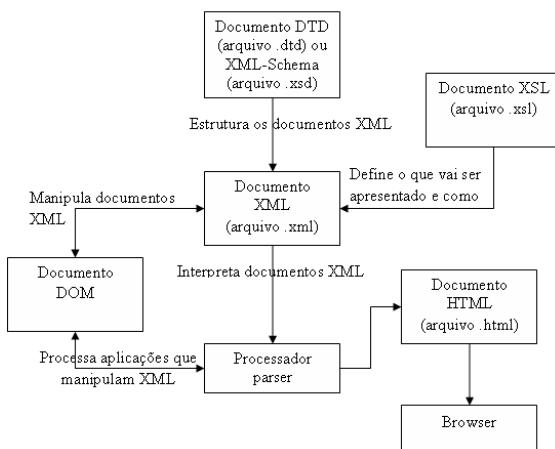


Figura 23.10: Interação entre as tecnologias XML

23.9 Benefícios da linguagem XML

Os principais objetivos da linguagem XML:

- Buscas mais eficiente;
- Desenvolvimento de aplicações flexíveis para a Web;

- Integração de dados de fontes diferentes;
- Computação e manipulação local;
- Múltiplas formas de visualizar os dados;
- Atualização granulares dos documentos;
- Fácil distribuição na Web;
- Escalabilidade;
- Compreensão.

23.10 Ferramentas de desenvolvimento

Parsers:

- Expat;
- XML4J;
- MSXML;

Editores:

- Editores de texto comum;
- Xeena;

Capítulo 24

XSLT

Uma das principais idéias de XML é deixar explícita a separação entre conteúdo, estrutura e apresentação de um documento para Web. Como XML trata apenas do conteúdo dos documentos, devem existir outros padrões para descrever o formato de apresentação dos dados. Uma das formas de permitir apresentação de tais documentos é por meio das folhas de estilo.

Linguagens de folhas de estilo (*style sheets*) foram projetadas para permitir que a descrição de estilo de apresentação de documentos seja separada da representação do conteúdo dos documentos. Isto facilita a reutilização de um mesmo documento para diferentes aplicações ou diferentes tipos de usuários que requerem diferentes visões do mesmo. O princípio básico destas linguagens é prover uma sintaxe que permita associar partes específicas do conteúdo de documentos a estilos ou ações que devem ser realizadas sobre tais partes. As principais linguagens de estilo são a CSS (*Cascading Style Sheets*) e XSL (*Extensible Style Sheet*).

24.1 O que é uma folha de estilo?

Quando um site Web é complexo e consiste de muitas páginas produzidas por vários autores que trabalham na mesma empresa, frequentemente é interessante ter um meio de impedir que diferentes páginas tenham uma aparência distinta. Esse problema pode ser resolvido usando-se folhas de estilo.

Quando as folhas de estilo são utilizadas, as páginas individuais deixam de usar estilos físicos, como negrito e itálico. Em vez disso, os desenvolvedores utilizam estilos próprios como *< dn >* (definição), *< em >* (ênfase fraca), *< strong >* (ênfase forte) e *< var >* (variáveis). Os estilos lógicos são definidos na folha de estilo, referida no início de cada página. Uma folha de estilo pode ser comparada a um arquivo *#include* em um programa C: a mudança em uma única definição de macro provoca a alteração em todos os arquivos de programa que incluem o cabeçalho.

24.2 Comparaçāo entre o CSS e XSL

A semelhança entre as essas duas folhas de estilo, CSS e XSL, está em ambas podem ser usadas em documentos XML, mas o CSS não transforma documentos.

Essa possibilidade está reservada somente ao XSL. XSL é mais poderoso que o CSS, pois suporta transformações do documento antes de sua exibição. Por outro lado, o XSL não pode ser usado em linguagens HTML.

Desta forma essas duas linguagens completam-se uma à outra e podem ser utilizadas simultaneamente. A utilidade do XSL é mais perceptível, quando, por exemplo, é necessária a ordenação dos dados antes de serem exibidos.

24.3 O que é o XSL?

O XSL é uma linguagem para descrever folhas de estilo para documentos XML. É composta por três linguagens descendentes de XML:

XSLT (XSL Transformation) linguagem para transformar documentos XML;

XPath linguagem para definir parte de um documento XML;

XSL-FO (XSL Formatting Object) linguagem para formatar documentos XML.

A componente mais importante do XSL é o XSLT.

24.4 O que é o XSLT?

O XSLT é a parte do XSL usada para transformar um documento XML em outro documento XML, ou em outro tipo de documento (tx, pdf, rtf, HTML, XHTML, etc.). O XSLT utiliza o XPath para encontrar informações nos documentos XML. XPath é usado para navegar através dos elementos e atributos nos documentos XML.

O XSLT define um conjunto de regras de transformação que serão aplicadas a um documento XML (árvore fonte) para produzir outro documento XML (árvore resultado). Nesse processo de transformação, o XSLT utiliza o XPath para definir partes do documento de origem (fonte) que combinem com um ou mais templates pré-definidos. Quando uma combinação é encontrada, o XSLT transforma essas partes do documento de origem no documento de resultado. As partes do documento de origem que não são combinadas com o template permanecerão sem modificações no documento de resultado.

O XSLT permite:

- Adicionar texto ao conteúdo de elementos;
- Remover, criar, alterar e ordenar os conteúdos dos elementos;
- Converter conteúdo de elementos para valores de atributos, ou vice-versa;
- Alterar a ordem dos elementos;
- Substituir elementos por novos elementos.

24.5 Características do XSLT

Algumas características do XSLT:

- É uma linguagem declarativa – descreve a transformação desejada, ao invés de prover uma seqüência de instruções procedurais;
- É essencialmente uma "ferramenta" para transformar documentos XML;
- Manipula árvores;
- Uso de XSL NameSpaces – o W3C provê um namespace para as tags XSL.

24.6 Declarando um documento XSL

Como o XSL obedece a mesma sintaxe do XML, quando se cria uma especificação de folha de estilo XSL, deve-se incluir a instrução de processamento (por exemplo, <?xml version="1.0" standalone="yes" encoding="UCS-2"?>) para arquivos XML no início do arquivo que descreve a folha de estilo. Os arquivos XSL tem extensão .xsl. A figura 24.1 mostra um exemplo de um arquivo XSL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
    <body>
      <h2>My CD Collection</h2>
      <table border="1">
        <tr bgcolor="#9acd32">
          <th align="left">Title</th>
          <th align="left">Artist</th>
        </tr>
        <xsl:for-each select="catalog/cd">
          <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="artist"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Figura 24.1: Declaração de uma folha de estilo XSL

O elemento raiz que declara um documento ser uma folha de estilo XSL é o elemento <xsl:stylesheet> ou <xsl:transform>. Esses dois elementos são sinônimos e são as tags mais externas de um documento XSL.

O modo correto de declarar uma folha de estilo XSL de acordo com a recomendação W3C XSLT:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

ou

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Obs.:

1. Para obter acesso aos elementos, atributos e característica do XSLT, deve-se declarar o namespace XSLT no início do documento;
2. O `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` identifica a Recomendação NameSpace oficial do W3C. Caso utilize este namespace, deve incluir o atributo `version="1.0"`.

O restante da declaração na folha de estilo XSL é referente a como transformar o documento XML. Essa transformação é realizada utilizando os principais elementos XSLT:

- `<xsl:template>`;
- `<xsl:value-of>`;
- `<xsl:for-each>`;
- `<xsl:sort>`;
- `<xsl:if>`;
- `<xsl:choose>`;
- `<xsl:apply-templates>`.

Após a definição do arquivo XSL é preciso associar este arquivo no documento XML. A instrução que deve ser incluída no documento XML para que o interpretador XML faça uso da folha de estilo XSL definida é:

```
<?xml-stylesheet type="text/xsl" href="localização do arquivo.xsl"?>
```

A figura 24.2 mostra o arquivo XML com esta associação. A figura 24.1 é a declaração da folha de estilo para esse documento XML. A figura 24.3 mostra a visualização do documento XML no browser utilizando a folha de estilo definida anteriormente. Ao abrir o documento XML o browser transforma o documento XML em HTML.

24.7 Elemento `<xsl:template>`

O elemento `<xsl:template>` é utilizado para construir templates. Cada template contém regras para aplicar quando um nó (elemento) específico é combinado. O atributo `match` desse elemento é usado para associar um template com um elemento XML ou definir um template para toda uma seção de um documento

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
</catalog>
```

Figura 24.2: Documento XML com "link" para a folha de estilo XSL

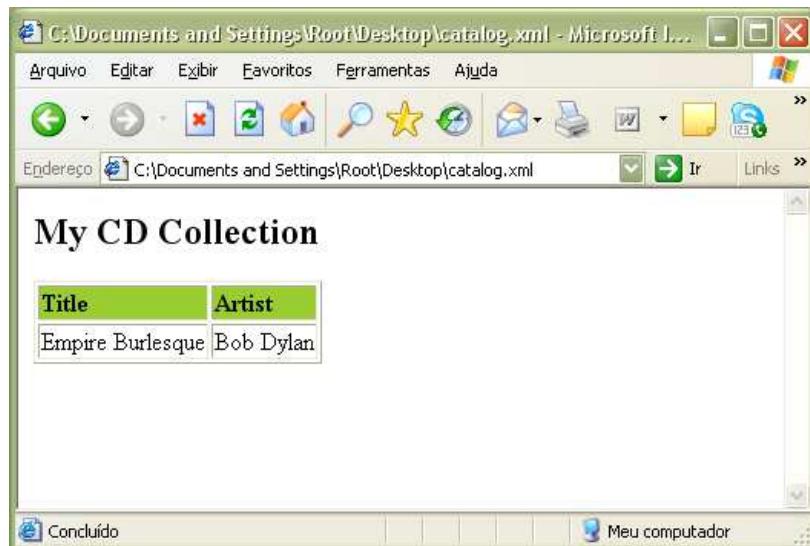


Figura 24.3: Visualização do arquivo XML no browser

XML. Para associar todo um documento XML o valor do atributo `match=" /"`. Para associar apenas a um elemento do documento XML o valor do atributo `match="nome do elemento"`. O valor do atributo `match` é uma expressão XPath. A figura 24.1 mostra um exemplo do uso do elemento `<xsl:template>`.

24.8 Elemento `<xsl:value-of>`

O elemento `<xsl:value-of>` é usado para extrair o valor de um elemento no documento XML e adicioná-lo ao documento resultado. O atributo `select` desse elemento informa qual o elemento que será extraído o valor. O valor do atributo `select` é uma expressão XPath. A figura 24.1 mostra um exemplo do uso do elemento `<xsl:value-of>`.

24.9 Elemento <xsl:for-each>

O elemento <xsl:for-each> permite a realização de loops no XSLT. Esse elemento é usado para selecionar todos os elementos XML de um determinado tipo. O atributo select desse elemento seleciona o elemento no documento XML que será processado. O valor desse atributo é uma expressão XPath. A figura 24.1 mostra um exemplo do uso do elemento <xsl:for-each>.

O elemento <xsl:for-each> também pode ser usado para filtrar a saída de um documento XML. Basta acrescentar um critério de seleção ao atributo select desse elemento. Por exemplo, <xsl:for-each select="catalog/cd[artist='Bob Dylan']">, mostra apenas que o catálogo com o nome de artista Bob Dylan será colocado no documento de saída. Os operadores de filtro são: = (igual); != (diferente); < (menor que); > (maior que).

24.10 Elemento <xsl:sort>

O elemento <xsl:sort> é usado para ordenar a saída de um documento. Para ordenar a saída, insere-se esse elemento dentro do elemento <xsl:for-each>. O atributo select desse elemento informa por qual elemento XML ocorre a ordenação. A figura 24.4 mostra o documento XML da figura 24.2 com a saída ordenada pelo elemento artist.

```
...
<xsl:for-each select="catalog/cd">
  <xsl:sort select="artist"/>
  <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="artist"/></td>
  </tr>
</xsl:for-each>
...
```

Figura 24.4: Arquivo XSL ordenando a saída de um documento

24.11 Elemento <xsl:if>

O elemento <xsl:if> é usado para realizar testes condicionais. Para realizar um teste condicional, basta inserir esse elemento dentro do elemento <xsl:for-each>. O atributo test contém a expressão booleana. A figura 24.5 mostra o documento XSL com o teste condicional, o qual seleciona apenas os cds com preços maiores que 10.

```
...
<xsl:for-each select="catalog/cd">
  <xsl:if test="price > 10">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
  </xsl:if>
</xsl:for-each>
...
```

Figura 24.5: Arquivo XSL com teste condicional

24.12 Elemento `<xsl:choose>`

O elemento `<xsl:choose>` é usado em conjunto com os elementos `<xsl:when>` e `<xsl:otherwise>` para expressar múltiplos testes condicionais. A mostra o uso do elemento `<xsl:choose>`. Neste exemplo, os elementos cd que possui preço maior que 10 aparecem pintados de rosa e os outros aparecem normais no documento de saída.

```
...
<xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <xsl:choose>
      <xsl:when test="price > 10">
        <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/></td>
      </xsl:when>
      <xsl:otherwise>
        <td><xsl:value-of select="artist"/></td>
      </xsl:otherwise>
    </xsl:choose>
  </tr>
</xsl:for-each>
...
```

Figura 24.6: Arquivo XSL com múltiplos testes condicionais

24.13 Elemento `<xsl:apply-templates>`

O elemento `<xsl:apply-templates>` aplica um template ao elemento corrente ou aos filhos do elemento nó corrente. Se adicionar o atributo `select` ao elemento `<xsl:apply-templates>`, o template só será aplicado aos elementos filhos que combinem com o valor do atributo. O valor do atributo `select` pode ser usado para especificar a ordem na qual os nós filhos são processados. A figura 24.7 mostra um exemplo com a utilização desse elemento. A figura 24.8 mostra como é a visualização do documento resultado.

Apesar do XSL ter elementos de loop, condições de teste, e etc., as páginas Web em XML e XSL ainda são estáticas como o HTML.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="cd">
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
</p>
</xsl:template>
<xsl:template match="title">
Title: <span style="color:#ff0000">
<xsl:value-of select=". "/></span>
<br />
</xsl:template>
<xsl:template match="artist">
Artist: <span style="color:#00ffff">
<xsl:value-of select=". "/></span>
<br />
</xsl:template>
</xsl:stylesheet>

```

Figura 24.7: Arquivo XSL com o uso do elemento <xsl:apply-template>

24.14 XSL no lado Cliente

Foi explicado anteriormente como o XSL pode ser usado para transformar um documento XML em HTML. O "truque" foi adicionar uma descrição XSL no arquivo XML e deixar a transformação para o browser. Embora isso funcione bem, não é sempre desejado incluir a referência ao XSL no arquivo XML essa solução não funcionaria com um navegador sem nenhuma facilidade XSL.

Uma solução mais versátil é usar o JavaScript para fazer a transformação de XML para HTML. Usando-se JavaScript pode-se:

- Permitir que o JavaScript faça um teste de navegação específico;
- Usar diferentes modelos de estilo de acordo com o navegador e/ou as necessidades de usuário.

Usando JavaScript é possível transformar dados de um formato para outro, suportando diferentes navegadores e diferentes necessidades dos usuários.

24.15 XSL no lado Servidor

A solução apresentada anteriormente utilizando JavaScript esbarra no problema de o browser não suportar um parser XML, pois a transformação não

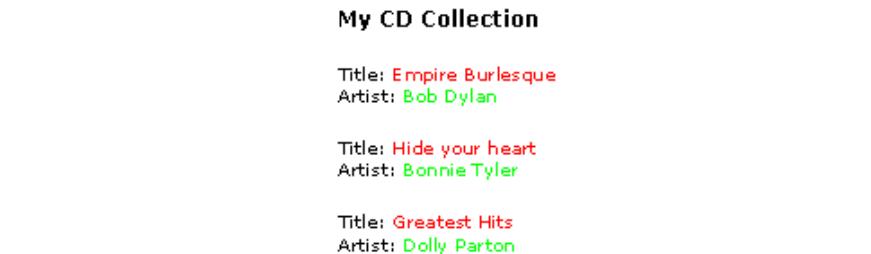


Figura 24.8: Visualização do arquivo XSL com uso do <xsl:apply-template>

funcionaria.

Para disponibilizar os dados XML para todos os tipos de navegadores, o documento XML é transformado no servidor e enviado como HTML puro para o browser. Isso pode ser feito utilizando a linguagem ASP.

24.16 Processadores XSLT

Os processadores XSLT aplicam uma folha de estilo XSLT a um documento XML e produz um documento resultado. Alguns exemplos de processadores: Saxon, XT, MSXML3, Xalan (Apache) e os browsers (IE 6.0 e NetScape 6.0).

Capítulo 25

Gerenciador de Conteúdo Web Zone/Plone

25.1 Gestão de Conteúdo

Os produtos e serviços de informação – dados, textos, imagens, sons, softwares, etc. – são identificados na rede com o nome genérico de conteúdos. A idéia básica da Gestão de Conteúdos (GC) é agilizar o processo de criação, gestão e publicação de informação. Em geral, os sistemas de gestão de conteúdos automatizam o processo de gestão e publicação e permitem que usuários não técnicos possam criar conteúdos com maior facilidade.

O processo de gestão de conteúdo, figura 25.1, envolve as seguintes etapas:

1. Criação de documentos;
2. Revisão de documentos;
3. Inclusão de metadados (Indexação) e controle de qualidade;
4. Publicação;
5. Revisão periódica
6. Arquivamento ou eliminação dos documentos.

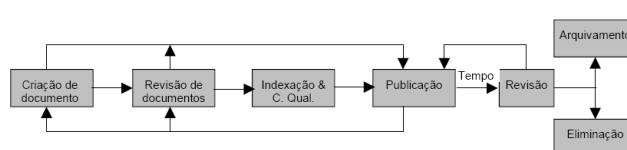


Figura 25.1: O processo de gestão de conteúdo

O processo de gestão pode ser representado em três partes básicas: criação, gestão e publicação. A fase de gestão envolve as etapas de indexação e controle de qualidade, revisão, arquivamento e eliminação de documentos.

Os componentes do gerenciamento de conteúdo são a administração de conteúdo, gerenciamento de workflow, acesso e segurança, bem como a customização e integração com sistemas legados.

Os sistemas de gestão de conteúdo são úteis em diversas aplicações, como por exemplo, comunidades práticas, portais corporativos e sites editoriais.

25.2 Sistema de Gestão de Conteúdo

Em geral, um sistema de gestão de conteúdo é composto por módulos que fornecem serviços que garantem um processo mais ágil na criação, gestão e publicação de conteúdos. As funcionalidades essenciais existentes nesses sistemas são:

Gestão de usuários e de seus direitos permite o controle de acesso por usuários, incluindo ferramentas de criação de perfis de usuários, autenticação, autorização e auditorias;

Criação, edição e armazenamentos de conteúdos suporte à criação, edição e manipulação de conteúdos, considerando múltiplos tipos (áudio, vídeo, imagem, xml, html, texto, etc.);

Metadados (propriedades que descrevem o conteúdo) descrevem características importantes do conteúdo como descrição, autor, linguagem, data da criação, data da revisão, etc. Os metadados possibilitam o controle de acesso, controle de qualidade, classificação e eliminação automática de documentos;

Gestão de qualidade inclui regras associadas aos tipos de conteúdo que permitem controle e acompanhamento do ciclo de vida. Um sistema workflow consiste na automatização de procedimentos na qual documentos, informações ou tarefas são passadas de um participante a outro, governado por regras;

Classificação, indexação e busca de conteúdo inclui mecanismos automatizados de classificação e indexação e de recursos de busca eficientes baseados em metadados;

Gestão de interface com usuários o conteúdo é independente da lógica da aplicação e da apresentação visual. A publicação é dinâmica em função do usuário e do dispositivo de saída. O acesso ao conteúdo é controlado e pode ser agrupado em três áreas de controle: Acesso público, Acesso restrito por licença e acesso privilegiado (controlado a partir de regras relacionadas ao conteúdo e usuário);

Sindicalização (syndication) técnica que permite compartilhar informação entre diferentes sites por meio do formato RSS;

Gestão de versões permite manipular diferentes versões do site, ou de um conjunto de conteúdos;

Gravação das ações e possibilidade de desfazê-las permite que se recupere de erros. Todas as mudanças no site (incluindo mudanças na lógica, apresentação e conteúdo) podem ser desfeitas.

No mercado de gestão de conteúdos, o ambiente Zope, uma framework para gestão de conteúdos, e o Plone, sistema de gestão de conteúdos, ambos de código aberto, tem atraído um número crescente de empresas e usuários.

Obs.: Um framework (literalmente: "moldura, esqueleto, armação") é um amálgama de serviços de "softwares" atômicos coerentes. Considerando como "tijolos" de base, estes serviços são montados para formar uma aplicação. A equipe pode concentrar-se nas regras de negócios e não nos aspectos técnicos dos projetos.

25.3 Zope

O Zope (Z Object Publishing Environment) é uma plataforma de desenvolvimento de aplicações Web baseada em Python e oferece um sistema de gestão de conteúdos, onde o usuário pode publicar e gerenciar o conteúdo Web. É um ambiente totalmente Orientado a Objetos e o permite o desenvolvimento via Web. O Zope integra um grande número de ferramentas e funcionalidades como uma base de dados de objetos, um módulo de publicação de objetos Web e uma linguagem de geração dinâmica de páginas.

O Zope é um ambiente multiplataforma (funciona em Unix, Linux, Mac Os e Windows). É registrado com a licença ZPL (Zope Public License), compatível com GPL (General Public License).

Diferente das outras soluções do mercado, a finalidade do Zope não é publicar páginas HTML, mas objetos que podem ser montados automaticamente a partir de componentes cujo comportamento, dados e aparência são configuráveis pelo projetista do site. Esta abordagem torna o Zope mais apto à publicação de conteúdo Web que os outros produtos.

A arquitetura do Zope, mostrada na figura 25.2, é constituída pelos seguintes módulos:

Servidor Web (ZServer) O Zope possui seu próprio servidor web, o Zope Server, e dispensa a presença de qualquer outro servidor web. Entretanto não impede a utilização de outro servidor web como Apache e Microsoft IIS. ;

Banco de dados orientado a objetos (ZODB) o Zope possui seu próprio servidor de Banco de Dados Orientado a Objetos (ZODB), onde armazena os objetos em uma representação simples e eficiente, que é formada pela classe do objeto e uma estrutura de dicionário contendo nas chaves as propriedades do objeto e nos valores associados às chaves que cada propriedade possui;

ZClasses Funcionam como molduras para os novos objetos no Zope. São estruturas utilizadas para criar novos objetos Zope, que podem ter como base outros objetos. Um objeto Zope pode ser uma pasta, um documento DTML, um método, etc. ZClasses podem ser criadas e ampliadas utilizando apenas a interface web;

Zope Products São produtos implementados em Python;

Integração com banco de dados relacionais O Zope apresenta componentes de acesso a vários bancos de dados, por exemplo, MYSQL, Oracle, PostgreSQL, Interbase, etc;

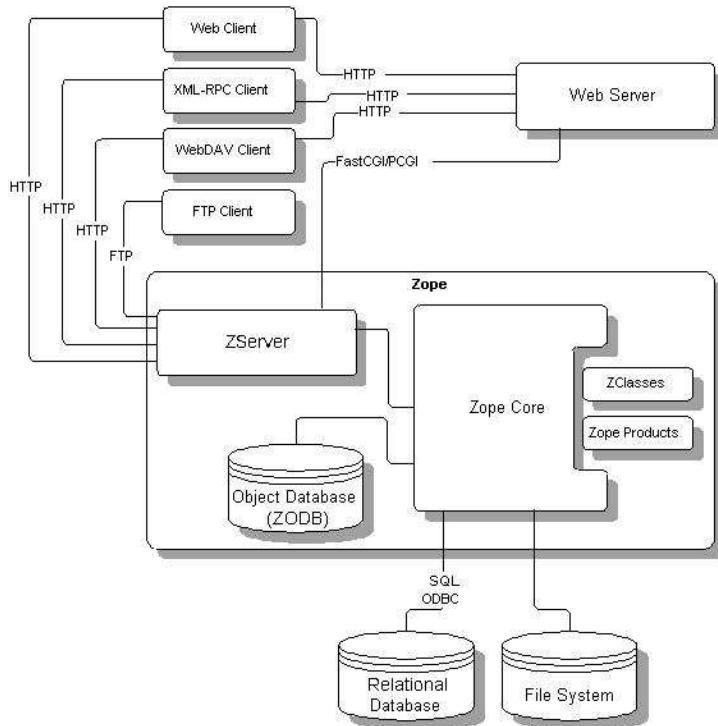


Figura 25.2: Arquitetura do Zope

Suporte a linguagens scripts Suporte a Phyton, ZPT (Zope Page Templates) e DTML (Document Template Markup Language). A ZPT permite a criação de modelos de páginas para apresentação. A DTML é uma linguagem de criação de scripts que permite ligar dados dinâmicos e modelos, definindo a apresentação visual desses dados;

Interface Web de gerenciamento e desenvolvimento Zope apresenta uma poderosa interface de gerenciamento e desenvolvimento (ZMI – Zope Management Interface) que permite adicionar, editar e remover objetos, configurar propriedades, via browser padrão. Inclui acesso via ftp ou WebDAV e oferece suporte aos protocolos DOM, XML, SOAP e XML-RPC.

A idéia principal em torno do Zope é a construção de um ambiente de publicação de objetos. Nesse contexto, elementos constituintes de uma página HTML são vistos como objetos que podem ser publicados e gerenciados através do Zope. Sua arquitetura flexível e baseada em componentes atende a diversas aplicações:

Gerenciar Websites o Zope inclui recursos como a possibilidade de delegação de acesso a diferentes níveis, separação do conteúdo, lógica e apresentação, controle de acesso, controle de versões, segurança, etc;

Construir sistemas de gestão de conteúdo a arquitetura Zope inclui componentes e disponibiliza produtos, que fornecem ferramentas que possi-

bilitam criar, gerenciar e publicar conteúdos, de modo fácil e seguro, por usuários não técnicos;

Desenvolver aplicações Web personalizadas o Zope fornece um framework com aplicações sofisticadas, como comércio-eletrônico, portais, fóruns e aplicações personalizadas. O Zope fornece componentes funcionais de acesso a dados, segurança e personalização, que integrados facilitam a construção de aplicações Web rapidamente;

Desenvolver portais corporativos o Zope possibilita o desenvolvimento de um site organizacional ou de uma comunidade de usuários, tal que o conteúdo, como notícias, documentos, eventos, seja fornecido por membros da organização. O ambiente Zope, através de sua arquitetura flexível e de produtos desenvolvidos pela comunidade de usuários, especialmente o Plone (um produto para portais), fornece elementos necessários para a construção rápida de um portal (apresentação e personalização, organização e gerenciamento, integração com diversas fontes de dados, mecanismos de indexação e busca, segurança, escalabilidade, etc.).

25.4 Plone

O Plone é um sistema gestão de conteúdo livre e de código aberto que roda em servidores Zope. Surgiu de uma evolução do CMF (Content Management Framework) do Zope. O Plone foi escrito em Python e roda sobre o Zope o CMF. O Plone vem com um sistema de workflow, segurança e funções pré-configuradas, um conjunto de tipos de conteúdo e suporte a várias línguas. O Plone não substitui o CMF, ele o complementa em funcionalidade e também em ambigüidade de interface com usuário, apresentando uma interface mais amigável para o usuário final. A figura 25.3 mostra a arquitetura do Zope, CMF e Plone.

Algumas características do Plone:

Internacionalização a interface de usuários é traduzida para diversas línguas;

Usabilidade permite desenvolver sites com alto nível de usabilidade e acessibilidade;

Personalização de templates o Plone separa o conteúdo do template no qual o conteúdo está sendo exibido. Os templates do Plone são escritos em códigos HTML com folhas de estilo CSS;

Personalização de registro de usuários o Plone possui um completo sistema de registro de usuários, onde é possível personalizar o registro de usuários. É possível usar informações de usuários de outros bancos de dados relacional (LDAP, AD, e outros);

Workflow e segurança o workflow controla a lógica de processamento de conteúdo de um site. É possível configurar o workflow na web utilizando ferramentas gráficas. Para cada item de conteúdo em um site Plone é possível construir uma lista de usuários que podem acessá-lo e se estão habilitados a interagir com este conteúdo (editar, ver, excluir, comentar, etc.);

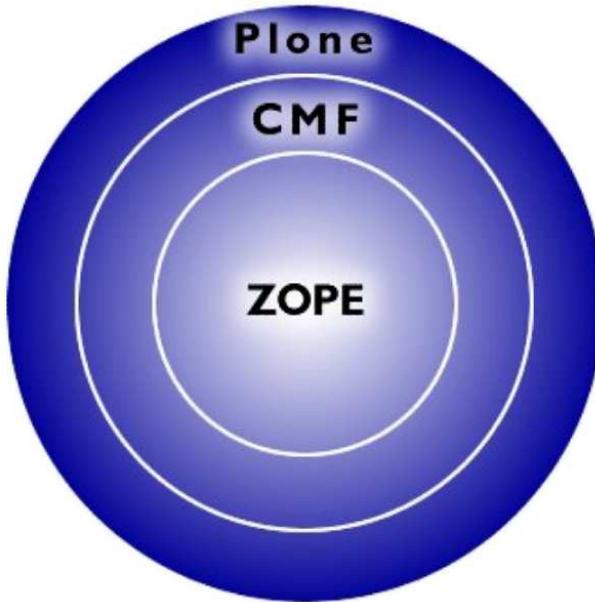


Figura 25.3: Arquitetura Zope, CMF e Plone

Extensível como o Plone é de código aberto é possível alterá-lo sem problemas.

Com ferramentas de desenvolvimento como Archetype, é possível gerar e alterar código do Plone por meio da Web;

Personalização de conteúdo o administrador de sites pode desenvolver seus próprios tipos de conteúdos e gerenciá-los conforme as necessidades. Com a ferramenta Archetype, é possível gerar novos tipos de conteúdo usando ferramentas de UML;

Comunidade Plone uma das melhores características do Plone é comunidade de desenvolvedores e empresas que prestam suporte ao desenvolvimento.

Os tipos básicos de conteúdo do Plone são:

Documento apresenta informações estáticas para o usuário. É o mais usado e pode ser considerado como uma página de um site;

Evento representam reuniões, eventos, encontros, seminários, e são utilizados nas pesquisas por tópico de calendário;

Arquivo objetos podem conter arquivos passíveis serem baixados;

Pasta serve para adicionar outros tipos de conteúdo. Pode ser usada para definir visões de informações personalizadas. É como uma pasta no disco rígido;

Imagem é uma imagem do tipo GIF ou JPEG;

Link são endereços Web;

Notícia contém pequenos artigos e podem possuir título assim como uma descrição opcional.

Algumas definições relacionadas ao Plone:

Portlet Um site Plone possui uma três colunas por padrão. As colunas da esquerda e da direita possuem uma série de caixas que mostram algum tipo de informação. Cada uma dessas caixas é chamada de portlet;

ZPT (Zope Page Template) é um sistema de templates usados pelo Plone para exibição de algum conteúdo;

Plone workflow por meio das permissões padrão (pendente, público, privado e esboço) do workflow Plone é possível criar níveis de acesso a informações, visões de conteúdos diferenciadas e regras de administração de conteúdo específicas para cada tipo de conteúdo;

Archetype é um framework para o desenvolvimento automatizado de produtos (conteúdos) no Plone;

ArchGenXML é uma ferramenta de geração de código que permite desenvolver novos tipos de conteúdo por meio de um modelo UML.

O Plone é hoje a mais recomendada opção para criação, publicação e edição de portais web. Ele pode ser usado como um servidor para intranets ou extranets, um sistema publicador de conteúdos e uma ferramenta de colaboração interativa. O Plone roda em diversos sistemas operacionais (como Linux, Windows, Solaris, FreeBSD e Mac OS X).

O uso do Plone está associado a vários casos de sucesso no Brasil e no mundo. No Brasil, pode-se citar o projeto Interlegis e o Serpro. O propósito do Interlegis é integrar as casas do poder legislativo e Serpro é uma empresa especializada em fornecer serviços de TI para organizações públicas federais.

Capítulo 26

Web Services

26.1 O que é Web Services?

Web Service é um componente de software identificado por uma URI que independe de implementação ou de plataforma e pode ser descrito, publicado e invocado sobre uma rede por meio de mensagens padrão XML. As mensagens XML são transportadas usando protocolos padrões da Internet. Com web service é possível realizar a integração entre sistemas desenvolvidos em diferentes linguagens e plataforma, e disponibilizar serviços interativos na Web. É uma tecnologia de padrão aberto e padronizada pelo W3C.

A arquitetura do Web Service é constituída por três componentes básicos: o servidor de registro (broker server ou service registry), o provedor de serviços (service provider) e o solicitante de serviços (service requestor). As interações entre esses componentes são de busca, publicação e interação de operações.

Na operação de publicação o provedor publica a descrição do serviço de tal forma que um solicitante possa localizá-la. Na operação de busca o solicitante obtém a descrição do serviço diretamente ou consulta o servidor de registro procurando pelo tipo de serviço desejado. Essa operação pode ser executada em duas fases distintas: desenvolvimento ou execução. Na operação de interação o solicitante chama ou inicia uma interação com o provedor, em tempo de execução, utilizando os detalhes contidos na descrição do serviço para localizar, contactar e chamar o serviço. A figura 26.1 mostra os componentes e a interação entre eles.

O provedor de serviços representa a plataforma que hospeda o web service permitindo que os clientes accessem o serviço. O provedor de serviços fornece o serviço e é responsável por publicar a descrição do serviço que provê. O solicitante de serviços é a aplicação que está procurando, invocando uma interação com o web service, ou seja, requisita a execução de um serviço. O solicitante de serviço pode ser uma pessoa acessando por meio do browser ou uma aplicação realizando uma invocação aos métodos descritos na interface do web service. O servidor de registro é um repositório central que contém a descrição (informação) de um web service, e é por meio do servidor de registro que essas descrições são publicadas e disponibilizadas para localização.

Os clientes buscam por serviços no servidor de registro e recuperam informações referentes à interface de comunicação para os web service durante

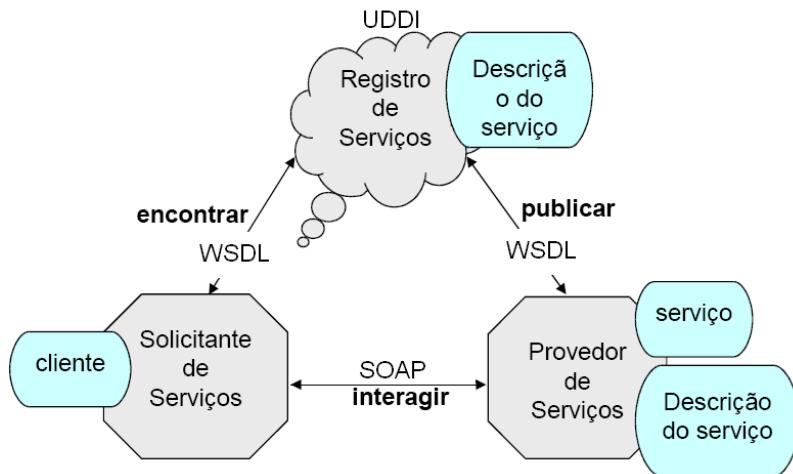


Figura 26.1: Componentes básicos da arquitetura do Web Service

a fase de desenvolvimento ou durante a execução do cliente, denominadas interação estática (static bind) e interação dinâmica (dynamic bind), respectivamente. Na interação estática, o cliente recupera a assinatura do serviço, necessária à codificação. Na interação dinâmica, o cliente recupera os valores de parâmetros e a localização do serviço.

O ciclo de vida de um web service compreende quatro estados distintos, figura 26.2:

Publicação processo, opcional, por meio do qual o fornecedor do web services dá a conhecer a existência do seu serviço, efetuando o registro do mesmo no repositório do web service;

Descoberta processo, opcional, por meio do qual uma aplicação cliente toma conhecimento da existência do web services pretendido pesquisando num repositório UDDI;

Descrição processo pelo qual o web service expõe a sua API (documento WSDL). Desta maneira a aplicação cliente tem acesso a toda a interface do web service, onde encontram descritas todas as funcionalidades por ele disponibilizadas;

Invocação (Mensagens) processo pelo qual o cliente e o servidor interagem, por meio do envio de mensagens;

A conjugação desses quatro estados permite constituir o ciclo de vida de um web service:

O fornecedor constrói o serviço utilizando a linguagem de programação que entender;

- De seguida, especifica a interface/assinatura do serviço que definiu em WSDL;

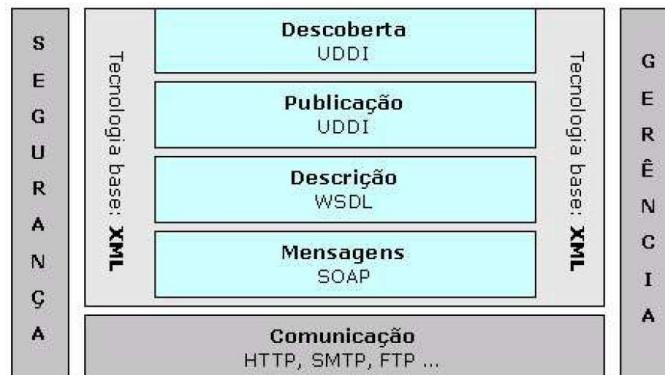


Figura 26.2: Ciclo de vida do web service

- Após a conclusão dos dois primeiros passos, o fornecedor registra o serviço no UDDI;
- O utilizador (aplicação cliente) pesquisa num repositório UDDI e encontra o serviço;
- A aplicação cliente estabelece a ligação com o web service e estabelece um diálogo com este, via mensagens SOAP.

A interação entre os web services se dá sob vários protocolos abertos, em diferentes níveis de abstração. Os protocolos utilizados para realizar a comunicação são o: UDDI (Universal Description Discovery and Integration), WSDL (Web Services Description Language), XML, SOAP (Simple Object Access Protocol) e o HTTP, conforme figura 26.3.

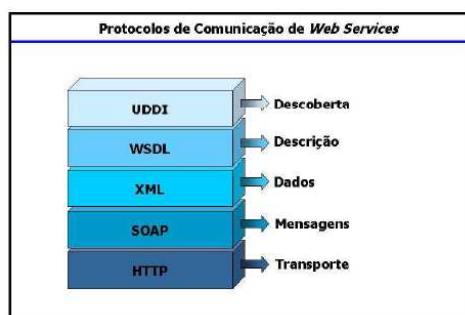


Figura 26.3: Protocolos de comunicação de Web services

As mensagens trocadas são formatadas no protocolo SOAP, o que permite a interoperabilidade entre diferentes plataformas, em um processo denominado serialização XML. Porém, antes que as mensagens SOAP sejam trocadas, suas características são explicitadas por meio de documentos WSDL, que descrevem quais dados estarão sendo trocados, e como estes dados estarão organizados nas mensagens SOAP. Adicionalmente, os serviços dos web services podem ser publicados através de UDDI, que é um formato utilizado para seu armazenamento

em repositórios disponíveis na Internet. Assim, se um desenvolvedor precisar resolver uma determinada tarefa, pode encontrar o web service que mais se adequar à sua necessidade.

Como os firewalls convencionais e proxies não bloqueiam a porta utilizada pelo protocolo HTTP, não existem grandes restrições para o uso deste tipo de aplicação em redes de longa distância.

Pode-se definir, resumidamente, o web service como sendo um serviço de software disponibilizado na Internet, descrito com um arquivo WSDL, registrado via UDDI, acessado utilizando SOAP e com dados representados em XML sendo transmitidos via HTTP.

A disseminação no uso de web services nos últimos anos incentivou o mercado a oferecer uma grande variedade de ferramentas e aplicações para prover suporte a essa tecnologia. Atualmente, as principais plataformas para web services são: Sun Microsystems, IBM, BEA, Apache, Systinet e Microsoft.

26.2 SOAP

O SOAP (Simple Object Access Protocol) é um protocolo leve para troca de informações em um ambiente descentralizado e distribuído que permite comunicação entre aplicações de forma simples e completamente independente de sistema operacional, linguagem de programação ou plataforma. É o protocolo de comunicação para os Web Services.

A comunicação é realizada por meio de trocas de mensagens, transmitidas em formato XML, incluindo os parâmetros usados nas chamadas, bem como os dados de resultados. Também pode ser utilizado para invocar, publicar e localizar web services no registro UDDI.

Parte da sua especificação é composta por um conjunto de regras de como utilizar o XML para representar os dados. Outra parte define o formato de mensagens, convenções para representar as chamadas de procedimento remoto (RPC – Remote Procedure Calls) utilizando o SOAP, e associações ao protocolo HTTP. O protocolo HTTP é o único protocolo padronizado pelo SOAP, mas existem algumas implementações que suportam outros protocolos como SMTP, TCP/IP, FTP e etc.

Uma mensagem SOAP nada mais é do que um fragmento XML bem formado, encapsulado por um par de elementos SOAP. A mensagem SOAP consiste dos seguintes elementos, figura 26.4:

Envelope toda mensagem SOAP deve contê-lo. É o elemento raiz do documento XML. Define o início e o fim das mensagens;

Header é um cabeçalho opcional. Ele carrega informações adicionais, como exemplo, se a mensagem deve ser processada por um nó intermediário. Quando utilizado, o Header deve ser o primeiro elemento do envelope;

Body é obrigatório e contém o payload, os dados em XML a serem transportados. O elemento Body possui um campo opcional Fault, usado para carregar mensagens de status e erros retornadas pelos "nós" ao processarem a mensagem;

RPCs ou chamadas remotas de procedimento são chamadas locais a métodos de objetos (ou serviços) remotos. Portanto, pode-se acessar os serviços de um

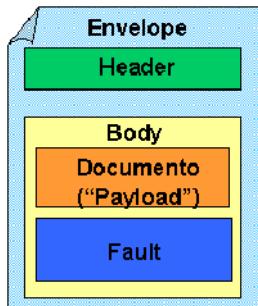


Figura 26.4: Estrutura da mensagem SOAP

objeto localizado em outro ponto da rede, através de uma chamada local a este objeto. Cada chamada ou requisição exige uma resposta.

O processo de uma chamada RPC: Antes de serem enviadas pela rede, as chamadas de RPC (emitidas pela aplicação cliente) são encapsuladas (ou serializadas) segundo o padrão SOAP. O serviço remoto, ao receber a mensagem faz o processo contrário, desencapsulando-a e extraíndo as chamadas de método. A aplicação servidora então processa esta chamada, e envia uma resposta ao cliente. O processo então se repete: a resposta é também serializada e enviada pela rede. Na máquina cliente, esta resposta é desencapsulada e é repassada para a aplicação cliente.

A especificação SOAP define as seguintes informações, como necessárias em toda chamada de RPC:

- A URI do objeto alvo;
- O nome do método;
- Os parâmetros do método (requisição ou resposta);
- Uma assinatura do método opcional;
- Um cabeçalho (header) opcional.

26.3 WSDL

O WSDL (Web Services Description Language) é uma linguagem baseada em XML, utilizada para descrever um web service. Um web service deve definir todas as suas interfaces, operações, esquemas de codificação, o conteúdo das mensagens, onde o serviço está disponível e quais os protocolos de comunicação são usados para conversar com o serviço e entre outros neste documento. Um documento WSDL define um XML Schema para descrever um web service.

A descrição de um serviço consiste de duas partes, figura 26.5: definição de implementação do serviço e definição da interface do serviço. A separação entre as duas camadas permite que elas sejam utilizadas separadamente.

A parte de definição de interface do serviço descreve o web service, incluindo métodos que são invocados e parâmetros que são enviados. A parte de definição



Figura 26.5: Camada de descrição de serviços da WSDL

de implementação de serviços descreve como uma interface de serviço é implementada por um provedor: onde o serviço está instalado e como pode ser acessado. As descrições dos elementos das duas partes:

Binding descreve protocolos, formato de data, segurança e outros atributos para uma interface (portType) em particular;

Porttype informa elementos de operações do web service;

Message define entrada e saída de dados referentes a operações;

Type define tipos de dados complexos em uma mensagem;

Service contém uma coleção de elementos port com elementos binding;

Port é a combinação de um binding com endereço de rede, fornecendo o endereço alvo para a comunicação dos serviços.

Tão logo o cliente tenha acesso à descrição do serviço a ser utilizado, a implementação do Web Service pode ser feita em qualquer linguagem de programação. Normalmente são utilizadas linguagens construídas para interação com a Web, como exemplo Java Servlets ou ASP, que, em seguida, chamam um outro programa ou objeto.

Basicamente, quando o cliente deseja enviar uma mensagem para um determinado web service, ele obtém a descrição do serviço (através da localização do respectivo documento WSDL), e em seguida constrói a mensagem, passando os tipos de dados corretos (parâmetros, etc) de acordo com a definição encontrada no documento. As duas partes envolvidas em uma interação web service precisam ter acesso à mesma descrição WSDL para conseguirem entender uma à outra. Em seguida, a mensagem é enviada para o endereço onde o serviço está localizado, a fim de que possa ser processada. O web service, quando recebe esta mensagem valida-a conforme as informações contidas no documento WSDL. A partir de então, o serviço remoto sabe como tratar a mensagem, sabe como processá-la (possivelmente enviando-a para outro programa) e como montar a resposta ao cliente.

26.4 UDDI

Para que um serviço seja utilizado é necessário que o cliente consiga localizá-lo, e essa localização pode ser feita por meio do UDDI (Universal Description, Discovery and Integration), que é uma especificação técnica para descrever, descobrir e publicar web services.

Uma implementação de UDDI corresponde a um registro web service, que provê um mecanismo para busca e publicação de web services. Um registro UDDI contém informações categorizadas sobre os serviços e as funcionalidades que eles oferecem, e permite a associação desses serviços com suas informações técnicas, geralmente definidas usando WSDL.

O UDDI possui um componente central chamado UDDI Project, que manipula um registro global e público chamado business registry. A informação oferecida pelo business registry consiste de três componentes: white pages, yellow pages e green pages.

A informação fornecida por um registro UDDI pode ser comparada à uma lista telefônica. As páginas brancas (white pages), fornecem informações tais como nome da organização, contato e identificadores. As páginas amarelas (yellow pages) são compostas por um índice de serviços e produtos e as páginas verdes (green pages) contém informações a respeito de transações, descrições de serviço e invocação de aplicações.

As informações contidas em arquivos de descrição de serviço (WSDL) completam aquelas que estão no registro. No entanto, UDDI não fornece suporte a vários tipos de descrição de serviço, mas não suporta a criação de descrições WSDL de forma direta.

Uma descrição WSDL completa consiste da combinação dos documentos de interface e de implementação de serviço. A primeira é publicada no registro UDDI como businessservice e a segunda como tmodel.

26.5 Segurança

A segurança no envio de mensagens SOAP é um tópico importante. Em um nível mais baixo, mensagens SOAP podem ser trocadas pela rede utilizando HTTPS ao invés de HTTP. Como HTTPS utiliza SSL no seu transporte, fica garantida a proteção contra possíveis intervenções. Além disso, o cliente e servidor podem verificar cada um suas respectivas identidades.

Embora HTTPS resolva o problema de proteção das mensagens contra possíveis invasores, este não ajuda muito quando se necessita da segurança necessária para a autenticação de usuários de web services específicos. Estes serviços irão fornecer algum tipo de combinação de usuário/senha durante a fase inicial de registro no serviço e então esta será utilizada para acessos futuros. Não há ainda um padrão de autenticação para Web services, mas pode utilizar os firewalls, VPNs, NTFS, produtos de single-in para oferecer a autenticação e autorização de usuários.

Parte VI

Redes de Comunicação

Capítulo 27

Técnicas Básicas de Comunicação

27.1 Base Teórica da Comunicação de Dados

As informações podem ser transmitidas por fios fazendo-se variar alguma propriedade física, como voltagem ou corrente. Representando o valor dessa voltagem ou corrente como uma função de tempo com uma valor único, $f(t)$, pode-se criar um modelo para o comportamento do sinal e analisá-lo matematicamente.

Tal análise é conhecida como Análise de Fourier e ela afirma que qualquer função periódica razoavelmente estável, $g(t)$, com o período T pode ser construída como a soma de um número (possivelmente infinito) de senos e co-senos. A decomposição dos harmônicos que compõe que compõe a função é a chamada série de Fourier.

Nenhum recurso de transmissão é capaz de transmitir sinais sem perder parte da energia no processo. Se todos os coeficientes da série de Fourier fossem igualmente reduzidos, o sinal resultante seria reduzido em amplitude, mas não seria distorcido. Infelizmente, todos os meios de transmissão reduzem diferentes componentes de Fourier por diferentes valores e, em consequência disso, introduzem distorções. Em geral, as amplitudes são transmitidas sem redução, de 0 a alguma freqüência f_c , com todas as freqüências acima dessa freqüência de corte sendo atenuadas. A faixa de freqüências transmitidas sem serem fortemente atenuadas denomina-se largura de banda. Na prática, o corte não é nítido; assim, muitas vezes a largura de banda varia desde 0 até a freqüência em que a metade da potência é transmitida.

A largura de banda é uma propriedade física do meio de transmissão, e em geral depende da construção, da espessura e do comprimento do meio. Em alguns casos um filtro é introduzido no circuito para limitar o volume de largura de banda disponível para cada cliente. Por exemplo, uma linha telefônica pode ter uma largura de banda de 1MHz para curtas distâncias, mas as empresas de telefonia limitam essa faixa a cerca de 3100 Hz.

27.2 Taxa Máxima de Dados em um Canal

Em 1924, Henry Nyquist percebeu que até mesmo um canal perfeito tem uma capacidade de transmissão finita. Ele derivou uma equação expressando a taxa máxima de dados de um canal sem ruído com largura de banda finita. Em 1948, Claude Shannon aprofundou o trabalho e o estendeu ao caso de uma canal com ruído aleatório (isto é, termodinâmico).

Nyquist provou que, se um sinal arbitrário atravessar um filtro com baixa freqüência de corte H , o sinal filtrado poderá ser completamente reconstruído a partir de apenas $2H$ amostras (exatas) por segundo. Fazer uma amostragem da linha com uma rapidez maior que $2H/s$ seria inútil, pois os componentes de freqüência mais alta que essa amostragem poderia recuperar já teriam sido filtrados. Se o sinal consistir em V níveis discretos o teorema de Nyquist afirma que:

$$\text{Maximum Data Rate} = 2H\log_2(1 + \frac{S}{N}) \quad (27.1)$$

Se houver ruído aleatório, a situação irá se deteriorar mais rapidamente. Além disso, sempre existe ruído aleatório (térmico) presente, devido ao movimento das moléculas no sistema. O volume de ruído térmico presente é medido pela relação entre a potência do sinal e a potência do ruído, chamada relação sinal/ruído. Dessa forma, o principal resultado de Shannon é que a taxa máxima de dados de um canal com ruídos cuja largura de banda é H Hz, e cuja relação sinal/ruído é S/N , é dada por:

$$\text{Maximum Number of Bits/sec} = H\log_2(1 + \frac{S}{N}) \quad (27.2)$$

Por exemplo, um canal com largura de banda de 3000 Hz com uma relação sinal/ruído de 30 db nunca pode transmitir muito mais que 30000 bps, independente da quantidade de níveis de sinal utilizados e da freqüência com que as amostras são obtidas.

27.3 Sinais Digitais Binários

Os sinais digitais binários podem ser estudados como se fossem um sinal quadrado. Através da "Série de Fourier", é possível reconstruir um sinal quadrado a partir da correspondente senóide fundamental. Ao incluirmos as freqüências harmônicas de ordem ímpar, o sinal reconstruído aproxima-se cada vez mais do sinal quadrado original. A figura 27.1 ilustra a reconstituição de um sinal quadrado a partir da senóide fundamental e suas harmônicas ímpares até a quinta ordem.

Assim sendo, na transmissão digital entre dois pontos, deveríamos ter um meio de transmissão com largura de faixa (banda passante) infinita, para que o sinal digital transmitido fosse recebido sem nenhuma distorção. Entretanto, fisicamente isso não é possível. Poderia-se pensar, então, em aumentar a largura de faixa do canal ao máximo para atenuar as distorções, no entanto isso não é viável economicamente.

A solução é adaptar o sinal digital aos tipos de degradação inerentes aos meios de transmissão. Para isto, foram desenvolvidos dispositivos capazes de transformar o sinal digital do computador em uma forma possível de ser transmitida pelo meio sem que ocorram danos graves. Estes dispositivos são chamados

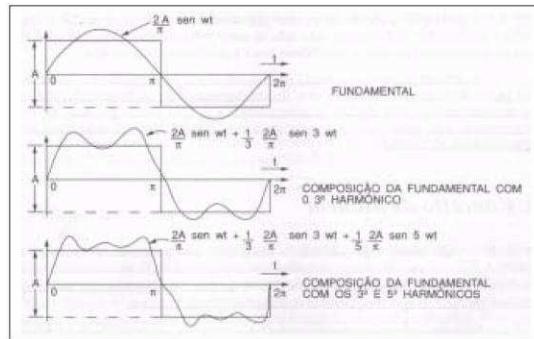


Figura 27.1: Reconstituição de um sinal binário

de "Modems" (Modulator/Demodulator). Este equipamento executa uma transformação por modulação (modem analógico) ou por codificação (modem digital) dos sinais digitais gerados pelo computador. A rigor, os modems digitais não deveriam receber esse nome, pois eles não executam a modulação/demodulação do sinal digital, apenas a codificação. Em complemento, os modems digitais também recebem os nomes modem banda-base e modem data set. Mesmo usando o modem digital, devido às características da onda quadrada, a transmissão só poderá ocorrer a curtas distâncias e com linhas de boa qualidade.

27.4 Transmissão em Banda Base

Banda base é freqüentemente utilizada para a transmissão digital de dados, um único canal utiliza a largura de banda total disponível. Assim, a transmissão de um sinal em banda base consiste em enviar o sinal de forma digital através da linha (forma de onda digital), ou seja, enviar os bits conforme a necessidade, de acordo com um padrão digital. Desde que a distância entre o transmissor e receptor seja de alguns quilômetros, a banda de transmissão disponível seja em torno de 15 KHz e o meio de transmissão tenha certas características, é possível realizar a Codificação Banda Base do sinal digital. Nesse sentido, a distância alcançada na transmissão diminui conforme aumenta a velocidade transmissão. O alcance máximo é definido como sendo a distância máxima em que é possível transmitir mantendo a taxa de erro abaixo de um valor predeterminado.

27.5 Classificação dos Sinais

Os sinais em Banda Base podem ser classificados quanto a duração e polaridade de seus pulsos:

- Duração:
 - o NRZ (non return to zero): cada bit 0 é representado por um pulso OFF e cada bit 1 por um pulso ON ocupando todo o intervalo significativo do bit;

- RZ (return to zero): os bits 1 são representados por pulsos ON com duração de meio intervalo significativo bit.
- Polaridade:
 - Unipolar: os dois níveis têm a mesma polaridade (exemplo: 0 e "+"). Esse tipo de sinal resulta em codificação com componente DC que não leva informação, mas consome energia. Além disso, a ocorrência de uma longa seqüência de bits 0 resulta em um sinal que não apresenta transições, dificultando a sincronização dos equipamentos.
 - Polar: este sinal possui pulsos com polaridades opostas (exemplo: o bit 0 é representado por pulso "-" e o bit 1 por pulso "+"), zerando a componente DC se a mensagem contiver uma proporção igual de bits 0 e 1. O número de transições dependerá completamente do sinal transmitido.
 - Bipolar: os sucessivos bits 1 são representados com pulsos de polaridade alternada.

27.6 Técnicas de Codificação de Linha

As diversas técnicas de codificação do sinal digital procuram gerar o sinal codificado com muitas transições, a fim de facilitar a recuperação do sincronismo no modem receptor (recuperação do relógio). Além disso, procura-se concentrar o espectro de transmissão do sinal codificado dentro de uma faixa de freqüência com pouco componente DC. Diversas técnicas de codificação são ilustradas na figura 27.2 e descritas adiante.

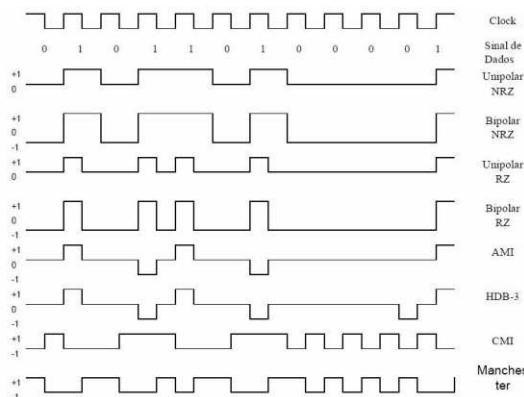


Figura 27.2: Exemplos de ondas codificadas

27.6.1 Codificação NRZ

Com o código NRZ, o nível do sinal é mantido constante em uma das tensões possíveis, pela duração de um intervalo de bit. Se as duas voltagens

permitidas são 0 e V, a forma de onda NRZ é dita UNIPOLAR. Este sinal tem uma componente DC diferente de zero. Por outro lado, o sinal NRZ BIPOLAR usa duas polaridades, +V e -V, deste modo provê uma componente DC nula.

A codificação NRZ apresenta carência de transições de dados, o que resulta em um pobre desempenho na recuperação de relógio no receptor. A recuperação do relógio nesse contexto significa recuperar o relógio usado no transmissor para temporizar a codificação do sinal e assim permitir a recuperação da informação no lado receptor. Esta característica limita o seu uso apenas para pequenas distâncias de transmissão e conexões entre estações.

27.6.2 Codificação RZ

O nível do sinal que representa o bit de valor 1 dura a primeira metade do intervalo do bit, após o qual o sinal retorna para o nível de referência (zero) para o restante meio intervalo de bit. Um bit 0 é indicado por uma não mudança, com o sinal continuando no nível de referência (zero).

Sua principal vantagem reside no aumento das transições e comparação com o NRZ, com uma resultante melhoria na recuperação do relógio no receptor. Nota-se que uma seqüência muito grande de 0 resulta em um sinal sem transições, o que representa um problema para os circuitos de recuperação de relógio.

27.6.3 Codificação AMI (Alternate Mark Inversion)

Na codificação AMI (bipolar), o bit 0 é sempre codificado como nível zero, e os bits 1 são codificados como +V ou -V, onde a polaridade é alternada para cada ocorrência de bit 1. A codificação AMI resulta em uma componente DC nula. A representação AMI pode ser NRZ (100 % do tempo de bit) ou RZ (50% do tempo de bit).

A garantia de transição dos níveis para cada bit 1, proporciona um ótimo desempenho na recuperação de relógio, melhorando ainda mais quando o sinal for RZ. Esta codificação apresenta ainda a capacidade de detecção de erro, pois amplitudes positivas consecutivas sem uma amplitude negativa intermediária (e vice-versa) constituem uma violação da regra AMI e indicam que ocorreu um erro na transmissão. Entretanto, quando ocorrer uma seqüência longa de zeros, o sinal codificado fica muito tempo sem transições na linha, o que dificulta a obtenção do relógio de sincronismo.

27.6.4 Codificação HDB-3 (High Density Bipolar with 3 Zero Maximum Tolerance)

Nesse tipo de codificação, o sinal digital a ser transmitido é analisado e, cada vez que é detectada uma seqüência de quatro zeros consecutivos, esta seqüência é substituída por uma outra seqüência padronizada. Para isso, é utilizado o recurso da "violação", que consiste no uso de um pulso que tenha a mesma polaridade que o pulso anterior (o que viola o princípio básico do AMI).

No HDB-3, os quatro zeros consecutivos são substituídos pela seqüência 000V ou V00V, onde "V" é a violação, e a substituição dependerá do último pulso transmitido, observando sempre o princípio da alternância de pulsos. Caso o último pulso transmitido não seja uma violação e tenha polaridade oposta à

polaridade da violação anterior, transmitirá 000V. No caso em que o último pulso transmitido seja uma violação ou tenha polaridade idêntica à polaridade da violação anterior, transmitirá V00V. Essa escolha entre 000V e V00V faz com que violações sucessivas sejam de polaridades opostas, o que garante a não existência de sinal DC na linha.

Na recepção, o decodificador tem de verificar, inicialmente, a violação AMI e, posteriormente, o número de zeros que precede esta violação, para determinar se o último pulso transmitido é também uma violação. Isto é feito da seguinte forma: se na recepção houver dois pulsos, com mesma polaridade, separados por três zeros, o segundo pulso é violação, logo, é eliminado. Se na recepção houver dois pulsos, com mesma polaridade, separados por dois zeros, ambos os pulsos são violação, logo, ambos são eliminados. Nota-se que na codificação HDB-3 são contornados os problemas do aparecimento do nível DC e da falta de transições para recuperação do sinal de relógio.

27.6.5 Codificação Manchester

Na codificação Manchester, cada período de bits é dividido em dois intervalos iguais. Um bit 1 binário é enviado quando a voltagem é definida como alta durante o primeiro intervalo, e como baixa no segundo intervalo. Um bit 0 binário é exatamente o oposto: primeiro baixo depois alto. Esse esquema garante que cada período de bit terá uma transição na parte intermediária, tornando fácil para o receptor sincronizar-se com o transmissor. Uma desvantagem da codificação Manchester é que ela exige duas vezes mais largura de banda que a codificação binária direta, pois os pulsos são a metade da largura. Por exemplo, para transmitir dados a 10 Mbps, o sinal precisa mudar 20 milhões de vezes por segundo.

A codificação Manchester diferencial é uma variação da codificação Manchester apresentada. Nela um bit 1 é indicado pela ausência de uma transição no início do intervalo. Um bit 0 é indicado pela presença de uma transição no início do intervalo. Em ambos os casos também existe uma transição na parte intermediária. O esquema diferencial exige equipamento mais complexo, mas oferece maior imunidade a ruídos. Todos os sistemas Ethernet usam a codificação Manchester devido a sua simplicidade.

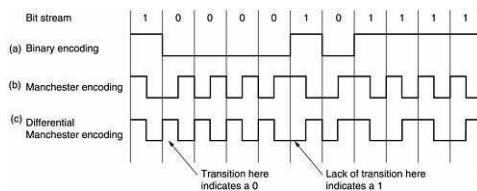


Figura 27.3: Codificações Binária, manchester e diferencial

27.7 Modulação

Modulação é o processo pelo qual alguma característica de uma onda portadora é variada de acordo com a mensagem (sinal modulante), produzindo um sinal

modulado cujas propriedades são mais compatíveis com as características do canal. Essa descrição é resumida na figura 27.4.

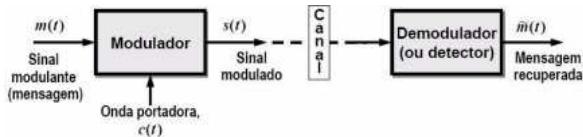


Figura 27.4: Processo de Modulação

A figura 27.5 resume a classificação dos tipos de modulação.

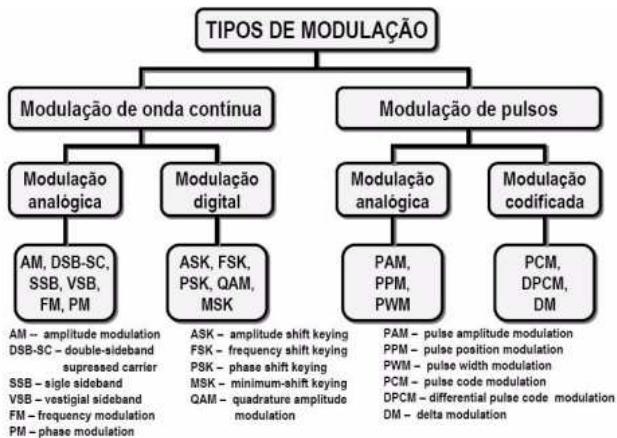


Figura 27.5: Tipos de Modulação

27.7.1 Modulação de Onda Contínua

As linhas de transmissão enfrentam três problemas principais: atenuação, distorção de retardamento e ruído. Devido a esses problemas e principalmente ao fato de a atenuação e a velocidade de propagação variarem em função da freqüência, não é interessante ter uma grande variedade de freqüências no sinal. Infelizmente, as ondas quadradas usadas em sinais digitais têm um amplo espectro de freqüências e, portanto, estão sujeitas a uma forte atenuação e a uma distorção de retardamento. Esses efeitos tornam a sinalização em banda base (DC) inadequada, exceto em baixas velocidades e em distâncias curtas.

Para contornar os problemas associados à sinalização DC é usada a sinalização AC. É introduzido no sinal um tom contínuo na faixa de 1000 a 2000 Hz, denominado onda portadora senoidal. Sua amplitude, freqüência ou fase pode ser modulada para transmitir informações.

Basicamente, existem dois tipos de modulação de onda contínua, a analógica e a digital. Pode-se citar como exemplo de modulação analógica a modulação AM (Amplitude Modulation) onde a variação da amplitude da portadora ocorre de acordo com a variação do sinal modulante. Este tipo de modulação tem algumas derivações, tais como, AM-DSB (Double-Sideband Amplitude Modulation), AM-DSB-SC (Double-Sideband Suprisse Carrier Amplitude Modulation)

e AM-VSB (Vestigial-Sideband Amplitude Modulation). A modulação FM (Frequency Modulation) tem melhor qualidade que a AM por ser menos suscetível a ruídos. Neste tipo de modulação ocorre a variação de freqüência da portadora em função da variação do sinal modulante. A Modulação PM (Phase Modulation) tem como principal característica a variação da fase da portadora em função da variação do sinal modulante. A figura 27.6 mostra alguns exemplos de modulação analógica.

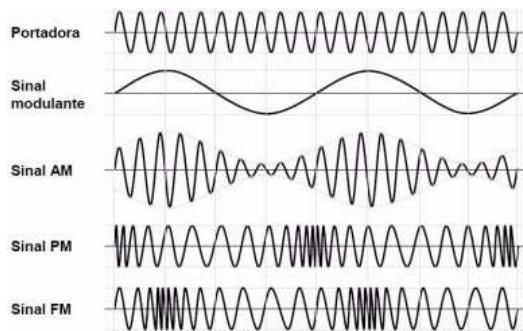


Figura 27.6: Modulação Analógica

A modulação digital consiste em aplicar um sinal digital (binário) como sinal modulante da portadora ao invés de um sinal analógico como ocorre na modulação analógica descrita anteriormente. As modulações digitais mais usadas são ASK (Amplitude Shift Keying), FSK (Frequency Shift Keying), PSK (Phase Shift Keying). Essas técnicas de modulação são representadas na figura 27.7.

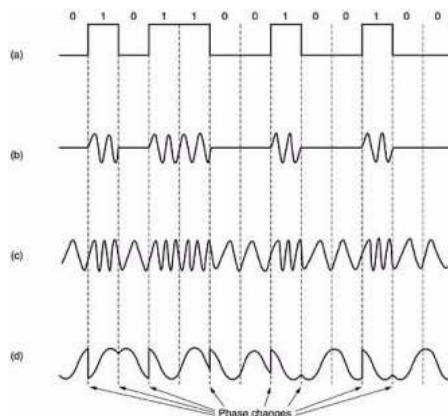


Figura 27.7: Modulação Digital: Amplitude, FrequênciA e Fase

Para atingir velocidades cada vez mais altas não basta apenas aumentar a taxa de amostragem. O teorema de Nyquist afirma que mesmo com uma linha de 3000Hz perfeita não há razão para uma amostragem mais rápida que 6000Hz. Na prática, a maioria dos modens realizam amostragens 2400 vezes/s e se concentra em obter mais bits por amostra.

O número de amostras por segundo é medido em baud. Durante cada baud, é enviado um símbolo. Desse modo, uma linha de n bauds transmite n símbolos/s. Se o símbolo consiste em 0 volts para representar um valor lógico 0 e 1 volt para indicar um valor lógico 1, a taxa de bits é 2400 bps. Porém se as voltagens 0, 1, 2, 3 volts são usadas, cada símbolo consiste em dois bits, e assim uma linha de 2400 bauds pode transmitir 2400 símbolos/s a uma taxa de dados de 4800 bps. De modo semelhante, com quatro deslocamentos de fase possíveis, também existem 2bits/símbolo, e portanto mais uma vez a taxa de bits é o dobro da taxa em bauds. Essa última técnica é amplamente usada e se denomina QPSK (Quadrature Phase Shift Keying). Desse modo, a técnica de modulação determina o número de bits/símbolo.

Todos os modems avançados utilizam uma combinação de técnicas de modulação para transmitir vários bits por baud. Com freqüência, várias amplitudes e vários deslocamentos de fase são combinados para transmitir diversos bits/símbolo. A figura 27.8 (b) abaixo apresenta uma estrutura de modulação na qual são usadas quatro amplitudes e quatro fases, dando um total de 16 combinações diferentes. Esse esquema de modulação pode ser usado para transmitir 4 bits por símbolo. Ele é chamado QAM-16 (Quadrature Amplitude Modulation). A figura 27.8 (c) permite 64 combinações diferentes. Diagramas como os apresentados na figura 27.8 são denominados de diagramas de constelação.

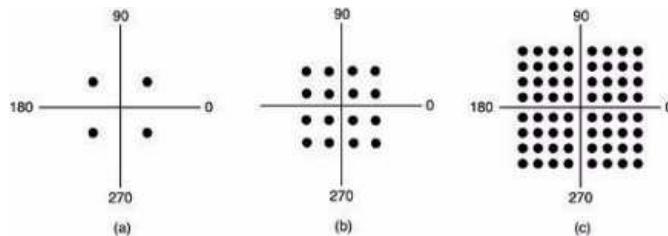


Figura 27.8: QPSK, QAM 16 e QAM 64

Com muitos pontos no padrão de constelação, até mesmo uma pequena quantidade de ruído na amplitude ou fase detectada pode resultar em um erro e, potencialmente, em muitos bits incorretos. Para reduzir a chance de um erro, é efetuada a correção de erros adicionando bits extras a cada amostra. Os esquemas são conhecidos como TCM (Trellis Coded Modulation).

27.7.2 Modulação de Pulso

A modulação por pulsos iniciou a partir da teoria da amostragem, a qual estabelece que a informação contida em qualquer sinal analógico pode ser recuperada a partir de amostras do sinal tomadas a intervalos regulares de tempo.

A modulação por pulsos pode ser analógica ou codificada. No caso analógico, os valores das amostras do sinal são transferidos para as amplitudes, durações ou posições de pulsos de formato fixo conhecido. No caso digital, os valores das amostras são convertidos para números binários que por sua vez são codificados em seqüências de pulsos que representam cada um dos valores binários. Um exemplo desse tipo de modulação é o PCM.

Modulação de Código de Pulso

O processo de transformação das redes telefônicas em redes totalmente digitais tanto no que diz respeito à comutação como à transmissão teve início quando da introdução, em escala comercial, dos sistemas de transmissão PCM. O PCM (Modulação por Código de Pulso) transforma um sinal analógico em uma série de pulsos binários que podem ser manipulados. Sendo assim, o objetivo da modulação PCM é fazer com que um sinal analógico possa ser transmitido através de um meio físico com transmissão digital.

O processo de modulação PCM consiste basicamente em três etapas, nessa ordem: amostragem, quantização e codificação. A figura 27.9 ilustra o processo de modulação PCM, passo a passo. Uma etapa adicional chamada compressão pode ser inserida após a etapa de quantização para manipular a saída desse processo e transformá-lo em uma quantização não-linear, melhorando assim a robustez a problemas de ruído no canal.

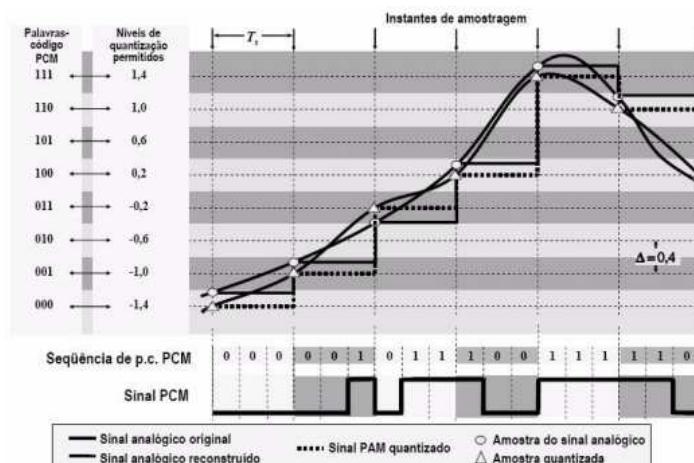


Figura 27.9: Pulse Code Modulation

O sinal PCM resultante é enviado através do canal por transmissão em banda base. Esse sinal ainda pode usar alguma das técnicas de codificação de linha apresentadas anteriormente, como, por exemplo, a AMI que, inclusive, foi desenvolvida originalmente para esse fim. Cabe ressaltar que o processo de codificação do PCM tem a função de transformar os valores quantificados em uma seqüência de bits. Essa seqüência de bits constitui o sinal PCM que pode ser transmitido pelo canal. Entretanto, o sinal PCM pode ser codificado novamente, mas agora usando uma técnica de codificação de linha para então ser transmitido pelo canal.

27.8 Técnicas de Multiplexação

Multiplexação é a transmissão simultânea de vários sinais por um único canal. Desse modo, várias fontes com esquemas de modulação individuais podem ser multiplexados em um mesmo canal. O objetivo básico para a utilização desta

técnica é economia, pois utilizando o mesmo meio de transmissão para diversos canais economiza-se em linhas, suporte, manutenção, instalação, etc.

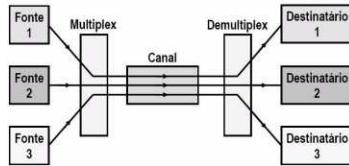


Figura 27.10: Multiplexação de 3 fontes

As principais técnicas de multiplexação são FDM, TDM e WDM. Cada uma delas serão discutidas a seguir.

27.8.1 FDM - Frequency Division Multiplexing

Com a técnica FDM (Frequency Division Multiplexing) a multiplexação é realizada dividindo todo o espectro de freqüência em diversos canais mais estreitos que são alocadas entre os terminais em uma relação um para um. Bandas de guarda são usadas entre subportadoras adjacentes para não ocorrer sobreposição, entretanto parte da capacidade de transmissão é desperdiçada para esse fim. Além disso, FDM é utilizada apenas em sistemas analógicos. A Figura 27.11 ilustra a técnica segundo os eixos de tempo e freqüência.

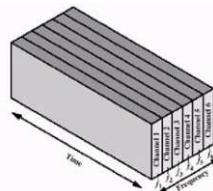


Figura 27.11: FDM - Frequency Division Multiplexing

27.8.2 TDM - Time Division Multiplexing

A técnica de multiplexação TDM (Time Division Multiplexing) é baseada no domínio do tempo e só pode ser empregado em sistemas digitais. Nesse esquema, a linha do tempo é dividida em quadros que por sua vez é dividida em slots. Dessa forma, o acesso ao meio é garantido através da alocação desses slots de tempo entre as estações que compõe o sistema. É mais eficiente em comparação ao sistema anterior, entretanto necessita acrescentar bits adicionais ao sinal original para sincronia e gerenciamento da rede. A Figura 27.12 ilustra o esquema.

27.8.3 OFDM

O método de multiplexação OFDM é uma técnica de transmissão multiportadora que surgiu no fim da década de sessenta como uma evolução de FDM.

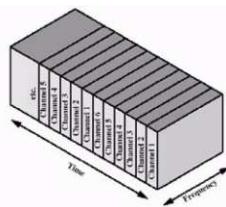


Figura 27.12: TDM - Time Division Multiplexing

O princípio básico em OFDM é eleger freqüências ortogonais entre si, dito de outra forma, deve-se garantir que nenhuma subportadora seja produto de combinação linear das demais presentes no canal. Isso permite prescindir da banda de guarda presente na técnica FDM. O benefício imediato obtido com OFDM é a economia de largura de banda. Essa técnica é particularmente útil nos casos de transmissões sem fio, pois além da eficiência espectral, a técnica OFDM se apresenta mais robusta em relação aos problemas inerentes ao canal sem fio.

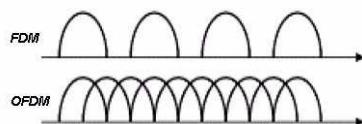


Figura 27.13: Comparação entre uso de banda FDM e OFDM

27.8.4 WDM -Wavelength Division Multiplexing

A técnica WDM (Wavelength Division Multiplexing) é uma variação da multiplexação por divisão de freqüência empregada em canais de fibra óptica. As informações são transportadas sobre faixas de comprimentos de onda por meio da tecnologia fotônica.

Nesse contexto, os sinais transmitidos são combinados em um multiplexador óptico para ser enviado através de um único cabo de fibra óptica. Dessa forma, a capacidade de transmissão é aumentada e a largura de banda da fibra é usada efetivamente. A figura 27.14 representa o esquema de multiplexação por divisão de comprimento de onda.

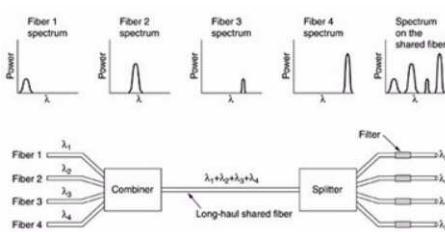


Figura 27.14: WDM -Wavelength Division Multiplexing

27.9 Protocolos de Acesso Múltiplo

Basicamente existem dois tipos de enlaces: ponto-a-ponto e broadcast. O primeiro caso consiste em um único remetente em uma extremidade do enlace e um único receptor na outra extremidade do enlace. O segundo tipo pode ter múltiplos nós remetentes e receptores, todos conectados ao mesmo, único compartilhado canal de transmissão. Nesse contexto surge uma questão: como coordenar o acesso dos múltiplos nós remetentes e receptores a um canal broadcast compartilhado. Essa coordenação é realizada pelos protocolos de acesso múltiplo. Formalmente, podem-se classificar esses protocolos em: protocolos de alocação fixa (também chamado protocolo de divisão do canal), protocolos de acesso aleatório e protocolo de alocação dinâmica (também chamado protocolo de revezamento).

Os protocolos de alocação fixa caracterizam-se por atribuir uma parte do canal para cada estação, de maneira fixa. O TDMA (Time-Division Multiple Access, baseado na técnica de multiplexação TDM) e o FDMA (Frequency-Division Multiple Access, baseado na técnica de multiplexação FDM) são exemplos deste tipo de protocolo. São extremamente fáceis de implementar. Porém, quando uma estação não tem pacotes para enviar durante o período de tempo em que o meio está alocado para ela, o canal ficará ocioso enquanto outros terminais poderiam estar utilizando-o.

Por outro lado, os protocolos de acesso aleatório não possuem um controle rígido para alocação do canal. O ALOHA e o CSMA (Carrier Sense Multiple Access) são exemplos desta categoria. Também são relativamente simples de implementar, porém, a possibilidade de ocorrer colisões quando duas ou mais estações tentam transmitir ao mesmo tempo provoca desperdício no canal de transmissão.

Nos protocolos de alocação dinâmica o canal é alocado de acordo com as necessidades das estações. Este controle pode ser implementado de várias maneiras, como por exemplo: na forma de polling (onde uma estação espera ser "questionada" se necessita acessar o canal), ou na forma de reservas explícitas, como no RPAC (Reservation-Priority Access Control). Com a utilização destes protocolos não existe a possibilidade de colisões e o canal é alocado sob demanda, evitando períodos de tempo ociosos no canal. Porém, há uma sobrecarga na rede devido aos sinais de controle transmitidos no meio. A tabela 27.1 resume as principais vantagens e desvantagens de cada tipo de protocolo.

	Canal Ocioso	Colisões	Sobrecarga
Alocação Fixa	sim	não	não
Acesso Aleatório	não	sim	não
Alocação Dinâmica	não	não	sim

Tabela 27.1: Alocação de Canais

Capítulo 28

Topologias de Redes

A topologia de uma rede de comunicação refere-se à forma como os enlaces físicos e os nós estão organizados, determinando os caminhos físicos existentes e utilizáveis entre quaisquer pares de estações conectadas a essa rede. A topologia de uma rede muitas vezes caracteriza o seu tipo, eficiência e velocidade. As topologias mais comuns são as seguintes:

Malha

A interconexão é total garantindo alta confiabilidade, porém a complexidade da implementação física e o custo inviabilizam seu uso comercial;

Estrela

A conexão é feita através de um nó central que exerce controle sobre a comunicação. Sua confiabilidade é limitada à confiabilidade do nó central, cujo mau funcionamento prejudica toda a rede;

Barramento

As estações são conectadas através de um cabo com difusão da informação para todos os nós. É necessária a adoção de um método de acesso para as estações em rede compartilharem o meio de comunicação, evitando colisões. É de fácil expansão, mas de baixa confiabilidade, pois qualquer problema no barramento impossibilita a comunicação em toda a rede;

Anel

O barramento toma a forma de um anel, com ligações unidirecionais ponto a ponto. A mensagem é repetida de estação para estação até retornar à estação de origem, sendo então retirada do anel. Como o sinal é recebido por um circuito e reproduzido por outro há a regeneração do sinal no meio de comunicação; entretanto há também a inserção de um atraso a cada estação. O tráfego passa por todas as estações do anel, sendo que somente a estação destino interpreta a mensagem;

Árvore

É a expansão da topologia em barra herdando suas capacidades e limitações. O barramento ganha ramificações que mantêm as características de difusão das mensagens e compartilhamento de meio entre as estações;

Mistas

Combinam duas ou mais topologias simples. Alguns exemplos são o de estrelas conectadas em anel e as árvores conectadas em barramento. Procuram explorar as melhores características das topologias envolvidas, procurando em geral realizar a conexão em um barramento único de módulos concentradores aos quais são ligadas as estações em configurações mais complexas e mais confiáveis.

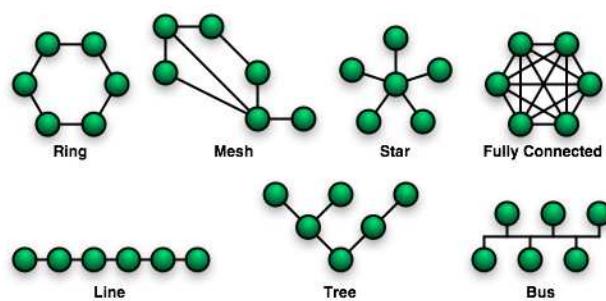


Figura 28.1: Topologias de Rede

Capítulo 29

Arquitetura de Redes

29.1 Organização em Camadas

Para reduzir a complexidade do projeto, a maioria das redes é organizada como uma pilha de camadas colocadas umas sobre as outras. O número de camadas e suas funções podem ser diferentes de uma rede para outra. No entanto, em todas as redes o objetivo principal da implementação em camadas é fazer com que uma camada ofereça *serviços* às camadas superiores, isolando a camada superior dos detalhes de implementação. A camada n de uma máquina se comunica com a camada n de outra máquina utilizando um conjunto de regras e convenções chamado *protocolo*. Essas entidades situadas em máquinas diferentes são chamadas de pares. Na verdade, a camada n de uma máquina não se comunica com a camada n da outra máquina diretamente. As informações são passadas para as camadas inferiores até que se alcance a camada mais baixa.

Abaixo desta está o meio físico através do qual a comunicação propriamente dita ocorre. Entre as camadas adjacentes existe uma *interface*. A interface define o conjunto de operações e serviços que a camada inferior pode oferecer a camada imediatamente superior. Em uma arquitetura em camadas deve-se estarem bem definidos quais são as funções de cada uma das camadas.

Este tipo de arquitetura proporciona a independência das camadas permitindo que a implementação de uma camada possa ser totalmente substituída tendo como único requisito da nova implementação oferecer no mínimo os mesmos serviços oferecidos pela implementação anterior. Uma arquitetura em camadas deve se preocupar com diversos aspectos como: Endereçamento, Modo de Transmissão(Simplex,Half-Duplex,Full-Suplex), Canais Lógicos(Dados,Controle), Controle de Erros, Controle de Fluxo, Fragmentação / Montagens, Multiplexação / Demultiplexação e Roteamento.

Capítulo 30

Protocolos de Rede

30.1 ARP - Address Resolution Protocol

O protocolo ARP é destinado a realizar a associação entre um endereço de camada 3 e um endereço de camada 2. O endereço de camada 3 é um endereço lógico e está relacionado com protocolos como IP, IPX, Appletalk,etc. O endereço mac é um endereço físico que está relacionado com a interface de hardware. Um computador pode ter inúmeros endereços de camada 3 por interface, porem só pode ter um endereço físico por interface de rede.

Quando o computador tem dados para enviar para algum destinatário representado por um endereço IP, por exemplo, ele encapsula esses dados em um pacote IP e então tem que realizar algumas etapas antes de enviar. Primeiro ele verifica se o endereço IP destino pertence ou não a mesma rede em que ele está conectado. Em caso afirmativo, ele precisa então determinar qual o endereço físico correspondente ao endereço IP destino, para que então possa encapsular o pacote IP em um frame e colocá-lo no meio de transmissão. Para determinar o endereço físico destino o computador primeiramente verifica se existe uma entrada em sua tabela de Cache de ARP. Caso exista,ele encapsula o pacote IP em um frame e envia, caso contrário é necessário que se envie uma mensagem ARP em broadcast com a seguinte pergunta: Quem possui o endereço IP *end*? Todas as máquinas da rede local irão receber a mensagem e analisar o conteúdo. Aquela que possuir endereço *end* responderá a mensagem arp enviando seu endereço físico.

30.2 DHCP - Dynamic Host Configuration Protocol

O DHCP (Dynamic Host Configuration protocol) é um protocolo que objetiva realizar a configuração dinâmica de hosts em uma rede ou na Internet. O DHCP consiste de dois componentes básicos que são um protocolo para trocar mensagens e transportar informações relativas a configurações do host e um mecanismo para controlar a alocação de endereços de rede.

O DHCP é baseado no modelo cliente-servidor, onde o cliente requisita os parâmetros de configuração e os servidores recebem os pedidos, os analisa e prove as configurações para os hosts. Vale ressaltar que é possível a coexistência de mais de um servidor DHCP em uma rede.

Também é possível o servidor DHCP estar fora da rede em que se encontram os hosts solicitantes (clientes). Neste caso, é necessária a instalação de um agente de retransmissão DHCP (DHCP Relay Agent). A única informação que este agente precisa ter é o endereço IP do servidor DHCP que está fora da rede.

As três formas de assinalar um endereço IP são:

- Alocação Automática - um endereço IP é fornecido para de forma permanente pelo servidor de DHCP;
- Alocação Dinâmica - as configurações do host são fornecidas por um determinado período de tempo chamado "lease". As configurações podem ser canceladas antes desse período expirar caso seja de interesse do host;
- Alocação Manual - as configurações do host são ajustadas manualmente pelo administrador da rede.

Em geral em uma rede essas formas de assinalar endereços são utilizadas em conjunto. Entre os requisitos de um serviço de DHCP em uma rede estão: garantir que um mesmo endereço não estará em uso mais de um host ao mesmo tempo; estar preparado para funcionar em mais de um em uma rede de forma aumentar a performance e redundância; estar preparado para lidar com hosts estaticamente configurados.

Entre os parâmetros de configuração fornecidos pelo DHCP, estão, o endereço de rede, a máscara de subrede, o tempo de duração da licença (*lease*), endereço do gateway padrão, endereço dos servidores de DNS, etc.

A alocação dinâmica de endereço é o mais importante serviço fornecido pelo DHCP. Para um host adquirir configurações as seguintes etapas são necessárias:

- Um cliente faz um broadcast de uma mensagem DHCPDISCOVER para na sua rede local física. Caso o servidor de DHCP não esteja naquela rede física, os *agentes relay* encaminham as mensagens para frente através dos roteadores;
- Os servidores recebem a mensagem DHCPDISCOVER e respondem com uma mensagem DHCPOFFER com as configurações que ele pode fornecer;
- O cliente recebe uma ou mais mensagem DHCPOFFER e escolhe qual delas responderá enviando um broadcast da mensagem DHCPREQUEST indicando qual servidor escolheu e possivelmente adicionando novos requisitos de configuração;
- O servidor recebe a mensagem DHCPREQUEST e checa em sua base de dados se é possível fornecer as configurações desejadas pelo host. Caso seja possível, envia uma mensagem com o DHCPACK, senão envia um

DHCPNAK. O processo é reiniciado após um tempo determinado por backoff exponencial;

- O host, caso tenha recebido ACK, checa as configurações fornecidas usando ARP, por exemplo, e caso detecte que configurações não são válidas, envia uma mensagem do tipo DHCPDECLINE, cancelando a negociação.

30.3 DNS - Domain Name System

Originalmente, o mapeamento de nomes em endereços era todo registrado em um único arquivo chamado HOSTS.TXT. Esse arquivo era gerenciado por uma entidade chamada NIC (Network Information Center). A distribuição das atualizações desse arquivo era feita via FTP, o que consumia uma banda muito grande. O perfil do usuário de redes e da Internet mudou muito, ocorrendo um grande crescimento das redes locais, com suas próprias necessidades de mapear nomes que só faziam sentido em seus ambientes. O perfil das aplicações de Internet também evoluiu bastante, surgindo a necessidade de se criar um sistema de mapeamento de nomes mais geral e eficiente. Foi neste contexto que surgiu o DNS.

O DNS é (1) um banco de dados distribuído implementado em uma hierarquia de servidores de nome, e (2) um protocolo de camada de aplicação que permite que hospedeiros consultem o banco de dados distribuído. Para tratar a questão da escala, o DNS usa um grande número de servidores organizados, de maneira hierárquica e distribuída, por todo o mundo. Os principais componentes do DNS são:

- Domain Name Space (Espaço de Nomes): é uma especificação para uma árvore estruturada para armazenar os espaços de nomes. Esta árvore é dividida em ZONAS não-superpostas. Normalmente, cada zona tem um servidor de nome principal (onde os registros são mantidos em disco) e um ou mais servidores secundários;
- Resource Records (Registro de Recursos): A principal função do DNS é mapear nomes de domínios em registros de recursos (não somente endereço IP). Um registro de recurso é uma tupla de cinco campos: Domain_name (normalmente existem muitos registros, de tipos diferentes, para cada domínio); Time_to_live (tempo de vida do registro); Class (IN quando relacionado à Internet, outros códigos são raramente encontrados); Type (tipo de registro); e Value (sua semântica depende do tipo de registro). Os tipos de registros existentes são:
 - Name Servers (Servidores de Nomes): são programas servidores que detêm informação sobre a estrutura da árvore de domínio e também tem a capacidade de registrar informações. Resumidamente, há quatro classes de servidores de nomes: servidores de nomes raiz (cerca de 13 servidores); servidores DNS de domínio de nível superior (.int, .com, .mil, .net, .br, .jp, etc. - cerca de 200 servidores); servidores DNS com autoridade (chamados de AUTHORITY - os que detêm os registros de recursos de servidores, localizados em sua zona, que podem ser acessados publicamente); e servidores intermediários;

Tipo	Significado	Valor
SOA	Início de autoridade	Parâmetro para essa zona
A	Endereço IP de um host Inteiro de 32 bits	
MX	Troca de mensagens de correio	Prioridade, domínio disposto a aceitar correio eletrônico
NS	Servidor de nomes	Nome de um servidor para este domínio
CNAME	Nome canônico	Nome de domínio
PTR	Ponteiro	Nome alternativo de um endereço IP
HINFO	Descrição de host	CPU e sistema operacional em ASCII
TXT	Texto	Texto ASCII não-interpretado

Tabela 30.1: Principais tipos de registros de recursos do DNS para o IPv4

- Resolvers (Resolvedores): são programas, executados tanto nos clientes quanto nos servidores, que extraem informações dos Name Servers em resposta a requisições feitas por clientes. Esses programas são aptos a responder as queries diretamente ou então encaminhar a queries caso a informação não esteja disponível naquele Name Server. São programas que podem ser acessados diretamente por programas de usuários ou rotinas do sistema operacional sem a necessidade de utilização de protocolos;
- Cache DNS: uma característica muito importante usada para aumentar o desempenho quanto a atraso e reduzir o número de mensagens DNS na Internet. Em cada elemento da cadeia (cliente, servidores local, secundários, primários, com autoridade, de nível superior, etc.) informações (registros) de respostas podem ser armazenadas em memória local. Para evitar o uso de registros desatualizados, é associado a cada registro um número TTL (Time to Live) que determina seu tempo de vida;

Existem dois métodos de consulta: recursiva, onde cada servidor que não tiver as informações solicitadas poderá encontrá-las, por meio de novas consultas, em algum lugar e informar ao solicitante o que encontrou. Iterativa, onde quando a consulta não pode ser satisfeita no local, haverá uma falha, mas é retornado o nome do próximo servidor a ser consultado, pelo solicitante, ao longo da linha. Para cada consulta, os tipos básicos de resposta são: authoritative answer (consulta resolvida pelo servidor com autoridade); positive answer; referral answer (não contém resolução da consulta, e sim uma referência para uma nova consulta - usada no método iterativo); e negative answer (ou o nome pesquisado não existe ou o tipo de registro não confere - sempre indicado pelo servidor com autoridade).

É importante observar que o DNS não somente provê o serviço de tradução de nomes em endereços IPs. Ele também provê outros serviços mais sofisticados, tais como:

- Distribuição de carga: implementado por meio do tipo de registro A com mais de um IP de servidores replicados (ex. Web ou FTP) utilizados na distribuição de carga. Ao resolver o nome, o DNS faz um rodízio entre os IPs cadastrados;
- Apelidos de hospedeiros: implementado por meio do tipo de registro CNAME (um hospedeiro com nome complicado pode ter um ou mais apelidos);
- Apelidos de servidores de correio: implementado por meio do tipo de registro MX;

O DNS utiliza UDP/53 para resolução de nomes e TCP/53 para realizar transferência de zonas.

O nome completo de um host da rede é conhecido como FQDN - Full Qualified Domain Name. Por exemplo, ftp.abc.com.br é um FQDN. ftp (a primeira parte do nome) é o nome de host e o restante representa o domínio DNS no qual está o host.

Dica Importante: O administrador do servidor DNS local pode desabilitar o recurso de recursão em um servidor DNS em situações onde os usuários devem estar limitados a utilizar apenas o servidor DNS da Intranet da empresa. Desta forma, somente os endereços internos (armazenados no disco do servidor DNS local - AUTHORITY) serão resolvidos.

O nslookup é uma ferramenta, comum ao Windows e ao Linux, utilizada para se obter informações sobre registros de DNS de um determinado domínio, host ou IP. Podemos utilizar esta ferramenta em dois modos diferentes. No modo direto, onde digitamos o comando nslookup e mais alguns parâmetros, e no modo interativo, onde digitamos somente o comando nslookup e um prompt é aberto para que sejam realizadas consultas ao servidor DNS. Alguns exemplos de parâmetros/consultas são: nslookup "nome do host"/nomes e endereços do servidor DNS e do host; nslookup -querytype=ANY "nome do host"/todas as informações disponíveis do servidor DNS e do host.

30.4 TCP - Transmission Control Protocol

O protocolo TCP (Transmission Control Protocol) é um protocolo da camada de transporte cujo principal objetivo é prover um serviço de conexão confiável entre um par de processos que desejam se comunicar. Prover facilidades como transferência básica de dados, controle de fluxo, multiplexação, precedência, controle de conexões, e alguma segurança. A expressão conexão confiável está relacionada com a capacidade do protocolo TCP em lidar com dados danificados, perdidos ou duplicados utilizando para isso números de seqüência, ACKs, buffers de retransmissão. O TCP implementa também a multiplexação para permitir que múltiplos processos utilizem as facilidades do TCP em um mesmo host. Para isso utiliza um identificador de conexão chamado soquete, que é formado pela *porta* e endereço de rede.

O cabeçalho do TCP é formado pelos seguintes campos: (i) Source Port - porta de origem; (ii) Destination Port - porta de destino; (iii) Sequence Number - numero de seqüência do segmento; (iv) Acknowledgment Number - o numero de seqüência que o sender do segmento espera receber; (v) Data Offset - Indica onde os dados começam. Múltiplo de 32 bits; (vi) Control Bits - (URG/ACK/PSH/RST/SYN/FIN); (vii) Window - tamanho da janela aceitável. Indica range de números de seqüência aceitos; (viii) Checksum - Checksum calculado sobre headers e dados; (ix) Urgent Pointer - Indica o offset dos dados urgentes em um dado segmento. Só tem sentido com o flag URG setado. (x) Options: opções adicionais (Ex: Maximum Segment Size. Esta opção só deve ser utilizada no estabelecimento da conexão. Caso contrário qualquer tamanho de segmento é aceito); (xi) Padding - usado para garantir que o header seja múltiplo de 32 bits e o campo data offset trabalhe corretamente.

Para manter uma conexão TCP é necessário que se guarde o status de várias variáveis referentes a recebimento e envio de segmentos, informações sobre o segmento corrente, informação sobre o estado da conexão e identificador da conexão. Todas essas conexões são guardadas em uma estrutura chamada *TCB* - *Transmission Control Block*. Entre as informações guardadas nos TCBs estão, por exemplo, o tamanho da janela, numero de seqüência inicial da conexão e o estado dos bits de controle.

As conexões TCP podem se encontrar nos seguintes estados:

- LISTEN - A espera de um pedido de conexão;
- SYN-SENT - A aplicação começou a abrir uma conexão;
- SYN-RECEIVED - Uma solicitação chegou. Espera por um ack;
- ESTABLISHED - Estado normal para envio de dados;
- FIN-WAIT-1 - A aplicação informou que acabou de transmitir;
- FIN-WAIT-2: O outro lado concordou em encerrar;
- TIMED-WAIT - Aguarda a entrega de todos os pacotes;
- CLOSE-WAIT - Um lado deu inicio ao encerramento;
- LAST-ACK - Aguarda entrega de todos os pacotes;
- CLOSING - Ambos tentaram encerrar simultaneamente.

A conexão passa de um estado para o outro em resposta a eventos que são as chamadas as funções OPEN, SEND, RECEIVE, CLOSE, ABORT, STATUS ou então segmentos contendo flags SYN, ACK, RST, e FIN, além dos timeouts.

As duas causas de congestionamento são a sobrecarga na rede e a sobrecarga no receptor. É necessário lidar com cada uma das causas separadamente. Se o mecanismo de *Janela de Transmissão* for obedecido pelo transmissor, não haverá problemas devido a sobrecarga de buffers mas somente devido a congestionamentos internos da própria rede. Para isso utiliza-se uma segunda janela

chamada *Janela de Congestionamento*.

O valor mínimo entre essas janelas reflete o numero máximo de bytes que um transmissor pode enviar. O mecanismo de janela de congestionamento funciona da seguinte maneira: Ao estabelecer a conexão o valor da janela de congestionamento é setado para o valor máximo do segmento (MSS) cordado na conexão. É feita então a transmissão de um segmento de tamanho MSS. Caso não tenha ocorrido timeout, o valor da janela de congestionamento é aumentado para duas vezes o anterior. Esse procedimento segue até que uma transmissão resulte em timeout. Caso isso ocorra o procedimento é reiniciado a partir do valor inicial. Esse é um algoritmo exponencial. Existe uma variante que permite a utilização de um limiar a partir do qual o crescimento passa a ser linear e não exponencial. Na internet essa técnica é utilizada.

O mecanismos de abertura de conexão é conhecido como *Three-Way Handshake*. É formado pelas seguintes etapas: (1) A → B (SYN my sequence number is X); (2) A ← B ACK (your sequence number is X) e SYN (my sequence number is Y); (3) A → B ACK (your sequence number is Y).

Os timers utilizados pelo TCP são: Timer de Retransmissão - responsável por indicar se um segmento deve ser retransmitido; Timer de Persistência - utilizado para evitar o impasse quando o anúncio do tamanho de janela se perder, Timer de keepAlive - pode expirar quando uma conexão fica inativa por muito tempo. É feita então uma checagem para ver se o outro lado ainda está ativo; TIME WAIT - utilizado no estado da conexão.

30.5 UDP - User Datagram Protocol

O UDP é um dos principais protocolos da Internet. É um protocolo destinado ao envio de mensagens custas. Por isso se diz que ele é um protocolo orientado a mensagens e também stateless. Não proporciona nenhuma garantia de entrega nem ordenação. No entanto, é muito mais leve e eficiente do que o TCP. Esse é um dos motivos pelos quais o UDP é muito utilizado em aplicações sensíveis ao tempo.

Entre as inúmeras aplicações do protocolo UDP estão os serviços de RIP, SNMP, DHCP, DNS, VoIP, jogos online, aplicações de multimídia, entre outras. Em um serviço de transmissão de voz, a retransmissão do dado não é útil devido ao atraso, assim como em uma transmissão de vídeo. É um protocolo da camada de transporte e possui um cabeçalho simplificado com apenas 4 campos que são:(i) *Source Port*;(ii) *Destination Port*;(iii) *Lenght*;(iv) *Checksum*.

Cada um é formado por 16 bits. Os campos Source Port e Checksum são opcionais. Na prática, o campo é quase sempre utilizado, enquanto o campo Source Port pode ser utilizado quando se deseja receber uma resposta. É importante lembrar que na comunicação UDP não existem mensagens de confirmação. Quando essas ocorrem se devem a iniciativa da aplicação que está utilizando o UDP e não das regras de comunicação do protocolo em si. Exemplo: Uma querie DNS é enviada ao servidor que por sua vez responde utilizando também

uma mensagem UDP.

Como o campo Source Port é opcional, caso não seja preenchido, no momento do cálculo do Checksum ele é considerado 0. Nos casos em que o Checksum também não é calculado, o valor enviado consiste em 16 bits 0. Caso o checksum calculado seja 16 bits 0, o valor do checksum enviado é composto por 16 bits 1.

30.6 HTTP - Hyper Text Transfer Protocol

É o protocolo de CAMADA DE APLICAÇÃO da Web definido no RFC 1945 (HTTP 1.0) e no RFC 2616 (HTTP 1.1). Ele é implementado em dois programas, um cliente e outro servidor, portanto, ele é classificado como aplicação cliente-servidor. Estes dois programas, executados em sistemas finais distintos, conversam entre si por meio de troca de mensagens HTTP. O HTTP define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam.

Uma página Web (ou documento) é constituída de OBJETOS. Um objeto é simplesmente um arquivo. Exemplos de objetos são: arquivo HTML, imagem JPEG, imagem GIF, applet JAVA, etc. Cada objeto é referenciado por uma única URL. Cada URL tem dois componentes: o nome do hospedeiro e o nome do caminho do objeto. Por exemplo, na URL <http://www.universidade.br/departamento/figura.gif>, www.universidade.br é o nome do hospedeiro e /departamento/figura.gif é o nome do caminho do objeto.

Um BROWSER é um AGENTE DE USUÁRIO para a Web. Eles implementam o lado cliente do HTTP a fim de apresentar páginas Web e fornecer numerosas características de navegação e de confirmação. Vale salientar que dois browsers distintos podem apresentar páginas Web de formas ligeiramente diferentes, pois o protocolo HTTP não define nada sobre apresentação. Os SERVIDORES WEB implementam o lado servidor do HTTP a fim de abrigar objetos Web. Os Servidores Web mais utilizados atualmente são os Apache e o Microsoft Internet Information Server (IIS).

O HTTP usa o TCP como seu protocolo de transporte. Por padrão, as portas utilizadas são as 80 (primária) e 8080 (alternativa). Desta forma, as trocas de mensagens são realizadas de forma orientada a conexão. Estas conexões podem ser persistentes ou não-persistentes.

As conexões não-persistentes seguem o seguinte fluxo:

- O processo cliente abre uma conexão TCP executando a APRESENTAÇÃO DE TRÊS VIAS (three way handshake);
- Durante a terceira via do three way handshake, o processo cliente envia uma mensagem de requisição HTTP ao servidor;
- O processo servidor recebe a mensagem de requisição, encapsula o objeto requisitado em uma mensagem de resposta HTTP, e a envia ao cliente;
- O processo servidor solicita o encerramento da conexão TCP;

- Após o tempo de transmissão da mensagem resposta, o processo cliente recebe a mensagem de resposta, extrai o arquivo da mensagem de resposta e o apresenta.

Desta forma, para cada objeto transferido há a exigência de abertura de uma nova conexão TCP. Como geralmente as páginas Web são consistidas de um arquivo-base HTML e diversos objetos referenciados, este tipo de conexão se mostrou um tanto quanto ineficiente.

O tipo de conexão persistente permite um aumento considerável na eficiência das trocas de mensagens. Neste tipo de conexão, o servidor deixa a conexão TCP aberta após enviar resposta. Ou seja, requisições e respostas subsequentes entre os mesmos cliente e servidor podem ser enviadas por meio da mesma conexão. Em geral, o servidor HTTP fecha uma conexão quando ela não é usada durante certo tempo (configurável).

Há duas versões de conexões persistentes: SEM PARALELISMO e COM PARALELISMO. Na versão sem paralelismo, o cliente emite uma nova requisição somente quando a resposta anterior foi recebida. O modo default do HTTP usa conexões persistentes com paralelismo. Nesse caso, o cliente emite uma requisição logo que encontra uma referência, mesmo antes de receber uma resposta a uma requisição anterior (PIPELINE).

Uma outra forma de paralelismo é o cliente abrir conexões paralelas com um servidor. Nos modos default, a maioria dos browsers abre de cinco a dez conexões TCP paralelas, cada uma delas manipula uma transação requisição/resposta.

É importante salientar que o servidor envia as respostas HTTP ao cliente sem armazenar nenhuma informação de estado sobre este. Se um determinado cliente solicita o mesmo objeto duas vezes em um período de poucos segundos, o servidor não responde dizendo que acabou de enviar o objeto. Em vez disso, ele envia novamente o objeto ao cliente. Em outras palavras, o HTTP é um PROTOCOLO SEM ESTADO.

Como já se pode notar, as mensagens são de dois tipos: mensagem de requisição e mensagem de resposta. Antes de apresentar os principais campos do tipo de mensagem de requisição, vamos a um exemplo típico.

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language: fr
```

Em primeiro lugar, vemos que esta mensagem está escrita em texto ASCII comum. Embora ela tenha cinco linhas, ela poderia ter de uma a diversas linhas. A primeira linha sempre é denominada LINHA DE REQUISIÇÃO. As subsequentes são denominadas LINHAS DE CABEÇALHO. A linha de requisição é composta por três campos: o campo do MÉTODO; o da URL; e o da VERSÃO do HTTP. A linha de cabeçalho Host especifica o hospedeiro no qual o

objeto reside (este campo é utilizado para buscar objetos em servidores proxy). Ao incluir a linha de cabeçalho Connection: close, o browser está encerrando a conexão. A linha de cabeçalho User-agent especifica o agente de usuário, ou seja, o tipo de browser (o servidor pode enviar versões diferentes de um mesmo objeto). Por fim, o cabeçalho Accept-language mostra que o usuário prefere receber uma versão em francês do objeto se este existir.

O formato geral de uma mensagem de requisição é mostrado na figura abaixo.

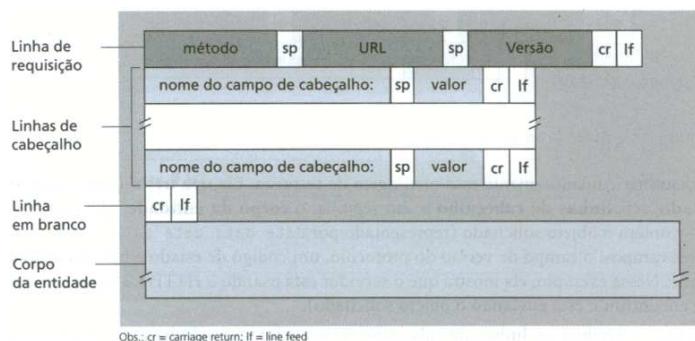


Figura 30.1: Formato geral de uma mensagem de requisição HTTP

Agora que já apresentamos um exemplo de mensagem de requisição, vamos a um exemplo típico de mensagem de resposta.

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 03 Jul 2003 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Sun, 5 May 2003 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
```

(data data data data data data ...)

Este exemplo tem três seções: a LINHA DE ESTADO; seis LINHAS DE CABEÇALHO; e CORPO DA ENTIDADE (contém os objetos propriamente ditos). A linha de estado tem três campos: o campo da VERSÃO do HTTP; um CÓDIGO DE ESTADO; e uma MENSAGEM de texto correspondente. A linha de cabeçalho Connection: close indica que o servidor fechará a conexão após o envio da mensagem. A linha de cabeçalho Date indica a hora e a data em que a resposta foi criada e enviada pelo servidor (não se refere à data de criação do objeto nem de sua última alteração). A linha de cabeçalho Server mostra por qual tipo de servidor a mensagem foi criada. A linha de cabeçalho Last-Modified indica a hora e a data em que o objeto foi criado ou sofreu a última modificação (fundamental para a implementação de servidores proxy). A linha de cabeçalho Content-Length indica o número de bytes do objeto que está sendo enviado. Por fim, a linha de cabeçalho Content-Type mostra o tipo de objeto presente

no corpo da mensagem. É importante notar a linha em branco antes do corpo da entidade. Esta linha é obrigatória e indica que não há mais nenhuma linha de cabeçalho.

O formato geral de uma mensagem de resposta é mostrado na figura abaixo.

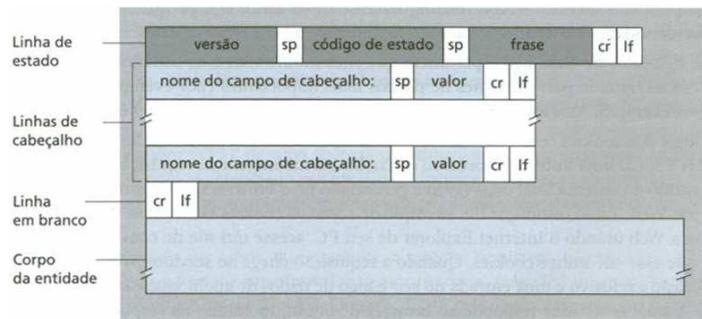


Figura 30.2: Formato geral de uma mensagem de resposta HTTP

Os métodos implementados pelo HTTP 1.1 são:

- GET - Este método solicita um objeto, identificado por uma URL indicada na mensagem de requisição. Um "GET condicional" é um tipo específico de GET. Ele deve possuir pelo menos uma linha de cabeçalho do tipo If-Modified-Since ou If-Unmodified-Since. De uma forma geral, ele permite melhorar o desempenho transmitindo apenas os dados que atendem determinada condição. Existe também o "GET parcial" que permite que apenas parte da informação seja transmitida. Isso pode ser utilizado quando o cliente já possui uma parte dos dados. Para isso o campo "Range" do cabeçalho deve ser preenchido. A grande maioria das mensagens de requisição tem a forma de método GET;
- HEAD - Este método é semelhante ao GET, contudo é utilizado para trazer apenas os cabeçalhos da mensagem que são exatamente os retornados pelo GET e não os objetos requisitados;
- POST - Este método é utilizado para acrescentar informações a uma requisição. Por exemplo, quando o cliente preenche algum formulário de busca. Com esta mensagem, o cliente continua requisitando um objeto (ou objetos), mas seu conteúdo depende do conteúdo do formulário. Os conteúdos do formulário são transmitidos no corpo da entidade. OBS: não necessariamente as requisições geradas com um formulário utilizam o método POST. É possível utilizar o método GET com os conteúdos do formulário como parte da URL (ex. www.somesite.com/animalsearch?monkey&bananas);
- PUT - Este método solicita o armazenamento de um objeto, enviado no corpo da entidade, em um diretório em um determinado Servidor Web (indicados por uma URL). Na prática este método não é muito utilizado hoje em dia;

- **DELETE** - Este método solicita que um objeto determinado por uma URL seja deletado no Servidor Web;
- **TRACE** - Este método requisita o destino final a refletir (enviar de volta) a mensagem que está recebendo. O campo de cabeçalho Max-Fowards pode ser preenchido de forma limitar o número de proxies para detectar possíveis loops;
- **CONNECT**: Não é utilizado atualmente. Ele é reservado para uso futuro;
- **OPTIONS**: fornece um meio para que o cliente consulte o servidor sobre suas propriedades ou sobre as de um objeto específico.

Os CÓDIGOS DE ESTADO são códigos para representar o atual estado do servidor. São separados nas seguintes classes:

- 1xx (Informational) - Raramente utilizados na prática. Ex. 100 - Indicam que servidor concorda em atender ou continuar atendendo requisição do cliente;
- 2xx (Successful) - Indicam que a requisição foi corretamente recebida, entendida, aceita e que o conteúdo (se houver) está sendo retornado. Ex. 200 - requisição bem-sucedida; 204 - sem conteúdo;
- 3xx (Redirection) - Indica que o cliente deve procurar em outro lugar, usando um URL diferente ou seu próprio cache. Ex. 301 - a página foi movida; 304 - a página no cache ainda é válida;
- 4xx (Client Error) - Indica um erro do cliente ao fazer a requisição. Ex. 403 - página;
- 5xx (Server Error) - Indica um erro do servidor ao processar a requisição.

Cabe ressaltar que o HTTP não é somente utilizado no contexto Web. Ele também é frequentemente utilizado em aplicações de comércio eletrônico para transferir arquivos XML de uma máquina para outra, sem que nenhuma dessas máquinas envolva um browser ou um usuário. O HTTP também é utilizado para transferir VoiceXML, WML e outros tipos de XML. Outra utilidade do HTTP é servir como protocolo de transferência de arquivos no compartilhamento de arquivos em redes P2P.

Outra observação importante é o fato do HTTP não exigir conexões específicas e distintas para transferência de mensagens de controle e para transferência de mensagens de dados. Desta forma, dizemos que as mensagens de controle são enviadas NA BANDA.

A versão 1.1 do HTTP é totalmente compatível com a versão 1.0, contudo há algumas diferenças. As duas principais são:

- A versão 1.0 não suporta conexões persistentes. Já a versão 1.1 suporta ambos os tipos de conexão, persistente e não-persistente;
- A versão 1.0 permite somente três tipos de métodos: GET, POST e HEAD.

30.7 SMTP - Simple Mail Transfer Protocol

É o protocolo de CAMADA DE APLICAÇÃO do modelo OSI definido pela RFC 2821. Ele é o principal protocolo do correio eletrônico da Internet. Ele é implementado em dois programas, um cliente e outro servidor, portanto, ele é classificado como aplicação cliente-servidor. Estes dois programas, executados em sistemas finais distintos, conversam entre si por meio de troca de mensagens SMTP.

Antes de apresentarmos as principais características do SMTP, se faz necessário abordar alguns assuntos relevantes a cerca do sistema de correio eletrônico na Internet. Uma grande característica deste sistema é o fato das comunicações serem assíncronas. Ou seja, as pessoas enviam e recebem mensagens quando for conveniente para elas. Este sistema é composto basicamente por três componentes: AGENTES DE USUÁRIOS; SERVIDORES DE CORREIO (também chamado de AGENTE DE TRANSFERÊNCIA DE MENSAGEM); e o SMTP.

- Os Agentes de Usuários são executados localmente na máquina do usuário, permitindo-o a ler, responder, retransmitir, salvar e escrever mensagens. Exemplos de Agentes de Usuários com interface gráfica são: Eudora, Outlook e o Messenger da Netscape. Outros Agentes sem interface gráfica são: mail, pine e elm;
- Os Servidores de Correio formam o núcleo da infra-estrutura do sistema. Cada usuário tem uma CAIXA POSTAL. Diversos Agentes de Usuários podem se conectar ao Servidor de Correio no intuito de enviar/receber mensagens. Eles implementam tanto o lado cliente quanto o lado servidor do SMTP. Quando um Servidor de Correio envia correspondência para outros, age como um cliente SMTP. Quando um Servidor de Correio recebe correspondência de outros, age como um servidor;
- O SMTP define a estrutura das mensagens trocadas, principalmente entre os Servidores de Correio.

O SMTP usa o TCP como seu protocolo de transporte. Desta forma, as trocas de mensagens são realizadas de forma orientada a conexão. Por padrão, a porta utilizada é a 25 e as conexões são sempre persistentes. Ou seja, em uma mesma sessão SMTP, diversas mensagens podem ser enviadas.

Para ilustrar uma operação típica do SMTP, vamos recorrer a um cenário bem comum descrito pela figura 30.3. Suponha que Alice queira enviar a Bob uma simples mensagem ASCII. Os seguintes passos são seguidos:

- Alice compõe a mensagem e solicita ao seu Agente de Usuário a entregar esta mensagem a Bob;
- O Agente de Usuário de Alice envia a mensagem para seu Servidor de Correio por meio do SMTP. Esta mensagem será colocada em uma fila de mensagens. Na verdade, este Agente de Usuário poderia dialogar diretamente com o Servidor de Correio de Bob, porém na prática esta não é uma abordagem tão comum;

- O lado cliente do SMTP, que funciona no Servidor de Correio de Alice, vê a mensagem na fila e abre uma conexão TCP para um servidor SMTP, que funciona no Servidor de Correio de Bob. Caso esta conexão não possa ser aberta, em geral, novas tentativas serão feitas a cada 30 minutos, se não obtiver sucesso após alguns dias, o Servidor de Correio de Alice removerá a mensagem da fila e a notificará. Os Servidores de Correio se encontram na Internet usualmente consultando o DNS, mais especificamente o registro MX (Mail eXchange);
- Após alguns procedimentos iniciais de apresentação, o cliente SMTP envia a mensagem de Alice para dentro da conexão TCP;
- No Servidor de Correio de Bob, o lado servidor do SMTP recebe a mensagem e a coloca na caixa postal dele;
- Bob chama seu Agente de Usuário para ler a mensagem quando for mais conveniente para ele. Vale salientar que a comunicação entre o Agente de Usuário de Bob e seu Servidor de Correio não pode se dar via SMTP. Este último passo do processo deverá ser feito via um dos protocolos: HTTP, POP3 ou IMAP.

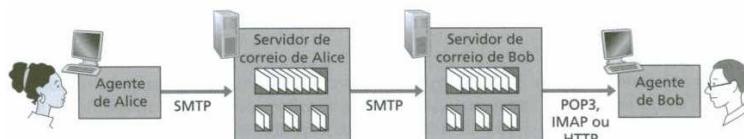


Figura 30.3: Protocolos de e-mail e suas entidades comunicantes

Uma diferença marcante entre o HTTP e o SMTP é o fato do HTTP ser um PROTOCOLO DE RECUPERACAO DE INFORMAÇÕES (pull protocol). Nesse tipo de protocolo, as conexões são abertas pelas máquinas que desejam receber as informações. Já o SMTP é um PROTOCOLO DE ENVIO DE INFORMAÇÕES. Neste tipo de protocolo, as conexões são abertas pelas máquinas que desejam enviar as informações (cliente SMTP). É por este motivo que o último passo da transferência entre dois usuários, entre Agente de Usuário destinatário e seu Servidor de Correio, não pode ser feito via SMTP.

É importante observar que o SMTP normalmente não usa servidores de correio intermediários para enviar correspondência, mesmo quando os dois servidores estão localizados em lados opostos do mundo. Isto porque ele utiliza o TCP como protocolo de transporte, que é orientado a conexão. Portanto, geralmente os Servidores de Correio cliente e servidor se conectam diretamente (virtualmente falando).

A cada comando enviado pelo cliente SMTP, haverá uma resposta enviada pelo servidor SMTP via um código de resposta e eventualmente uma mensagem de explicação. Os principais comandos são:

- DATA: inicializa a transmissão do corpo da mensagem;

- HELO: identifica o emissor da mensagem;
- HELP: solicita ao servidor SMTP informações de ajuda;
- MAIL FROM: inicializa uma transação de mail;
- QUIT: determina que o servidor SMTP envie um ok e então feche a conexão;
- RCPT TO: identifica o destinatário da mensagem. Múltiplos destinatários são definidos por múltiplos usos deste comando;
- TURN: faz com que os cliente e servidor troquem de papel, o cliente passa a ser servidor e vice-versa;
- . (ponto): identifica o fim do corpo da mensagem.

Todo correio eletrônico é dividido em duas partes: MENSAGEM e ENVELOPE. O conteúdo da parte envelope é determinado pelo protocolo de apresentação SMTP. A mensagem é subdividida em CABEÇALHO e CORPO. As informações contidas no cabeçalho são determinadas no RFC 822. Cada cabeçalho deve ter obrigatoriamente uma linha From: e uma To: e pode incluir também uma Subject:, bem como outras linhas opcionais. É importante notar que essas linhas de cabeçalho são diferentes dos comandos SMTP, ainda que contenham algumas palavras em comum.

Uma característica marcante do SMTP é o fato dele exigir que o formato de representação, tanto dos cabeçalhos quanto do corpo da mensagem, seja ASCII. Desta forma, conteúdos não ASCII, por exemplo, imagens, áudio, vídeo e caracteres especiais, devem ser codificados em ASCII antes de serem enviados pelo SMTP. Após a transferência, estes dados são decodificados novamente em formato ASCII. Para que isto seja possível, há necessidade de incluir cabeçalhos adicionais na mensagem. Esses cabeçalhos extras são definidos no RFC 2045 e no RFC 2046, que são extensões do RFC 822 referentes ao MIME (Multipurpose Internet Mail Extentions). Os dois cabeçalhos MIME fundamentais para suporte de multimídia são: Content-Type e Content-Transfer-Encoding. O primeiro permite que o Agente de Usuário destinatário realize uma ação adequada sobre a mensagem, por exemplo, execução de uma rotina de descompressão JPEG. O segundo alerta o Agente de Usuário destinatário que o corpo da mensagem foi codificado em ASCII.

30.8 POP3 - Post Office Protocol Version 3

É o protocolo de CAMADA DE APLICAÇÃO do modelo OSI definido no RFC 1939. Ele é um protocolo de acesso de correio eletrônico extremamente simples, e como consequência bastante limitado. Por padrão, ele utiliza o TCP como protocolo de transporte via porta 110.

Seu funcionamento típico é da seguinte forma. O Agente de Usuário, agindo como cliente, abre uma conexão TCP com o Servidor de Correio que age como servidor. Com a conexão ativada, o protocolo passa por três fases: AUTORIZAÇÃO,

TRANSAÇÃO e ATUALIZAÇÃO.

Durante a fase de autorização, o Agente de Usuário envia um nome de usuário e uma senha (às claras) para autenticar o usuário. Na fase de transação, as mensagens são transferidas ao cliente. É também nesta fase que o Agente de Usuário pode marcar/desmarcar as mensagens que deverão ser apagadas, e obter estatísticas de correio. A fase de atualização ocorre após o cliente ter solicitado o encerramento da conexão. Nesse momento, o Servidor de Correio apaga as mensagens que foram marcadas.

Em uma transação POP3, o Agente de Usuário emite comandos e o Servidor de Correio emite uma resposta a cada comando recebido. Há duas respostas possíveis:

- +OK: indica que correu tudo bem com o comando anterior. Esta resposta às vezes vem seguida de dados do servidor para o cliente;
- -ERR: indica que houve algo errado com o comando anterior.

Os comandos do POP3 são definidos na RFC 1939. Os principais são:

- USER: informa o nome de usuário;
- PASS: informa a senha de usuário;
- LIST: solicita a listagem das mensagens, e seus respectivos tamanhos, presentes na caixa postal;
- RETR: solicita o descarregamento de uma mensagem;
- DELE: marca uma mensagem para ser deletada na fase de atualização;
- QUIT: determina que o servidor envie um ok e então feche a conexão.

Uma característica importante deste protocolo é o fato dele realizar uma conexão persistente, ou seja, em uma mesma sessão TCP entre cliente e servidor é possível a transferência de diversas mensagens.

Um Agente de Usuário que utiliza POP3 frequentemente pode ser configurado pelo usuário final para LER-E-APAGAR ou para LER-E-GUARDAR. A seqüência de comandos emitida por um Agente de Usuário depende desta configuração.

No modo ler-e-apagar, os seguintes comandos são emitidos: LIST, RETR e DELE. Múltiplas mensagens são transferidas por meio de múltiplas seqüências de comandos RETR e DELE, uma seqüência para cada mensagem. Há um problema com o modo ler-e-apagar. O usuário destinatário pode ser nômade e querer acessar sua caixa de postal de muitas máquinas. Desta forma, as mensagens ficam repartidas entre as máquinas que estabeleceram conexão com o Servidor de Correio, não permitindo a visualização em uma máquina de mensagens baixadas por outra máquina diferente.

No modo ler-e-guardar, o Agente de Usuário deixa as mensagens no Servidor de Correio após descarregá-las. Desta forma, o problema anterior de repartição de mensagens é resolvido.

Cabe ressaltar que o Agente de Usuário, ao se conectar ao Servidor de Correio, descarrega todas as mensagens que estão na caixa postal. Por este fato, diz-se que este protocolo é off-line. Ou seja, o usuário final recebe todas as suas mensagens e então pode lê-las e apaga-las do servidor de arquivo local mesmo estando offline. Portanto, este protocolo se mostra interessante para usuários que pagam pelo acesso a Internet por tempo de conexão, por exemplo, conexão discada.

Uma característica bastante importante do POP3 é o fato dele guardar informações de estados durante uma sessão POP3. Em particular, monitora as mensagens do usuário marcadas para apagar. Contudo, não mantém informações de estado entre sessões POP3. Portanto, de certa forma diz-se que este protocolo é COM ESTADO.

Este protocolo é bastante difundido atualmente. Contudo, há desde 2003 uma proposta de atualização que recebe o nome de POP4. Porém, praticamente nenhum progresso tem sido observado na utilização desta atualização.

30.9 IMAP - Internet Mail Access Protocol

É o protocolo de CAMADA DE APLICAÇÃO do modelo OSI definido no RFC 2060. Ele é um protocolo de acesso de correio eletrônico complexo e com muitos recursos. Por padrão, ele utiliza o TCP como protocolo de transporte via porta 143.

Seu funcionamento típico se assemelha bastante com o funcionamento do POP3. Contudo, há alguns novos recursos bastante interessantes para o usuário. A principal diferença é com relação à possibilidade do usuário poder criar pastas, mover/copiar mensagens de uma pasta para outra e também realizar procurar por palavras-chaves. No POP3 todas essas funcionalidades só podem ser executadas com as mensagens na máquina local do cliente. Enquanto que o IMAP implementa comandos que permitem fazer essas manipulações com as mensagens no Servidor de Correio, o que trouxe um avanço considerável.

Uma outra característica contrastante com o POP3 é com relação à manutenção de informações de estado. O POP3 somente mantém informações de estado durante uma sessão POP3, enquanto o IMAP mantém informações tanto durante sessões quanto entre sessões IMAP. Por exemplo, os nomes das pastas e quais mensagens estão associadas a elas devem ser mantidos. Portanto, diz-se que o POP3 é um protocolo COM ESTADO.

Uma explicação sucinta do fluxo das mensagens é a seguinte. O servidor IMAP sempre associa as mensagens que chegam à pasta INBOX do destinatário, que, então, pode transferir estas mensagens para uma nova pasta criada por ele, lê-las, apagá-las e assim por diante.

Outra característica importante do IMAP é que ele tem comandos que permitem que um Agente de Usuário obtenha componentes de mensagens. Por exemplo, um Agente de Usuário pode obter apenas o cabeçalho ou somente uma das partes de uma mensagem MIME multiparte. Essa característica é útil quando há uma conexão de largura de banda estréia entre o Agente de Usuário e seu Servidor de Correio. Com uma conexão deste tipo, o usuário pode decidir não baixar todas as mensagens de sua Caixa Postal, evitando, em particular, mensagens longas que possam conter, por exemplo, arquivos grandes.

Hoje, um número cada vez maior de usuários está enviando e acessando e-mails por meio de seus browsers Web. Esse tipo de serviço é provido por praticamente todos os sites ISPs, bem como universidades e empresas importantes. Com esse serviço, o Agente de Usuário é um browser Web comum e o usuário se comunica com sua Caixa Postal, tanto enviando quanto recebendo mensagens, via HTTP, e não via os protocolos SMTP, POP3 ou IMAP. Contudo, os Servidores de Correio continuam se comunicando entre si via SMTP. Essa solução também é extremamente conveniente quando o usuário está em trânsito. Pois, como ocorre com o IMAP, usuários podem organizar suas mensagens em uma hierarquia de pastas no servidor remoto. Na verdade, muitas implementações de e-mail pela Web utilizam um servidor IMAP para prover a funcionalidade de pastas e busca. Nesse caso, o acesso às pastas e mensagens é provido por scripts que rodam em um servidor HTTP e usam o protocolo IMAP para se comunicar com um servidor IMAP.

A tabela a seguir apresenta os principais aspectos na comparação entre IMAP e POP3.

Característica	POP3	IMAP
Onde o protocolo é definido	RFC 1939	RFC 2060
Porta TCP usada	110	143
Onde as mensagens são armazenadas	PC do usuário	Servidor
Como as mensagens são lidas	Offline	Online
Tempo de conexão exigido	Pequeno	Grande
Utilização de recursos do servidor	Mínima	Intensa
Várias caixas de correio (pastas)	Não	Sim
Bom para usuário em trânsito	Não	Sim
Controle do usuário sobre o download	Pequeno	Grande
Downloads de mensagens parciais	Não	Sim
Quotas de disco consistem um problema	Não	Possível
Implementação simples	Sim	Não

Tabela 30.2: Comparação entre IMAP e POP3

30.10 LDAP - LightWeight Directory Access Protocol

O LDAP é mais um protocolo de camada de aplicação que roda sobre o protocolo TCP. O serviço LDAP segue o modelo cliente servidor, onde um ou mais servidores contêm as informações sobre os diretórios em uma rede. Um cliente LDAP se conecta ao servidor e faz um request, que é respondido pelo servidor ou então redirecionado para outro servidor. Os servidores LDAP armazenam informações como direitos de acesso, espaço disponível por usuários, entre outras. Cada um dos diretórios gerenciados pelo LDAP contém informações a seu respeito como CreatorsName, CreateTimestamp, ModifiersName, ModifiersTimestamp. As estruturas no LDAP são armazenadas de forma hierárquica e não existe limitação quanto a organização hierárquica como nos sistemas de arquivos. Pode-se, por exemplo, adicionar as estruturas partes de nomes de domínio, ou sistema operacional. As entradas no LDAP são chamadas Distinguished Names (DNs). Ao configurar um servidor utiliza-se o conceito de sufixo (suffixe) para determinar qual o nível mais alto da hierarquia está sob seu controle.

30.11 SNMP - Simple Network Management Protocol

O SNMP é um protocolo da camada de aplicação que foi desenvolvido para permitir a execução de várias funções como: (i) Configurar Dispositivos Remotos; (ii) Monitorar Performance da Rede (iii) Detectar Problemas na Rede e Acessos Indevidos; (iv) Auditar a utilização da Rede.

Os componentes do serviço de SNMP são o sistema de gerenciamento, que pode ser qualquer máquina rodando um software de gerenciamento, e os agentes que são as máquinas gerenciadas. Essas máquinas podem ser computadores pessoais, servidores, switches, roteadores, etc. O sistema de gerenciamento envia um request para um agente ou também pode enviar informações em alguns casos. O serviço de SNMP pode ser configurado para especificar quais informações devem ser informadas ao sistema de gerenciamento e quais sistemas de gerenciamento estão autorizados a requisitar informações. Em geral, são os sistemas de gerenciamento que requisitam informações dos agentes, porém os agentes podem utilizar mensagens chamadas *traps* para comunicar algo ao sistema em ocasiões especiais. Exemplos dessas situações são temperatura elevada, tentativas de invasão, altas taxas de utilização de memória ou CPU, etc.

Os sistemas de gerenciamento são organizados em *comunidades* por propósitos administrativos e segurança. Quando um sistema de gerenciamento requisita alguma informação dos agentes, os agentes recuperam as informações de uma base chamada *MIB - Management Information Base*. As mensagens do protocolo SNMP são as seguintes: (i) Get - A mensagem mais simples onde o sistema de gerencia solicita uma informação específica; (ii) Get-Next - Pesquisa por uma informação em toda a MIB; (iii) Set - Se for permitida a escrita na MIB; (iv) Getbulk - Solicita que os dados enviados pelo agente sejam o maior possível

respeitando um determinado valor e o MTU da linha; (v) Trap - Mensagem não solicitada enviada do agente para o sistema de gerenciamento. O protocolo SNMP utiliza as portas UDP/161 e UDP/162 (traps).

30.12 FTP - File Transfer Protocol

É o protocolo de CAMADA DE APLICAÇÃO do modelo OSI no RFC 959. Ele é implementado em dois programas, um cliente e outro servidor, portanto, ele é classificado como aplicação cliente-servidor.

Como segurança mínima o protocolo FTP implementa um processo de autenticação e outro de permissão. A autenticação é verificada através de um código de usuário e senha, já a permissão, é dada em nível de diretórios e arquivos.

Em uma sessão FTP típica, o usuário, sentado à frente de um hospedeiro (o local), quer transferir arquivos de ou para um hospedeiro remoto. Na realidade, um usuário também pode transferir arquivos de um hospedeiro remoto para outro hospedeiro remoto. Para acessar a conta remota, o usuário deve fornecer uma identificação e uma senha (processo de autenticação). O usuário final interage com o FTP por meio de um AGENTE DE USUÁRIO FTP. Alguns dos agentes mais utilizados atualmente são: SmartFTP; Cute FTP; FTP via Web; Filezilla; Core FTP; WS FTP; LeechFTP; e gFTP.

Assim como o HTTP, o FTP também usa o TCP como protocolo de camada de transporte. Contudo, há algumas diferenças importantes. Diferente do HTTP, o FTP usa duas conexões TCP paralelas para transferir um arquivo: uma CONEXÃO DE CONTROLE e uma CONEXÃO DE DADOS. A primeira é utilizada para enviar informações de controle, por exemplo, identificação de usuário, senha, comandos para trocar diretório remoto e comandos de inserir/pegar arquivos. A conexão de dados é usada para enviar efetivamente os dados. Como são usadas conexões distintas para controle e dados, dizemos que o FTP envia informações de controle FORA DA BANDA. Por padrão, as portas utilizadas são as 21 (conexão de controle) e 20 (conexão de dados).

Diferentemente do HTTP, o FTP precisa manter informações de estado sobre cada usuário conectado ao servidor. Em particular, o servidor deve associar a conexão de controle com uma conta de usuário específica e também deve monitorar o diretório corrente do usuário enquanto este passeia pela árvore do diretório remoto. Portanto, diz-se que o FTP é um PROTOCOLO COM ESTADO.

Uma característica bastante importante do FTP é o fato da conexão de controle permanecer aberta durante toda uma sessão enquanto é necessária uma nova conexão de dados para transferência da cada arquivo. Ou seja, se for necessária a transferência de dois arquivos entre os mesmos cliente e servidor, em uma mesma sessão FTP, somente uma conexão de controle será aberta, mas serão abertas duas conexões de dados, uma para cada arquivo.

O FTP implementa três modos de transferências. São eles:

- Modo Ativo: o cliente é responsável pela abertura da conexão de controle e o servidor é responsável pela abertura da conexão de dados. O processo acontece da seguinte forma. O cliente abre uma conexão de controle em uma porta randômica, por exemplo 1553, com o servidor que está escutando em sua porta 21. O cliente envia ao servidor, por meio do comando PORT, uma outra porta randômica, por exemplo 1500, e seu endereço IP. Então, o servidor abre uma conexão de dados com o cliente utilizando sua porta 20 e a porta 1500 do cliente;
- Modo Passivo: o cliente é responsável pela abertura de ambas as conexões. O processo acontece da seguinte forma. O cliente abre uma conexão de controle em uma porta randômica, por exemplo 1553, com o servidor que está escutando em sua porta 21. O cliente envia um comando PASV ao servidor informando que o modo de conexão será passivo. Em resposta, o servidor envia, por meio do comando PORT, uma porta randômica, por exemplo 1728, e seu endereço IP. Então, o cliente abre uma conexão de dados com o servidor utilizando uma outra porta randômica, por exemplo 1500, e a porta 1728 do servidor;
- Modo Passivo Estendido: este modo é muito similar ao modo passivo. Contudo, o servidor, ao enviar o comando PORT, transmite ao cliente somente o número da porta randômica (não transmite seu endereço IP) que estará escutando para o estabelecimento da conexão de dados. O cliente assume então que ele deverá se conectar ao mesmo endereço IP que foi originalmente conectado. Este modo foi adicionado no protocolo por meio da RFC 2428.

Cabe salientar que toda conexão randômica acontece em portas maiores que 1023 e são conexões não privilegiadas. Os processos de conexão em modo ativo e passivo são exemplificados na figura 30.4.

O FTP implementa dois tipos de conexão: a anônima e a com autenticação. A primeira é a mais utilizada, onde não é necessário o usuário informar login e senha, basta identificar-se como anonymous. Contudo, somente arquivos e diretórios públicos do servidor de arquivo remoto serão acessados. Dependendo da implementação do servidor FTP, é solicitada ao usuário a entrada de seu endereço de e-mail como campo de senha. Isto permite aos administradores algum tipo de controle. Em alguns casos o próprio servidor FTP suprime a solicitação de senha preenchendo este campo com valores padrões (ex. lftp@ e mozilla@example.com).

O protocolo FTP, assim como o HTTP, envia comandos e respostas por meio da conexão de controle no formato ASCII. Contudo, o FTP permite que o cliente especifique o formato dos dados armazenados a serem transferidos na conexão de dados. Os formatos mais utilizados são: ASCII, Binary, EBCDIC e USASCII.

A lista de comandos do FTP é de tamanho considerável. Os principais são:

- CD: permite alterar o diretório de trabalho remoto;



Figura 30.4: Modos de transmissão do FTP

- CLOSE: solicita o encerramento da sessão FTP, ou seja, o encerramento da conexão de controle;
- DELE: solicita ao servidor a deletar um arquivo no servidor remoto;
- EPSV: informa ao servidor a entrada no modo de transferência passivo estendido (não especificada na RFC 959, especificada em outra RFC);
- GET: solicita ao servidor a transferência de uma cópia de um arquivo do servidor remoto (não especificada na RFC 959, especificada em outra RFC);
- HELP: retorna a documentação sobre como usar o FTP, inclusive a lista de comandos;
- FTP: inicia uma seção FTP (inicializa o programa FTP);
- LIST: retorna a listagem de arquivos em um diretório remoto ou conteúdo de um arquivo do servidor remoto;
- OPEN: abre uma conexão com um servidor FTP, ou seja, abre uma conexão de controle servidor remoto;
- PASS: informa ao servidor a senha do usuário;
- PASV: informa ao servidor a entrada no modo de transferência passivo;
- PORT: envia um endereço IP e uma porta que serão utilizados na abertura de uma conexão futura, ou pelo servidor ou pelo cliente;
- PUT: realiza a transferência de uma cópia de um arquivo da máquina local ao servidor remoto (não especificada na RFC 959, especificada em outra RFC);
- PWD: informa o diretório de trabalho corrente no servidor remoto;

- QUIT: fecha uma seção FTP (finaliza o programa FTP);
- RETR: solicita ao servidor a transferência de uma cópia de um arquivo do servidor remoto;
- REST: solicita ao servidor o reinício da transferência de uma cópia de um arquivo do servidor remoto a partir de certo ponto;
- STOR: realiza a transferência de uma cópia de um arquivo da máquina local ao servidor remoto;
- TYPE: configura o formato de arquivo a ser transferido;
- USER: informa ao servidor remoto o nome do usuário.

No FTP, cada comando é seguido de uma resposta, que é enviada do servidor ao cliente. Assim como no HTTP, as respostas são codificadas em números de três dígitos com uma mensagem opcional após o número. As respostas mais típicas, junto com suas possíveis mensagens são as seguintes:

- 1xx: Resposta preliminar positiva. A ação requisitada se iniciou, mas haverá uma outra resposta antes de seu começo;
- 2xx: Resposta definitiva positiva. A ação requisitada se completou. Agora o cliente pode fazer outra solicitação;
- 3xx: Resposta intermediaria positiva. O comando teve sucesso, mas um outro comando é necessário antes do servidor poder executar efetivamente a solicitação feita;
- 4xx: Resposta preliminar negativa. O comando não foi bem sucedido, mas o cliente pode tentá-lo novamente;
- 5xx: Resposta definitiva negativa. O comando não foi bem sucedido e o cliente não deve solicitá-lo novamente;
- x0x: A falha se deu por motivos de erro de sintaxe;
- x1x: Resposta a uma solicitação de informação. Por exemplo, 125 - Conexão de dados já aberta, iniciando transferência;
- x2x: Resposta a uma solicitação de informação referente à conexão. Por exemplo, 425 - Não é possível abrir a conexão de dados;
- x3x: Resposta a uma solicitação de informação referente autenticação e conta de usuário. Por exemplo, 331 - Nome de usuário OK, senha requisitada;
- x4x: Ainda não especificada;
- x5x: Indica o estado do servidor de arquivo. Por exemplo, Erro ao escrever o arquivo.

Vale mencionar que o protocolo TFTP (Trivial File Transfer Protocol) é uma opção para quem não necessita da robustez do protocolo FTP. O TFTP usa o protocolo UDP/69 para fazer a entrega dos pacotes, ao contrário do protocolo FTP que usa o protocolo TCP. As principais características deste protocolo são: não permite visualização dos diretórios; não implementa autenticação de usuários; e não implementa mecanismos de criptografia. As operações de escrita/leitura são descritas pelas figuras 30.5.

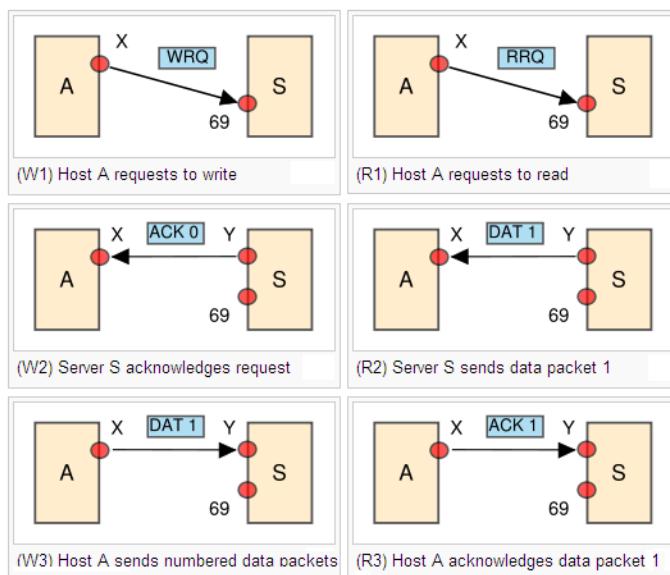


Figura 30.5: Processo de escrita/leitura do TFTP

30.13 IP - Internet Protocol

O IP (Internet protocol) é um protocolo de camada de rede que prove mecanismos de fragmentação e remontagem de unidades chamadas datagramas para a transmissão por diversas redes com diferentes unidades máximas de transferências. O protocolo IP não é orientado a conexão e não possui controle de fluxo, confirmações ou sequenciamento. O IP utiliza um esquema de endereçamento de 32 bits que é dividido em três partes que são rede, subrede e host. Os endereços IP podem ser divididos em classes de acordo com o número de bits que reservam para identificação da rede. As máscaras de sub-rede servem para identificar qual a rede e quantos host são possíveis na rede. As máscaras são compostas por diversos bits 1 seguidos de bits 0. As formas de endereçar um pacote IP são o unicast, multicast e broadcast.

O cabeçalho do IP é formado pelos seguintes campos: (1) Versão: Indica a versão utilizada. (2) IP Header Length: Indica o tamanho do cabeçalho em múltiplo de 32 bits. (3) Type of Service: Especifica a maneira com a qual os protocolos de camadas superiores querem que o pacote IP seja tratado. Os níveis

de QoS são assinalados por meio desse campo do cabeçalho. (4) Total Length: Tamanho total do datagrama (dados e cabeçalho). (5) Identification: Identifica a qual datagrama pertence um fragmento. (6) Time to Live: Contador de tempo de vida do pacote. Decrementado a cada hop. (7) Protocol: Identifica qual protocolo de camada superior está encapsulado. (8) Header Checksum: Identifica se cabeçalhos estam corrompidos. (9) Source/Destination Address. (10) Options e Padding: Suporta opções (Ex: Segurança) e Preenchimento.

Com o crescimento das redes os endereços IP estão se tornando um problema para os administradores da Internet. Apesar de matematicamente serem possíveis em torno de 4 bilhões de endereços, muitos deles não podem ser utilizados ou simplesmente são desperdiçados devido ao esquema de distribuição de endereços inadequado baseado em classes. Um outro problema decorrente é a sobrecarga das tabelas de roteamento. Para solucionar o problema surgiu o IPv6 porém até que a migração ocorra por completo serão necessários alguns anos segundo especialistas. Uma outra solução para este problema é chamada *IP Classless*. O esquema de endereçamento padrão divide os endereços nas classes A,B e C onde a diferença entre a quantidade de endereços em cada uma das classes é muito grande, fazendo com que ocorra um desperdício muito grande de endereços. O esquema *IP Classless* utiliza uma máscara de 32 bits para identificar a rede. Exemplos de máscara são:

```
11111111 11111111 11111111 00000000 255.255.255.0
11111111 11111111 00000000 00000000 255.255.0.0
11111111 11111111 11111111 11111000 255.255.255.252
11111111 11100000 00000000 00000000 255.32.0.0
```

Com esta estratégia é possível criar redes com tamanhos mais adequados às necessidades. Para identificar como um datagrama deve ser encaminhado por um roteador, por exemplo, é feito um AND lógico da máscara com o endereço IP de destino do datagrama para pesquisar por qual linha ele deve sair.

O endereço de broadcast em uma rede IP é sempre o último endereço da faixa. Por exemplo: Na rede 10.0.0.0/24, o endereço de broadcast é 10.0.0.255, enquanto em na rede 172.22.4.0/22 o endereço de broadcast seria 172.22.7.255.

30.14 TELNET - TELetype NETwork

É o protocolo de CAMADA DE APLICAÇÃO do modelo OSI definido no RFC 854 (na verdade também em diversos outros RFCs). Ele é um protocolo popular utilizado para fazer login remoto entre qualquer par de hospedeiros. Por padrão, ele utiliza o TCP como protocolo de transporte via porta 23. Na verdade, nada impede que sejam utilizadas outras portas.

Este protocolo é implementado em dois programas, um cliente e outro servidor, portanto, ele é classificado como aplicação cliente-servidor. Porém, o termo Telnet também é utilizado para se referir ao software que implementa o programa cliente do protocolo. A máquina que abre a conexão é a que pretende enviar caracteres, e portanto, é a considerada cliente. A máquina que recebe os

caracteres enviados é a considerada servidora.

As três características mais marcantes deste protocolo são:

- Todos os dados, inclusive senhas, não são criptografados (o que o torna bastante vulnerável);
- Não há autenticação para se garantir que o servidor é realmente quem ele diz que é;
- Cada caractere digitado pelo usuário (no cliente) será enviado ao hospedeiro remoto e este devolverá uma cópia (eco) do caractere digitado que será apresentado na tela do Telnet do usuário. Esse eco é utilizado para garantir que os caracteres vistos pelo usuário do Telnet já foram recebidos e processados no local remoto. Assim, cada caractere atravessa literalmente a rede duas vezes.

A figura 30.6 apresenta o cenário típico de envio de caracteres via Telnet. A seguir, são examinados os segmentos TCP que são enviados entre o cliente e o servidor deste cenário. Admite-se que os números de seqüência iniciais sejam 42 e 79 para cliente e servidor, respectivamente. Lembre-se que estes números representam os números dos segmentos aguardados cada máquina. Vamos aos segmentos:

- O primeiro é enviado do cliente ao servidor, contendo em seu campo de dados um byte com a representação ASCII para a letra C (digitada pelo usuário). Este segmento também tem 42 em seu campo de número de seqüência. E mais, como o cliente ainda não recebeu nenhum dado do servidor, esse segmento terá o número 79 (não foi incrementado) em seu campo de número de reconhecimento;
- O segundo é enviado do servidor ao cliente. Esse segmento tem dupla finalidade. A primeira é fornecer um reconhecimento para os dados que o servidor recebeu. Ao colocar 43 no campo de reconhecimento, o servidor está dizendo ao cliente que recebeu com sucesso tudo até o byte 42 e que está aguardando os bytes de 43 em diante. A segunda finalidade é ecoar a letra C. Ele tem o número de seqüência 79, que é o numero de seqüência inicial do fluxo de dados de servidor para cliente dessa conexão TCP. Note que o reconhecimento para dados do cliente para servidor é levado em um segmento que carrega dados do servidor para o cliente. Tecnicamente essa carona recebe o nome de PIGGYBACK;
- O terceiro é enviado do cliente ao servidor. Seu único propósito é reconhecer os dados que recebeu do servidor. Ele tem o campo de dados vazio, tem o numero 80 no campo de número de reconhecimento porque o cliente agora está aguardando os bytes de 80 em diante.

Vale a pena comentar que este protocolo vem perdendo bastante espaço para o protocolo SSH. Pois o SSH provê todas as funcionalidades do Telnet com a adição de uma forte criptografia de dados, inclusive senhas, e de uso de chaves públicas para a realização de autenticação. Com este tipo de autenticação, se garante que o servidor é realmente quem ele diz que é.

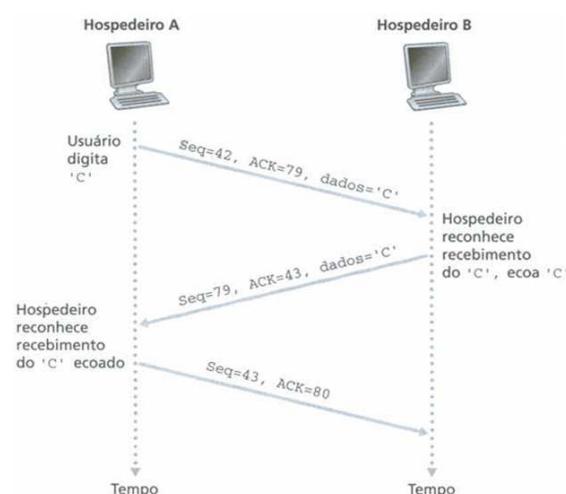


Figura 30.6: Cenário típico de envio de caracteres via Telnet

Capítulo 31

O Modelo de Referência OSI

O modelo de referência OSI (Open Systems Interconnection) foi o primeiro passo para a padronização internacional dos processos de comunicação de sistemas abertos. Sistemas abertos são aqueles que estão abertos a comunicação com outros sistemas. Os princípios aplicados para se chegar ao modelo de 7 camadas foram os seguintes:

- Uma camada deve ser criada onde houver necessidade de abstração acional;
- Cada camada deve executar tarefa bem definida;
- O limite das camadas deve minimizar o fluxo de informações sobre as interfaces.

O modelo OSI não é uma arquitetura, pois não especifica serviços e protocolos exatos de cada camada. Ele apenas indica o que cada camada deve fazer. As 7 camadas propostas no modelo OSI e suas respectivas funções são:

1. *Camada Física* - Responsável pela transmissão de bits brutos por um canal de comunicação. As questões mais comuns são representação dos bits 0 e 1, tempo de vida de um bit, permitir ou não transmissões simultâneas nos dois sentidos, pinagens, etc; (2)
2. *Camada de Enlace* - Transformar o canal de comunicação bruto em uma linha que pareça livre de erros para a camada de rede. Deve implementar mecanismos de fragmentação e montagem, controle de fluxo, tratamentos de erros e mecanismos de confirmação caso o serviço seja confiável. Para melhorar o desempenho pode-se utilizar a técnica de Pipelining. Deve possuir também mecanismos de retransmissão;
3. *Camada de Rede* - Determinar como os pacotes são roteados da origem ao destino. Evitar congestionamentos e atrasos excessivos. Deve se preocupar também com endereçamento e tamanhos de pacotes, que podem ser diferentes nas diversas redes. É papel da camada proporcionar interoperabilidade entre as redes. Deve implementar serviços orientados a conexão (circuitos virtuais) e não orientados a conexão (datagramas).

4. *Camada de Transporte* - Deve receber dados da camada acima, dividi-los em unidades menores e repassar essas unidades para a camada de rede. A camada de transporte deve assegurar que todos os dados chegarão corretamente ao destino. É a camada que realiza o controle fim-a-fim. Um processo em uma máquina há mantém uma conversação diretamente com um processo em alguma outra máquina. Entre as tarefas mais comuns realizadas pelos protocolos dessa camada estão transferência básica de dados, multiplexação, abertura de conexões, controle de fluxo e congestionamento;
5. *Camada de Sessão* - Permite que usuários de diferentes máquinas estabeleçam sessões entre eles. Os serviços são controle de diálogo (quem deve transmitir a cada momento), gerenciamento de token (impedindo que duas máquinas tentem executar sessão crítica ao mesmo tempo) e a sincronização (que permite que transmissões longas parem e reiniciem do ponto onde ocorreu interrupção);
6. *Camada de Apresentação* - Preocupa-se com a sintaxe e a semântica das informações transmitidas. O objetivo é permitir a transferência de dados entre entidades com diferentes representações de dados. Definem as estruturas de dados a serem intercambiadas juntamente com a codificação dos dados;
7. *Camada de Aplicação* - Contém os protocolos que implementam de fato as aplicações dos usuários.

Capítulo 32

Roteamento

Três importantes protocolos roteáveis são o IP, o IPX e o AppleTalk. O IP é o protocolo oficial da Internet e faz parte da pilha de protocolos TCP/IP, por isso é o mais importante. Embora a maior atenção deve ser dada no IP, é importante saber que existem outros protocolos roteáveis (IPX/SPX e o AppleTalk). Protocolos como o IP, o IPX/SPX e o AppleTalk fornecem suporte à camada 3 e são, portanto, roteáveis.

Entretanto, há protocolos que não suportam a camada 3. Eles são classificados como protocolos não-roteáveis. O mais comum desses protocolos não-roteáveis é o NetBEUI. O NetBEUI é um protocolo pequeno, rápido e eficiente, cuja execução se limita a um segmento. Para um protocolo ser roteável, ele deve propiciar a habilidade de atribuir um número de rede, assim como um número de host, a cada dispositivo individual. Alguns protocolos, como o IPX, exigem apenas que você atribua um número de rede, porque usam um endereço MAC de host para o número físico. Outros protocolos, como o IP, exigem que você forneça um endereço completo, além de uma máscara de sub-rede. O endereço de rede é obtido pela operação AND do endereço com a máscara de sub-rede.

Os protocolos de roteamento determinam os caminhos que os protocolos roteados seguem para seus destinos. Exemplos de protocolos de roteamento incluem o Routing Information Protocol (RIP), o Interior Gateway Routing Protocol (IGRP), o Enhanced Interior Gateway Routing Protocol (EIGRP) e o Open Shortest Path First (OSPF). Os protocolos de roteamento permitem que os roteadores conectados criem internamente um mapa de outros roteadores na rede ou na Internet. Isso permite o roteamento (ou seja, seleção do melhor caminho e comutação). Tais mapas tornam-se parte da tabela de roteamento de cada roteador.

Os dois tipos de protocolos de roteamento são os Exterior Gateway Protocols (EGPs) e os Interior Gateway Protocols (IGPs). Os Exterior Gateway Protocols roteiam os dados entre sistemas autônomos. Um exemplo de EGP é o BGP (Border Gateway Protocol), o principal protocolo de roteamento externo da Internet. Os protocolos RIP, OSPF, IGRP e EIGRP são exemplos de IGPs.

O administrador de rede pode inserir as informações manualmente no roteador.

Os roteadores podem conhecer as informações uns dos outros durante o processo. As entradas manuais nas tabelas de roteamento são chamadas de *rotas estáticas*. As rotas descobertas automaticamente são chamadas de *rotas dinâmicas*. Quando um algoritmo de roteamento atualiza uma tabela de roteamento, seu principal objetivo é determinar as melhores informações para incluir na tabela. Cada algoritmo de roteamento interpreta à sua maneira o que é melhor. O algoritmo gera um número, chamado valor métrico, para cada caminho através da rede. Geralmente, quanto menor o número da métrica, melhor é o caminho. Você pode calcular a métrica com base em uma única característica do caminho; você pode calcular métricas mais complexas combinando várias características. As métricas comumente usadas pelos roteadores são as seguintes:

- Largura de banda - a capacidade de dados de um link;
- Atraso - o tempo necessário para mover um pacote por cada link, da origem até o destino;
- Carga - a quantidade de atividade em um recurso de rede, como um roteador ou um link;
- Confiabilidade - geralmente se refere à taxa de erros de cada link da rede;
- Número de hops - o número de roteadores através dos quais um pacote deve trafegar antes de chegar ao seu destino;
- Pulses (ticks) - o atraso em um enlace de dados que usa os pulsos de clock do PC IBM (aproximadamente 55 milissegundos);
- Custo - um valor arbitrário, geralmente baseado na largura de banda, despesas monetárias ou outras medidas, atribuído por um administrador de rede.

32.1 Link State e Distance Vector

A maioria dos algoritmos de roteamento pode ser classificada como um dos dois algoritmos básicos:

- Vetor de Distâncias (*Distance Vector*);
- Estado do Link (*Link State*).

A abordagem do roteamento de vetores de distância determina a direção (vetor) e a distância de todos os links na internetwork. A abordagem do link state (também chamado de *shortest path first*) recria a topologia exata da internetwork inteira (ou de pelo menos da parte onde o roteador está situado). A abordagem híbrida balanceada combina aspectos dos algoritmos do link state e do vetor de distância. A seguir são apresentados os procedimentos e os problemas de cada um desses algoritmos de roteamento e apresentam técnicas para minimizar os problemas.

Cada roteador que usa roteamento de vetor de distância começa identificando seus próprios vizinhos. À medida que o processo de exploração de rede de vetor

de distância prossegue, os roteadores descobrem o melhor caminho para as redes de destino, com base nas informações que recebem de cada vizinho. Quando a topologia em uma rede de protocolo de vetor de distância é alterada, devem ocorrer atualizações na tabela de roteamento. Da mesma forma que acontece com o processo de exploração da rede, as atualizações das alterações na topologia prosseguem passo a passo, de roteador para roteador. Os algoritmos de vetor de distância solicitam que cada roteador envie **toda a sua tabela** de roteamento para cada um dos vizinhos adjacentes. As tabelas de roteamento incluem informações sobre o custo total do caminho (definido pela sua métrica) e o endereço lógico do primeiro roteador no caminho para cada rede contida na tabela.

Loops de roteamento podem ocorrer se a convergência lenta de uma rede em uma configuração nova provoca entradas de roteamento inconsistentes. Os algoritmos de roteamento de vetores de distância são autocorrigíveis, mas um problema de loop de roteamento pode exigir primeiro uma contagem até o infinito. Para evitar esse problema prolongado, os protocolos de vetores de distância definem o infinito como um número máximo específico. Esse número se refere a uma métrica de roteamento (por exemplo, um contador de saltos simples).

O segundo algoritmo básico usado para roteamento é o algoritmo de link state. Os algoritmos de roteamento baseados em link state, também conhecidos como algoritmos SPF (shortest path first - primeiro caminho mais curto), mantêm um banco de dados complexo de informações sobre a topologia. Enquanto o algoritmo de vetor de distância tem informações não específicas sobre redes distantes e nenhum conhecimento sobre roteadores distantes, um algoritmo de roteamento de link state mantém conhecimento completo sobre roteadores distantes e de como estão interconectados. O roteamento de link state usa:

- LSAs (Link-State Advertisements - aviso de estado do link);
- Um banco de dados topológico;
- O algoritmo SPF e a árvore SPF resultante;
- Uma tabela de roteamento de caminhos e portas para cada rede.

Os engenheiros implementaram esse conceito de link state no roteamento OSPF (Open Shortest Path First). Sempre que uma topologia de link state é alterada, os roteadores que primeiro tomam conhecimento da alteração enviam informações para outros roteadores ou para um roteador designado, as quais todos os outros roteadores podem usar para atualizações. Isso envolve o envio de informações comuns de roteamento a todos os roteadores na internetwork. Existem duas questões relacionadas ao link state:

- Requisitos de processamento e memória: Executar protocolos de roteamento de link state, na maior parte das situações, requer que os roteadores usem mais memória e executem mais processamento que os protocolos de roteamento de vetor de distância. Para roteamento de link state, a memória deve ser capaz de reter informações de diversos bancos de dados, da árvore de topologia e da tabela de roteamento. Usar o algoritmo de *Dijkstra* para calcular o SPF requer uma tarefa de processamento proporcional ao número de links na internetwork, multiplicado pelo número de roteadores na internetwork;

- Requisitos de largura de banda: consumida para a sobrecarga inicial do pacote de link state. Durante o processo inicial de exploração, todos os roteadores usando protocolos de roteamento de link state enviam pacotes LSA para todos os outros roteadores. Essa ação sobrecarrega a internetwork à medida que os roteadores fazem uma demanda em massa de largura de banda e reduzem temporariamente a largura de banda disponível para o tráfego roteado que transporta dados de usuários. Depois dessa sobrecarga inicial, os protocolos de roteamento de link state requerem geralmente apenas uma largura de banda mínima para enviar pacotes LSA raros ou desencadeados por eventos que refletem alterações na topologia.

32.1.1 Vetor de Distâncias vs. Estado do Link

Você pode comparar o roteamento de vetor de distância com o roteamento de link state em diversas áreas-chave:

- O roteamento de vetor de distância obtém dados topológicos das informações da tabela de roteamento dos vizinhos. O roteamento de link state obtém uma ampla visão de toda a topologia da internetwork acumulando todos os LSAs necessários;
- O roteamento de vetor de distância determina o melhor caminho, adicionando ao valor métrico recebido à medida que informações de roteamento são passadas de roteador para roteador. Para o roteamento de link state, cada roteador opera separadamente para calcular o seu caminho mais curto para as redes de destino;
- Com a maior parte de protocolos de roteamento de vetor de distância, atualizações de alterações na topologia chegam em atualizações periódicas de tabelas. As informações passam de roteador para roteador, geralmente resultando em uma convergência mais lenta. Com protocolos de roteamento de link state, as atualizações são normalmente desencadeadas por alterações na topologia. LSAs relativamente pequenos passados para todos os outros roteadores resultam geralmente em um tempo de convergência mais rápido em qualquer alteração na topologia da internetwork.

Os protocolos de roteamento híbrido balanceado usam vetores de distância com métricas mais precisas para determinar os melhores caminhos até as redes de destino. Entretanto, eles diferem da maior parte dos protocolos de vetores de distância porque utilizam alterações na topologia para desencadear atualizações de bancos de dados de roteamento. O protocolo de roteamento híbrido balanceado converge rapidamente, como os protocolos de link state. Entretanto, ele difere dos protocolos de vetor de distância e de link state porque usa menos recursos, como largura de banda, memória e sobrecarga de processador. Exemplos de protocolos híbridos são o IS-IS (Intermediate System-to-Intermediate System) da OSI e EIGRP (Enhanced Interior Gateway Routing Protocol) da Cisco.

32.2 Protocolos de Roteamento

32.2.1 RIP - Routing Information Protocol

O protocolo mais comumente usado para transferir informações de roteamento entre roteadores localizados na mesma rede é o Routing Information Protocol (RIP). Esse Interior Gateway Protocol (IGP) calcula a distância para um host de destino em termos do número de hops. O RIP permite aos roteadores atualizar suas tabelas de roteamento em intervalos programáveis, normalmente a cada 30 segundos. Uma desvantagem dos roteadores que usam RIP é o fato deles estarem constantemente se conectando aos roteadores vizinhos para atualizar as suas tabelas de roteamento, criando assim uma grande quantidade de tráfego de rede. O RIP é um protocolo de vetor distância. Como o contador de saltos é a única medida de roteamento usada pelo RIP, ele não seleciona necessariamente o caminho mais rápido para um destino.

Quando se usa o RIP, o número máximo de saltos pelos quais os dados podem ser encaminhados é 15. Alguns avanços foram introduzidos na nova versão do RIP, chamada de RIP2. Esta nova versão trata da questão de VLSMs (máscara de tamanho variável), autenticação, e atualizações de roteamento simultâneas (multicast). RIP2 não apresenta avanços expressivos em relação ao RIP porque ainda apresenta limitações na contagem de hops e lenta convergência, o que é essencial nas grandes redes atuais.

32.2.2 OSPF - Open Shortest Path First

Uma descrição pode ser determinação de um caminho ótimo, pois esse Interior Gateway Protocol realmente usa vários critérios para determinar a melhor rota para um destino. Esses critérios incluem as medidas de custo, que são subdivididas em itens como a velocidade de rota, o tráfego, a confiabilidade e a segurança.

Há duas características principais no OSPF. A primeira, é um protocolo de padrão aberto,. A segunda, é um protocolo baseado no algoritmo SPF (link-state), também chamado de algoritmo de Dijkstra. O OSPF utiliza o conceito de roteamento hierárquico.

OSPF apresenta algumas características importantes que são:

- Não há limite na contagem de hops;
- O uso eficiente de VLSM é muito útil na alocação de endereços IP;
- OSPF usa multicast de IP para enviar atualizações de link-state. Isto garante menor processamento nos roteadores que não estão escutando os pacotes OSPF;
- Apresenta melhor convergência que o RIP;
- Permite um melhor balanceamento de carga;.
- Permite uma divisão lógica da rede, onde roteadores podem ser divididos em **áreas**. Isto limita a explosão de atualizações de link state por toda a

rede. Também fornece um mecanismo para agregar roteadores a limita a propagação desnecessária de informações de subrede;

- Permite a autenticação de rota utilizando diferentes métodos para a autenticação de senha;
- Permite a transferência e marcação de rotas externas inseridas em um Sistema Autônomo-AS. Isto rastreia rotas externas inseridas por protocolos externos como o BGP.

32.2.3 IGRP e EIGRP

O IGRP(Interior Gateway Routing Protocol) e o EIGRP(Enhanced Interior Gateway Routing Protocol) são protocolos de roteamento desenvolvidos pela Cisco Systems, Inc. e, portanto, são considerados protocolos de roteamento proprietários. O IGRP foi desenvolvido especificamente para tratar problemas associados ao roteamento em grandes redes de vários fabricantes, que estivessem além do escopo de protocolos como o RIP. Como o RIP, o IGRP é um protocolo de vetor de distância. Entretanto, ao determinar o melhor caminho, ele também leva em consideração itens como largura de banda, carga, atraso e confiabilidade. Os administradores de rede podem determinar a importância dada a qualquer uma dessas métricas ou permitir que o IGRP calcule automaticamente o melhor caminho. O EIGRP é uma versão avançada do IGRP. Especificamente, o EIGRP fornece eficiência operacional superior e une as vantagens dos protocolos de Link State com as dos protocolos de vetor de distância.

Capítulo 33

Redes Ethernet

A rede Ethernet é um padrão de rede local que foi padronizado pelo comitê IEEE 802 como IEEE 802.3. As redes Ethernet vêm sendo, desde 1990, as mais utilizadas para implementação de LAN's.

A Ethernet foi originalmente baseada na idéia de comunicar computadores a partir de um cabo coaxial compartilhado que funcionava como um meio de transmissão em broadcast. Dessa forma, os computadores disputam o meio para realizar uma transmissão e podem ocorrer colisões. Para controlar o acesso ao meio de transmissão e as colisões, as redes Ethernet utilizam um protocolo chamado CSMA/CD (*Carrier Sense for Multiple Access with Collision Detection*), que será detalhado mais adiante.

No entanto, essa configuração era problemática na medida que problemas em qualquer parte do cabo interrompiam a comunicação em toda a rede. Para contornar esse problema, foram introduzidos os hubs. Porém, em termos do domínio de colisão, não há diferenças entre um hub e um cabo único compartilhado, o que impõe restrições práticas ao tamanho da rede. A solução para esse problema é a utilização de switches, que limitam o domínio de colisão à um enlace ponto-a-ponto.

33.1 Protocolo CSMA/CD

O CSMA/CD é um protocolo que permite acesso múltiplo ao meio de transmissão baseando-se em três mecanismos básicos que são: (i) escuta da portadora; (ii) detecção de colisões e (iii) mecanismo de contenção. O protocolo funciona da seguinte maneira:

1. Um adaptador pode começar a transmitir a qualquer tempo, ou seja, não são utilizados compartimentos *slots*;
2. Um adaptador nunca transmite um quadro quando percebe que algum outro adaptador está transmitindo. Para isso, um adaptador que deseja transmitir precisa escutar a portadora;

3. Um adaptador que está transmitindo aborta sua transmissão quando percebe que algum outro adaptador está transmitindo, ou seja, detecta colisões. Além disso, ao detectar uma colisão, um adaptador que está transmitindo também emite um sinal de reforço de 48 bits (*jam signal*) para garantir que os demais adaptadores também tomaram conhecimento da colisão;
4. Após abortar a transmissão o adaptador entra em fase de contenção, durante a qual terá que esperar um tempo aleatório antes de tentar transmitir novamente. Esse tempo é determinado pelo algoritmo de backoff exponencial.

O algoritmo de backoff exponencial diz que, após a detecção da n -ésima colisão, um adaptador deverá esperar por um tempo igual a $512Kt_0$, onde t_0 é o tempo de transmissão de um bit e K é um valor escolhido com igual probabilidade entre o conjunto $\{0, 1, 2, \dots, 2^m - 1\}$, sendo m igual ao mínimo entre n e 10.

Dessa forma, após um adaptador detectar uma colisão pela primeira vez ($n = 1$) escolherá um valor de K no conjunto $\{0, 1\}$. No caso de uma rede de 10Mbps, o tempo de transmissão de um bit é igual a 0.1 microsegundos e supondo que o K escolhido seja 1, então o adaptador terá que esperar por um tempo igual a 51.2 microsegundos antes de uma nova tentativa de transmissão. Após a segunda detecção de colisão, o valor de K será escolhido entre 0, 1, 2, 3. A partir da décima colisão o valor de K será escolhido no conjunto $\{0, 1, 2, 4, \dots, 1023\}$.

33.2 Fast Ethernet

O Padrão Fast Ethernet (IEEE 802.3u) foi lançado em 1995 como a evolução do padrão Ethernet original. As principais modificações introduzidas foram o aumento da velocidade de transmissão de 10 para 100 Mbps, a possibilidade de transmissão no modo full-duplex. O protocolo de acesso ao meio utilizado é o CSMA/CD, assim como no padrão Ethernet. No modo full-duplex, um adaptador está apto a receber e transmitir a qualquer momento e a utilização do CSMA/CD é dispensada, na medida em que não existem mais colisões nem disputa pelo meio.

No padrão Fast Ethernet também foi introduzido um mecanismo de controle de fluxo. Quando um receptor deseja que o transmissor interrompa a transmissão por um tempo, ele envia um quadro especial chamado *pause frame*, que indica o tempo que o transmissor deve esperar antes de continuar com a transmissão (*time-to-wait*).

O padrão Fast Ethernet é subdividido em sub-padrões de acordo com o meio de transmissão utilizado. As principais sub-divisões do padrão Fast-Ethernet são as seguintes:

- 100Base-TX: É o sub-padrão mais utilizado para redes Fast Ethernet. Ele funciona sobre cabos UTP de categoria 5 ou superior, embora utilize apenas os pares laranja e verde do cabo de acordo com os padrões de terminação TIA/EIA-568A e TIA/EIA-568B. Cada enlace pode ter no

máximo 100 metros de comprimento. O esquema de codificação utilizado no padrão 100Base-TX é o MLT-3;

- 100Base-FX: Essa é uma versão que utiliza um par de fibras ópticas multimodo com terminadores do tipo SC, ST ou MIC. O esquema de codificação utilizado é o NRZI;
- 100Base-SX: Assim como o padrão 100Base-FX, também utiliza um par de fibras ópticas multimodo. O padrão SX difere do FX por utilizar óptica de ondas de comprimento curto (*short wavelength*), o que o torna mais acessível. No entanto, o padrão FX permite alcançar distâncias maiores;
- 100Base-BX: Transmissão e recebimento são realizados por um único cabo de fibra, utiliza para isso, técnicas de multiplexação por comprimento de onda.

33.3 Gigabit Ethernet

Em 1999, o padrão Gigabit Ethernet (IEEE 802.3z) estendeu a velocidade das redes Ethernet de 100 Mbps para 1 Gbps. Assim como o Fast-Ethernet, o padrão Gigabit permite operação nos modos half e full-duplex e utiliza o mecanismo de controle de fluxo baseado em *pause frames*. No modo half-duplex ainda é utilizado o protocolo CSMA/CD.

No entanto, com aumento da velocidade de transmissão, para garantir que as colisões sejam corretamente detecadas pelo CSMA/CD, foi necessário extender o slot-time mínimo de 64 para 512 bytes. Para cumprir este requisito é utilizado o campo *carrier extension*, que é preenchido até que o slot-time de um quadro pequeno alcance 512 bytes. No receptor, o conteúdo do campo de extensão é descartado. A diferença entre o quadro Ethernet original e o quadro do padrão Gigabit é mostrada na figura 33.3.

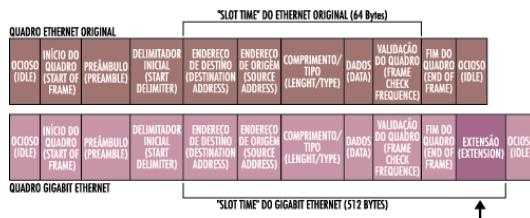


Figura 33.1: Quadros Ethernet e Gigabit Ethernet

Utilizada puramente, esta modificação afeta o desempenho na transmissão de quadros pequenos devido a grande quantidade de carga inútil transmitida. Para contornar esse problema, o padrão Gigabit também utiliza o recurso de transmissão em rajadas (*frame bursting*), permitindo que um conjunto de pacotes pequenos sejam transmitidos em conjunto dentro de um único quadro.

Assim como o padrão Fast Ethernet, o padrão Gigabit também tem suas sub-divisões. As principais são:

- 1000Base-SX: Este padrão opera utilizando óptica de ondas de comprimento curto sobre um par de fibras multimodo. Com fibras ópticas de qualidade já é possível alcançar distâncias maiores que 500 metros com esse padrão;
- 1000Base-LX: Opera utilizando óptica de ondas de comprimento longo, permitindo alcançar distâncias de até 20 quilômetros com fibras monomodo. Quando utilizado com fibras multi-modo permite distâncias de até 500 metros;
- 1000Base-T: É o padrão para redes gigabit que utiliza cabos UTP. O requisito mínimo do padrão é um cabo da categoria 5, porém, cabos da categoria 5e ou 6 são recomendados. A distância máxima alcançada com este padrão é de 100 metros;
- 1000Base-CX: Criado como uma alternativa para distâncias de até 25 metros. Ao invés de fibra óptica, utiliza cabos twiaxiais. Na prática são pouco utilizados, uma vez que o padrão 1000Base-T usa cabos UTP, que são mais baratos e podem desempenhar a mesma função;
- 1000BASE-ZX e 1000BASE-LH: Esses dois padrões utilizam transmissão em fibra monomodo, usando um comprimento de onda de 1550 nanômetros para alcançar distâncias de até 70 quilômetros.

Capítulo 34

Cabeamento Estruturado

34.1 Par Trançado

O cabeamento por par trançado (Twisted pair) é um tipo de fiação na qual dois condutores são enrolados ao redor dos outros para cancelar interferências magnéticas de fontes externas e interferências mútuas (crosstalk) entre cabos vizinhos. A taxa de giro (normalmente definida em termos de giros por metro) é parte da especificação de certo tipo de cabo. Quanto maior o número de giros, mais o ruído é cancelado. Foi um sistema originalmente produzido para transmissão telefônica analógica. Utilizando o sistema de transmissão por par de fios aproveita-se esta tecnologia que já é tradicional por causa do seu tempo de uso e do grande número de linhas instaladas.

Existem dois tipos de cabos par trançado:

- Unshielded Twisted Pair - UTP (cabo sem blindagem): São quatro pares de fios entrelaçados e revestidos por uma capa de PVC é o mais usado atualmente e mais barato.
- Shield Twisted Pair - STP (cabo com blindagem): É igual ao UTP a diferença é que possui uma blindagem feita com a malha do cabo, que o protege mais que o UTP. Porém é mais caro, menos usado e necessita de aterramento. Este gênero de cabo, por estar revestido diminui as interferências eletromagnéticas externas, protege mais da umidade, etc.

Considerando que o fator principal para determinar o alcance máximo possível de um sistema é a atenuação do sinal ao longo do cabo, foi necessário estabelecer alguns modos de classificação para o cabeamento em par metálico e o respectivo hardware de conexão. Criou-se então a subdivisão em uma série de categorias e classes por capacidades de desempenho. Nessa classificação, uma categoria ou classe de desempenho superior do cabo significa maior eficiência e uma menor atenuação. Dentre as categorias destacam-se a 5e e a 6.

34.1.1 Interferências nos Cabos de Par Trançado

O Crosstalk não ocorre apenas no par adjacente (pair to pair NEXT), mas em todos os outros pares de um cabo UTP podem interferir com seus próprios níveis

em ambas as extremidades do cabo, multiplicando o efeito dessa interferência sobre o par transmissor ou receptor.

Em razão destes níveis de interferência poderem debilitar redes de alta velocidade, alguns fabricantes de cabos começaram a apresentar as taxas de NEXT, FEXT, PSNEXT, ELFEXT e PS-ELFEXT para seus cabos CAT5e e Categoria 6. A Figura ?? ilustra essas interferências

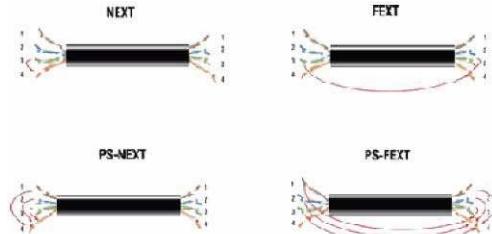


Figura 34.1: Interferências entre fios em um cabo de par trançado

O NEXT (Near-End Crosstalk) é a interferência no sinal de um par sobre um outro na mesma extremidade do cabo.

O PS-NEXT é uma medida de crosstalk mais rigorosa que inclui a soma total de todas as interferências que podem ocorrer entre um par e todos os pares adjacentes de um cabo.

O FEXT mede a interferência de um par em uma extremidade do cabo em outro par na outra extremidade do cabo. Essa medida utiliza operação full duplex para detectar onde os sinais são gerados simultaneamente em ambas as extremidades.

O ELFEXT (Equal-Level Far-End Crosstalk) mede o FEXT em relação ao nível do sinal recebido medido no mesmo par. Ele mede basicamente a interferência sem os efeitos da atenuação.

O PS-ELFEXT, uma medida comum em progressão, mede a soma total de todas as interferências dos pares de uma extremidade em um par da outra extremidade sem os efeitos da atenuação.

34.2 Categorias 5e

A letra "e" no nome da categoria significa Enhanced, ou seja, a categoria 5e representa uma melhoria das características dos materiais utilizados na categoria 5, permitindo um melhor desempenho. Pode ser usado para freqüências de até 100MHz em redes 1000BASE-T e 1000BASE-TX. Ele é igual a categoria 5, porém com especificações adicionais como os parâmetros PS NEXT, Balanço, PS ELFEXT, Return Loss.

O cabeamento Categoria 5 é normalmente direcionado para o mercado residencial, mas sua utilização vem caindo devido ao seu custo ser praticamente o mesmo da Categoria 5e. Reforçando essa afirmativa, nos projetos atuais de infra-estrutura é recomendada a utilização de cabeamento de, no mínimo, Categoria 5e para pequenas redes com poucos serviços ou que tenham caráter provisório e Categoria 6 para as redes novas ou de maior porte.

34.3 Categoria 6

A Categoria 6 pode ser vista como um aperfeiçoamento no projeto de infra-estrutura das redes locais. Ela segue seus predecessores, as categorias 3, 4, 5 e 5e, cada uma provendo maior capacidade de transporte de informação para usuários finais. Torna-se uma opção que oferece alta performance para a distribuição horizontal em um sistema estruturado, permitindo suporte para aplicações como voz tradicional (telefone analógico ou digital), VoIP, Ethernet (10Base-T), Fast Ethernet (100Base-TX) e Gigabit Ethernet a 4 pares (1000Base-T), com melhor performance em relação a Categoria 5e. Ela permite ainda suporte para aplicações ATM e novas tecnologias como Ethernet a 10Gbps sem investimentos adicionais na infra-estrutura existente.

Os sistemas Categoria 6 foram projetados para atender basicamente os seguintes objetivos:

- manter boa relação custo x benefício dos sistemas UTP, bem como facilitar sua instalação e operação;
- garantir a interoperabilidade com os atuais sistemas Categoria 5e;
- proporcionar uma nova infra-estrutura com capacidade para serviços futuros (redes de próxima geração).

34.4 Categoria 5e vs. Categoria 6

A principal diferença entre a Categoria 5e e a Categoria 6 está na performance de transmissão e na largura de banda estendida de 100MHz da Categoria 5e para 250MHz da Categoria 6. A largura de banda é a medida da faixa de freqüência que o sinal de informação ocupa. O termo é também usado em referência às características de resposta em freqüência de um sistema comunicação. No sentido mais qualitativo, a largura de banda é proporcional à complexidade dos dados transmitidos. Já a performance se traduz em uma menor atenuação, melhor NEXT, perda de retorno e ELFEXT, possibilitando uma melhor relação sinal/ruído.

Devido a esses fatores (performance e largura de banda), associando uma melhor imunidade às interferências externas, os sistemas que operam em Categoria 6 são mais estáveis em relação aos sistemas baseados na Categoria 5e. Isto significa redução nas retransmissões de pacotes, proporcionando uma maior confiabilidade e estabilidade para a rede.

Outro fato que deve ser considerado é que os requisitos para o link (meio de transmissão entre dois pontos, não incluindo a conexão de equipamentos) e canal (meio de transmissão fim-a-fim entre dois pontos no qual existem equipamentos de aplicações específicos conectados) na Categoria 6 são compatíveis com os da Categoria 5e, fazendo com que os projetistas escolham a Categoria 6, substituindo as redes Categoria 5e. A tabela 34.2 resumo as principais informações sobre as categorias de cabos UTP.

ISO	EIA/TIA	Utilização
	Cat 1	Serviços telefônicos e dados de baixa velocidade
	Cat 2	RDSI e circuitos T1/E1 - 1.536 Mbps/2.048 Mbps
Classe C	Cat 3	Dados até 16 MHz, incluindo 10Base-T e 100Base-T
Classe B	Cat 4	Dados até 20 MHz, incluindo Token-Ring e 100B-T (extinto)
Classe D	Cat 5	Dados até 100 MHz, incluindo 100Base-T4 e 100Base-TX (extinto)
	Cat 5e	Dados até 100 MHz, incluindo 1000Base-T e 1000Base-TX
Classe E	Cat 6	Dados até 200/250 MHz, incluindo 1000Base-T e 1000Base-TX
Classe F	Cat 7	Dados até 500/600 MHz

Figura 34.2: Principais Características do cabeamento UTP

34.5 Cabeação Estruturada – Norma EIA/TIA 568

Pode-se definir a cabeação estruturado como um sistema baseado na padronização das interfaces e meios de transmissão, de modo a tornar o cabeamento independente da aplicação e do layout. O cabeamento estruturado descreve ainda os sistemas de rede interna e de campus e sua interconexão com a planta externa.

34.5.1 Sistemas de Cabeamento Estruturado

Reconhecendo a necessidade de padronizar o Sistema de Cabeação Estruturado diversos profissionais, fabricantes, consultores e usuários reuniram-se sob a orientação de organizações como ISO/IEC, TIA/EIA, CSA, ANSI, BICSI e outras para desenvolver normas que garantissem a implementação do conceito do mesmo. Apesar deste trabalho resultar em diversas normas a mais conhecida no Brasil é a ANSI/TIA/EIA 568-A (versão revisada da ANSI/TIA/EIA 568 que inclui as especificações para cabeação categoria 4 e 5)

O conceito de Sistema de Cabeação Estruturada baseia-se na disposição de uma rede de cabos, com integração de serviços de dados e voz, que facilmente pode ser redirecionada por caminhos diferentes, no mesmo complexo de cabeação, para prover um caminho de transmissão entre pontos da rede distintos. Um Sistema de Cabeação Estruturada EIA/TIA 568A é formado por seis subsistemas conforme ilustrado na figura 34.3 e descritos a seguir.

- Entrada do Edifício;
- Sala de Equipamentos;
- Cabeação do Backbone;
- Armário de Telecomunicações;
- Cabeação Horizontal;
- Área de Trabalho.

As instalações de entrada no edifício fornecem o ponto no qual é feita a interface entre a cabeação externa e a cabeação intra-edifício e consistem de cabos,

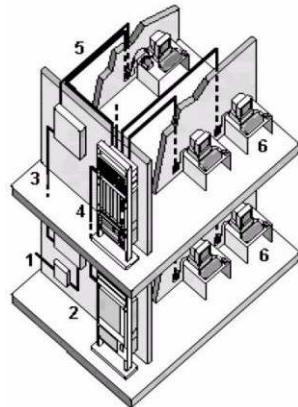


Figura 34.3: Sistema de Cabeação Estruturada

equipamentos de conexão, dispositivos de proteção, equipamentos de transição e outros equipamentos necessários para conectar as instalações externas ao sistema de cabos locais.

As salas de equipamento geralmente alojam equipamentos de maior complexidade que os do armário de telecomunicações. Qualquer uma ou todas das funções de um armário de telecomunicações podem ser atendidas por uma sala de equipamentos. Ela também contém a conexão cruzada principal ou a conexão secundária, usada conforme a hierarquia do sistema de cabeação backbone.

O subsistema de cabeamento backbone, também chamado de cabeamento vertical, propicia a interligação entre os armários de telecomunicações, salas de equipamento e instalações de entrada. Ele consiste dos cabos de Backbone, cross-connects intermediário e principal, terminações mecânicas e cabos de conexão ou de jumper utilizados para a ligação de backbone para backbone. Isto inclui:

- Ligação vertical entre os pisos (subidas ou risers)
- Cabos entre a sala de equipamentos e o local das instalações de entrada dos cabos no prédio
- Cabos entre os prédios (inter-prédios)

Os cabos homologados na norma EIA/TIA 568A para utilização como Backbone são mostrados na tabela 34.1:

Tipos	Distâncias Máximas
Cabo UTP de 100 ohms (22 ou 24 AWG)	800 metros(2625 pés) Voz
Cabo STP (par trançado blindado) de 150 ohms	90 metros(295 pés) Dados*
Fibra Óptica Multimodo de 62,5/125 μ m	2000 metros (6560 pés)
Fibra Óptica Monomodo de 8,5/125 μ m	3000 metros (9840 pés)

Tabela 34.1: Cabos homologados para cabeação Backbone e as respectivas distâncias máximas

Vale ressaltar que o alcance do Backbone depende da aplicação. As distâncias máximas especificadas acima são baseadas na transmissão de voz em UTP e de dados em STP e fibras ópticas. A distância de 90 metros para STP dá-se para aplicações com um espectro de largura de banda de transmissão de 20 a 300 MHz. Esta mesma distância também se aplica ao UTP para espectros com largura de banda de 5 a 16 MHz para Cat 3, 10 a 20 MHz para Cat 4 e de 20 a 100 MHz para Cat 5.

Outros requisitos de projeto são:

- Topologia em estrela
- Não possuir mais do que dois níveis hierárquicos de cross-connects
- Não são permitidos Bridge Taps
- Os cabos de conexão ou de jumper no cross-connect principal ou intermediário não podem exceder 20 metros (66 pés)
- Evitar a instalação em áreas onde existam fontes de interferências eletromagnéticas ou de rádio freqüência.

As figuras 34.4 e 34.5 representam o modelo geral de cabeamento backbone em topologia estrela e as respectivas configurações limites.

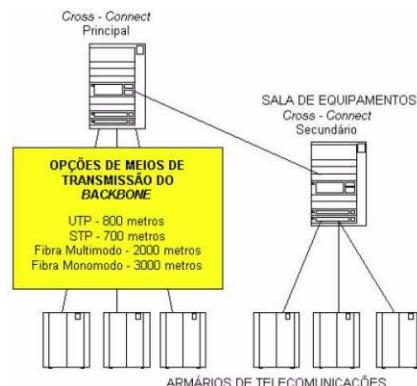


Figura 34.4: Cabeamento Backbone em topologia estrela

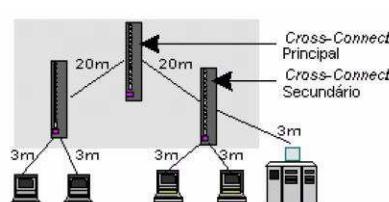


Figura 34.5: Estrutura geral e as configurações limites para o subsistema de cabeamento Backbone

O armário de telecomunicações é a área dentro de um edifício que aloja o equipamento do sistema de cabeamento de telecomunicações. Inclui as terminações mecânicas e/ou cross-connects para o sistema de cabeamento horizontal e backbone.

O subsistema de Cabeação Horizontal compreende os cabos que vão desde a Tomada de Telecomunicações da Área de Trabalho até o Armário de Telecomunicações. O subsistema de Cabeação Horizontal possui os seguintes elementos:

- Cabeação Horizontal;
- Tomada de Telecomunicações, também chamado Saída de Informação;
- Terminações de Cabo;
- Cross-Connections.

Existem três tipos de meios de transmissão a serem considerados como opções para o cabeamento horizontal, todos para a distância máxima de 90 metros:

- Cabo UTP de 4-pares, 100 ohms (condutores sólidos de 24 AWG)
- Cabo STP de 2-pares, 150 ohms
- Cabo de Fibra Óptica de 2-fibras, 62,5/125 μ m

Atualmente o cabo coaxial de 50 ohms é reconhecido como um meio de transmissão. Entretanto, não recomenda-se a sua utilização em instalações novas. Espera-se a remoção deste meio na próxima revisão desta norma.

A figura 34.6 apresenta as distâncias limites para o subsistema de cabeação horizontal bem como para o subsistema da área de trabalho que será apresentado em seguida.

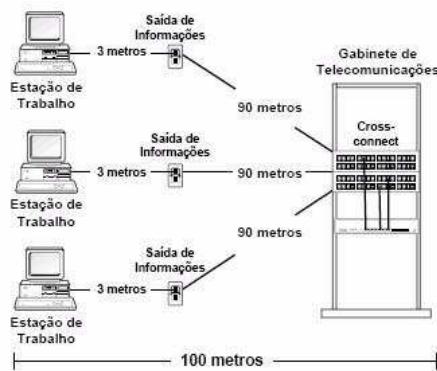


Figura 34.6: Distâncias limite para os subsistemas de cabeação horizontal e área de trabalho

A norma prevê 100 metros total para a Cabeação Horizontal: 90 metros entre o Armário de Telecomunicações e as Tomadas de Telecomunicações (conectores de parede); 10 metros para cabos entre uma estação de trabalho e o conector

de parede, (em geral, 3 metros) mais as conexões internas do Armário de Telecomunicações e entre este e os equipamentos ativos (7 metros restantes). Em complemento, cada área de trabalho deve ter no mínimo DUAS posições de saída de informação: uma para voz e outra para dados.

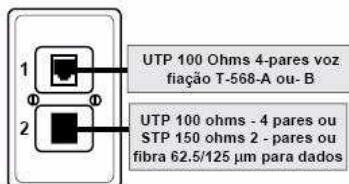


Figura 34.7: Tomada de Telecomunicações

Os componentes da área de trabalho estendem-se da tomada de telecomunicações até o equipamento da estação. A fiação da área de trabalho é projetada para ser de interconexão relativamente simples, de forma que deslocamentos, expansões e alterações possam ser efetuados com facilidade. Os componentes da área de trabalho são:

- Equipamento da estação: computadores, terminais de dados, telefone, etc.;
- Cabos de ligação - cordões modulares, cabos de adaptação, jumpers de fibra;
- Adaptadores.

Nesse contexto, a figura 34.8 ilustra o Sistema de Cabeamento Estruturada EIA/TIA 568:

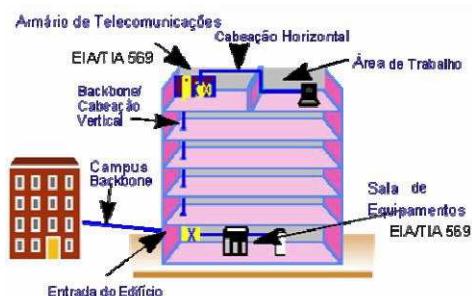


Figura 34.8: Sistema de Cabeamento Estruturada

34.6 Desempenho do Hardware e Meios de Transmissão

A norma EIA/TIA 568 classifica o sistema de cabeamento em categorias levando em consideração aspectos de desempenho, largura de banda, comprimento, atenuação e outros fatores de influência neste tipo de tecnologia. A seguir,

serão apresentadas as categorias de cabeação com tecnologia de par trançado UTP e de fibra óptica.

34.6.1 Cabeamento UTP

Os cabos UTPs são compostos de pares de fios trançados não blindados de 100 Ohms. Este tipo de cabo, nos dias de hoje, são projetados para alto desempenho na transmissão de dados ou voz. Segundo a norma, o cabo UTP pode ser classificado em três categorias como mostrado abaixo:

- Categoria 3 - Utiliza cabos com pares de fios trançados sólidos de bitola 24 AWG. Os fios AWG24 apresentam uma impedância típica de 100 Ohms, a 16 MHz. Estes cabos são utilizados para transmissão de sinais até 16 MHz.
- Categoria 4 - Utiliza cabos com pares de fios trançados sólidos de bitola 22 ou 24 AWG, com impedância de 100 Ohms a 20 MHz. Este cabos são utilizados para transmissão até uma largura de banda de 20 MHz;
- Categoria 5 - Utiliza cabos com pares de fios trançados sem blindagem de bitola 22 ou 24 AWG e impedância de 100 Ohms a 100 MHz. Este tipo de categoria é recomendável para aplicações com taxa de transmissão elevada, por exemplo, para transmissão de imagens e dados a 100 Mbps.

A atenuação é comumente derivada da medida do sinal de varredura da freqüência na saída de um cabo de comprimento maior ou igual a 100 metros (328 ft), ou seja, é a perda de potência do sinal no meio, em função da distância a uma determinada freqüência. As perdas por diafonia ou NEXT são comumente derivadas de medidas de varredura de freqüência. Por exemplo, na comunicação de voz, seus efeitos são sentidos por linhas cruzadas, isto é, vozes estranhas que são escutadas durante uma ligação telefônica. A impedância característica do cabo UTP para Cabeação Horizontal e Backbone é de 100 Ohms + 15% de 1 MHz até a maior freqüência da categoria (16, 20 ou 100 MHz).

Em complemento, os Terminadores para cabo UTP devem utilizar contatos por deslocamento por isolador (IDC). Os limites máximos para jumper/cordões de ligação são:

- 20 m para cross-connect principal;
- 20 m para cross-connect intermediário;
- 6 m no armário de telecomunicações;
- 3 m na estação de trabalho.

A terminação dos cabos horizontais deverá ser feita com material de conexão da mesma categoria ou superior do cabo UTP utilizado na Cabeação Horizontal. Por outro lado, os cabos utilizados para cordões de ligação e jumpers de cross-connect devem pertencer à mesma categoria do cabo UTP usado na Cabeação Horizontal. Assim, um sistema de cabeação UTP só poderá ser classificado como categoria 3, 4 ou 5 se todos os componentes do sistema de cabeação atenderem aos requisitos da categoria.

34.6.2 Fibra Óptica

A fibra óptica pode ser utilizada tanto para a Cabeação Horizontal como para a Vertical. A fibra para Cabeação Horizontal é do tipo multimodo de 62,5/125mm com um mínimo de duas fibras. A Cabeação Vertical ou Backbone utiliza fibras dos tipos multimodo de 62,5/125mm e monomodo formados em grupos de 6 ou 12 fibras.

As premissas para uma Cabeação Backbone com fibra ópticas, têm sido e continuam a ser baseadas em fibras multimodo de 62,5/125mm, devido à possibilidade de uso de transmissores ópticos com LED nessas fibras. Com o rápido crescimento dos requisitos de largura de banda, atualmente, tem-se instalado fibras ópticas monomodo em adição às fibras multimodo, para atender os requisitos atuais e futuros. Sistemas de fibras monomodo atendem tanto maiores bandas de freqüências como também têm maior capacidade para longas distâncias do que as fibras ópticas multimodo. A Figura 34.9 ilustra os tipos de fibras ópticas empregados.

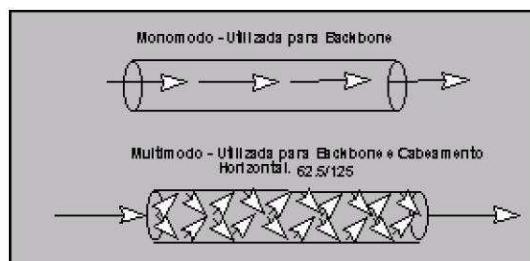


Figura 34.9: Tipos de Fibras Ópticas

Os conectores especificados para fibra óptica são os 568SC. Os conectores ópticos seguem um esquema de cores para sua identificação. A cor bege especifica o conector/acoplamento multimodo de 62,5/125mm e a cor azul especifica o conector/acoplamento monomodo de 8,3/125mm.

Para assegurar que os conectores 568SC manterão uma correta polarização através do sistema de cabeação, deve-se ter uma correta orientação do adaptador utilizado. A Cabeação Horizontal deve ser instalada de tal forma a casar um número ímpar da fibra com o próximo número par da fibra, por exemplo: fibra 1 com fibra 2; fibra 3 com fibra 4 e assim sucessivamente. Cada segmento da cabeação deve ser instalado seguindo a orientação invertida (cross-over) do par, de tal modo que fibras de número ímpar são posição A numa ponta e posição B na outra ponta, enquanto que fibras de número par são posição B numa ponta e posição A na outra ponta.

A orientação invertida (cross-over) deve ser conseguida pelo uso consecutivo da numeração das fibras (por exemplo 1, 2, 3, 4, ...) em ambos os lados da fibra, mas os adaptadores 568SC devem ser instalados de maneira oposta em cada ponta (por exemplo A-B, A-B, ... numa ponta e B-A, B-A, ... na outra ponta).

O principal motivo para especificação dos conectores de fibra 568SC é a padronização com a norma IEC Européia. Hoje são muito utilizados conectores ST. Entretanto, é recomendado a substituição gradativa dos conectores ST por 568SC.

A norma EIA/TIA 568A especifica, também, as saídas de telecomunicações para fibra óptica com as seguintes características:

- A caixa de montagem em superfície deve ser fixada diretamente sobre a caixa elétrica, seguindo um padrão de 4"x 4";
- A capacidade de terminação para um mínimo de duas fibras, por acoplamento 568SC;
- Possibilidade de armazenar um mínimo de 1 metro de cabo de duas fibras.

34.7 Código de Cores para Sistemas de Cabeção UTP

A EIA/TIA 568A define um sistema de codificação com quatro cores básicas, em combinação com o branco, para os condutores UTP de 100 Ohms, assim como a ordem dos pares no conector RJ-45, conforme ilustrado na figura 34.10.

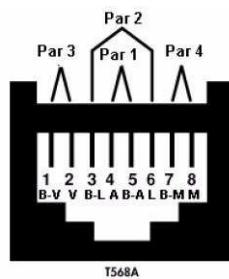


Figura 34.10: Código de cores EIA/TIA 568A

Um outro padrão de cores da cabeação UTP, derivado da EIA/TIA 568A, o padrão EIA/TIA 568B, não muito utilizado nos dias atuais, define a seqüência de cores da figura 34.11.

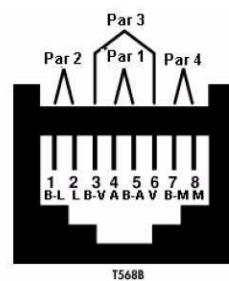


Figura 34.11: Código de cores EIA/TIA 568B

Capítulo 35

Redes sem fio

Os principais elementos de uma rede sem fio são:

- Hospedeiros sem fio: São os sistemas finais que executam as aplicações, por exemplo, um celular, um PDA, um notebook etc;
- Enlace sem fio: Os hospedeiros se conectam ao restante da rede por meio de um enlace de comunicação sem fio. Tecnologias de enlace sem fio têm diferentes taxas de transmissão e podem transmitir à distâncias diferentes. Outra característica relevante dos enlaces sem fio é a grande taxa de erros de bits, que obriga os protocolos de enlace sem fio a empregarem mecanismos mais sofisticados de detecção de erros e retransmissão;
- Estação-base: É a responsável pelo envio e recebimento dos dados de e para os hospedeiros sem fio à ela associados. Exemplos de estação-base são torres celulares, pontos de acesso de uma rede 802.11 etc. Em geral, as estações-base estão ligadas à uma infra-estrutura maior de rede (uma LAN cabeadas, a internet etc.) que é responsável pelo fornecimento dos serviços mais comuns como atribuição de endereço e roteamento. Quando isso acontece é dito que a rede está operando no modo de infra-estrutura.

Algumas tecnologias de rede sem fio permitem que os dispositivos móveis se comuniquem diretamente sem a necessidade de uma estação base. Esse modo de operação é chamado *ad hoc*. Nesse caso os próprios hospedeiros sem fio devem ser capazes de fornecer serviços básicos de atribuição de endereço, roteamento, resolução de nomes etc.

35.1 O padrão IEEE 802.11

O IEEE 802.11, também conhecido como *Wi-Fi*, é a tecnologia mais utilizada atualmente para implementação de redes locais sem fio. Entre os padrões 802.11 que mais se destacam estão o 802.11a, 802.11b e 802.11g. Esses três padrões compartilham características como formato do quadro, protocolo de acesso ao meio, capacidade de reduzir taxa de transmissão para alcançar distâncias maiores e a possibilidade de operar tanto em modo de infra-estrutura como em modo *ad hoc*. As diferenças entre os padrões a, b e g se concentram na camada física de acordo com a tabela 35.1.

Padrão	Faixa de Frequência	Taxa de Dados
802.11b	2.4-2.485 GHz	até 11 Mbps
802.11a	5.1-5.8 GHz	até 54 Mbps
802.11g	2.4-2.485 GHz	até 54 Mbps

Tabela 35.1: Resumo dos Padrões 802.11

O padrão 802.11b é atualmente o mais utilizado devido à sua boa relação custo-benefício. Em seguida está o padrão 802.11g, que vêm se estabelecendo por permitir operação à taxas mais elevadas e distâncias equivalentes ao padrão 802.11b. Embora alcance taxas tão elevadas quanto o o padrão 802.11g, o padrão 802.11a alcança distâncias menores. Em contrapartida, o padrão 802.11a é menos suscetível às interreferências introduzidas por telefones celulares, microondas e bluetooth, que operam na faixa de 2.4 GHz assim como os padrões 802.11b e 802.11g.

35.1.1 CSMA/CA

O protocolo de acesso ao meio utilizado pelas redes do padrão 802.11 é o CSMA/CA (*Carrier Sense for Multiple Access with Collision Avoidance*). Esse protocolo se baseia no fato de que é extremamente difícil detectar colisões em enlaces sem fio, e por isso, tenta evitá-las sempre que possível. O CSMA/CA funciona da seguinte maneira:

1. Se a estação perceber que o canal está ocioso, ela transmitirá o quadro após um curto espaço de tempo conhecido como DIFS (*Distributed Inter-Frame Space*);
2. Caso contrário, a estação escolherá um valor aleatório de backoff e iniciará a contagem regressiva a partir desse valor assim que detectar o canal ocioso;
3. Quando o contador chegar a zero, a estação transmitirá o quadro inteiro e ficará esperando por uma confirmação. Caso o receptor receba o quadro corretamente, ele esperará um tempo curto chamado SIFS (*Short Inter-Frame Space*) e então enviará a confirmação;
4. Se a estação receber a confirmação e tiver outro quadro para transmitir, então ela irá iniciar o protocolo a partir da fase 2. Caso não receba nenhuma confirmação, voltará para fase 2 porém irá escolher por um backoff dentro de um intervalo de tempo maior.

A característica mais importante do protocolo CSMA/CA é que mesmo que uma estação detecte que o canal está ocioso, ela adia a transmissão por um curto período de tempo aleatório. Se ao detectar o canal ocupado cada uma das estações escolher um valor de backoff diferente das demais, então uma delas irá começar a transmitir primeiro. As demais, ao escutar o canal ocupado, interromperam seus contadores e aguardaram pelo término da transmissão.

O padrão 802.11 conta ainda com um mecanismo de reserva de canal que ajuda a evitar colisões. O mecanismo é baseado no envio de quadros especiais chamados RTS (*Request to Send*) e CTS (*Clear to Send*). Quando um transmissor deseja enviar um quadro de dados DATA, ele pode solicitar a reserva do canal enviando um quadro RTS para a estação-base. O RTS indica o tempo total necessário para a transmissão de DATA. Ao receber o RTS, a estação-base envia um quadro CTS em broadcast para toda a rede. O quadro CTS tem como finalidade dar permissão para o envio do quadro DATA e instruir os demais transmissores a não enviarem nada durante o tempo reservado.

35.1.2 Formato do Quadro 802.11

A característica mais marcante de um quadro 802.11 é que ele têm 4 campos de endereço, cada um podendo conter um endereço MAC de 6 bytes. Em redes 802.11 operando no modo infra-estrutura 3 desses campos são necessários (o outro só é utilizado no modo *ad hoc*). O campo endereço 1 contém o MAC do dispositivo sem fio destinatário, enquanto o endereço 2 contém o MAC do dispositivo sem fio remetente. O campo endereço 3 contém o endereço MAC do dispositivo Ethernet com o qual a estação base está conectada, por exemplo, um roteador. A estação-base é, portanto, o dispositivo de camada de enlace responsável por intermediar a comunicação dos dispositivos sem fio com a rede cabeada.

Além dos campos de endereço, também merecem destaque os campo CRC (*Cyclic Redundancy Check*), que permite detecção de erros de bits, e o campo de número de sequência, que permite que o mecanismo de confirmação funcione. O formato completo do quadro 802.11 pode ser visto na figura 35.1.2.

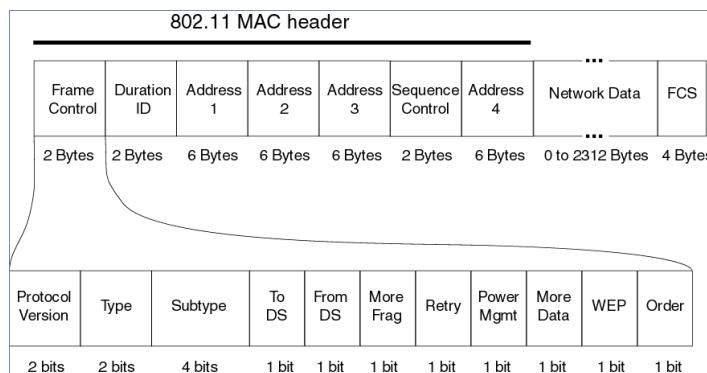


Figura 35.1: Formato do quadro 802.11

Capítulo 36

Elementos de Interconexão de Redes de Computadores

36.1 Repetidores

São elementos que interligam dois segmentos de um mesmo domínio de colisão. Os sinais elétricos que trafega em ambos sentidos são amplificados analogicamente permitindo comunicações em maiores distâncias. Portanto, estes elementos operam na CAMADA FÍSICA do modelo OSI.

Estes equipamentos deixaram de ser largamente utilizados devido ao fato dos hubs ativos, switches, gateways e roteadores terem passado a exercer também a função de amplificação.

36.2 Hubs

Este é um termo genérico para CONCENTRADORES. Eles são elementos de CAMADA FÍSICA do modelo OSI. Portanto, todos os nós conectados a um hub, ou a um conjunto de hubs interconectados, fazem parte de um mesmo domínio de colisão (topologia barramento). Desta forma, cada nó deve ser capaz de detectar colisões e decidir o momento de realizar transmissões e/ou retransmissões.

Duas características importantes são: como um hub define um único domínio de colisão comum a todas as portas, a princípio ele não permite a interconexão de segmentos que utilizam tecnologias distintas (taxas, protocolo de camada 2, etc.); um hub não implementa nenhum tipo de controle de acesso, e sim encaminha os bits de uma porta para as demais.

Estes concentradores podem ser classificados em quatro grupos:

- Hubs Passivos: hubs que não implementam a função de repetição, ou seja, não funcionam como repetidores. Estes equipamentos não necessitam de nenhuma alimentação elétrica. Eles não possuem nenhum módulo microprocessado. Um exemplo são os patch painels;

- Hubs Ativos: hubs que implementam a função de repetição. Assim como os passivos, eles não possuem nenhum módulo microprocessado. A grande maioria dos hubs utilizados ultimamente pertence a este grupo;
- Hubs Inteligentes: hubs que possuem módulos microprocessados que realizam monitoramento e diagnóstico. As funcionalidades dependem do projeto do fabricante, por exemplo, eles podem: detectar e se preciso desconectar da rede estações com problemas, evitando que uma estação faladora prejudique o tráfego ou mesmo derrube a rede inteira; detectar e impedir tentativas de invasão ou acesso não autorizado à rede; monitorar/informar o nível de utilização de cada porta; e possibilitar operação de taxas diferentes entre as portas (ex. 10Mbps e 100Mbps). Geralmente estes hubs também são classificados como ativos;
- Hubs Empilháveis: hubs que possuem portas denominadas Up-Links. Estas portas permitem o empilhamento de hubs através de CABOS DIRETOS. Na prática, qualquer hub pode ser empilhado a outro, a diferença é que no caso de hubs não-empilháveis (sem porta do tipo Up-Link) deve ser utilizado um CABO CROSS-OVER.

36.3 Switches

Estes são elementos de CAMADA DE ENLACE do modelo OSI também denominados de COMUTADORES. Geralmente eles funcionam também como repetidores. Diferentemente dos hubs, os switches dividem a rede em domínios de colisão (um domínio por porta). Desta forma, o switch ao comutar um quadro de um domínio para outro deve ser capaz de implementar algum tipo de controle de acesso para ser possível detectar colisões e decidir o momento de realizar transmissões e/ou retransmissões.

Uma outra característica importante presente no switches é a possibilidade deles interconectarem segmentos com tecnologias distintas. Ou seja, este tipo de comutador pode ser multiprotocolo (exige conversão de um tipo de quadro em outro - ex. Ethernet em FDDI). Alguns switches também são capazes de administrar múltiplas taxas de transmissão. Quando ele é capaz de detectar automaticamente a taxa de transmissão utilizada em uma determinada porta e se auto configurar para esta taxa, diz-se que este switch é AUTO-SENSING.

As comutações são realizadas por meio de consultas a tabelas dinâmicas que armazenam endereços físicos, e seus respectivos, número da interface (porta) e horário da última comutação. Os switches são equipamentos plug-and-play devido ao fato de sua tabela de comutação ser preenchida de forma automática ao longo do funcionamento da rede. Este preenchimento acontece da seguinte forma. Inicialmente a tabela se encontra vazia. Quando um quadro chega para ser comutado, e o switch ao perceber que não há nenhuma entrada na tabela para o endereço físico de destino, ele replica este quadro em todas as outras interfaces (portas). Todas as vezes que um nó envia um quadro a qualquer outro nó, seu endereço físico é registrado na tabela juntamente com o número da interface onde ele está conectado ao switch.

Todo switch realiza no mínimo duas operações básicas utilizando sua tabela de comutação. A FILTRAGEM que é a capacidade de determinar se um quadro deve ser repassado para alguma interface ou se ele deve ser descartado. E o REPASSE que é a capacidade de determinar para qual interface o quadro deve ser dirigido.

Estes comutadores podem operar em duas formas distintas. São elas:

- Store-and-forward - a seqüência de comutação é a seguinte: o quadro é recebido na porta de entrada; este quadro é armazenado em um buffer interno; é realizada a filtragem/repassagem; o quadro é copiado no buffer da porta de saída. Este tipo de comutação é geralmente implementado por elementos de hardware e software (principalmente) em conjunto;
- Cut-Through (Acelerada ou De Corte) - a seqüência de comutação é a seguinte: à medida que o quadro vai sendo recebido de forma seqüencial na porta de entrada, assim que os primeiros bits que definem o endereço físico de saída acabam de chegar, a filtragem/repassagem é realizada e então este quadro já vai sendo copiado no buffer da porta de saída. É importante notar que o quadro já começa a ser copiado no buffer de saída antes mesmo dele ter sido totalmente recebido na porta de entrada. Nos casos onde os buffers de saída estão geralmente vazios (sem fila de transmissão), esta forma de operação traz benefícios consideráveis no desempenho da comutação. Este tipo de comutação é geralmente implementado por somente elementos de hardware.

36.4 Bridges

As bridges (PONTES) são muito semelhantes aos switches. Na verdade, muitas pessoas usam ambos os termos de forma intercambiável. A principal diferença é que os switches são usados com maior freqüência para conectar computadores individuais, enquanto as bridges são mais usadas para conectar redes. Contudo, todas as características dos switches apresentadas na seção anterior também estão presentes nas bridges.

36.5 Roteadores

São elementos da CAMADA DE REDE do modelo OSI. Eles também geralmente funcionam como repetidores, ou seja, regeneram os sinais recebidos antes de realizarem a comutação. Assim como os switches e bridges, os roteadores também são COMUTADORES, porém com uma complexidade muito maior. O que reflete no tempo de comutação e também nos recursos disponíveis.

Seu papel fundamental é escolher, por meio de algum algoritmo de roteamento, uma dentre as eventuais muitas rotas entre um nó de origem e um nó de destino. Vale ressaltar que no caso dos switches e bridges a comutação é mais fácil, visto que a "rota" é única, o que nem sempre acontece no caso das roteadores. Uma característica muito importante dos roteadores é a capacidade

de interconectar redes que utilizam tecnologias diferentes (arquiteturas diferentes). Isto só é possível quando o roteador é capaz de "entender" e converter um datagrama de uma tecnologia em outro datagrama de outra tecnologia (ex. IP e ATM).

Outras características também importantes são: a comutação não se limita a uma SPANNING-TREE (grafo sem loopings - caminhos únicos entre quaisquer dois nós) como os comutadores de camada de enlace; possibilita proteção contra broadcast em excesso; possibilita proteção de firewall; e não é um equipamento plug-and-play tendo em vista que os endereços de rede são lógicos (não-fixos) e não físicos (fixos) e devem ser "programados" de alguma forma; e a comutação é necessariamente realizada no modo store-and-forward (nunca no modo cut-through).

36.6 Gateways

Estes elementos podem ser divididos em duas grandes classes: Gateways Conversores de Meio e Gateways Tradutores de Protocolos.

Os Conversores de Meio são exatamente os roteadores multiprotocolo. Ou seja, são elementos de CAMADA DE REDE do modelo OSI.

Já os Tradutores de Protocolos podem ser elementos de CAMADA DE TRANSPORTE ou CAMADA DE APLICAÇÃO. Fundamentalmente os primeiros realizam conversões entre segmentos (unidade de camada de transporte) de protocolos distintos e os últimos realizam conversões entre mensagens (unidade de camada de aplicação) de protocolos distintos. Vale ressaltar que durante as conversões o que se busca e manter a semântica dos dados. Por isso, nem sempre é possível converter um protocolo em outro.

Capítulo 37

Redes Multimídia

37.1 Qualidade de Serviço

Um fluxo é uma sequência de pacotes de uma origem até um destino. Em uma rede orientada a conexões os pacotes de um determinado fluxo seguem uma mesma rota, enquanto em uma rede não orientada a conexões, eles podem seguir rotas diferentes. Nesse contexto, Qualidade de Serviço (QoS) pode ser definido em termos do conjunto de requisitos de um determinado fluxo de pacotes. Os requisitos mais comuns são os seguintes:

- Confiabilidade: Garantia da entrega dos pacotes e da integridade dos dados transmitidos.
- Retardo: Atraso total na transmissão de um pacote da origem até o destino.
- Flutuação (*Jitter*): É a variação do atraso entre os pacotes. Em outras palavras, é a variação do intervalo de tempo entre os recebimento de pacotes subsequentes de um determinado fluxo.
- Largura de Banda: Taxa com a qual os dados são transmitidos.

A rigidez dos requisitos das aplicações mais comuns é mostrada na tabela a 37.1:

No que diz respeito à confiabilidade, por exemplo, pode-se dizer que o serviço de correio eletrônico é mais sensível do que uma aplicação de vídeo sob demanda, já que o corrompimento dos dados pode invalidar por completo uma mensagem. Em uma aplicação de vídeo sob demanda pode conviver com alguns pacotes com erro sob pena da perda parcial da qualidade do vídeo. É por esse motivo que em redes TCP/IP aplicações de multimídia, geralmente, utilizam serviços da camada de transporte baseados em UDP e não em TCP, evitando os atrasos gerados pelo estabelecimento de conexão, confirmações e retransmissões.

No entanto, para alcançar qualidade de serviço de forma eficiente e segura é necessário combinar várias técnicas que podem ir além da escolha de um protocolo. Exemplos de técnicas para alcançar QoS são:

Aplicação	Confiabilidade	Retardo	Jitter	Largura de Banda
Correio Eletrônico	Alta	Baixa	Baixa	Baixa
Transferência de Arquivos	Alta	Baixa	Baixa	Média
Acesso à Web	Alta	Média	Baixa	Média
Login Remoto	Alta	Média	Média	Baixa
Áudio por demanda	Baixa	Baixa	Alta	Média
Vídeo por demanda	Baixa	Baixa	Alta	Alta
Telefonia	Baixa	Alta	Alta	Baixa
Videoconferência	Baixa	Alta	Alta	Alta

Tabela 37.1: Rígidez dos Requisitos das Alicações de Rede

- Superdimensionamento: Consiste em fornecer tanta capacidade de buffers e largura de banda de forma que dificilmente os pacotes serão descartados ou sofrerão atrasos. Logicamente essa é a solução tem um custo altamente elevado;
- Armazenamento em Buffers: Consiste em armazenar os fluxos em buffers no lado do receptor e entregá-los apenas no momento oportuno suavizando o *jitter*. Essa técnica não inclui alterações na largura de banda, porém aumenta o atraso e exige buffers maiores do lado do receptor. Muitos reprodutores de áudio e vídeo se utilizam dessa técnica;
- Moldagem de Tráfego: Muitas vezes uma máquina pode transmitir pacotes com espaçamento não uniforme, o que pode gerar congestionamento na rede. A moldagem de tráfego está relacionada à regulagem da taxa média de transmissão dos dados. Para realizar tal tarefa são utilizadas técnicas como o algoritmo do balde furado (*leaky bucket*) ou o algoritmo do balde de símbolos (*token bucket*);
- Reserva de Recursos: Se for possível fazer com que todos os pacotes de um fluxo sigam a mesma rota, torna-se possível reservar recursos para garantir que a capacidade necessária estará disponível. Portanto, As aplicações são responsáveis por definir os seus requisitos quanto a utilização dos recursos de CPU, espaço em buffers e largura de banda. O conjunto desses parâmetros é chamado especificação de fluxo, e é com base nele que os roteadores ao longo da rota decidem se vão ou não aceitar o fluxo. Ao longo do caminho da origem ao destino, os requisitos da especificação de fluxo podem ser alterados para baixo (ex: diminuição da largura de banda, diminuição do pacote de tamanho máximo etc.). Ao fim do caminho origem destino os parâmetros estão ajustados conforme a capacidade dos roteadores intermediários, e a transmissão pode ser iniciada ou não. Esse processo é conhecido como controle de admissão;
- Enfileiramento Prioritário: Utilização de múltiplas filas, cada uma com uma prioridade associada. Cada pacote que chega é direcionado à uma das filas de acordo com sua classe. Os pacotes a serem transmitidos são sempre aqueles da fila de maior prioridade que ainda não está vazia. Algumas

implementações permitem a interrupção de uma transmissão de um pacote de prioridade mais baixa para transmissão de um pacote de prioridade mais alta (preempção).

- Varredura Cílica e WQF: Na varredura cílica os pacotes são classificados e colocados em filas de saída da mesma forma que no enfileiramento prioritário. No entanto, nessa disciplina o escalonador alterna o direito de transmitir de forma cílica a cada período (Ex: fila 1, fila 2, fila 3, fila 1, fila 2, fila 3 etc.). No WQF (*Weighted Fair Queuing* ou Enfileiramento Justo Ponderado) o direito de transmitir é ponderado entre as filas. (Ex: fila 1, fila 1, fila 1, fila 2, fila 3, fila 3, fila 1, fila 1, fila 1, fila 2, fila 3, fila 3 etc.).

37.2 Serviços Integrados - *IntServ*

A arquitetura de serviços integrados tem duas características fundamentais que são a reserva de recursos e o estabelecimento de chamada. O Intserv é fornecer qualidade de serviço a fluxos individuais garantindo que, antes do estabelecimento de uma sessão, todos os roteadores entre a origem e o destino possuem recursos suficientes para garantir o atendimento das exigências de Qos. As etapas envolvidas no processo de aceitação da chamada são as seguintes:

- Caracterização do Tráfego e Especificação de Qos desejada: Para que um roteador defina se pode ou não atender as exigências de uma sessão ele precisa conhecer a exigências de QoS (Rspec) e as características de tráfego (Tspec). Tspec e Rspec são chamados especificações de fluxo *flowspecs*;
- Sinalização para o estabelecimento da chamada: A Tspec e o Rspec devem ser transportados para todos os roteadores nos quais serão reservados recursos;
- Aceitação da chamada: Assim que recebe Tspec e Rspec o roteador pode determinar se pode ou não aceitar a chamada.

Na Internet, o protocolo RSVP (*Resource Reservation Protocol*) é o protocolo de sinalização preferido. Ele foi especialmente desenvolvido para serviços integrados de Internet e permite que as próprias aplicações requeiram da rede reservas de recursos necessários para seus diversos serviços. O RSVP é executado nos *hosts*, para definir as especificações de fluxo, e nos roteadores, para propagação das especificações pela rota e para manutenção do estado da conexão.

Outra característica importante na arquitetura IntServ são as classes de serviço oferecidos. As classes são os modelos de qualidade de serviço oferecidos. As duas grandes classes de serviço do intServ são: (i)Serviço de Qualidade Garantida e (ii) Serviço de Rede de Carga Controlada.

A primeira grande desvantagem da arquitetura IntServ estão relacionadas com a escalabilidade, uma vez que os roteadores precisam manter o estado de cada um dos fluxos transmitidos, o que pode ocasionar sobrecarga em roteadores de backbone. A segunda desvantagem do IntServ é que ela atende apenas um número pequeno de classes de serviço.

37.3 Serviços Diferenciados - *DiffServ*

A arquitetura de serviços diferenciados (Diffserv) tem por objetivo prover diferenciação de serviços mais escalável e flexível. Para isso o DiffServ propõe que as operações de controle mais complexas devem ser feitas pela borda da rede, aliviando a sobrecarga no núcleo da rede. Dessa forma, a arquitetura Intserv consiste de dois elementos funcionais que são:

- Funções de Borda: Entende-se por borda como o primeiro dispositivo habilitado a DiffServ (host ou roteador) no caminho entre a origem e o destino. A função dos dispositivos de borda são a classificação dos pacotes e o condicionamento do tráfego. A classificação do tráfego consiste na marcação do campo DS (Differentiated Service), que na verdade é o campo ToS (Type of Service) do pacote IP. O condicionamento do tráfego está relacionado com a limitação de parâmetros pré-acordados como taxa média de envio, taxa de pico etc.
- Função Central: Quando um pacote marcado com DS chega a um roteador habilitado a DiffServ ele é repassado de acordo com o seu comportamento de salto (*per-hop behavior* - PHB) associado a sua classe. O PHB influencia na maneira como os buffers e a largura de banda são compartilhados entre as classes de tráfego no roteador. Os dois tipos de PHB definidos até agora são o PHB de repasse acelerado (*expedited forwarding* - EF), que garante a taxa de partida de uma determinada classe será maior ou igual do que uma taxa configurada, e o PHB de envio assegurado (*assured forwarding* - AF), que permite criação de regras de descarte preferencial entre as classes. Um aspecto de fundamental importância na arquitetura DiffServ é que o comportamento de salto dos pacotes depende única e exclusivamente de sua classe de tráfego.

Capítulo 38

Redes X.25 e Frame Relay

38.1 X.25

O conjunto de protocolos X.25 foi projetado no final da década de 70. Nessa época, os PCs e as estações de trabalho não estavam amplamente disseminados e não dispunham de muito suporte para rede. Basicamente, eram usados os chamados terminais "burros" para acessarem os mainframes através das redes de computadores. Dessa forma, para dar o suporte necessário aos terminais "burros" os projetistas da X.25 decidiram "injetar inteligência na rede". Como sabemos hoje, essa filosofia é oposta à filosofia da Internet, que coloca muito da complexidade nos sistemas finais e espera o mínimo dos serviços de camada de rede.

Outra parte importante do contexto tecnológico do final dos anos 70 e do começo dos anos 80 se refere aos enlaces físicos. Naquela época, quase todos os enlaces terrestres eram de cobre, apresentavam ruídos e estavam propensos a erros. Os enlaces de fibra óptica ainda não tinham saído dos laboratórios de pesquisa. As taxas de erros de bits sobre enlaces de longa distância eram muitas ordens de grandeza maiores do que são agora sobre os enlaces ópticos. Devido às altas taxas de erros, tinha sentido projetar o protocolo X.25 com recuperação de erros em cada enlace. Em particular, sempre que um protocolo X.25 envia um pacote, ele conserva uma cópia do pacote até que o comutador seguinte (na rota do pacote) devolva um reconhecimento, indicando que o pacote foi recebido livre de erros. A recuperação de erros por enlace reduz significativamente a taxa de transmissão, o que era consistente com o contexto tecnológico da época – altas taxas de erros de enlace e terminais não inteligentes. Além disso, o projeto da X.25 também pede controle de fluxo por enlace.

38.2 Frame Relay

Projetada no final da década de 80 e amplamente disseminada na década de 90, a Frame Relay é, em vários aspectos, uma X.25 de segunda geração. Como a X.25, ela usa circuitos virtuais, opera na camada 2 do modelo OSI e usa multiplexação estatística e compartilhamento de porta. Contudo, como se baseia em enlaces de fibras ópticas, ela naturalmente foi projetada para taxas de erros muito mais baixas. A essência da Frame Relay é um serviço de comutação de pacotes de

circuitos virtuais sem recuperação de erros e sem controle de fluxo. Quando um comutador Frame Relay detecta um erro em um pacote, seu único curso de ação possível é descartar os dados. Isso resulta em uma rede com carga de processamento mais baixas e taxas de transmissão mais altas que a X.25, mas exige sistemas finais inteligentes para garantir a integridade dos dados.

Embora o protocolo Frame Relay tenha sido desenvolvido para ser o mais simples possível, e a sua premissa básica determinar que os eventuais problemas de erros da rede deveriam ser resolvidos pelos protocolos dos equipamentos de usuário, surgiram ao longo do tempo necessidades que levaram os órgãos de padronização a definir mecanismos de sinalização para três tipos de situações:

- Aviso de congestionamento
 - Aviso Explícito de Congestionamento
 - Aviso Implícito de Congestionamento
 - Elegibilidade para Descarte
- Estado das conexões
- Sinalização SVC

Entretanto, a implementação desses mecanismos é opcional e, embora a rede seja mais eficiente com a sua adoção, os equipamentos que não os implementam devem atender pelo menos as recomendações básicas do Frame Relay.

Atualmente, na maioria dos casos, as redes Frame Relay são de propriedade de um provedor de serviços de rede pública (por exemplo AT&T, Sprint e etc) e seu uso é contratado em base multianual por clientes empresariais. Hoje, a Frame Relay é usada de maneira extensiva para permitir que LANs localizadas em diferentes ambientes enviem dados ao mesmo tempo a velocidades razoavelmente altas.

As redes Frame Relay podem usar circuitos virtuais comutados (switched virtual circuits – SVCs) ou circuitos virtuais permanentes (permanent virtual circuits – PVCs).

38.2.1 Estrutura do Frame

O protocolo da Frame Relay utiliza um frame com estrutura comum e bastante simplificada, conforme demonstram a figura e a descrição a seguir:

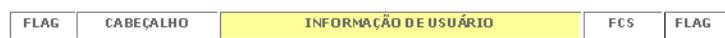


Figura 38.1: Estrutura do Frame

Byte 1			Byte 2					
DLCI	C/R	EA	DLCI	FE CN	BE CN	DE	EA	
8 7 6 5 4 3	2	1	8 7 6 5	4	3	2	1	

Figura 38.2: Estrutura do Cabeçalho

- Flags - indicam o início e o fim de cada frame.
- Cabeçalho - carrega as informações de controle do protocolo. É composto por 2 bytes com as seguintes informações:
 - DLCI (Data Link Connection Identifier), com 10 bits, representa o número (endereço) designado para o destinatário de um PVC dentro de um canal de usuário, e tem significado local apenas para a porta de origem (vide figura 38.2);
 - C/R (Command / Response), com 1 bit, é usado pela aplicação usuária;
 - FECN (Forward Explicit Congestion Notification), com 1 bit, é usado pela rede para informar um equipamento receptor de informações que procedimentos de prevenção de congestionamento devem ser iniciados;
 - BECN (Backward Explicit Congestion Notification), com 1 bit, é usado pela rede para informar um equipamento transmissor de informações que procedimentos de prevenção de congestionamento devem ser iniciados;
 - DE (Discard Eligibility Indicator), com 1 bit, indica se o frame pode ser preferencialmente descartado em caso de congestionamento na rede;
 - EA (Extension Bit), com 2 bits, é usado para indicar que o cabeçalho tem mais de 2 bytes, em caso especiais;
- Informação de usuário - contém as informações da aplicação usuária a serem transportadas através da rede Frame Relay.
- FCS - o FCS (Frame Check Sequence) representa o CRC padrão de 16 bits usado pelo protocolo Frame Relay para detectar erros existentes entre o Flag de início do frame e o próprio FCS, e pode ser usado apenas para frames com até 4096 bytes.

38.2.2 Envio de um datagrama IP de Ethernet para Frame Relay e Ethernet

Considere a transmissão de um datagrama IP entre dois sistemas finais que estão em duas redes Ethernet interconectadas por uma rede Frame Relay. Quando um quadro Ethernet chega ao roteador fonte, a placa Ethernet do roteador retira os campos Ethernet e passa o datagrama IP à camada de rede. A camada de rede passa o datagrama IP à placa de interface Frame Relay. Essa placa encapsula o datagrama IP em um quadro Frame Relay. Ela também calcula o CRC (2 bytes) e insere o valor resultante no campo CRC. O campo de camada de enlace (2 bytes) contém um campo de número de circuito virtual de 10 bits. A placa de interface obtém o número do CV de uma tabela que associa números de rede IP a números de CV. Ela então transmite o pacote.

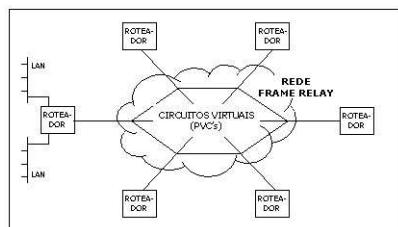
A placa de interface transmite o pacote Frame Relay a um comutador Frame Relay próximo, de propriedade de um provedor de serviços Frame Realy. O comutador examina o campo de CRC. Se o quadro tiver um erro, o comutador o descartará. Se não houver erro no quadro, o comutador usará o número de

CV do quadro para roteá-lo até o próximo comutador (ou até o roteador de destino). O roteador de destino remove os campos frame relay e, então, passa o datagrama pela Ethernet para o hospedeiro de destino. Se os segmentos TCP forem perdidos ou chegarem fora de ordem, o TCP dos hospedeiros comunicantes corrigirá o problema.

38.3 Interligação de Redes LAN

A interligação das redes LAN de vários escritórios compõe uma rede WAN, é uma aplicação típica para o uso da tecnologia Frame Relay. O tráfego usual das redes de dados é normalmente de 2 tipos: interativo (comando - resposta), ou seja, solicitação de usuários e aplicações clientes e respostas de aplicações servidoras, e por rajadas (bursty), quando grandes quantidades de dados são transferidas de forma não contínua.

O Frame Relay, através de roteadores ou equipamentos de acesso (FRAD) instalados nos escritórios, permite utilizar uma porta única em cada escritório para compor redes do tipo malha (meshed) onde a comunicação de um escritório com todos os outros é possível sem a complexidade do uso de múltiplas portas e múltiplos circuitos dedicados.



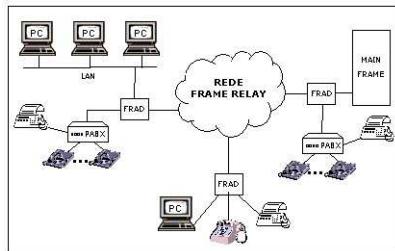


Figura 38.4: Voz sobre Frame Relay

38.3.2 Interação entre Frame Relay e ATM

Para buscar aumentar a interoperabilidade do Frame Relay com outros protocolos de dados, o FR Fórum e o ATM Fórum, os órgãos responsáveis pelo desenvolvimento de Acordos de Implementação (IA's), desenvolveram padrões para interligar equipamentos dessas tecnologias através de PVC's.

Foram padronizadas duas formas de interoperabilidade. A primeira, chamada de Frame Relay/ATM Network Interworking for PVC's, padroniza uma funcionalidade responsável pelo encapsulamento dos PVC's para que os mesmos possam ser transportados indistintamente nas redes da 2 tecnologias. Seu uso típico ocorre quando a rede Frame Relay tem com núcleo uma rede ATM, para otimizar ainda mais o uso de banda e a segurança. A figura a seguir apresenta esta solução.

A segunda forma de interoperabilidade, chamada de Frame Relay/ATM Service Interworking for PVC's, padroniza uma funcionalidade responsável pela conversão dos protocolos (FR → ATM), que pode ser incorporada tanto aos equipamentos de acesso como aos equipamentos da rede. Seu uso típico ocorre quando o usuário possui redes Frame Relay em alguns escritórios que devem se interligar com a rede ATM da matriz. A figura a seguir apresenta esta solução.

38.3.3 CIR (Taxa de Informação Comprometida)

A Frame Relay faz uso de um mecanismo inovador chamado de taxa de informação comprometida (committed information rate - CIR), de forma que cada CV possui um CIR. Em termos gerais, a CIR representa um compromisso que a rede Frame Relay assume de dedicar ao CV uma taxa de transmissão determinada pela CIR. Pode-se dizer que, em muitos aspectos, o serviço CIR é um predecessor do serviço diferenciado da Internet.

Nas redes Frame Relay, os pacotes podem pertencer a um de dois níveis de prioridade: alta ou baixa. Atribui-se prioridade aos pacotes marcando um bit especial no cabeçalho do pacote – o denominado bit de descarte preferencial (discard eligibility- DE) -, com 0 para alta prioridade e 1 para baixa prioridade. Se um quadro for de alta prioridade, a rede deverá entregar o pacote no destino sob todas e quaisquer condições de rede, incluindo períodos de congestionamento e falha de enlaces de backbone. Contudo, para pacotes de baixa prioridade, permite-se que a rede descarte o quadro quando ela estiver congestionada. Na verdade, sob condições extremas, a rede pode até descartar pacotes de alta prioridade.

A CIR, nesse contexto, está envolvida no processo de marcação dos pacotes com valores 1 ou 0. Entretanto alguns conceitos devem ser discutidos para se entender exatamente como a CIR funciona. A taxa de acesso é a taxa de acesso ao enlace, isto é, a taxa do enlace do roteador fonte até o comutador de borda Frame Relay. Essa taxa é freqüentemente 64Kbps, mas múltiplos inteiros de 64 Kbps até 1544 Mbps também são comuns. Chamemos essa taxa de R . O comutador de borda é responsável pela marcação dos pacotes que chegam do roteador fonte. Para fazer a marcação, o roteador de borda examina os horários de chegada dos pacotes vindos do roteador fonte em intervalos fixos de tempo, chamados de intervalos de medição e designados por T_c .

Dessa forma, a cada CV que provém do roteador fonte é atribuída uma CIR, que é expressa em unidades de bits/seg. A CIR nunca é maior que R , a taxa de acesso. Os clientes pagam por uma CIR específica; quanto maior a CIR, maior é o valor pago ao provedor. Se o CV gerar pacotes a uma taxa menor do que a CIR, então todos os pacotes são marcados como de alta prioridade. Contudo, se a taxa na qual a CV gerar pacotes exceder a CIR, então a fração de pacotes do CV que excederem a taxa será marcada como pacotes de baixa prioridade. Mais especificamente, a cada intervalo de medição T_c , para os primeiros $CIR \cdot T_c$ bits que o CV enviar, o comutador de borda marca os pacotes correspondentes como sendo de baixa prioridade.

Por exemplo, suponha que o provedor de serviço Frame Relay use um intervalo de medição $T_c=500$ ms. Suponha que a taxa de enlace de acesso seja $R=64$ Kbps e que a CIR atribuída àquele CV em particular seja 32 Kbps. Suponha também, para facilitar, que cada pacote Frame Relay consista em exatamente $L=4000$ bits. Isso significa que a cada 500 ms o CV pode enviar $CIR \cdot T_c / L = 4$ pacotes como pacotes de alta prioridade. Todos os pacotes adicionais dentro do intervalo de 500 ms são marcados como pacotes de baixa prioridade.

Devemos ter em mente que muitos CVs podem provir do roteador fonte e transitar pelo enlace de acesso. É interessante notar que se permite que a soma das CIRs para todos esses CVs exceda a taxa de acesso R . Isso é chamado de excesso de reserva. Como o excesso de reserva é permitido, um enlace de acesso pode transmitir pacotes de alta prioridade a uma taxa de bits correspondente que excede a CIR (mesmo que cada CV individual envie pacotes prioritários a uma taxa que não excede a CIR).

Capítulo 39

Redes Virtuais Locais

39.1 VLANs

39.1.1 Definição

Uma Virtual Lan, ou simplesmente vlan, é um método para se criar redes lógicas independentes dentro de uma rede física. As vlans facilitam a administração da rede separando logicamente segmentos lógicos de uma lan, por exemplo, departamentos distintos. As vlans reduzem o domínio de broadcast, diminuindo o tráfego na rede. Ou seja, os pacotes ARP broadcast enviados por um host A que deseja descobrir o endereço MAC de um host B, ambos de uma mesma vlan V, não serão escutados por um host C que não pertence a vlan V. Uma vlan é uma subrede na qual os computadores não necessariamente precisam estar conectados no mesmo segmento físico. O que torna as vlans extremamente flexíveis é o fato de os administradores da rede podem configurar as vlans através de software.

39.1.2 Protocolo 802.1q

As vlans operam na camada de enlace do modelo de referência OSI, no entanto, os administradores de rede geralmente configuram uma vlan para mapear diretamente uma rede ou subrede IP, o que dá a impressão de que esta é uma tecnologia que envolve a camada de rede. Atualmente, o protocolo IEEE 802.1q é o mais popular para implementação de vlans.

Para desenvolver a tecnologia de vlans, a tarefa mais difícil enfrentada pelo comitê 802 do IEEE foi definir como armazenar o identificador da vlan dentro de um quadro Ethernet. Depois de muita discussão o comitê fez o impensável e mudou o cabeçalho do quadro Ethernet, adicionando uma tag VLAN. Uma mudança do cabeçalho Ethernet só foi possível pois o comitê 802 mostrou que o campo VLAN só é realmente utilizado pelos switches e pontes e não pelas máquinas dos usuários. Assim, a saída para o problema foi a seguinte: se a origem não gerar a tag VLAN, o primeiro switch ou bridge que for capaz de identificar a VLAN para o quadro acrescentará o quadro, enquanto o último dispositivo do percurso removerá a tag para entregar o quadro à máquina de

destino. Embora muitas das novas placas Ethernet já sejam compatíveis com o padrão 802.1q, geralmente o quadro é marcado com a tag vlan por um switch. Dessa forma, apenas os switches precisam ser configurados.

Na verdade, o padrão 802.1q adicionou ao cabeçalho Ethernet um campo chamado Tag Protocol ID (TPID) e um campo Tag Control Information (TCI), que por sua vez é subdividido em três partes que são CFI, Pri, e VID. Um quadro 802.1q é reconhecido quando o campo TPID possui o valor 0x8100. O campo CFI é utilizado para compatibilização com redes Token Ring, enquanto o campo VID é o identificador de vlan de fato. O campo Pri possui 3 bits e é utilizado para implementar mecanismos de prioridade no nível de enlace. O modo como deve ser utilizado o campo Pri é definido no padrão IEEE 802.1p. O novo cabeçalho do Ethernet pode ser visto na figura 39.1.2.

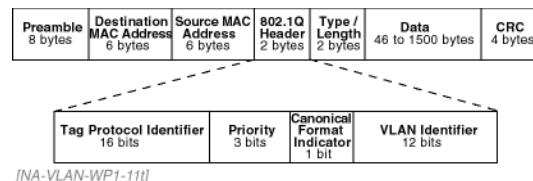


Figura 39.1: Frame 802.1q

Capítulo 40

Redes de Circuito Virtuais

40.1 Redes ATM

Uma rede ATM (Asynchronous Transfer Mode) tem como principal característica o fato de ser orientada a conexão. Portanto, antes do início da transmissão de dados é necessário que todos os roteadores entre a origem e o destino registrem a existência da conexão e reservem recursos para ela. Essas conexões são chamadas de circuitos virtuais, que podem ser dinâmicos (*Switched Virtual Circuit - SVC*) ou permanentes (*Permanent Virtual Circuit - PVC*). O protocolo de sinalização utilizado para estabelecimento das conexões é o Q.2931.

A idéia básica do ATM é transmitir as informações em pequenos pacotes chamados células. As células ATM possuem o tamanho fixo de 53 bytes, sendo 5 para o cabeçalho e 48 para a carga útil. O fato das células terem tamanho fixo permite que todo o roteamento das células seja feito via hardware. A estrutura de uma célula ATM pode ser vista na figura (está faltando ??).

- VPI (*Virtual Path Identifier*): representa o número da rota virtual até o destinatário da informação útil;
- VCI (*Virtual Channel Identifier*): representa o número do canal virtual dentro de uma rota virtual específica;
- PT (*Payload Type*): identifica o tipo de informação que a célula contém: de usuário, de sinalização ou de manutenção;
- CLP (*Cell Loss Priority*): indica prioridade da célula caso sejam necessários descartes por motivos de congestionamento;
- HEC (*Header Error Correction*): Permite correção de erros de um bit e detecção de erros de mais de um bit.

Os campos VPI e VCI são utilizados no processo de comutação das células. É importante destacar que os valores de VPI e VCI são alterados a medida que a célula trafega pela rede. Por ser baseada em circuitos virtuais, a rede ATM garante a ordenação na entrega das células. No entanto, não é capaz de garantir a entrega. Quando um detecta um erro de mais de um bit a partir do campo

HEC, a célula é descartada.

As redes ATM têm seu próprio modelo de referência, diferente do modelo OSI e do modelo TCP/IP. O ATM é um modelo tridimensional, sendo composto não só por camadas, mas também por planos, como mostrado na figura 40.1.

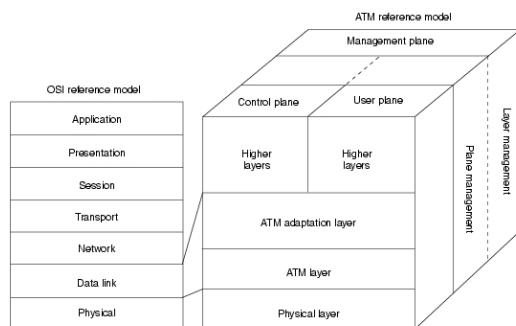


Figura 40.1: Modelo de Referência ATM

O plano de usuário é responsável pelo transporte de dados, pelo fluxo de controle e pela correção de erros, enquanto o plano de controle trata do gerenciamento de conexões. No modelo ATM todas as camadas possuem funcionalidades de usuário e de controle. A descrição de cada uma das camadas é mostrada a seguir:

- Camada Física: provê os meios para transmitir as células ATM. A sub-camada TC (*Transmission Convergence*) mapeia as células ATM no formato dos frames da rede de transmissão (SDH, SONET, PDH, etc.). A sub-camada PM (*Physical Medium*) temporiza os bits do frame de acordo com o relógio de transmissão;
- Camada ATM: é responsável pela construção, processamento e transmissão das células, e pelo processamento das conexões virtuais. Esta camada também processa os diferentes tipos e classes de serviços e controla o tráfego da rede;
- Camada de Adaptação ATM ou AAL (*ATM Adaptation Layer*): é responsável pelo fornecimento de serviços para a camada de aplicação superior. A sub-camada CS (*Convergence Sublayer*) converte e prepara a informação de usuário para o ATM, de acordo com o tipo de serviço, além de controlar as conexões virtuais. A sub-camada SAR (*Segmentation and Reassembly*) fragmenta a informação para ser encapsulada na célula ATM. A camada AAL implementa ainda os mecanismos de qualidade de serviço e sinalização.

Os tipos de serviço oferecidos pelas redes ATM são os seguintes:

- CBR (*Constant Bit Rate*): Garantida uma taxa de transmissão constante. Aplicações típicas que necessitam desse tipo de serviço são telefonia e distribuição de áudio e vídeo (televisão, pay-per-view etc.);

- VBR (*Variable Bit Rate*): Garantida uma taxa média de transmissão e um valor máximo de pico. Aplicações típicas deste serviço são voz com taxa variável de bits e vídeo comprimido (MPEG, por exemplo);
- ABR (*Available Bit Rate*): Garantida uma taxa mínima de transmissão. Aplicado a conexões que transportam tráfego em rajadas que podem prescindir da garantia de banda, variando a taxa de bits de acordo com a disponibilidade da rede ATM. Aplicações típicas deste serviço também são as interligações entre redes e a emulação de LAN's;
- UBR (*Unspecified Bit Rate*): A capacidade de transmissão restante é alojada ao tráfego. Utilizada para tráfego que não possui requisitos de atraso ou *jitter*, como transferência de arquivos e email.

40.2 MPLS - Multiprotocol Label Switching

A medida em que a Internet foi crescendo e os serviços nela oferecidos foram se tornando mais sofisticados foi se tornando necessário desenvolver novas técnicas para garantir níveis de qualidade de serviço altos. Enquanto o IETF se concentrou na concepção das arquiteturas de serviços integrados e serviços diferenciados, vários fabricantes se concentravam desenvolvimento de métodos de encaminhamento melhores. Esse trabalho se concentrou na inclusão de um rótulo no início de cada pacote, de forma que o roteamento pudesse ser feito com base nos rótulos e não mais a partir do endereço de destino. Dessa forma, os engenheiros pretendiam tornar o processo de encaminhamento muito mais eficiente. Uma das motivações da época era o fato de que pacotes IP não podiam ser encaminhados completamente via hardware (hoje já é possível), e com o MPLS, isso poderia ser feito.

O MPLS é um protocolo que permite emular algumas propriedades de redes de comutação de circuitos utilizando para isso a técnica de circuitos virtuais. O MPLS opera entre as camadas de enlace e de rede de forma independente, e por isso muitas vezes é dito ser um protocolo de camada 2.5. Por esse motivo é possível construir switches MPLS capazes de encaminhar tanto pacotes IP, células ATM ou outros. Daí o nome Multiprotocol Label Switching.

O MPLS funciona adicionando aos pacotes um cabeçalho adicional contendo um ou mais rótulos (*labels*), formando uma pilha de rótulos. Cada um dos rótulos é composto por quatro campos que são:

- Label: 20 bits utilizados para indicar o valor do rótulo propriamente dito. É com base no valor desse campo que os pacotes são encaminhados;
- CoS: 3 bits utilizados para indicar a classe de serviço;
- S: 1 bit que quando setado para 1 indica que o rótulo é o último da pilha;
- TTL: 8 bits para indicar o máximo número de hops que um pacote pode dar antes de ser descartado.

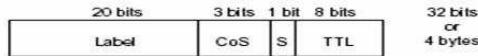


Figura 40.2: Formato do rótulo MPLS

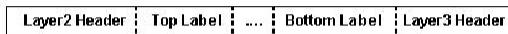


Figura 40.3: Empilhamento de rótulos MPLS entre as camadas 2 e 3

O processo de encaminhamento ocorre da seguinte maneira. Quando um pacote rotulado com MPLS chega a um roteador, o rótulo o campo label é analisado a fim de determinar por qual linha de saída o pacote deve sair. O roteador também é responsável por determinar com qual rótulo o pacote deverá sair. A essa operação de troca de rótulos dá-se o nome de *swap*.

Quando um pacote ainda não rotulado chega a um roteador e precisa passar por um túnel MPLS, o roteador deve primeiro determinar a classe de equivalência do pacote (*forwarding Equivalence Class - FEC*) para então adicionar um ou mais rótulos ao pacote. O FEC é um mecanismo de agregação que permite usar um único rótulo para pacotes de conexões diferentes mas têm o mesmo destino ou pertencem a mesma classe de serviço. Isso permite diminuir o número da tabela de encaminhamento e aumentar a velocidade de encaminhamento.

Além da operação de troca de rótulos (*swap*), o MPLS possui ainda mais duas operações que são a adição de um novo rótulo (*push*) e a remoção de um rótulo (*pop*). A adição de um novo rótulo serve para encapsular o pacote em outra camada de MPLS, permitindo por exemplo, a criação de VPN's. A operação de remoção do rótulo é dita desencapsulamento. Quando o último rótulo é removido diz-se que o pacote deixou o túnel MPLS.

Assim como em outras tecnologias de circuito virtual, o MPLS utiliza protocolos auxiliares para criar e desfazer suas conexões. Atualmente são utilizados o CR-LDP (*Constraint-based Routing - Label Distribution Protocol*) e o RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*). Ambos realizam a distribuição de rótulos com base em restrições de qualidade de serviço, rotas obrigatórias etc.

Outra consideração importante sobre a tecnologia MPLS é que, diferentemente de outras tecnologias de circuito virtual como ATM, uma conexão MPLS é unidirecional. Uma conexão MPLS é também chamada LSP (*Label Switched Path*). Para estabelecimento de uma comunicação bidirecional é necessário estabelecer dois LSP's. Isso é feito de forma independente entre origem e destino, de forma que os dados em um sentido podem seguir por uma rota diferente dos dados no sentido oposto.

Capítulo 41

Arquitetura TCP/IP

41.1 Visão geral

A arquitetura TCP/IP surgiu com a criação de uma rede chamada ARPANET, que foi uma rede criada para manter comunicação entre os órgãos do governo dos EUA e as universidades. A ARPANET cresceu e tornou-se a rede mundial de computadores, a Internet.

A arquitetura TCP/IP trata de um conjunto de protocolos divididos em quatro camadas: Física (host/Rede), Rede (Inter-Rede ou Internet), Transporte e Aplicação; onde cada uma executa um conjunto bem definido de funções de comunicação. Nesta arquitetura não existe uma estruturação formal para cada camada conforme ocorre no modelo OSI. Ela procura definir um protocolo próprio para cada camada, assim como a interface de comunicação entre duas camadas adjacentes. A figura ?? 1 mostra a arquitetura TCP/IP.

41.2 Comparação entre a arquitetura OSI e TCP/IP

Os modelos de referência OSI e TCP/IP têm muito em comum. Os dois se baseiam no conceito de pilha de protocolos independentes. Além disso, as camadas têm praticamente as mesmas funções. Apesar dessas semelhanças, os modelos têm muitas diferenças.

O modelo OSI torna explícita a diferença do conceito de serviço, de interface e de protocolo. Enquanto que o modelo TCP/IP não diferencia com clareza esses conceitos. Por esse motivo, o modelo OSI os protocolos são bem mais encapsulados que os do TCP/IP e podem ser substituídos com relativa facilidade. O modelo TCP/IP não é nem um pouco abrangente e não consegue descrever outras pilhas de protocolo que não a pilha TCP/IP.

Uma diferença explícita está no número de camadas. O modelo OSI possui sete e o TCP/IP possui quatro. Outra diferença está na área de comunicação sem conexão e orientada a conexões. Na camada de rede, o modelo OSI é compatível com a comunicação sem conexão e orientada a conexões. No entanto, na camada de transporte, o modelo aceita apenas comunicação orientada a conexões. O modelo TCP/IP só tem um modo de operação na camada de rede (sem conexão), mas aceita ambos os modos na camada de transporte.

No modelo TCP/IP, a camada física não é realmente uma camada no sentido em que o termo é usado no contexto dos protocolos hierarquizados. Trata-se, na verdade, de uma interface entre a camada de redes e de enlace de dados. E ainda, o modelo não faz distinção entre as camadas físicas e de enlace de dados.

41.3 Camada Física (host/rede)

A arquitetura TCP/IP não defini muito bem o que acontece nesta camada, apenas especifica que o host tem que se conectar à rede utilizando algum protocolo para que seja possível enviar pacotes IP. Esse protocolo não é definido e varia de host para host e de rede para rede.

Esta camada também é chamada de camada de abstração de hardware, pois sua função principal ser uma interface do modelo TCP/IP com os diversos tipos de rede (X25, ATM, FDDI, Ethernet, Token Ring, Frame-Relay, etc.). Como há uma grande variedade de tecnologias de rede, que utilizam diferentes velocidades, protocolos, meios de transmissão, etc., esta camada não é normatizada pelo modelo TCP/IP, o que provê uma das grandes vantagens do modelo: a possibilidade de interconexão e interoperabilidade de redes heterogêneas.

Os protocolos desta camada são:

- PPP (Point-to-Point Protocol): é um protocolo ponto a ponto utilizado para transportar datagramas através de uma conexão serial entre dois dispositivos de rede, por exemplo, entre modems (do usuário e do provedor de Internet). Ele aceita a detecção de erros, negociação de opções, compactação de cabeçalhos e, opcionalmente, a transmissão confiável com o uso de um formato de quadro do tipo HDLC.
- ARP (Address Resolution Protocol): é um protocolo utilizado para descobrir o endereço físico (MAC) de uma máquina a partir de seu endereço IP;
- RARP (Reverse ARP): é um protocolo utilizado descobrir o endereço IP de uma máquina a partir de um endereço físico.

Pode-se dizer que os protocolos ARP e RARP pertencem também à camada Inter-Rede.

41.4 Camada de Inter-Rede

A camada de Inter-Rede é a primeira normatizada pelo modelo TCP/IP. Conhecida também como camada Internet (Rede), esta camada define o protocolo IP (Internet Protocol) responsável pelo endereçamento dos hosts e roteadores. A tarefa desta camada é entregar pacotes IP onde eles são necessários. O roteamento de pacotes é uma questão de grande importância nesta camada, assim como a necessidade de evitar o congestionamento.

Além do protocolo IP, a camada de Inter-Rede define alguns outros protocolos:

- ICMP (Internet Control Message Protocol): é um protocolo utilizado para transmissão de mensagens de controle ou ocorrência de problemas;

- OSPF (Interior Gateway Routing Protocol): é um protocolo de roteamento em um Sistema Autônomo;
- IP (Routing Information Protocol): é um protocolo que permite a troca de informações de roteamento entre gateways utilizando o algoritmo Vector-Distance;
- BGP (Border Gateway Protocol): é um protocolo de roteamento entre Sistemas Autônomos.

41.5 Camada de Transporte

A camada de transporte é o núcleo de toda a hierarquia de protocolos. É uma camada fim a fim, isto é, uma entidade desta camada da máquina de origem só se comunica com uma entidade par da máquina de destino. Sua função é prover uma transferência de dados confiável e econômica entre a máquina de origem e a máquina de destino, independente das redes físicas em uso no momento.

Dois protocolos fim a fim são definidos nesta camada. O primeiro deles, o TCP (Transmission Control Protocol) é um protocolo orientado a conexão confiável que permite a entrega sem erros de um fluxo de bytes, verificando se a ordem e a seqüência dos dados recebidos e enviados estão corretas. O segundo protocolo, O UDP (User Datagram Protocol) é um protocolo sem conexão e não-confiável, ou seja, não oferece nenhuma garantia em relação à entrega dos dados ao destino.

Para distinguir entre várias aplicações, a camada de Transporte associa um identificador a cada processo de aplicação. Esse identificador é chamado de porta. Para uma aplicação "falar" com outra numa máquina remota, é preciso conhecer não apenas o endereço IP da máquina destino, mas também a porta associada a cada aplicação. O UDP e o TCP fornecem um conjunto de portas que permite a múltiplos processos dentro de uma única máquina usarem os serviços providos pelo UDP e TCP simultaneamente. O protocolo TCP utiliza o conceito de sockets para caracterizar uma conexão entre a origem e o destino. O socket consiste no endereço IP da máquina e a porta.

41.6 Camada de Aplicação

É formada pelos protocolos utilizados pelas diversas aplicações do modelo TCP/IP. Esta camada não possui um padrão comum, cada aplicação define o seu próprio protocolo. É esta camada que trata a compatibilidade entre os diversos formatos representados pelos variados tipos de estações da rede.

Os principais protocolos desta camada são:

- TELNET (TeleType Network): é um protocolo utilizado para acessar sistemas remotos por meio de um terminal. Utiliza a porta 23 do protocolo TCP;
- FTP (File Transfer Protocol): é um protocolo utilizado para serviços de transferência, renomeação e eliminação de arquivos, além da criação, modificação e exclusão de diretórios. Utiliza duas conexões TCP: uma para

controle, porta 21, e outra para dado, porta 20. As transferências de arquivos podem ser no modo texto ou binário;

- SNMP (Simple Network Management Protocol): é um protocolo utilizado trafegar informações sobre dispositivos da rede, ou seja, gerenciamento da rede. Utiliza duas conexões UDP: uma para requisições, porta 161, e uma para as mensagens de trap, porta 162;
- DNS (Domain Name Server): é um protocolo utilizado para realizar o mapeamento entre nomes e endereço IP. Utiliza a porta 43 do protocolo UDP para resolução de nomes e a porta 53 do protocolo TCP para transferência de zonas;
- DHCP (Dynamic Host Configuration Protocol): é um protocolo que permite realizar a configuração automática de endereços de hosts em uma rede ou na Internet;
- SMTP (Simple Mail Transfer Protocol): é um protocolo utilizado para enviar mensagens de correio eletrônico. Usualmente, utiliza a porta 25 do protocolo TCP;
- HTTP (HyperText Transfer Protocol): é um protocolo de transferência de mensagens utilizado na WWW. Usualmente, utiliza a porta 80 ou 8080 do protocolo TCP;
- NFS (Network File System): é um protocolo que permite "montar" discos ou parte deles de dispositivos remotamente e operá-los como se fossem locais. Inicialmente este protocolo utiliza a porta 2049 do protocolo UDP, mas a versão NFS4 utiliza a porta 2049 do protocolo TCP;

Capítulo 42

Camada de Aplicação

42.1 Proxy Cache

Um proxy cache, ou simplesmente proxy, é uma entidade de rede que atende requisições HTTP em nome de um servidor Web de origem. Um proxy tem seu próprio disco de armazenagem onde mantém cópias dos objetos recentemente requisitados. Alguns dos objetivos da utilização de um proxy é a redução na utilização dos links de acesso a internet e a diminuição do atraso percebido pelo usuário no atendimento de uma requisição. Os dois principais tipos de proxy são:

- Interceptação: Trabalha interceptando o tráfego da rede de forma transparentemente, não sendo necessária nenhuma configuração adicional nos browsers. São utilizados especialmente pelos ISP's.
- Intermediário: Geralmente utilizados em ambientes que constituem um domínio administrativo (empresas, universidades etc.), uma vez que é necessária a configuração dos browsers. Esse tipo de proxy pode ainda implementar outras funções como autenticação e filtros de conteúdo.

Para que um proxy possa atender as requisições dos usuários de forma correta é necessário que ele determine se um documento que está em cache é válido ou não. Para isso, os servidores proxy utilizam alguns campos do cabeçalho das respostas HTTP. O campo Max-Age, por exemplo, informa por quanto tempo em segundos a resposta será válida. Quando o documento em cache não possui a informação max-age, o proxy pode utilizar o método GET em conjunto com o campo If-Modified-Since (GET condicional) para solicitar ao servidor uma nova cópia do mesmo caso ele tenha sido modificado desde a data definida.

Os proxies podem ainda serem organizados de forma hierárquica, de modo que um proxy possa consultar outros sobre seu conteúdo e, baseado nos tempos de respostas, decide qual cache entregará um dado objeto. Para essa comunicação são utilizados protocolos como o ICP (*Internet Cache Protocol*) e HTCP (*Hyper Text Caching Protocol*). Outro mecanismo de implementar a comunicação entre proxies é utilizando o chamado *Cache-Digest*, que consiste de um sumário do conteúdo do cache de um servidor que é trocado periodicamente

com outros servidores pertencentes à hierarquia.

Parte VII

Gerência de Redes

Capítulo 43

O protocolo SNMP

O SNMP (*Simple Network Management Protocol*) é um protocolo da camada de aplicação que facilita a troca de informações de gerenciamento entre dispositivos de rede. O SNMP é o padrão de fato para gerenciamento de redes.

Uma rede gerenciada por SNMP é composta basicamente por três elementos que são: (i) os dispositivos gerenciados; (ii) os agentes de monitoramento e (iii) os sistemas de gerenciamento de rede (*Network Management Systems* - NMS's).

Um dispositivo gerenciado é qualquer elemento da rede (hub, switch, roteador, servidor etc.) que contenha um agente SNMP. Os dispositivos gerenciados coletem e armazenam informações e as tornam disponíveis para os NMS's através do protocolo SNMP. Essa informações são armazenadas nas chamadas MIB's (Management Information Base).

Um agente é um software que reside em um dispositivo gerenciado e tem conhecimento sobre as informações de gerenciamento e é responsável pela tradução dessas para um formato compatível com o SNMP.

Um NMS executa uma aplicação que monitora e controla os dispositivos gerenciados. Existem duas formas básicas de comunicação entre os agentes e os NMS's.

Na primeira, o NMS envia uma mensagem GET para o agente solicitando o envio de alguma informação. Geralmente os NMS's são configurados para coletar informações dos agentes periodicamente para armazená-las em bases de dados para serem analisados posteriormente. Um NMS pode ainda solicitar a alteração de alguma informação enviando uma mensagem SET ao agente. As mensagens enviadas do NMS para o agente utilizam a porta UDP 161. Para recuperar todos os objetos de uma determinada subárvore da MIB pode ser utilizada a operação GETNEXT.

Na segunda forma, o agente envia uma mensagem para o NMS quando detecta alguma situação pré definida. As mensagens enviadas do agente para o NMS são chamadas *traps*. O protocolo SNMP conta com algumas *traps* nativas, porém os agentes podem ser contruídos para enviar traps em outras

situações não previstas pelo protocolo SNMP. As traps são enviadas utilizando a porta UDP 162.

Um conjunto de dispositivos SNMP (dispositivos gerenciados e NMS's) podem ser agrupados em comunidades. Uma mensagem SNMP originada por um dispositivo SNMP que de fato pertence a comunidade SNMP referenciada é considerada uma mensagem SNMP autêntica.

Atualmente existem duas versões do protocolo SNMP que são SNMPv1 e SNMPv2. O SNMPv2 introduziu duas novas operações que são GETBULK e INFORM. A GETBULK permite o NMS recuperar grandes blocos de dados aumentando a eficiência do protocolo, enquanto a operação INFORM permite a comunicação entre dois NMS's na mesma rede.

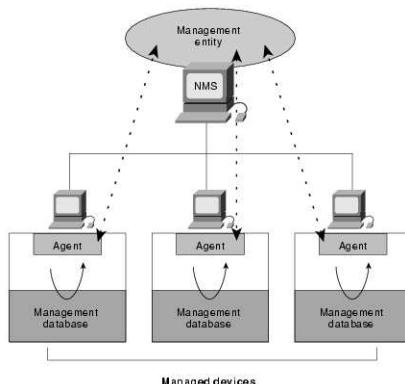


Figura 43.1: Arquitetura SNMP

43.1 Management Information Base

A Management Information Base (MIB) é uma coleção de informações organizada de forma hierárquica como uma árvore de raiz não nomeada. A MIB é composta por objetos gerenciados, cada um deles identificado por um OID (*object identifiers*) único.

Os ID's de nível mais alto pertencem a diferentes organizações de padronização (ISO, ITU etc.), enquanto os ID's de nível imediatamente inferior são alocados para organizações associadas. Fabricantes podem definir ramos na árvore para incluir objetos gerenciados para seus produtos. A estrutura de uma MIB é mostrada na figura 43.2.

Os ramos mais comuns na estrutura de uma MIB são os seguintes:

- .1.3.6.1.2.0 ou (iso.org.dod.internet.mgmt.mib): Caminho para MIB versão I;

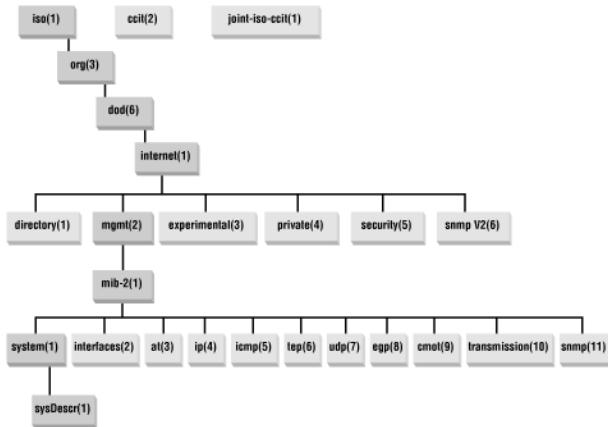


Figura 43.2: Estrutura da MIB

- .1.3.6.1.2.1 ou (iso.org.dod.internet.mgmt.mib-2): Caminho para MIB II, definida pela RFC 1213 para ser utilizada no gerenciamento de redes baseadas na pilha TCP/IP. A MIB-II é subdividida nos 11 grupos de informação mostrados na figura 43.2;
- .1.3.6.1.4.1 ou (iso.org.dod.internet.private.enterprises): Caminho para MIB's proprietárias. A MIB dos equipamentos da Cisco, por exemplo, ficam abaixo de .1.3.6.1.4.1.9 ou (iso.org.dod.internet.private.enterprises.cisco).

Parte VIII

Segurança da Informação

Capítulo 44

Políticas de Segurança de Informação

44.1 Políticas de Segurança

A política de segurança tem por objetivo prover à administração uma direção e apoio para a segurança da informação. A administração deve estabelecer uma política clara e demonstrar apoio e comprometimento com a segurança da informação através da emissão e manutenção de uma política de segurança da informação para toda a organização (ISO/IEC 17799:2000).

Uma política de segurança é a expressão formal das regras pelas quais é fornecido acesso aos recursos tecnológicos da empresa. O principal propósito de uma política de segurança é informar aos usuários, equipe e gerentes, as suas obrigações para a proteção da tecnologia e do acesso à informação. A política deve especificar os mecanismos através dos quais estes requisitos podem ser alcançados.

Outro propósito é oferecer um ponto de referência a partir do qual se possa adquirir, configurar e auditar sistemas computacionais e redes, para que sejam adequados aos requisitos propostos. Portanto, uma tentativa de utilizar um conjunto de ferramentas de segurança na ausência de pelo menos uma política de segurança implícita não faz sentido (RFC 2196).

A política deve especificar as metas de segurança da organização, onde as responsabilidades recaem, e qual o comprometimento da organização com a segurança.

Uma vez que a política é um estatuto, é necessário que a sua elaboração, aprovação e aplicação sigam os ritos internos da instituição na qual será aplicada. O caráter estratégico de uma política de segurança deve garantir que a mesma aborde questões que são essenciais para a corporação como um todo. Cada regra da política serve como referência básica para a elaboração do conjunto de regras particulares e detalhadas que compõem as normas e os procedimentos de segurança.

Com o intuito de tornar a política de segurança um instrumento que viabilize a aplicação prática e a manutenção de uma infra-estrutura de segurança para a instituição, é necessário que a política seja desdobrada em estatutos mais detalhados. Estes estatutos podem ser referidos como políticas específicas, normas,

regras complementares, ou controles. Outros níveis podem existir, tal qual numa hierarquia, sendo que o limite será ditado pelas necessidades e conveniências da instituição para a qual são elaborados as regras de segurança. Cabe ressaltar que, quanto mais baixo o nível hierárquico de um documento de segurança em relação à política, mais detalhado e de caráter operacional será.

É importante lembrar que toda regra aplicada a uma instituição deve estar em consonância com os objetivos fins da mesma. A segurança não é um fim em si mesma, mas um meio para se chegar a um objetivo maior.

A política de segurança como um elemento institucional da organização possui um ciclo de vida indefinido e deve prever todos os mecanismos de defesa contra qualquer ameaça conforme estabelecido no estudo de custos x benefícios. Considerando a mutabilidade de tais elementos e dos próprios objetivos e metas da organização, uma política só apresentará efetividade ao longo do tempo se sofrer constantes reavaliações e atualizações.

ISMS - Information Security Management System, ou Sistema de Gerenciamento da Segurança da Informação (SGSI) - é o resultado da aplicação planejada de objetivos, diretrizes, políticas, procedimentos, modelos e outras medidas administrativas que, de forma conjunta, definem como são reduzidos os riscos para segurança da informação.

44.2 Projeto de Segurança

A estratégia de segurança da informação de uma empresa exige a elaboração de um projeto de segurança (nível mais alto de abstração) que descreva todos os aspectos da segurança da informação na empresa. Um desses aspectos consiste na elaboração de um plano de segurança. O projeto de segurança, segundo Oppenheimer (1999), envolve várias etapas de trabalho:

- Identificação dos ativos da empresa em termos de informações;
- Análise dos riscos de segurança;
- Análise dos requisitos de segurança e compromissos;
- Desenvolvimento de um plano de segurança;
- Definição de uma norma de segurança;
- Desenvolvimento de procedimentos para implantar a norma e uma estratégia de implementação;
- Implementação, gerenciamento e auditoria dos procedimentos de segurança.

44.3 Plano de Segurança

Plano de Segurança é um documento de alto nível que propõe o que uma organização deve fazer para satisfazer os requisitos de segurança, contendo a relação dos serviços de TI disponibilizados, quais áreas da empresa disponibilizam os serviços, quem terá acesso aos serviços, a descrição detalhada de sua implementação, dos procedimentos de controle dos ambientes, incidentes e

contingências. O plano especifica o tempo, as pessoas e outros recursos que serão necessários para desenvolver uma norma de segurança e alcançar a implementação técnica da norma.

O plano deve estar baseado na análise de ativos de redes e riscos. Deve fazer referência à topologia de rede e incluir uma lista de serviços de rede que serão fornecidos, como por exemplo, FTP, Web, correio eletrônico e outros. Esta lista deve especificar quem fornecerá os serviços, quem terá acesso aos serviços, o modo como o acesso será fornecido e quem irá administrar os serviços.

Um dos aspectos mais importantes do plano de segurança é uma especificação das pessoas que devem estar envolvidas na implementação da segurança de rede:

- Serão contratados administradores de segurança especializados?
- Como os usuários finais e seus gerentes estarão envolvidos?
- Como os usuários finais, gerentes e pessoal técnico serão treinados sobre normas e procedimentos de segurança?

Para ser útil, um plano de segurança precisa ter o apoio de todos os níveis de funcionários dentro da organização. É muito importante que a administração corporativa apoie plenamente o plano de segurança. O pessoal técnico da rede e de locais remotos deve se envolver no plano, da mesma forma que os usuários finais (Oppenheimer, 1999).

44.4 Normas de Segurança

Norma de segurança é uma declaração formal das regras às quais as pessoas que têm um determinado acesso à tecnologia e aos ativos de informações de uma organização devem obedecer. (RFC 2196, The Site Security Handbook).

Pode-se definir ainda, norma de segurança como sendo um estatuto no qual estão transcritas regras de nível intermediário, ou seja, entre o nível estratégico e o de descrição de procedimentos, cujo cumprimento visa garantir a segurança das informações e recursos de uma instituição, dentro de um segmento particular do ambiente desta corporação.

A norma de segurança informa aos usuários, gerentes e ao pessoal técnico de suas obrigações para proteger os ativos de tecnologia e informações. A norma deve especificar os mecanismos pelos quais estas obrigações podem ser cumpridas. Da mesma forma que o plano, a norma de segurança deve ter o comprometimento de funcionários, gerentes, executivos e pessoal técnico.

Uma vez desenvolvida, a norma de segurança deve ser explicada a todos pela gerência superior. Muitas empresas exigem que o pessoal assine uma declaração indicando que leu, compreendeu e concorda em cumprir as normas. A norma de segurança é um documento vivo. Pelo fato de as organizações mudarem continuamente, as normas de segurança devem ser atualizadas com regularidade a fim de refletirem novas orientações comerciais e mudanças tecnológicas (Oppenheimer, 1999).

44.4.1 ISO/IEC 17799

A ISO/IEC 17799 é a versão internacional da BS7799, homologada pela International Standardization Organization em dezembro de 2000. A NBR ISO/IEC

17799 é a versão brasileira da norma ISO, homologada pela ABNT em setembro de 2001. A norma ISO é rigorosamente idêntica a norma BS7799. A norma brasileira é a tradução literal da norma ISO.

BS7799 - British Standard 7799 - é uma norma de segurança da informação destinada a empresas. Criada na Inglaterra, teve seu desenvolvimento iniciado em 1995, dividindo-se em duas partes: A BS7799-1 e a BS7799-2. A BS7799-1 é a primeira parte da norma que contém uma introdução, definição de extensão e condições principais de uso da norma. Disponibiliza 148 controles divididos em dez partes distintas. É planejada como um documento de referência para implementar "boas práticas" de segurança na empresa.

A BS7799-2 é a segunda parte da norma e tem por objetivo proporcionar uma base para gerenciar a segurança da informação dos sistemas das empresas. Uma empresa que implante a norma BS/ISO acaba por constituir um ISMS. A forma de como implementar um ISMS é descrita na norma BS7799-2.

As normas publicadas pela Organização Internacional de Normalização, a ISO, têm uma grande aceitação no mercado. Um exemplo disso é a norma ISO 9001:2000, que trata da Gestão da Qualidade, considerada como a mais difundida norma da ISO que existe no mundo. No caso da NBR ISO IEC 17799, que é um Código de Boas Práticas para a Segurança da Informação, a sua aplicação é um pouco mais restrita que a ISO 9001:2000, pois ela não é uma norma voltada para fins de certificação.

Entretanto, a NBR ISO IEC 17799 pode ser usada pela maioria dos setores da economia, pois todas as Organizações, independentemente do seu porte ou do ramo de atuação, do setor público ou privado, precisam proteger suas informações sensíveis e críticas.

As principais recomendações da NBR ISO IEC 17799 são organizadas em 11 seções:

- Política de Segurança de Informação;
- Organizando a Segurança da Informação;
- Gestão de Ativos;
- Segurança em Recursos Humanos;
- Segurança Física e do Ambiente;
- Gerenciamento das Operações e Comunicações;
- Controle de Acesso;
- Aquisição, Desenvolvimento e Manutenção de Sistemas de Informação;
- Gestão de Incidentes de Segurança da Informação;
- Gestão da Continuidade de Negócios;
- Conformidade.

Ela permite que uma empresa construa de forma muito rápida uma política de segurança baseada em controles de segurança eficientes. Os outros caminhos para se fazer o mesmo, sem a norma, são constituir uma equipe para pesquisar

o assunto ou contratar uma consultoria para realizar essas tarefas. Ambas as opções são caras e demoradas.

A ISO/IEC 17799:2000 tem como objetivo permitir que companhias que cumprem a norma demonstrem publicamente que podem resguardar a confidencialidade, integridade e disponibilidade das informações de seus clientes.

A ISO/IEC 17799:2000 fornece mais de 127 orientações de segurança estruturadas em 10 títulos principais para possibilitar aos leitores identificarem os controles de segurança apropriados para sua organização ou áreas de responsabilidade. Além de fornecer controles detalhados de segurança para computadores e redes, a ISO/IEC 17799:2000 dá orientações sobre políticas de segurança, conscientização sobre segurança para os funcionários, plano de continuidade dos negócios e requisitos legais.

44.4.2 Família ISO 27000

A série de normas ISO 27000, encabeçadas pela ISO 27001 estão sendo elaboradas para substituir e completar os padrões definidos pela BS7799.

Como forma de dar suporte à implantação da ISO IEC 27001:2005, o Comitê da ISO que trata da segurança de informa decidiu pela criação de uma família de normas sobre Gestão da Segurança da Informação. Esta família foi batizada pela série ISO IEC 27000, a exemplo da série ISO 9000 das normas de qualidade e da série ISO 14000 das normas sobre meio ambiente.

Esta nova família está relacionada com os requisitos mandatórios da ISO IEC 27001:2005, como, por exemplo, a definição do escopo do Sistema de Gestão de Segurança da Informação, a avaliação de riscos, a identificação de ativos e a eficácia dos controles implementados.

As normas da família ISO IEC 27000 são:

27000 O seu objetivo é apresentar os principais conceitos e modelos de SI. Ainda está em processo de desenvolvimento (previsão 2008-2009).

27001 Define requisitos para estabelecer, implementar, operar, monitorar, revisar, manter e melhorar um Sistema de Gestão de Segurança de Informação. É a norma usada para fins de certificação e substitui a norma britântica BS7799-2:2002. É a base para as Organizações que desejam implementar um SGSI.

27002 Guia prático de diretrizes e princípios gerais para iniciar, implementar, manter e melhorar a gestão de SI em uma Organização. Os objetivos de controle e os controles atendem aos requisitos identificados na análise de riscos.

27003 Guia prático para a implementação de um SGSI, baseado na ISO IEC 27001. Ainda está em processo de desenvolvimento (previsão 2008-2009).

27004 Fornece diretrizes com relação a técnicas e procedimentos de medição para avaliar a eficácia dos controles de SI implementados, dos processos de SI e do SGSI. Ainda está em processo de desenvolvimento (previsão 2008-2009).

27005 Fornece diretrizes para o gerenciamento de riscos de SI. Esta norma será constituída por indicações para implementação, monitoramento e melhoria

contínua do sistema de controles. O seu conteúdo deverá ser idêntico ao da norma BS 7799-3:2005 - *Information Security Management Systems - Guidelines for Information Security Risk Management*, a publicar em finais de 2005. A publicação da norma como ISO está prevista para meados de 2007.

27006 Esta norma será referente à recuperação e continuidade de negócio. Este documento tem o título provisório de *Guidelines for information and communications technology disaster recovery services*, não estando calendário a sua edição.

44.4.3 Diferenças entre a ISO/IEC 17799 e a ISO 27001

A norma ISO/IEC 27001 (Information Technology - Information Security Management Systems - Requirements) trata da implantação de um processo de gestão de segurança da informação (ISMS - Information Security Management Systems). Esta norma em conjunto com a ISO/IEC 17799 (Código de Boas Práticas da Gestão de Segurança da Informação) são as principais referências, atualmente, para a quem procura tratar a questão da segurança da informação de maneira eficiente e com eficácia.

A ISO 27001 é uma norma que gera segurança na corporação, ou seja, cria um sistema de segurança (SGSI) dentro de uma Organização. Isso em nenhum momento garante segurança, só torna o ambiente mais controlável. Nesta norma ela contém controles de segurança, apenas cita, por exemplo, o controle A.9.1.1 é Perímetro de Segurança Física, uma definição muito abrangente, e pode ter várias interpretações.

Já a ISO/IEC 17799 possui todos os controles que tem na ISO 27001, só que com explicações e exemplos de implementação. Isso ajuda muito na implementação numa corporação. Um fato importante é que só há certificação ISO 27001, e não a NBR 17799. Além disso, a certificação ISO 27001, contém um descriptivo do escopo, ou seja, quando uma empresa declara que é certificada ISO 27001, ela pode ser certificada apenas no CPD, por exemplo.

Como é uma norma de sistema de segurança, a ISO 27001 também contém controles de outras ISO, por exemplo a ISO 15408 (segurança no desenvolvimento), mas não quer dizer que ao atender completamente a ISO 27001, será atendida a ISO 15408 ou o ITIL, que também possui alguns de seus controles nessa norma.

44.5 Procedimentos de Segurança

Os procedimentos de segurança implementam normas de segurança, definem processos de configuração, login, auditoria e configuração.

Podem-se definir procedimentos de segurança como sendo um estatuto no qual estão transcritas regras de nível operacional, ou seja, em nível de descrição de execução de ações, cujo cumprimento visa garantir a segurança das informações de uma instituição, dentro de um segmento particular do ambiente da corporação.

Devem ser escritos procedimentos de segurança para usuários finais, administradores de redes e administradores de segurança. A divulgação deve ser restrita aos funcionários diretamente envolvidos. Os procedimentos de segurança

devem especificar como controlar incidentes (quer dizer, o que fazer e quem contatar se uma intromissão for detectada), fazer auditoria e desenvolver o plano de contingência com objetivo de manter o negócio sempre ativo.

Os procedimentos de segurança podem ser comunicados aos usuários e administradores em turmas de treinamento lideradas por instrutores qualificados.

44.6 Arquitetura de Segurança

Com base na norma de segurança, é criado um documento denominado política de segurança para ser divulgado em toda empresa. Para implementar a política de segurança deve ser criada uma arquitetura de segurança que consiste na aplicação de todos os mecanismos de controles físicos, lógicos, técnicos e administrativos necessários para a garantia da segurança da informação (ROBERTI, 2001). Com base nessa arquitetura, são criados o plano de contingência e o processo de auditoria.

Uma arquitetura de segurança representa um elenco de recomendações que define os princípios e fundamentos que devem ser observados na implementação de um ambiente considerado seguro em relação aos riscos, impactos e custos ao qual ele está submetido.

A arquitetura de segurança recomendada deve fornecer as bases para os aspectos de segurança dos seguintes elementos: aplicações, dados, comunicação de dados e gerência de sistemas e rede.

Uma arquitetura de segurança deve levar em consideração três elementos básicos: pessoas, o modelo de segurança e a junção de padrões e tecnologias.

É importante salientar que a arquitetura de segurança proposta deve conduzir a implementações que sejam financeiramente possíveis para a organização. Para tanto, a arquitetura deve possuir as seguintes qualidades:

- Ser independente de plataforma operacional, aplicação de rede;
- Ser alavancada por tecnologias de segurança amadurecidas, por exemplo: criptografias e cartão inteligente;
- Estar em conformidade com padrões infacto, como por exemplo a norma ISO/IEC 17799:2000 e CobiT;
- Definir relacionamentos entre os componentes de segurança: autenticação e permissão de acesso, por exemplo;
- Ter performance e disponibilidade dos mecanismos de segurança;
- Possuir um modo consistente de gerenciamento;
- Obter a conscientização de usuários finais.

44.7 Classificação de Informações

Segundo Claudia Dias (Dias, 2000), diferentes tipos de informação devem ser protegidos de diferentes maneiras. Por isso a classificação das informações é um dos primeiros passos para o estabelecimento de uma política de segurança de informações. Um vez classificada a informação, a política pode definir como

tratá-la de acordo com sua classe, escolhendo mecanismos de segurança mais adequados.

A classificação mais comum de informações é aquela que as divide em 04 níveis:

Públicas ou de uso irrestrito As informações e os sistemas assim classificados podem ser divulgados a qualquer pessoa sem que haja implicações para a instituição. Exemplo: serviços de informação ao público em geral, informações divulgadas à imprensa ou pela internet.

Internas ou de uso interno Podem ser chamadas também de corporativas. As informações e os sistemas assim classificados não devem sair do âmbito da instituição. Porém, se isto ocorrer, as consequências não serão críticas. Exemplo: Serviços de informação interna ou documentos de trabalho corriqueiros que só interessam aos funcionários.

Confidenciais Informações e sistemas tratados como confidenciais dentro da instituição e protegidos contra o acesso externo. O acesso a estes sistemas e informações é feito de acordo com sua estrita necessidade, isto é, os usuários só podem acessá-los se estes forem fundamentais para o desempenho satisfatório de suas funções na instituição. O acesso não autorizado a esses dados e sistemas pode comprometer o funcionamento da instituição, causar danos financeiros ou perdas de fatias do mercado para o concorrente. Exemplo: Dados pessoais de clientes e funcionários, senhas, informações sobre vulnerabilidades de segurança dos sistemas institucionais, contratos, balanços entre outros.

Secretas O acesso interno ou externo de pessoas não autorizadas a este tipo de informação é extremamente crítico para a instituição. É imprescindível que o número de pessoas autorizadas seja muito restrito e o controle sobre o uso dessas informações seja total. Exemplo: Informações dos contribuintes, declarações de imposto de renda.

Capítulo 45

Segurança Física e Lógica

45.1 Segurança Física

A segurança física consiste em proteger informações através do uso de controles de acesso não permitindo que elas sejam prejudicadas através da presença indevida de pessoas e de catástrofes naturais. Os controles de acesso podem ser divididos em controles físicos, técnicos e administrativos.

Controle físico Envolve os controles de acessos convencionais como guardas, iluminação, detectores de movimento etc.

Controle técnico Envolve crachás de acesso e dispositivos biométricos.

Controle administrativo Envolve procedimentos de emergência, controle de pessoal (tanto na contratação quanto na demissão), planejamento e implementação de políticas.

Em relação ao ambiente, devem ser considerados os aspectos relacionados ao fornecimento de energia elétrica, umidade e supressão de incêndio. Nesse caso, todos os tipos de controles de acesso acima são possíveis.

45.2 Segurança Lógica

Envolve os mecanismos de controle de acesso. Os controles de acessos são projetos para mitigar vulnerabilidades associadas ao acesso.

Define-se sujeito (*subject*) como sendo a representação do usuário dentro do sistema. O objeto (*object*) representa o recurso computacional cujo acesso é controlado. Operações são realizadas pelos sujeitos do sistema sob seus objetos. Uma permissão é a manifestação de que uma determinada operação é permitida para um determinado objeto; por exemplo: *permissão de leitura do arquivo da fila de impressão*.

45.2.1 Matrizes de acesso, listas de controle de acesso e capabilities

A **matriz de acesso** é organizada com colunas representando os objetos, e os usuários em linhas. Em cada célula da tabela representam-se as permissões

que o respectivo usuário possui sobre o objeto. Um exemplo é a convenção do UNIX, em que a letra *R* representa permissão de leitura, a letra *W* representa permissão de escrita e a letra *X* representa permissão de execução.

Matrizes de acesso podem ser utilizadas para modelar mecanismos de autorização simples. Mas não são recomendados para implementação, já que não escalam bem: um sistema com 50.000 usuários e 300 arquivos precisaria de uma matriz com 15 milhões de células, gerando problemas de espaço e maior possibilidade de erros na administração.

Lista de controle de acesso é uma maneira de simplificar o gerenciamento das permissões de acesso, pois indexam a matriz de controle de acesso pela coluna, indicando que usuários possuem que permissão para cada objeto. Esta lista de controle de acesso é armazenada junto com cada objeto e relaciona quais permissões cada usuário possui naquele objeto. Listas de controle de acesso (ACLs) são extensamente utilizadas em praticamente todos os sistemas operacionais modernos (UNIX, Windows), além de estarem presentes em dispositivos de rede como roteadores e firewalls.

A outra maneira de gerenciar a matriz de acesso é indexando-a pelos usuários, apontando para cada um suas permissões no sistema, sendo conhecida como **capabilities**.

ACLs são simples de implementar, mas difíceis de manter em ambientes de alta rotatividade de usuários ou arquivos, já que é necessário configurá-los para cada objeto. O agrupamento de usuários ajuda, mas não resolve a questão. Como as ACLs são indexadas pelo objeto a ser protegido, é mais custoso saber exatamente que usuários tem acesso a que arquivos do sistema. As vantagens e desvantagens do uso de capabilities são basicamente o contrário das mesmas de listas de controle de acesso. A implementação de capabilities tende a ser um pouco mais eficiente em sistemas operacionais, enquanto é mais difícil saber que usuários tem acesso a um arquivo, já que a informação está espalhada.

Listas de controle de acesso tem gozado de uma popularidade muito maior no mundo comercial de sistemas operacionais que capabilities. Isso se deve ao fato de que o modelo DAC (Discretionary Access Control) é o mais popular nos sistemas operacionais modernos, e ACLs adequam-se bem a este modelo, por facilitar que cada usuário gerencie as permissões de seus objetos no sistema.

45.2.2 Modelos de Controle de Acesso

Modelos de controle de acesso (ou, rigorosamente, modelos de autorização de acesso) definem características primitivas de um determinado conjunto de regras de autorização a serem utilizadas. Essas características influenciam os limites da semântica de autorização que pode ser expressa no modelo e consequentemente a sua implementação. Os principais modelos de controle de acesso hoje são DAC (Discretionary Access Control), MAC (Mandatory Access Control) e RBAC (Role-Based Access Control).

DAC: Discretionary Access Control

Controle de acesso discrecionário tem sua origem no contexto de pesquisa acadêmica em sistemas de tempo compartilhado que surgiram no começo dos anos setenta. DAC é baseado na noção de que usuários individuais são *donos* de objetos e

portanto tem controle (discreção) total em quem deve ter permissões para acessar o objeto. Um usuário transforma-se em dono do objeto ao criá-lo. No modelo discrecionário, se João é dono de um objeto (um arquivo, por exemplo), ela pode conceder a José a permissão de acessá-lo em um modo qualquer de operação. Posteriormente, ele pode revogar essa permissão a qualquer momento. O princípio básico de DAC é possessão do objeto pelo usuário que o criou.

Atualmente, o DAC é o modelo mais popular de controle de acesso, pela sua utilização em grande escala em sistemas operacionais comerciais. Todas as variantes do UNIX, o Netware e a série Windows NT, 2000 e XP utilizam o modelo DAC como seu modelo básico de controle de acesso. Estes sistemas operacionais utilizam extensamente a técnica de listas de controle de acesso para conceber a implementar as suas checagens de autorização, dispondo também do conceito de grupos de usuários para facilitar na administração e concessão de permissões.

MAC: Mandatory Access Control

Enquanto o ponto-chave do DAC é o fato de que os usuários são considerados donos do objeto e portanto responsáveis pelas suas permissões de acesso, o modelo mandatório prevê que usuários individuais não têm escolha em relação a que permissões de acesso eles possuem ou a que objetos podem acessar.

Neste modelo, os usuários individuais não são considerados donos dos objetos, e não podem definir suas permissões, isso é realizado pelos administradores do sistema.

O modelo MAC é conhecido, a tal ponto de ser às vezes confundido, pela sua utilização em políticas de acesso multinível, em que se deseja controlar o fluxo de informações em um sistema. Em geral, se quer garantir que a informação só flua em um determinado sentido: por exemplo, de níveis mais baixos de confidencialidade para níveis maiores de confidencialidade, nunca de níveis mais altos para níveis mais baixos.

DAC e MAC na atualidade

Tanto os modelos DAC e MAC são utilizados atualmente, o DAC em maior escala, estando presente em diversos sistemas operacionais comerciais como o UNIX, Windows e Netware. Implementações de sistemas MAC são comuns em ambientes militares e de mainframe.

Apesar da sua popularidade, eles apresentam problemas próprios que estão possibilitando o crescimento de outro modelo de acesso, o RBAC a crescer, visando resolver estas questões.

O MAC, apesar de ser reconhecido genericamente como mais controlável e potencialmente mais seguro que DAC não tem obtido grande uso fora dos circuitos militares. Isso se deve principalmente pela dificuldade em adaptar fluxos de negócio e hierarquia comerciais ao modelo formal. É imprático implantá-lo em sistemas que não sejam militares, pelo custo de administração e de overhead que seria gerado.

O DAC, por sua vez, goza de grande popularidade no mundo comercial, mas tem em seu maior problema a questão da dificuldade no gerenciamento das permissões. Sistemas operacionais modernos de rede possuem milhares de

usuários e potencialmente milhões de arquivos espalhados pelos seus sistemas. O gerenciamento das permissões de cada um destes objetos em uma escala como esta não é um problema simples de se resolver, já que cada objeto possui sua própria informação de acesso individual.

Por exemplo, quando um usuário é removido do sistema, costuma ser necessário remover suas informações de acesso a todos os objetos em que possuía o acesso. Da mesma forma, quando um objeto novo é criado, é necessário definir as suas permissões. Em grandes sistemas computacionais multi-usuário, essas questões pesam bastante.

Uma outra desvantagem tanto do MAC quanto do DAC é que é complicado estender suas regras de acesso para incluir mecanismos de proteção e exceções que acontecem normalmente em sistemas. Como exemplo, toma-se o caso de um sistema hospitalar. Expressar a seguinte regra em sistemas com MAC ou DAC tradicionais é difícil: *Na maior parte do tempo, cada médico só tem acesso aos dados dos seus pacientes. Quando estiver no seu turno na UTI, ele terá também acesso aos pacientes que estão internados na UTI. Ao acabar seu turno, perde este acesso adicional.* A complexidade de se expressar esse tipo de comportamento faz com que acabe-se configurando os sistemas com proteções mais relaxadas do que o estritamente necessário.

RBAC: Role-Based Access Control

RBAC foi projetado para gerenciar centralmente privilégios ao prover uma camada de abstração, conhecida como role (papel), mais alinhado à estrutura da organização. A noção central de RBAC é que permissões são associadas à papéis, e usuários são associados aos seus papéis corretos na organização. Isso simplifica bastante a administração e gerenciamento de permissões de acesso em grandes organizações. Papéis são criados para as várias funções de negócio da organização, e os usuários são associados a esses papéis de acordo com suas responsabilidades e qualificações. Usuários podem ser facilmente reassociados de um papel para outro. Papéis podem receber novas permissões de acesso à medida que novas aplicações ou funcionalidades são adicionadas ao sistema, e permissões podem ser revogadas sempre que necessário.

Um papel é apropriadamente entendido como uma construção semântica à volta do qual a política de controle de acesso é formulada. A coleção particular de usuários e permissões associadas a um papel é transitória. O papel é mais estável porque as atividades e funções da organização em geral mudam menos do que o conjunto de usuários ou de permissões.

Isso torna a administração de um sistema RBAC muito mais fácil e escalável do que a de um sistema DAC, por exemplo, que precisa associar usuários a objetos diretamente, sem a construção do papel entre os dois, atuando como componente estabilizador.

Além do forte motivo de facilitar o gerenciamento das permissões de acesso, um outro ponto motivador de RBAC é a sua flexibilidade de adaptação a regras de acesso particulares de cada sistema, através do recurso de **constraints**. Constraints são predicados que, aplicados a relações e funções do modelo, retornam um valor *aceito* ou *não aceito*. Isso permite expressar, na política de acesso do sistema, restrições como a separação de deveres, em que um mesmo usuário não pode subverter a segurança do sistema ao exercer dois papéis conflitantes ao mesmo tempo. Por exemplo, um mesmo usuário não poderia ser ao mesmo

tempo o gerente de compras (que toma a decisão de realizar uma compra), e o gerente financeiro (que passa o cheque da compra), já que isso poderia abrir espaço para fraudes em que ele indicaria uma compra fraudulenta e autorizaria a sua fatura por conta própria. A separação de deveres é um princípio clássico de segurança, utilizado extensamente no mundo dos negócios, e é possível de ser expressado como regra de acesso em sistemas RBAC.

Capítulo 46

Backup de Dados

Existem várias formas de se garantir a disponibilidade da informação, a mais importante é a cópia destes dados em local seguro, ou seja, o backup de dados. Os backups podem ser classificados em três tipos:

Backup total Realiza uma cópia de todos os dados para a mídia, não importando o conteúdo do último backup. seja, o atributo de arquivamento é desmarcado ou redefinido. Uma fita atualizada de backup total pode ser usada para restaurar um servidor completamente em um determinado momento.

Backup incremental Salva os arquivos que foram alterados desde o último backup. Neste processo o novo arquivo é armazenado na mídia e o arquivo original não é removido da mídia. No processo de restauração devemos ter o último backup completo e dos os backups incrementais desde então.

Backup diferencial Copia todos os arquivos que foram alterados desde o último backup completo, por este motivo ocupa mais espaço nas mídias de backup e é mais lento de ser gerado, contudo é mais fácil de recuperá-lo. Para restaurar os dados a partir deste tipo de backup deve-se ter em mãos apenas o último backup completo e o último backup diferencial.

Backup delta Só faz a cópia dos dados reais que foram modificados nos arquivos, é um processo de backup mais rápido e que ocupa menos espaço nas mídias de backup, contudo o processo de restauração é mais lento e complexo.

46.1 Meios de Armazenamento

Definida as necessidades básicas a serem atendidas devemos selecionar um do tipos de armazenamento, que podem ser: on-line, Próximos e off-line.

As mídias de **armazenamento on-line** consistem em discos rígidos ou arrays de discos. Estas mídias fornecem uma disponibilidade em tempo real e são normalmente utilizados para fornecer uma forma alternativa de armazenamento. Estas mídias não substituem os backups offline.

O **armazenamento próximo** é formado por *Jukeboxes* óticos e cópias locais, que estão rapidamente acessíveis, normalmente fazem uso de robôs para

gerenciarem as mídias fornecendo um acesso rápido aos dados quando o serviço on-line não está disponível.

Já o **armazenamento off-line** consiste no arquivamento de mídias fora da rede de dados em um local seguro e protegido contra roubo, catástrofes naturais e outros ameaças. Sempre que possível as mídias devem ser armazenadas em local geográficamente diferente e fora das instalações comerciais da empresa.

Para a realização deste tipos de backup podem ser utilizadas três tipos de mídias diferentes: fitas/discos magnéticos, armazenamento ótico e arrays de disco.

As **fitas magnéticas** são as mídias mais comuns, mais baratas utilizadas nos backups off-line, mas por outro lado são as mais lentas e que ocupam um grande espaço. Seus principais tipos são: 8mm, Travan, DLT, DAT e Magstar.

O **armazenamento ótico** é muito popular em ambientes onde a velocidade e a confiabilidade são as maiores preocupações, estes ambientes fazem uso de servidores com *jukeboxes* óticos de alta disponibilidade que são soluções caras porem muito eficientes.

Os **arrays de discos** ou simplesmente RAIDs (*Redundant Array of Independent Disks*) são um subsistema de discos rígidos que melhoram o desempenho e a tolerância a falhas, uma vez que os dados são gravados em mais de um disco ao mesmo tempo. Estas soluções podem ser tanto implementadas em software quanto em hardware. Neste caso quando uma unidade de disco falha o administrador do sistema pode substituí-la, em alguns casos, sem parar o funcionamento do servidor.

A solução de RAID fornece um melhor desempenho e tolerância a falhas, mas de forma alguma substitui o processo de backup off-line. Vale lembrar que dois ou mais discos podem falhar ao mesmo tempo, perdendo o acesso total aos dados armazenados no array.

Outra solução de proteção aos dados é o HSM (*Hierarchical Storage Management*), que é um sistema automatizado para o gerenciamento de dados e espaço em disco, muito utilizado em mainframes. Esta solução monitora a capacidade das unidades e move os dados para as mídias de armazenamento próximo ou offline, mais lentas.

O HSM pode mover os dados segundo sua idade, freqüência de uso ou baseado em outros critérios, permitindo neste modo uma migração de dados automática. Esta solução é relativamente cara e difícil de ser implementada.

Capítulo 47

Vírus e Ataques

Conceito básicos:

Cracker Termo usado para designar quem quebra um sistema de segurança, de forma ilegal ou sem ética. Este termo foi criado em 1985 pelos hackers em defesa contra o uso jornalístico do termo hacker.

Hacker Habitualmente (e erradamente) confundido com cracker, um hacker é um expert ou Problem Solver, aquele que apresenta soluções para problemas técnicos relativos à Internet.

White Hat (aka hacker ético) Hacker em segurança, utiliza os seus conhecimentos na exploração e detecção de erros de concepção, dentro da lei.

Black Hat (aka cracker ou dark-side hacker) criminoso ou malicioso hacker, um cracker.

Gray hat Tem as habilidades e intenções de um hacker de chapéu branco na maioria dos casos, mas por vezes utiliza seu conhecimento para propósitos menos nobres.

Script Kiddie Antigamente chamado de Lammer, é um indivíduo que não tem domínio dos conhecimentos de programação. É pouco experiente, com poucas noções de informática, porém tenta fazer-se passar por um cracker a fim de obter fama, o que acaba gerando antipatia por parte dos hackers verdadeiros.

Newbie Newbie, Noob ou a sigla NB, é aquele jovem aprendiz de hacker que possui uma sede de conhecimento incrível, pergunta muito e é ignorado e ridicularizado na maioria das vezes, ao contrário dos lammers não tenta se pôr acima dos outros.

Phreaker Hacker especialista em telefonia móvel ou fixa.

Vírus É um programa capaz de infectar outros programas e arquivos de um computador. Para realizar a infecção, o vírus embute uma cópia de si mesmo em um programa ou arquivo, que quando executado também executa o vírus, dando continuidade ao processo de infecção. Para que um computador seja infectado por um vírus, é preciso que de alguma maneira um programa previamente infectado seja executado.

Worms É um programa completo capaz de se propagar automaticamente através de redes, enviando cópias de si mesmo de computador para computador. Sua propagação se dá através da exploração de vulnerabilidades existentes ou falhas na configuração de softwares instalados em computadores. O worm pode trazer embutido programas que geram algum tipo de problema ou que tornam o computador infectado vulnerável a outros ataques. Um worm pode provocar danos apenas com o tráfego de rede gerado pela sua reprodução.

Cavalos de Tróia (Trojan) É um programa que além de executar funções para as quais foi aparentemente projetado, executa também outras funções normalmente maliciosas e sem o conhecimento do usuário como: alteração ou destruição de arquivos; furto de senhas e números de cartões de crédito; inclusão de backdoors. Não se replica, não infecta outros arquivos, ou propaga cópias de si mesmo automaticamente. Necessita ser explicitamente executado. Exemplos comuns de cavalos de tróia são programas que você recebe ou obtém de um site e que dizem ser jogos ou protetores de tela.

Traps São como backdoors. Garantir uma forma de retornar a um computador comprometido, sem precisar recorrer aos métodos utilizados na realização da invasão. A forma usual de inclusão de um backdoor consiste na adição de um novo serviço ou substituição de um determinado serviço por uma versão alterada, normalmente incluindo recursos que permitam acesso remoto (através da Internet).

Spyware/Adware Spyware (Software Espião) e Adware (Publicidade não desejada). Spyware são arquivos ou aplicativos que são instalados em seu computador, algumas vezes sem seu consentimento ou autorização, ou mesmo depois que você aceita as *Condições de Uso*. Os Spyware monitoram e capturam informações das atividades dos usuários, as enviando para servidores onde são armazenadas para fins em geral comerciais. Tais informações serão posteriormente vendidas a provedores de produtos e serviços como maillings. Estes provedores utilizam-se destas informações para difundir informações na forma de spam. Podem ser maliciosos, incluindo programas que capturam as informações de tudo o que é digitado no teclado. Adware, semelhante aos spyware, são aplicativos instalados da mesma forma que no caso anterior, fazendo com que banners publicitários de serviços e produtos aparecem na sua telinha.

DoS/DDoS O objetivo de tais ataques é indisponibilizar o uso de um ou mais computadores, e não invadi-los. Nos ataques de negação de serviço (DoS - Denial of Service) o atacante utiliza um computador para tirar de operação um serviço ou computador conectado à Internet. No DDoS (Distributed Denial of Service) constitui um ataque de negação de serviço distribuído, ou seja, um conjunto de computadores é utilizado para tirar de operação um ou mais serviços ou computadores conectados à Internet. Normalmente estes ataques procuram ocupar toda a banda disponível para o acesso a um computador ou rede, causando grande lentidão ou até mesmo indisponibilizando qualquer comunicação com este computador ou rede.

IP Spoofing O IP spoofing consiste na troca do IP original por um outro, podendo assim se passar por um outro host. A pessoa que irá realizar o spoofing tem então dois problemas na verdade: o de alterar o ip origem, mais simples de resolver, e o de manter a seqüência de números, que por serem geradas arbitrariamente, complica em muito esta tarefa. As seqüências de números são geradas no TCP para a comunicação entre os hosts.

Flooding O TCP Flood SYN attack tira vantagem do comportamento do 3 way handshake efetuado pelo protocolo TCP no processo de uma conexão. O attacker faz um pedido de conexão para o servidor da vítima com pacotes que carregam o endereço falsificado de IP da fonte (método IP spoofing). Como resultado o servidor da vítima perde tempo e recursos de máquina que poderiam estar sendo usados para outros processos. O UDP Flood attack simplesmente envia pacotes de UDP randomicamente para todas as portas do servidor da vítima. Ping Flood attack é uma técnica de attack que tenta saturar uma conexão de Internet através do envio contínuo de uma série de pings originados tipicamente em redes de alta velocidade para redes de baixa velocidade.

Além dos mecanismos mais tradicionais de infecção tais como setor de boot, macros e memória, três vetores também se destacam: correio eletrônico, compartilhamento de arquivos e falhas no sistema operacional. Curiosamente enquanto os vírus antigos faziam uso de mecanismos extremamente complexos para se infectar, hoje em dia vários vírus fazem uso dos próprios recursos do sistema operacional.

47.1 Estratégias de combate à pragas eletrônicas

A estratégia de mais longo prazo é a prevenção. Usuários, suporte e administradores devem ser alertados para os riscos de vírus de computador e mecanismos eficazes, em acordo com a política de segurança, que deve ser colocada em prática.

As tarefas mais comuns que devem ser realizadas são a análise e remoção de compartilhamento de redes Microsoft, análise e atualização do sistema operacional e análise e atualização do sistema de antivírus do usuário.

Campanhas de conscientização também são importantes, os usuários devem ser alertados sobre os perigos existentes em determinadas ações, como baixar arquivos executáveis por e-mail.

No caso em que uma reação é necessária, não basta somente remover a praga eletrônica. É importante registrar o incidente e analisar qual fato levou à infecção e tomar as medidas cabíveis para a remoção da vulnerabilidade. Também é importante que equipes de resposta à emergência sejam criadas dentro da estrutura do órgão.

47.1.1 Antivírus

Todos os antivírus agem de forma semelhante. Existem dois métodos básicos usados para combater vírus. O primeiro consiste em manter nos antivírus um grande banco de dados onde ficam registradas todas as assinaturas (parte do

vírus que o caracteriza) de vírus conhecidos. Daí a importância de manter seu antivírus atualizado, pois a cada dia surgem centenas de novos vírus. Assim, quando procuramos vírus no sistema, na verdade, o que estamos fazendo é comparar cada arquivo nosso com a assinatura dos vírus registrados.

A segunda forma de proteção é conhecida como inoculação, que nada mais é que a criação de um banco de dados contendo as principais informações (tamanho, data de criação e data da ultima alteração) sobre os arquivos inoculados. Assim, cada vez que procuramos por vírus no sistema, o programa antivírus compara as informações do banco de dados criado com as que estão no disco. Se houver alguma diferença é emitido um alerta. Mas note que não é qualquer arquivo que deve ser inoculado, uma vez que arquivos de dados sempre são alterados. Os arquivos executáveis, DLLs e arquivos de sistema são exemplos de arquivos que devem ser inoculados, pois são as principais vítimas de vírus e não mudam seu conteúdo com freqüência.

Capítulo 48

Princípios de Criptografia

Em um processo de comunicação, uma mensagem pode ser definida como um conjunto de informações que um remetente deseja enviar para um ou mais destinatários. O processo de modificar uma mensagem de forma esconder seu conteúdo é chamado encriptação. A ciência que estuda a encriptação e decriptação de mensagens é chamada Criptografia. No início, a criptografia era utilizada com o único intuito de garantir confidencialidade. Somente as pessoas que conhecessem o processo de criptografia utilizada poderiam ler a mensagem. Atualmente, a criptografia é utilizada para prover também as seguintes garantias:

- Autenticação: Provar a identidade dos participantes em um processo de comunicação;
- Integridade: Permitir que o receptor verifique se a mensagem não foi alterada ou corrompida ao longo do caminho;
- Incontestabilidade: Impedir que um remetente negue o envio de uma mensagem. A incontestabilidade também é chamada não-repúdio. Em outras palavras, é possível provar para um terceiro que uma mensagem só pode ter sido gerada por um remetente específico.

Um algoritmo de criptografia é uma função matemática utilizada para encriptar e decriptar uma mensagem. Inicialmente, o segredo da comunicação era baseado no segredo do algoritmo de criptografia, de forma que para que duas se comunicassem de forma segura, ambas precisavam conhecer o algoritmo. Nos sistemas criptográficos atuais o segredo de comunicação é baseado no segredo de uma chave de criptografia. A chave de criptografia é passada como parâmetro para o algoritmo pra que esse possa criptografar ou descriptografar uma mensagem. Neste modelo, o algoritmo de criptografia é público, e somente aqueles que conhecem a chave de criptografia utilizada para encriptar uma mensagem são capazes de decriptá-la.

O conjunto de valores que uma chave pode assumir é chamado espaço de chaves (*keyspace*). Na Internet, por exemplo, atualmente são utilizadas chaves com comprimento de até 4096 bits, o que permite escolher uma chave em um conjunto de 2^{4096} elementos. Portanto, utilizando uma chave com esse comprimento seria possível criptografar uma mensagem de 2^{4096} formas diferentes

utilizando o mesmo algoritmo.

48.1 Tipos de Criptografia

As técnicas de criptografia são divididas em três tipos principais que são: (i) Funções Hash; (ii) Criptografia Simétrica e (iii) Criptografia Assimétrica. O tipo de garantia (autenticação, integridade, não-repúdio e confiabilidade) que se deseja no processo de comunicação é o que determina qual o tipo de criptografia que se deve utilizar. Os tipos de criptografia são descritos a seguir:

- Funções Hash: Funções matemáticas que transformam um texto em uma sequência de caracteres de tamanho fixo (128, 160, 512 bits, por exemplo) independente do tamanho do texto. A sequência de caracteres geradas é conhecida como resumo da mensagem (*message digest*). A segurança das funções hash se baseiam no fato delas serem funções só de ida. A saída do processo de uma função hash não é dependente da entrada de uma forma clara, o que na prática torna impossível alterar uma mensagem de modo que o mesmo hash seja gerado e mensagem continue fazendo sentido. Boas funções hash também devem garantir que seja computacionalmente impossível gerar a mensagem original a partir de seu hash;
- Criptografia Simétrica: É composto por uma chave e um algoritmo que pode ser executado no modo de encriptação, para cifrar uma mensagem, ou decriptação, para decifrá-la. A chave utilizada nos processos de encriptação e decriptação é a mesma e deve ser mantida em segredo entre as partes comunicantes. Os algoritmos de criptografia simétrica geralmente são baseados em operações de *shift* e XOR, o que permite que sejam extremamente eficientes e de fácil implementação em hardware;
- Criptografia Assimétrica: É composto por um algoritmo e um par de chaves pública e privada geralmente denotadas por K^+ e K^- respectivamente. Uma mensagem cifrada com a chave K^+ só pode ser decifrada com a chave K^- e vice-versa. Dessa forma, se Bob deseja garantir que sua mensagem só poderá ser lida por Alice, então ele deve cifrar a mensagem com a chave pública de dela. Como Alice é, em tese, a única detentora de sua chave privada, ela será a única capaz de decifrar a mensagem enviada por Bob. Dessa maneira pode-se garantir a confidencialidade da mensagem. Repare que nem mesmo Bob será capaz de decifrar a mensagem. Bob pode ainda cifrar uma mensagem com sua própria chave privada e enviar para Alice. Nesse caso, para decifrar a mensagem Alice deverá utilizar a chave pública de Bob. Dessa forma, Alice poderia garantir que foi realmente Bob quem enviou a mensagem e não um impostor. Algoritmos de criptografia assimétrica geralmente são baseados em operações de fatoração, exponenciação e logaritmos de grandes números, o que os torna muito mais lentos do que os algoritmos simétricos. A criptografia de chave assimétrica é a base do sistema de criptografia de chave pública, que será discutido mais adiante;

48.2 Algoritmos de Criptografia Simétricos

Três exemplos de cifras (método de criptografia) do tipo chave simétrica são:

Cifra de César cada letra do texto cifrado recebe a letra do texto aberto mais uma constante, com rotação no alfabeto.

Cifra Monoalfabética Cada letra do texto aberto é substituída por uma outra única letra.

Cifra Polialfabética Usa-se várias cifras de César para se levar em consideração a posição da letra no texto. Ou seja, a constante de César se modifica em função da posição da letra no texto aberto.

Os algoritmos simétricos utilizados na prática são:

DES (Data Encryption Standard) O DES utiliza uma chave de 56 bits e opera em blocos de 64 bits de mensagem. Foi projetado inicialmente para ser utilizado em componentes de hardware, nos dias atuais, ele é usado na Internet em conexões Web segura, pois o SSL se utiliza do DES. Ele é um algoritmo seguro para a maioria das aplicações, entretanto, em aplicações altamente secretas, ele não deve ser usado, pois existe o perigo de violação.

3DES São usados 3 estágios e duas chaves. No primeiro estágio, o texto simples é criptografado com chave K1 da maneira usual do DES. No segundo estágio, o DES é executado no modo de descriptografia, com o uso de uma chave K2. Por fim, outra criptografia é feita com a chave K1. São utilizadas 2 chaves porque os criptógrafos concordam que 112 bits serão suficientes para aplicações comerciais durante um tempo. O uso de 168 bits só criaria overhead desnecessário de gerenciar e transportar outra chave, com pouco ganho real. Ao utilizar dessa maneira (criptografia, descriptografia, criptografia), um computador que utiliza a criptografia tripla pode se comunicar com outro que utiliza a criptografia simples apenas definindo $k1=k2$.

RC2 e RC4 Mais rápidos do que o DES, esses códigos podem se tornar mais seguros com o simples aumento do tamanho das chaves, O RC2 pode substituir perfeitamente o DES com a vantagem de ser 2 vezes mais rápido, já o RC4 fica 10 vezes mais rápido. Algumas chaves são fracas. Suas chaves são de 1 a 2048 bits.

IDEA (International Data Encryption Algorithm) Criado em 1991. Ele foi projetado para ser facilmente programado, é forte e resistente a muitas formas de criptoanálise. Possui chave de 128 bits, porém é patenteado.

AES (Advanced Encryption Standard) Foi promovido em um concurso um novo padrão cujas propostas eram: 1 - O algoritmo teria de ser uma cifra de bloco simétrica; 2 - Todo o projeto teria de ser público; 3 - Devem ser admitidos tamanhos de chaves iguais a 128, 192 e 256 bits; 4 - Teriam de ser possíveis implementações de software e de hardware; 5 - O algoritmo teria de ser público ou licenciado em termos não-discriminatórios. Os finalistas: 1 - Rijndael; 2 - Serpent; 3 - Twofish; 4 - RC6; 5 - MARS.

Cifra	Autor	Comprimento da chave	Comentários
Blowfish	Bruce Schneier	1 a 448 bits	Velho e lento
DES	IBM	56 bits	Muito fraco para usar agora
IDEA	Massey e Xuejia	128 bits	Bom, mas patenteado
RC4	Ronald Rivest	1 a 2048 bits	Algumas chaves são fracas
RC5	Ronald Rivest	128 a 256 bits	Bom, mas patenteado
Rijndael	Daemen e Rijmen	128 a 256 bits	Melhor escolha
Serpent	Anderson, Biham, Knudsen	128 a 256 bits	Muito forte
3DES	IBM	168 bits	Segunda melhor escolha
Twofish	Bruce Schneier	128 a 256 bits	Muito forte; amplamente utilizado

Tabela 48.1: Alguns algoritmos criptográficos de chave simétrica

48.3 Algoritmos de Criptografia Assimétricos

A criptografia de chave pública ou criptografia assimétrica, foi criada em 1970. Esse método funciona com uma chave para criptografar, e outra para descriptografar a mesma mensagem. No sistema de chave pública, cada pessoa tem que ter duas chaves, uma que fica publicamente disponível, e outra, que deve ser mantida em segredo. O algoritmo que se mantém até hoje é o RSA, que é patenteado pela RSADSI (RSA Data Security Incorporated) nos Estados Unidos. Para entender como funciona, observe abaixo:

- As pessoas (A) e (C), escrevem mensagens, utilizando a chave pública da pessoa (B), note que, a partir desse momento somente ela, poderá ler as mensagens;
- As mensagens são enviadas a pessoa (B) através da Internet;
- A pessoa (B), recebe as mensagens de (A) e (C), na qual ela usa a chave privada para descriptografar;
- A pessoa (B), lê as mensagens, e se, tiver que responde-las, deverá usar as chaves públicas de criptografia de (A) e ou (C).

Nesse momento, é importante enfatizar que o sigilo da chave privada é muito importante, pois, a criptografia assimétrica, se baseia no fato de que a chave privada, é realmente privada, por isso, somente seu detentor deve ter acesso. A descrição do algoritmo mais utilizado segue abaixo:

RSA (Rivest, Shamir, Adleman) O método se baseia em alguns princípios da teoria dos números. De forma resumida:

- Escolha dois números primos extensos, p e q (geralmente, de 1024 bits)
- Calcule $n = p \times q$ e $z = (p-1) \times (q-1)$
- Escolha um número d tal que z e d sejam primos entre si
- Encontre e de forma que $e \times d = 1 \text{ mod } z$

Com esses parâmetros calculados com antecedência, estamos prontos para começar a criptografia. Divida o texto simples (considerado um string de bits) em blocos, de modo que cada mensagem de texto simples P fique no intervalo $0 \leq P < n$. Isso pode ser feito agrupando-se o texto simples em blocos de k bits, onde k é o maior inteiro para o qual a desigualdade $2k < n$ é verdadeira.

Para criptografar a mensagem P , calcule $C = Pe(modn)$. Para descriptografar C , calcule $P = Cd(modn)$. É possível provar que, para todo P na faixa especificada, as funções de criptografia e descriptografia são inversas entre si. Portanto, a chave pública consiste no par (e,n) e a chave privada consiste em (d,n) .

Lento demais para codificar grande volume de dados, mas amplamente utilizado para a distribuição de chaves. A segurança do sistema está baseado na dificuldade de fatorar números grandes.

48.4 Técnicas de Quebra de Criptografia

Basicamente, os três tipos de ataque destinado à quebra de criptografia são:

Ataque Exclusivo a Texto Cifrado Não se conhece o texto aberto. Análise estatística geralmente é utilizada para se extrair algo do texto cifrado.

Ataque com Texto Aberto Conhecido Se conhece o texto cifrado e pelo menos parte do texto aberto. A descoberta da chave fica menos complexa em relação ao caso anterior.

Ataque com Texto Aberto Escolhido É possível escolher o texto aberto a ser encriptografado. Desta forma, a descoberta da chave fica mais facilitada.

Capítulo 49

Autenticação

49.1 Autenticação de Mensagens

Autenticação de mensagens é um mecanismo ou serviço utilizado para verificar a integridade de uma mensagem. Os métodos mais comuns são MAC (*message authentication code*) e funções *hashes* seguras. O MAC requer o uso de chave secreta e é produzido através do uso dessa chave em uma mensagem de tamanho variável. Uma função *hash* mapeia uma mensagem de tamanho variável em um *hash* de tamanho fixo e para garantir o não-repúdio é utilizado em conjunto com uma chave privada.

Quando o método de autenticação garante o não repúdio, então trata-se de uma assinatura digital. O MAC não provê assinatura digital, uma vez que o remetente e o receptor compartilham a mesma chave.

As funções *hash* são utilizadas para a geração de sumários de mensagens *message digests*. Com os sumários de mensagens é possível fornecer autenticação sem que haja sigilo e tem quatro propriedades:

1. Se a mensagem é fornecida, o cálculo de seu sumário é fácil.
2. Se o sumário é fornecido, será impossível encontrar a mensagem original.
3. Dada uma mensagem, ninguém pode encontrar uma mensagem diferente com o mesmo sumário que ela.
4. Uma mudança de um bit na mensagem produz drásticas mudanças no sumário.

As funções de *hash* mais utilizadas são o MD5 e o SHA-1. O sumário gerado pelo MD5 possui 128 bits e é um melhoramento do MD4. Já o SHA-1 trabalha com *hash* de 160 bits. O SHA-1 foi considerado o sucessor do MD5. Ambos têm vulnerabilidades comprovadas. Em algumas correntes, é sugerido que o SHA-256 ou superior seja usado para tecnologia crítica.

O MAC mais utilizado é o HMAC, que é utilizado no IPSec. O HMAC se baseia em algoritmos de *hash* como o SHA-1 e o MD5.

49.2 Protocolos de Autenticação

A autenticação é a técnica através de qual um processo confirma que seu parceiro na comunicação é quem deve ser e não um impostor.

49.2.1 Métodos de Autenticação

A autenticação pode ser feita com a combinação de um ou mais dos itens abaixo:

- Algo que você saiba, um número de identificação pessoal ou uma senha (autenticação tipo 1).
- Algo que você tenha, um cartão de banco ou um cartão com chip (autenticação tipo 2). Em alguns casos, esses dispositivos geram senhas automaticamente e são conhecidos como *tokens*.
- Algo que você é, impressão digital ou escaneamento de retina (autenticação tipo 3). Os parâmetros de desempenho mais importantes em um sistema de biometria são: FRR (*False Rejection Rate*), FAR(*False Acceptance Rate*), tempo de registro, tempo de atendimento e aceitabilidade (não invasivo).

49.2.2 Autenticação baseada em uma chave secreta compartilhada

Para efetuar uma autenticação baseada em uma chave secreta compartilhada, é necessário que Alice e Bob estabeleçam essa chave (K_{ab}).

Os protocolos de autenticação baseada em uma chave secreta é baseada no princípio do desafio-resposta (challenge-response), onde o emissor envia um número aleatório para o receptor (*nonce*); quando este recebe a mensagem, transforma o número recebido em uma forma especial e o retorna ao emissor.

Uma brecha que não deve existir no protocolo é a possibilidade de ataque por reflexão, em que uma pessoa no meio da comunicação recebe um desafio. A pessoa indesejada cria outra conexão para obter a resposta para aquele desafio, assim, utiliza a resposta para o sucesso da autenticação anterior.

Outras propriedades importantes para os protocolos dessa classe são:

- O transmissor deve provar sua identidade antes que o receptor responda;
- As pessoas envolvidas no processo devem utilizar chaves específicas provando suas identidades, mesmo que haja necessidade de existir duas chaves compartilhadas K_{ab} e K'_{ab} ;
- Condicionar para que tanto transmissor quanto receptor extraiam seus desafios de conjuntos distintos.

Na atualidade há protocolos baseados no uso de HMACs e na codificação de itens através do encadeamento de blocos de cifras.

49.3 Certificado Digital

Um certificado digital, ou identidade digital, pode ser visto como uma carteira de identidade para uso na internet. Tecnicamente, um certificado digital é um conjunto de dados (um arquivo), assinado digitalmente pela autoridade certificadora e contendo tipicamente informações como: Chave pública do certificado; Nome e endereço de e-mail do dono do certificado; Nome e assinatura digital da autoridade certificadora; Privilégios de acesso a sites seguros; Outras.

Uma Autoridade Certificadora é uma entidade de confiança que administra a gestão de certificados digitais através da emissão, revogação e renovação dos mesmos por aprovação individual. Uma Autoridade Certificadora pode emitir diferentes tipos de certificados, atribuindo diferentes níveis de confiança a cada tipo. Para cada tipo de certificado é utilizado um processo diferente para realizar a verificação da identidade do solicitante.

O padrão para descrever certificados é o X.509 (pela ITU). Um modo diferente de certificar chaves públicas é o PKI (Public Key Infrastructure). Uma PKI tem vários componentes, incluindo usuários, CAs, entidades de registro, certificados e diretórios. A função é fornecer um modo de estruturar esses componentes e definir padrões para os vários documentos e protocolos. Uma forma particularmente simples de PKI é uma hierarquia de CAs.

Um certificado digital é um arquivo de computador que contém um conjunto de informações referentes a entidade para o qual o certificado foi emitido (seja uma empresa, pessoa física ou computador) mais a chave pública referente a chave privada que acredita-se ser de posse unicamente da entidade especificada no certificado. Um certificado padrão X.509 contém os seguintes campos:

Versão Contém a versão do certificado X.509, atualmente versão 3

Número serial Todo certificado possui um, não é globalmente único, mas único no âmbito de uma AC, as LCRs usam o serial para apontar quais certificados se encontram revogados

Tipo de algoritmo Contém um identificador do algoritmo criptográfico usado pela AC para assinar o certificado juntamente com o tipo de função de hash criptográfica usada no certificado

Nome do titular Nome da entidade para o qual o certificado foi emitido

Nome do emitente Autoridade Certificadora que emitiu/assinou o certificado

Período de validade Mostra o período de validade do certificado no formato *Não antes e Não depois* (Ex. *Não antes de 05/03/2006 - 14:35:02, Não depois de 05/03/2007 - 14:03:2006*)

Informações de chave pública da entidade Algoritmo de chave pública e Chave pública.

Assinatura da AC A garantia que a AC provê sobre a veracidade das informações contidas neste certificado de acordo com as políticas da AC

Identificador da chave do titular É uma extensão do X.509 que possui um identificador numérico para a chave pública contida neste certificado, especialmente útil para que programas de computador possam se referir a ela

Identificador da chave do emitente A mesma idéia mencionada anteriormente, só que se referindo a chave pública da AC que emitiu o certificado

Atributos ou extensões A vasta maioria dos certificados X.509 possui campos chamados extensões (OID) que provêem algumas informações extras, como cadastros adicionais do titular e do emitente, especificações de propósito do certificado e etc.

Capítulo 50

Segurança em diversas camadas

50.1 *Secure Sockets Layer*

Em 1995, a então dominante do mercado de browsers, Netscape, resolveu introduzir um pacote de segurança chamado *Secure Sockets Layer* (SSL), para atender demandas por conexões mais seguras na internet. O SSL se tornou um padrão e até hoje é utilizado para prover conexões seguras. O SSL está posicionado entre a camada de transporte e a camada de aplicação da pilha TCP/IP e funciona provendo serviços de autenticação do servidor, comunicação secreta e integridade dos dados.

O SSL pode ser utilizado para prover segurança na comunicação de qualquer aplicação baseada em TCP. O uso do SSL com HTTP, geralmente é referenciado como HTTPS (*Secure HTTP*). O HTTPS geralmente utiliza a porta TCP 443, ao invés da porta 80.

Antes que o cliente e o servidor iniciem a troca de mensagens de forma segura é necessário que se estabeleça uma conexão SSL seguindo as seguintes etapas:

1. O cliente e o servidor negociam parâmetros de conexão como versão do protocolo, número da sessão, algoritmos de cifragem, algoritmo de compactação. Eles trocam ainda valores randômicos (*nonce*) que serão utilizados no processo de geração da chave de sessão;
2. O cliente solicita o certificado do servidor. Caso o certificado do servidor esteja assinado por uma CA confiável, o cliente extrai a chave pública do servidor e a utiliza para criptografar uma chave pré-mestra antes de enviá-la ao servidor. A chave pré-mestra é utilizada em combinação com os valores randômicos trocados na etapa 1 pra gerar a chave simétrica de sessão que será utilizada na comunicação;
3. O cliente e o servidor trocam mensagens de controle para sinalizar que todas as próximas mensagens serão cifradas utilizando a chave de sessão gerada.

A partir da etapa 3, todos os dados gerados pela camada de aplicação serão cifrados pelo SSL antes de serem repassados para a camada de transporte. É importante ressaltar que o SSL não cifra as informações do cabeçalho TCP, mas somente os dados gerados pela camada de aplicação.

50.2 IPSec

A IETF sabia há muitos anos da carência de segurança na Internet. Para aumentá-la, havia uma disputa para definir onde colocá-la. Para especialistas, a segurança deveria ser colocada no nível de aplicação. A dificuldade com essa abordagem é que ela exigiria que todas as aplicações fossem modificadas afim de torná-las seguras. Outra abordagem seria colocar a segurança em um nível intermediário entre aplicação e transporte, como é feito no SSL. Dessa forma, as aplicações não precisariam ser alteradas completamente.

Uma outra visão é que os usuários não conhecem segurança, e portanto não seriam capazes de implementá-la corretamente nos níveis de transporte ou aplicação. A partir desse princípio, IETF introduziu o IPSec.

O IPSec é uma suíte de protocolos que foi desenvolvida para proporcionar autenticação, confidencialidade e integridade dos dados no nível de rede. Todos os serviços oferecidos pelo IPSec são baseados em criptografia de chave simétrica porque o alto desempenho é importante. É importante ressaltar que o projeto do IPSec é independente do algoritmo de criptografia utilizado, de modo que a quebra de um algoritmo não represente o fim da utilidade do projeto.

Embora trabalhe na camada IP, o IPSec é orientado a conexões. Isso se deve à necessidade do estabelecimento de uma chave de criptografia que será utilizada por um determinado período. No contexto do IPSec, uma conexão é chamada SA (*Security Association*). Uma SA é uma conexão unidirecional, de forma que caso se deseje conexão segura em ambos os sentidos, serão necessárias duas SA's. Para o estabelecimento de chaves, o IPSec utiliza um protocolo chamado IKE (*Internet Key Exchange*).

O IPSec é baseado na adição de cabeçalhos adicionais que podem ser de dois tipos. O primeiro deles é o AH (*Authentication Header*), enquanto o segundo é o ESP (*Encapsulation Security Payload*).

O cabeçalho AH é capaz de prover autenticação e checagem de integridade dos dados por meio do campo HMAC (*Hashed Message Authentication Code*). Esse campo contém um hash da mensagem criptografado com a chave estabelecida na criação da conexão. Nesse modo de operação, o IPSec não é capaz de oferecer confidencialidade, já que os dados em si não são criptografados.

No modo ESP os dados são cifrados garantindo-se também a confidencialidade na comunicação. A integridade e a autenticação dos dados são obtidos com o campo HMAC, que também está presente no cabeçalho ESP. Com o ESP é possível operar de duas formas que são conhecidas como modo transporte e modo tunel. No modo transporte, o cabeçalho original do pacote IP não é

criptografado, e é utilizado para roteá-lo ao longo do caminho. Já no modo túnel, o pacote IP inteiro é criptografado e inserido dentro de um novo pacote IP juntamente com o cabeçalho ESP. O modo túnel é utilizado, por exemplo, para implementar VPN's seguras usando IPSec.

Os modos de operação do IPSec de acordo com o protocolo de extensão utilizado podem ser vistos na figura 50.1.

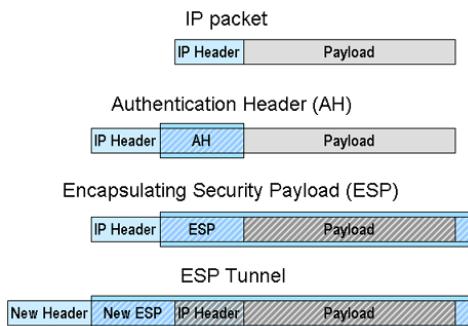


Figura 50.1: Modos de operação do IPSec

50.3 Virtual Private Network (VPN)

A idéia de utilizar uma rede pública como a Internet em vez de linhas privativas para implementar redes corporativas é denominada de Virtual Private Network (VPN) ou Rede Privada Virtual. As VPNs são túneis de criptografia entre pontos autorizados, criados através da Internet ou outras redes públicas e/ou privadas para transferência de informações, de modo seguro, entre redes corporativas ou usuários remotos.

A segurança é a primeira e mais importante função da VPN. Uma vez que dados privados serão transmitidos pela Internet, que é um meio de transmissão inseguro, eles devem ser protegidos de forma a não permitir que sejam modificados ou interceptados.

Outro serviço oferecido pelas VPNs é a conexão entre corporações (Extranets) através da Internet, além de possibilitar conexões dial-up criptografadas que podem ser muito úteis para usuários móveis ou remotos, bem como filiais distantes de uma empresa.

Uma das grandes vantagens decorrentes do uso das VPNs é a redução de custos com comunicações corporativas, pois elimina a necessidade de links dedicados de longa distância que podem ser substituídos pela Internet. As LANs podem, através de links dedicados ou discados, conectar-se a algum provedor de acesso local e interligar-se a outras LANs, possibilitando o fluxo de dados através da Internet. Esta solução pode ser bastante interessante sob o ponto de vista econômico, sobretudo nos casos em que enlaces internacionais ou nacionais de longa distância estão envolvidos. Outro fator que simplifica a operacionalização da WAN é que a conexão LAN-Internet-LAN fica parcialmente a cargo dos provedores de acesso.

Abaixo, são apresentadas as três aplicações ditas mais importantes para as VPNs.

- ACESSO REMOTO VIA INTERNET
- CONEXÃO DE LANS VIA INTERNET
- CONEXÃO DE COMPUTADORES NUMA INTRANET

No desenvolvimento de soluções de rede, é bastante desejável que sejam implementadas facilidades de controle de acesso a informações e a recursos corporativos. A VPN deve dispor de recursos para permitir o acesso de clientes remotos autorizados aos recursos da LAN corporativa, viabilizar a interconexão de LANs de forma a possibilitar o acesso de filiais, compartilhando recursos e informações e, finalmente, assegurar privacidade e integridade de dados ao atravessar a Internet bem como a própria rede corporativa. A seguir são enumeradas características mínimas desejáveis numa VPN:

Autenticação de Usuários: Verificação da identidade do usuário, restringindo o acesso às pessoas autorizadas. Deve dispor de mecanismos de auditoria, provendo informações referentes aos acessos efetuados - quem acessou, o quê e quando foi acessado.

Gerenciamento de Endereço: O endereço do cliente na sua rede privada não deve ser divulgado, devendo-se adotar endereços fictícios para o tráfego externo.

Criptografia de Dados: Os dados devem trafegar na rede pública ou privada num formato cifrado e, caso sejam interceptados por usuários não autorizados, não deverão ser decodificados, garantindo a privacidade da informação. O reconhecimento do conteúdo das mensagens deve ser exclusivo dos usuários autorizados.

Gerenciamento de Chaves: O uso de chaves que garantem a segurança das mensagens criptografadas deve funcionar como um segredo compartilhado exclusivamente entre as partes envolvidas. O gerenciamento de chaves deve garantir a troca periódica das mesmas, visando manter a comunicação de forma segura.

Suporte a Múltiplos Protocolos: Com a diversidade de protocolos existentes, torna-se bastante desejável que uma VPN suporte protocolos padrão de fato usadas nas redes públicas, tais como IP (Internet Protocol), IPX (Internetwork Packet Exchange), etc.

As redes virtuais privadas baseiam-se na tecnologia de tunelamento cuja existência é anterior às VPNs. Ele pode ser definido como processo de encapsular um protocolo dentro de outro. O uso do tunelamento nas VPNs incorpora um novo componente a esta técnica: antes de encapsular o pacote que será transportado, este é criptografado de forma a ficar ilegível caso seja interceptado durante o seu transporte. O pacote criptografado e encapsulado viaja através da Internet até alcançar seu destino onde é desencapsulado e decriptografado, retornando ao seu formato original. Uma característica importante é que pacotes de um determinado protocolo podem ser encapsulados em pacotes de protocolos diferentes. Por exemplo, pacotes de protocolo IPX podem ser encapsulados e transportados dentro de pacotes TCP/IP.

O protocolo de tunelamento encapsula o pacote com um cabeçalho adicional que contém informações de roteamento que permitem a travessia dos pacotes ao longo da rede intermediária. Os pacotes encapsulados são roteados entre as extremidades do túnel na rede intermediária. Túnel é a denominação do caminho lógico percorrido pelo pacote ao longo da rede intermediária. Após alcançar

o seu destino na rede intermediária, o pacote é desencapsulado e encaminhado ao seu destino final. A rede intermediária por onde o pacote trafegará pode ser qualquer rede pública ou privada.

Note que o processo de tunelamento envolve encapsulamento, transmissão ao longo da rede intermediária e desencapsulamento do pacote.

Para se estabelecer um túnel é necessário que as suas extremidades utilizem o mesmo protocolo de tunelamento.

O tunelamento pode ocorrer na camada 2 ou 3 (respectivamente enlace e rede) do modelo de referência OSI (Open Systems Interconnection).

Tunelamento em Nível 2 - Enlace - (PPP sobre IP): O objetivo é transportar protocolos de nível 3, tais como o IP e IPX na Internet. Os protocolos utilizam quadros como unidade de troca, encapsulando os pacotes da camada 3 (como IP/IPX) em quadros PPP (Point-to-Point Protocol). Como exemplos, podemos citar:

- PPTP (Point-to-Point Tunneling Protocol) da Microsoft permite que o tráfego IP, IPX e NetBEUI sejam criptografados e encapsulados para serem enviados através de redes IP privadas ou públicas como a Internet.
- L2TP (Layer 2 Tunneling Protocol) da IETF (Internet Engineering Task Force) permite que o tráfego IP, IPX e NetBEUI sejam criptografados e enviados através de canais de comunicação de datagrama ponto a ponto tais como IP, X25, Frame Relay ou ATM.
- L2F (Layer 2 Forwarding) da Cisco é utilizada para VPNs discadas.

Tunelamento em Nível 3 - Rede - (IP sobre IP): Encapsulam pacotes IP com um cabeçalho adicional deste mesmo protocolo antes de enviá-los através da rede.

O IP Security Tunnel Mode (IPSec) da IETF permite que pacotes IP sejam criptografados e encapsulados com cabeçalho adicional deste mesmo protocolo para serem transportados numa rede IP pública ou privada. O IPSec é um protocolo desenvolvido para IPv6, devendo, no futuro, se constituir como padrão para todas as formas de VPN caso o IPv6 venha de fato substituir o IPv4. O IPSec sofreu adaptações possibilitando, também, a sua utilização com o IPv4.

50.4 Filtragem de Pacotes e Firewalls

Filtragem de pacotes é o bloqueio ou liberação da passagem de pacotes de dados de maneira seletiva, conforme eles atravessam a interface de rede. Em sistemas Linux, por exemplo, a filtragem de pacotes é implementada diretamente no kernel. Esses filtros inspecionam os pacotes com base nos cabeçalhos de transporte, rede ou até mesmo enlace. Os critérios mais utilizados são os endereços IP e portas TCP/UDP de origem e destino.

Alguns filtros de pacotes também são capazes de transformar o conteúdo dos pacotes, como é o caso do netfilter do Linux. Normalmente às pessoas se referem ao filtro de pacotes do Linux pelo nome de iptables. Na verdade, o iptables é uma utilitário de linha de comando a partir do qual podem ser configuradas as

regras de filtragem do netfilter.

O netfilter é formado por um conjunto de cadeias. Cada cadeia é um ponto no caminho que um pacote IP percorre ao entrar ou sair de uma máquina. A figura 50.2 mostra as cadeias do netfilter.

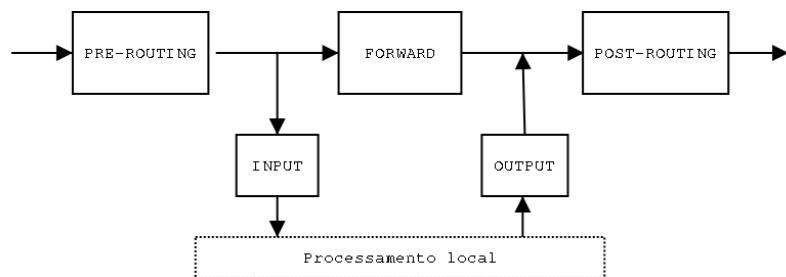


Figura 50.2: Estrutura do netfilter

A cadeia PREROUTING está ligada à entrada de pacotes na máquina. Após a decisão de roteamento, os pacotes que não são destinados à máquina local atravessam a cadeia FORWARD e finalmente saem da máquina passando pela cadeia POSTROUTING. A cadeia INPUT é atravessada pelos pacotes que chegam com destino à máquina local, enquanto que a cadeia OUTPUT é utilizada pelos pacotes originados localmente.

Para filtrar ou transformar pacotes IP, o netfilter possui estruturas chamadas tabelas. Cada tabela possui regras, que por sua vez se fazem referência a uma cadeia. As tabelas padrão do netfilter mais utilizadas são a filter, para implementação de um firewall, e a tabela nat, utilizada para fazer NAT.

As regras podem ser criadas pelo administrador de acordo com o que deseja implementar, por exemplo, um firewall ou NAT. Cada regra especifica um padrão ou critério a ser comparado com os pacotes e um alvo. Um alvo é uma ação pré-definida a ser tomada quando um pacote casa com a regra. Os alvos mais comuns são:

- ACCEPT: deixa o pacote passar;
- DROP: impede que o pacote siga adiante;
- REJECT: impede que o pacote siga adiante e envia uma mensagem ICMP ao sistema de origem;
- MASQUERADE: mascara os pacotes como se eles tivessem sido originados pela máquina local. Utilizado para implementação de NAT.

Um firewall pode ser configurado seguindo uma postura permissiva ou restritiva. Uma configuração permissiva permite tudo o que não for explicitamente negado, enquanto uma configuração restritiva nega tudo o que não for explicitamente permitido. A postura padrão de um firewall é conhecida como *policy*. A seguir são mostrados alguns exemplos de regras de firewall utilizando a sintaxe

do iptables.

50.4.1 Regras iptables - Exemplo 1

1. Pela interface ppp0, é permitido o tráfego de entrada com origem 172.16.100.200 e destino 192.168.130.0/24.

```
iptables -i ppp0 -A FORWARD -s 172.16.100.200  
-d 192.168.130.0/24 -j ACCEPT
```

2. Pela interface ppp0, é permitido o tráfego de saída com origem 192.168.130.0/24 e destino 172.16.100.200.

```
iptables -o ppp0 -A FORWARD -s 192.168.130.0/24  
-d 172.16.100.200 -j ACCEPT
```

3. Pela interface ppp0, tráfego de entrada, é negada toda e qualquer outra origem e outro destino. Loga este bloqueio.

```
iptables -i ppp0 -A FORWARD -j DROP --log
```

4. Pela interface ppp0, tráfego de saída, é negada toda e qualquer outra origem e outro destino. Loga este bloqueio.

```
iptables -o ppp0 -A FORWARD -j DROP --log
```

50.4.2 Regras iptables - Exemplo 2

1. Pela interface ppp0, permite a saída de pacotes com destino a porta TCP 23 de qualquer sistema remoto desde que a porta TCP de origem seja maior ou igual a 1024.

```
iptables -o ppp0 -A FORWARD -p tcp -s 192.168.10.0/24 -d 0/0  
--sport 1024:65535 --dport 23 -j ACCEPT
```

2. Pela interface ppp0, permite respostas aos pacotes aceitos na regra anterior, desde que os pacotes não tenham o flag SYN setado. Isso significa dizer que não é possível iniciar que um host externo inicie o processos de abertura de conexão TCP com hosts da rede 192.168.10.0/24 nas portas acima de 1023.

```
iptables -i ppp0 -A FORWARD -p tcp -s 0/0 -d 192.168.10.0/24  
--sport 23 --dport 1024:65535 ! -syn -j ACCEPT
```

3. Pela interface ppp0, tráfego de entrada, é negada toda e qualquer outra origem e outro destino. Loga este bloqueio.

```
iptables -i ppp0 -A FORWARD -j DROP --log
```

4. Pela interface ppp0, tráfego de saída, é negada toda e qualquer outra origem e outro destino. Loga este bloqueio.

```
iptables -o ppp0 -A FORWARD -j DROP --log
```

50.4.3 Firewall Stateful

Um firewall de filtro de pacotes é dito um firewall sem estado. Isso porque ele trata cada um dos pacotes que atravessam a interface de forma independente. Na segunda sequência de regras, por exemplo, foram necessárias duas regras para garantir que os hosts da rede 192.168.10.0/24 pudessem acessar a porta TCP 23 de um outro host qualquer. Cada uma das regras tratava os pacotes em um determinado sentido. Em uma conexão TCP, sempre existe fluxo de dados em ambos os sentidos entre origem e destino.

As conexões TCP são caracterizadas por uma série de atributos como IP's de origem e destino, portas de origem de destino, números de sequência etc. Em conjunto, esses atributos determinam o estado de uma conexão TCP.

Ao contrário dos firewalls de filtro de pacotes, um firewall stateful não filtra pacotes de forma isolada, mas sim com base em informações sobre o estado de conexões pré-estabelecidas. Para um firewall stateful, a comunicação bidirecional é implícita, de forma que não há necessidade de se escrever regras de filtragem para cada um dos sentidos. A seguir são mostrados alguns exemplos de regras para um firewall stateful utilizando a sintaxe do iptables.

1. Permite abertura de conexões TCP com origem na rede 10.0.0.0/24 e porta acima de 1023 para qualquer outro host na porta TCP 80. O detalhe é que, além de controlar quem tem o direito de abertura da conexão, a regra também cuida de todos os pacotes trocados em ambos os sentidos até o fim conexão.

```
iptables -A FORWARD -s 10.0.0.0/24 -d 0/0 -p tcp --sport  
1024:65535 --dport 80:80 -m state --state NEW -j ACCEPT
```

Na verdade, seria necessário uma outra regra para permitir o controle do fluxo de pacotes em ambos os sentidos. A regra é a seguinte:

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Essa regra aceita os pacotes que atravessam a cadeia FORWARD desde que eles pertençam a alguma conexão TCP que esteja no estado ESTABLISHED ou RELATED. No entanto, essa regra é válida para todos os

pacotes que atravessam a cadeia FORWARD, e não somente para os pacotes definidos na regra 1.

Alguns firewalls são capazes de implementar o conceito de estados para comunicações baseadas em UDP. Geralmente, isso é alcançado utilizando-se um temporizador. Dessa forma, os pacotes UDP transmitidos em ambos os sentidos entre uma origem e um destino são tratados como uma conexão pelo período de tempo definido pelo timeout.

50.4.4 Application Gateway

Embora os firewalls de filtro de pacotes e statefull apresentem uma diferenças em como pacotes de uma determinada conexão são tratados, ambos se baseiam fundamentalmente nas informações do cabeçalho dos protocolos da camada de transporte.

Um application gateway é um firewall stateful capaz de inspecionar os dados da camada de aplicação para tomar decisões mais inteligentes sobre a conexão. Exemplos de controles realizados por um application gateway são autenticação, filtros de URL e filtros de conteúdo.

Um application gateway pode ser utilizado para bloquear, por exemplo, aplicações peer to peer (eMule, Kaaza etc.), ou mensageiros instantâneos (IRC, MSN etc.) que tentam se esconder debaixo do protocolo HTTP. No entanto, os application gateway não são capazes de inspecionar dados criptografados via SSL, por exemplo.

50.4.5 Arquitetura de firewall e DMZ

A arquitetura de implementação do firewall também é um fator de análise importante. Por arquitetura de implementação entende-se a posição relativa que o firewall possui em relação às redes protegidas, aos recursos de conexão e às redes externas.

A análise da arquitetura de implementação de firewalls traz os conceitos de Bastion Host e DMZ, que precisam ser explicados: Bastion Hosts são servidores cuidadosamente implementados e de alta segurança que mantém contato com a rede externa, consequentemente estando expostos aos riscos de ataques. DMZ's são áreas intermediárias entre a rede interna e externa onde os servidores que recebem tráfego externo estão hospedados de maneira separada da rede interna de uma corporação.

a) Dual Homed Host - são Bastion hosts nos quais o firewall é implementado, expondo sua interface externa e desativando o roteamento entre as interfaces externas e as interfaces internas. Assim, conexões externas chegariam até o Bastion host apenas, e conexões internas teriam que obrigatoriamente passar pelo Bastion Host. Como mostra a figura abaixo, entre a internet e um Dual Homed host, poderia ser implementado um conjunto de filtros de pacotes, no roteador mais próximo, por exemplo, para diminuir as possibilidades de ataques,

diminuindo a necessidade de filtros de pacotes no próprio bastion host. Esta arquitetura gera um ponto focal na rede, com vantagens e desvantagens: O tráfego centralizado permite uma administração focalizada em um único host, porém o excesso de tráfego em um único host pode causar condições de concorrência e caso o bastion host seja tomado por um invasor, toda a rede estará vulnerável à ataques.

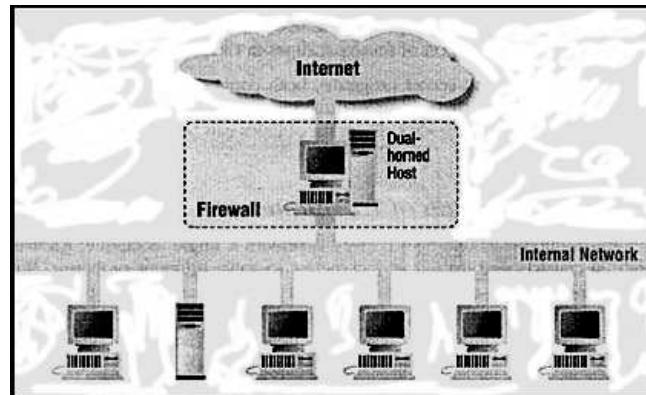


Figura 50.3: Arquitetura Dual Homed Host

b) Screened Host - conforme a figura abaixo, essa arquitetura apresenta uma conexão externa com ligação apenas a um host interno, que é um bastion host. Este host estaria conectado à rede interna, e não entre as redes, e o firewall teria que ser implementado no roteador, porém haveria pacotes externos entrando na rede local. Apesar de ser aparentemente menos seguro do que os dual homed hosts, este tipo de arquitetura permite o acesso aos serviços do bastion host sem causar condições de concorrência na rede, uma vez que todo o trabalho de filtragem e análise de tráfego ocorre no roteador.

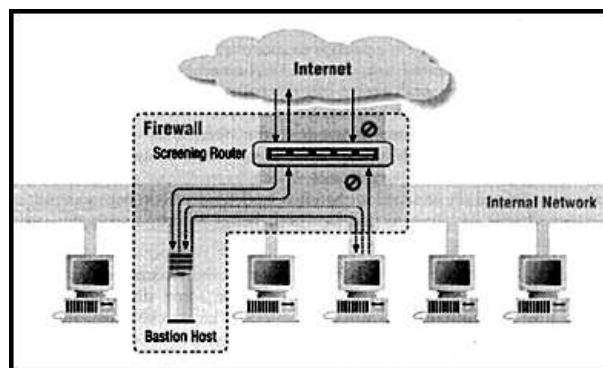


Figura 50.4: Arquitetura Screened Host

c) Screened Subnet - adicionam mais uma camada à segurança de redes através da utilização de DMZ para hospedagem dos bastion hosts e de um firewall adicional que separa a rede interna da DMZ. Em uma screened subnet

um firewall separa a rede externa da DMZ que hospeda os serviços que podem ser acessados pela rede externa, como por exemplo, um servidor de correio corporativo. Dentro da DMZ está também o bastion host que contém o firewall que dá acesso da rede interna à DMZ e roteia as requisições da rede interna para o roteador da DMZ. Esta arquitetura de screened subnets permite maior segurança a uma rede, uma vez que as tentativas de invasão serão efetuadas contra bastion hosts que não possuem acesso à rede interna, e que o bastion host de saída da rede é protegido por filtros de entrada no roteador externo.

50.5 Sistemas de Detecção de Intrusão (IDS)

Detecção de intrusão é uma tentativa de monitorar estações ou fluxos de rede com o intuito de descobrir ações de intrusos. Mais especificamente, um SDI tenta detectar ataques ou usos impróprios e alerta o responsável pela rede do acontecimento. O funcionamento é análogo ao de um sistema de detecção de ladrões, usado em casas para se proteger de eventuais incidentes. O sistema domiciliar inicialmente precisa ser configurado e ter especificado o que monitorar (janelas, portas, movimento) e para quem alertar ou chamar em caso de uma invasão (polícia, donos da casa). No sistema computacional, também precisamos determinar se queremos monitorar fluxos de rede, processos internos de uma estação ou servidor, ou ainda um sistema de arquivos, por exemplo. E devemos deixar claro para quem enviar os alarmes ou relatórios e como estes devem ser enviados, tendo como alternativas o e-mail, pager, ou ainda um pacote SNMP.

Teoricamente, esse tipo de sistema seria somente passivo, observando pacotes na rede ou processos em uma estação de trabalho e alertando os responsáveis. Porém, alguns sistemas possuem a habilidade de reagir às invasões, deixando de ser um sistema exclusivamente de detecção e pode ser definido como um Sistema de Prevenção de Intrusão (IPS). Exemplos de reações podem ser um fechamento de conexão, um bloqueio no firewall, execução de algum arquivo, ou ainda a desabilitação de uma conta.

Os sistemas de detecção de intrusão servem, como já foi dito, para indicar que alguma tentativa de intrusão foi feita no sistema. Para isso, existem dois tipos diferentes de detecção que podem ser empregados, os baseados na rede e os baseados na estação. Cada um tem uma maneira distinta de abordar o problema e o modo como isso é feito traz vantagens e desvantagens para cada tipo. Resumidamente, podemos dizer que os sistemas baseados na estação monitoram dados em uma determinada máquina, sejam eles processos, sistemas de arquivos e o uso de CPU, por exemplo; enquanto que os sistemas baseados na rede observam todos os dados trocados entre estações.

50.6 Segurança em Redes Wireless 802.11

50.6.1 WEP

Desde sua primeira versão, o padrão IEEE 802.11 tentou oferecer um mecanismo de segurança. Esse mecanismo é chamado WEP (*Wired Equivalent Privacy*) e se baseia em criptografia de chave simétrica. O processo de autenticação entre um host sem fio e um ponto de acesso (AP) funciona da seguinte maneira. O host solicita autenticação ao AP, que por sua vez envia um *nonce* de 128 bits ao

host. O host cifra o nonce com uma chave simétrica pré-compartilhada e envia de volta ao AP, que decifra o nonce e autentica o host. A partir daí, o host pode iniciar a transmissão dos dados de fato.

No entanto, o WEP não prevê nenhum mecanismo de distribuição de chaves e não possui autenticação individual. Isso significa dizer que todos os hosts sem fio utilizam a mesma chave simétrica para se autenticar no AP. Além disso, o processo de cifragem WEP 802.11 ainda apresenta outras falhas graves.

No WEP, cada um dos frames transmitidos é criptografado com o algoritmo RC4, que utiliza uma chave de 64 bits para gerar um fluxo de chaves de tamanho 1 byte (K_i^{IV}). Cada chave do fluxo gerado é utilizada para cifrar um byte de um frame enviado fazendo um XOR. A geração do fluxo de chaves é determinística, ou seja, uma chave de 64 bits sempre gera a mesma sequência de chaves de um byte.

Como foi dito, os hosts sem fio e o AP possuem uma chave pré-compartilhada. Essa chave possui comprimento de 40 bits. Os outros 24 bits necessários para formar a chave de 64 bits do RC4 são gerados dinamicamente a cada frame transmitido. Esses 24 bits são chamados vetor de inicialização (IV). O IV é transmitido em texto aberto no mesmo frame que carrega os dados cifrados. Além disso, os IV's eventualmente precisam ser reutilizados, já que seu range é de apenas 2^{24} . Dessa forma, ao detectar a reutilização um IV, o atacante pode fazer com que um transmissor cifice os dados conhecidos. Ao receber os dados cifrados, o atacante poderá calcular a sequência K_i^{IV} . Da próxima vez que IV for reutilizado, o atacante poderá decifrar os dados utilizando a sequência.

50.7 802.11i

O padrão 802.11i, também conhecido como WAP, propõe mecanismos de segurança para redes 802.11 mais avançados que o WEP. O WAP prevê formas de cifragem mais fortes, autenticação individual, e a utilização de um servidor de autenticação (AS) separado do AP. O processo de associação entre uma estação cliente (STA) e um AP no padrão 802.11i segue as seguintes etapas:

1. Descoberta das capacitações de segurança entre o STA e AP;
2. STA e AS se autenticam mutuamente. Juntos geram chave mestra (MK). O AP serve como passagem, permitindo a passagem de pacotes específicos;
3. STA e AS derivam uma chave PMK (*Pairwise Master Key*). O AS envia a chave PMK para o AP. A partir daí, o STA e o AS já conhecem a mesma chave PMK;
4. STA e AP derivam uma nova chave temporária TK (*Temporal Key*) a partir de PMK. A chave TK será utilizada para cifragem e integridade das mensagens.

Por ser formado por 4 etapas principais, o processo de autenticação definido no padrão 802.11i é conhecido como *four-way handshake*.

Parte IX

Alta Disponibilidade

Capítulo 51

Soluções de Armazenamento RAID, SAN e NAS

51.1 RAID

Tal como em diversas outras áreas, quando não é possível aumentar o desempenho de um componente, uma solução possível passa pelo uso de múltiplos componentes em paralelo. No caso do armazenamento, isto leva ao desenvolvimento de conjuntos de discos que operam independentemente e em paralelo. Com vários discos, pedidos de I/O podem ser tratados em paralelo, desde que os dados residam em discos separados. Do mesmo modo, um único pedido de I/O pode ser tratado de igual modo em paralelo caso os dados se encontrem distribuídos ao longo de diversos discos.

Com o uso de vários discos, torna-se claro que existem muitos modos de organizar os dados e nos quais a redundância pode ser adicionada para aumentar a confiabilidade. O esquema RAID (Redundant Array of Independent Disks) puro consiste em 7 níveis (de zero a seis). Não há qualquer relação hierárquica entre eles, mas cada um deles designa arquiteturas distintas que partilham 3 características diferentes.

- O conjunto dos discos físicos em RAID é visto pelo sistema operacional como sendo um único drive lógica
- Os dados são distribuídos pelos drives físicos de um array.
- Redundância de capacidade é usada para armazenar informação de paridade, que garante a recuperação dos dados no caso de uma falha num disco.

51.1.1 RAID 0

Este nível RAID refere-se a um array de discos onde os dados estão divididos em faixas, mas não existe nenhuma redundância para tolerância a falhas. Sempre que a performance e a capacidade forem a preocupação principal e o baixo

custo for mais importante que a confiabilidade adicional, esta é uma opção a considerar. A figura 51.1 exemplifica o tipo de estrutura existente num sistema deste tipo.

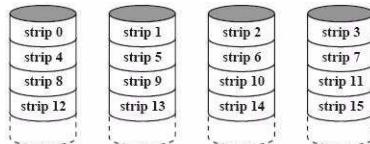


Figura 51.1: RAID 0

Os dados são subdivididos em segmentos consecutivos (stripes) que são escritos seqüencialmente através de cada um dos discos de um array. Cada segmento tem um tamanho definido em blocos. O striping oferece um melhor desempenho, quando comparado a um disco individual, se o tamanho de cada segmento for ajustado de acordo com a aplicação que utilizará o array.

- Em um ambiente com uso intensivo de E/S ou em um ambiente de banco de dados onde múltiplas requisições concorrentes são feitas para pequenos registros de dados, um segmento de tamanho grande é preferencial. Se o tamanho de segmento para um disco é grande o suficiente para conter um registro inteiro, os discos do arranjo podem responder independentemente para as requisições simultâneas de dados.
- Em um ambiente onde grandes registros de dados são armazenados, segmentos de pequeno tamanho são mais apropriados. Se um determinado registro de dados extende-se através de vários discos do arranjo, o conteúdo do registro pode ser lido em paralelo, aumentando o desempenho total do sistema.

Na verdade, a distribuição dos dados nos drives não é completamente uniforme. Os arquivos são divididos em fragmentos de tamanho configurável ("chunk size", ou "stripe size"). Assim, se está sendo usado 3 HDs em RAID 0, utilizando fragmentos de 32 KB, por exemplo, ao gravar um arquivo de 80 KB teríamos fragmentos de 32 KB gravados nos dois primeiros HDs e os 16 KB finais seriam gravados no terceiro, sendo que os 16 KB que "sobraram" no terceiro HD ficariam como espaço desperdiçado.

Arranjos RAID-0 podem oferecer alta performance de escrita se comparados a verdadeiros níveis de RAID por não apresentarem sobrecarga associada com cálculos de paridade ou com técnicas de recuperação de dados. Esta mesma falta de previsão para reconstrução de dados perdidos indica que esse tipo de arranjo deve ser restrito ao armazenamento de dados não críticos e combinado com eficientes programas de backup.

Entretanto, cabe ressaltar alguns pontos negativos desta implementação no que tange confiabilidade e desempenho. RAID 0 não terá o desempenho desejado com sistemas operacionais que não oferecem suporte de busca combinada de setores. Os resultados serão corretos, porém não haverá paralelismo e nenhum ganho de desempenho. Outra desvantagem desta organização é que a confiança se torna potencialmente pior. Um disco SLED com um tempo médio de vida de 20.000 horas será 4 vezes mais seguro do que 4 discos funcionando em paralelo

com RAID 0 (Admitindo-se que a capacidade de armazenamento somada dos quatro discos for igual ao do disco SLED). Como não existe redundância, não há muita confiabilidade neste tipo de organização. Por fim, RAID 0 é um nome até certo ponto equivocado, pois não há redundância, ele não se encontra na taxonomia original de níveis de RAID, e a aplicação de faixas é anterior ao RAID.

51.1.2 RAID 1

Esse esquema tradicional para tolerância a falhas de disco, chamado espelhamento (mirroring) ou sombreamento, utiliza o dobro da quantidade de discos do RAID 0. Formalmente, para esta implementação são necessários no mínimo dois discos. O funcionamento deste nível é simples: todos os dados são gravados em dois discos diferentes; se um disco falhar ou for removido, os dados preservados no outro disco permitem a não descontinuidade da operação do sistema. A figura 51.2 apresenta este esquema.

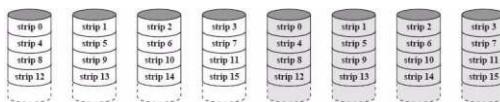


Figura 51.2: RAID 1

Apesar de muitas implementações de RAID 1 envolverem dois grupos de dados (daí o termo espelho ou mirror), três ou mais grupos podem ser criados se a alta confiabilidade for desejada. O RAID 1 é o que oferece maior segurança, pois toda informação é guardada simultaneamente em dois ou mais discos. Se ocorrer uma falha num dos discos do array, o sistema pode continuar a trabalhar sem interrupções, utilizando o disco que ficou operacional. Os dados então são reconstruídos num disco de reposição (spare disk) usando dados do(s) disco(s) sobrevivente(s). O processo de reconstrução do espelho tem algum impacto sobre o desempenho de I/O do array, pois todos os dados terão de ser lidos e copiados do(s) disco(s) intacto(s) para o disco de reposição.

Alguns pontos em relação ao desempenho deste tipo de esquema são descritos abaixo:

- Um pedido de leitura pode ser satisfeito por qualquer um dos dois discos que contenha os dados em questão, podendo ser escolhido o que implicar um tempo menor de procura e latência de rotação.
- Um pedido de escrita requer uma atualização em ambos os discos, mas isto pode ser feito em paralelo. Deste modo, a performance de escrita é ditada pela mais lenta das duas faixas físicas. Em outros níveis RAID em que são utilizados outros esquemas de redundância, quando uma única faixa é atualizada é necessário calcular e atualizar bits de paridade provocando um aumento no tempo necessário para fazer uma escrita no disco. Em RAID 1, este write penalty é inexistente.
- A recuperação de uma falha é simples. Quando ocorre uma falha num dos discos, é sempre possível aceder aos dados utilizando o outro disco.

- Conectando os discos primários e os discos espelhados em controladoras separadas, pode-se aumentar a tolerância a falhas pela eliminação da controladora como ponto único de falha.

Num ambiente de transações, RAID 1 consegue satisfazer altas taxas de pedidos I/O se a maioria dos pedidos forem leituras, onde se consegue aproximar do dobro do desempenho conseguido por RAID 0. No entanto, sempre que uma fração substancial dos pedidos forem escritas, pode não haver superioridade de desempenho relativamente a RAID 0.

Entre os não-híbridos, este nível tem o maior custo de armazenamento, pois serão utilizados dois discos para a mesma informação. Este nível adapta-se melhor em pequenas bases de dados ou sistemas de pequena escala que necessitem confiabilidade.

51.1.3 RAID 2

Raramente são usados, e em algum momento ficaram obsoletos pelas novas tecnologias de disco. RAID-2 é similar ao RAID-4, mas armazena informação ECC (error correcting code), que é a informação de controle de erros, no lugar da paridade. Isto ofereceu pequena proteção adicional, visto que todas as unidades de disco mais novas incorporaram ECC internamente. RAID 2 pode oferecer maior consistência dos dados se houver queda de energia durante a escrita. Baterias de segurança e um desligamento correto, porém, podem oferecer os mesmos benefícios. As figuras 51.3 e 51.4 ilustram o esquema RAID 2 bem como a forma em que as operações de leitura e escrita são realizadas.

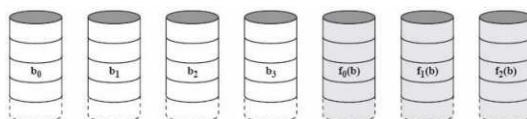


Figura 51.3: RAID 2

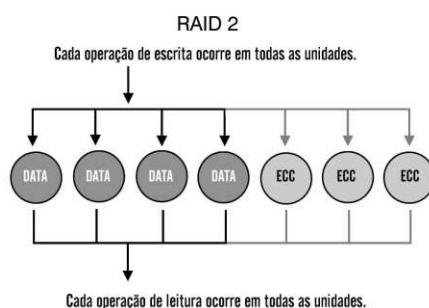


Figura 51.4: Operações de leitura e escrita em RAID 2

Em RAID 2 o ECC é calculado sobre os bits em posições análogas em cada disco. Os bits do código são armazenados nas posições correspondentes em múltiplos discos de paridade. Normalmente é utilizado o código de Hamming

que torna possível a correção de erros de um bit e a detecção de erros de dois bits. Apesar de requerer menos discos que RAID 1 é ainda bastante caro. O número de discos redundantes é proporcional ao logaritmo do número de discos de dados.

Numa leitura única, todos os discos são acessados simultaneamente. Os dados e o código corretor associado são entregues ao controlador do array. Se existir um erro de um só bit, o controlador consegue reconhecer e corrigi-lo instantaneamente. Do mesmo modo, numa escrita, todos os discos de dados e paridade devem ser acessados.

Esta pode ser uma opção em ambientes em que existe uma alta probabilidade de ocorrência de erros. No entanto, dada a alta confiabilidade dos discos atuais, normalmente não se implementa este esquema RAID.

51.1.4 RAID 3

Assim como RAID 2, raramente são usados, e é semelhante ao RAID 4, exceto por usar o menor tamanho possível para a stripe. Como resultado, qualquer pedido de leitura invocará todos os discos, tornando a execução de requisições diferentes em paralelo difíceis ou impossíveis. A fim de evitar o atraso devido a latência rotacional, o RAID-3 exige que todos os eixos das unidades de disco estejam sincronizados. A maioria das unidades de disco mais recentes não possuem a habilidade de sincronização do eixo, ou se são capazes disto, faltam os conectores necessários, cabos e documentação do fabricante.

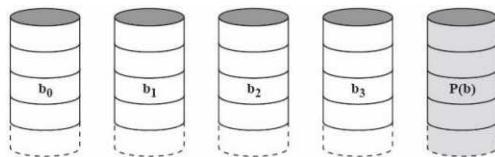


Figura 51.5: RAID 3

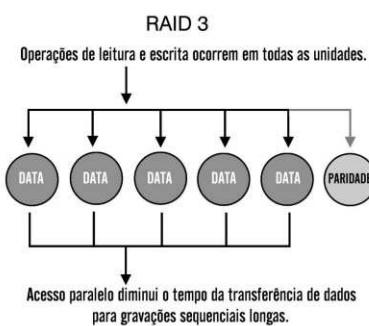


Figura 51.6: Operações de leitura e escrita em RAID 3

Outra diferença em relação ao RAID 2 é que RAID 3 requer apenas um disco redundante, qualquer que seja o tamanho do array de discos.

RAID 3 aplica acesso paralelo, com os dados distribuídos em pequenas faixas. Em vez de um código corretor de erros, um simples bit de paridade é calculado

para o conjunto de bits na mesma posição em todos os discos e armazenado no disco redundante referido. De uma forma simplificada, pode-se pensar no disco redundante como contendo a soma de todos os dados nos outros discos. Quando ocorre uma falha, subtraem-se todos os dados nos discos bons pelos dados contidos no disco de paridade. A informação restante terá inevitavelmente de ser a informação que falta. Por exemplo, o cálculo desta para o n-ésimo bit para um array de quatro discos será:

$$X3(n) = X2(n) \text{ xor } X1(n) \text{ xor } X0(n) \quad (51.1)$$

Supondo que ocorre uma avaria no disco X0, adicionando X3 xor X0 a ambos os lados da equação obtemos:

$$X0 = X3(n) \text{ xor } X2(n) \text{ xor } X1(n) \quad (51.2)$$

Este princípio é verdadeiro para cada os sistemas RAID de nível 3 a 6. Dado que os dados se encontram em faixas de tamanho bastante reduzido, em RAID 3 é possível atingir altas taxas de transferência de dados. Qualquer pedido I/O implicará a transferência paralela de dados de todos os discos. Este aumento de desempenho é mais visível em grandes transferências. Por outro lado, apenas um pedido de I/O pode ser executado de cada vez, portanto não constituirá a melhor opção para um ambiente de transações.

51.1.5 RAID 4

Tal como nos outros sistemas RAID, são usadas faixas, mas no caso das implementações RAID de 4 a 6, estas faixas são relativamente grandes. Em RAID 4, uma faixa de paridade é calculada bit a bit para faixas correspondentes em cada disco de dados. Os bits de paridade são armazenados no disco redundante. Ao contrário do sistema de RAID 3 que armazena a paridade bit-a-bit, em RAID 4 a paridade é armazenada sob a forma de blocos e associada a um conjunto de blocos de dados.

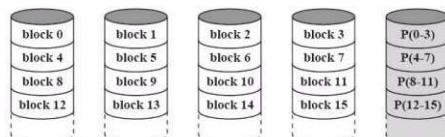


Figura 51.7: RAID 4

No RAID 3, todo o acesso era realizado em todos os discos. Entretanto, é possível que algumas aplicações preferiram acessos menores, permitindo que ocorressem acessos independentes em paralelo. Essa é a finalidade dos níveis de RAID 4-6. Tendo em vista que as informações de detecção de erros em cada setor são examinadas nas leituras para verificar se os dados estão corretos, essas "pequenas leituras" em cada disco podem ocorrer independentemente, desde que o acesso mínimo seja a um único setor.

As gravações são outra questão. Aparentemente, cada pequena gravação exigiria que todos os outros discos fossem acessados para ler o restante das informações necessárias para recalcular a nova paridade. Uma "pequena gravação" exigiria

a leitura dos dados antigos e da paridade antiga, adicionando as novas informações, e depois a gravação da nova paridade no disco de paridade e dos novos dados no disco de dados.

A idéia-chave para reduzir essa sobrecarga é que a paridade é simplesmente um somatório de informações; observando quais bits se alteram quando gravamos as novas informações, só precisamos mudar os bits correspondentes no disco de paridade. Dessa forma, temos de ler os dados antigos do disco que está sendo gravado, comparar os dados antigos com os novos para verificar quais bits mudaram, ler a paridade antiga, modificar os bits correspondentes, depois gravar os novos dados e a nova paridade. Desse modo, uma pequena gravação envolve quatro acessos a dois discos, em vez do acesso a todos os discos. Uma desvantagem do sistema é que o disco de paridade deve ser atualizado em cada gravação; assim ele é o gargalo de gravações (write bottleneck).

Os sistemas RAID dos níveis 4 a 6 fazem uso de uma técnica de acesso independente. Neste tipo de acesso, cada disco opera independentemente sendo assim possível satisfazer pedidos I/O em paralelo. Por esta razão, arrays deste tipo é um arranjo perfeitamente ajustado para ambientes transacionais que requerem muitas leituras pequenas e simultâneas e menos para aplicações que necessitem de altas taxas de transferência.

51.1.6 RAID 5

Este tipo de RAID largamente usado funciona de forma similar ao RAID 4, mas supera alguns dos problemas mais comuns sofridos por esse tipo. As informações sobre paridade para os dados do array são distribuídas ao longo de todos os discos do array, ao invés de serem armazenadas num disco dedicado, oferecendo assim mais desempenho que o RAID 4, e, simultaneamente, tolerância a falhas.

A idéia de paridade distribuída reduz o gargalo de escrita, pois agora as escritas concorrentes nem sempre exigem acesso às informações de paridade a um mesmo disco dedicado. Contudo, o desempenho de escrita geral ainda sofre por causa do processamento adicional causado pela leitura, recálculo e atualização da informação sobre paridade.

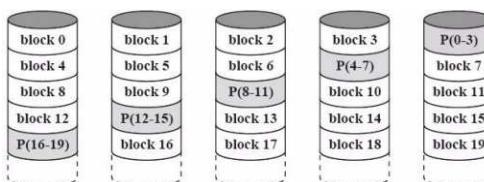


Figura 51.8: RAID 5

Para aumentar o desempenho de leitura de um array RAID 5, o tamanho de cada segmento em que os dados são divididos pode ser otimizado para o array que estiver a ser utilizado. O desempenho geral de um array RAID 5 é equivalente ao de um RAID 4, exceto no caso de leituras seqüenciais, que reduzem a eficiência dos algoritmos de leitura por causa da distribuição das informações sobre paridade. A informação sobre paridade ao ser distribuída ao longo de todos os discos, havendo a perda de um, reduz a disponibilidade de

ambos os dados e da informação sobre paridade, até à recuperação do disco que falhou. Isto pode causar degradação do desempenho de leitura e de escrita.

Como em outros arranjos baseados em paridade, a recuperação de dados em um arranjo RAID-5 é feita calculando a função XOR das informações dos discos restantes do arranjo. Pelo fato de que a informação sobre paridade é distribuída ao longo de todos os discos, a perda de qualquer disco reduz a disponibilidade de ambos os dados e informação sobre paridade, até a recuperação do disco que falhou. Isto pode causar degradação da performance de leitura e de escrita. Além disso, esta estrutura é vista como contendo uma limitação crítica, o throughput das aplicações sofre normalmente uma penalização até cerca de 4x, comparativamente a arrays não redundantes para pequenas escritas.

51.1.7 RAID 6 (Redundância de P+Q)

Os esquemas baseados em paridade protegem contra uma única falha auto-identificada. Quando uma única correção de falha não é suficiente, a paridade pode ser generalizada para ter um segundo cálculo sobre os dados e outro disco de verificação de informações. Esse segundo bloco de verificação permite a recuperação de uma segunda falha. Desse modo, a sobrecarga de armazenamento é o dobro do overhead do RAID 5. O atalho de pequena gravação apresentado anteriormente funciona bem exceto pelo fato de haver agora seis acessos ao disco, em vez de quatro, para atualizar as informações P e Q. A figura ?? ilustra essa configuração.

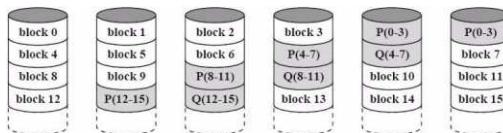


Figura 51.9: RAID 6 - Redundância P+Q

Em RAID 6 são efetuados dois cálculos diferentes para a paridade e armazenados em blocos separados em discos diferentes. Na realidade, um array de n discos organizados em RAID 6 requer n+2 discos.

P e Q são dois algoritmos diferentes de verificação de dados. Um dos dois é o cálculo por xor já referido e usado em RAID 4 e 5, sendo o outro um algoritmo independente. Isto torna possível a regeneração de dados mesmo que ocorra uma falha em dois dos discos.

A grande vantagem desta organização é o fato de providenciar alta disponibilidade de dados. Teria de ocorrer um erro em três dos discos durante o tempo médio para reparação para que isso tornasse os dados indisponíveis. Por outro lado, subjacente a este sistema está um write penalty substancial, pois cada escrita afeta obrigatoriamente dois blocos de paridade.

51.1.8 Tipos Híbridos

Dadas as limitações de cada nível RAID, diversos criadores de dispositivos de armazenamento tentaram criar novas formas RAID combinando características de cada um dos níveis originais.

- RAID 10: Nesta variedade combina-se o espelhamento existente em RAID 1 com a divisão em faixas do RAID 0. Numa implementação 0+1 os dados são divididos por conjuntos de drives duplicadas. Numa implementação 1+0, os dados são divididos por diversas drives e este array completo de discos é duplicado por um ou mais array de drives. O desempenho na reposição de dados é melhor neste tipo de arrays que em sistemas baseados em paridade, pois os dados não precisam ser regenerados com base nesta, mas sim simplesmente copiados para a nova drive.
- RAID 30 e 50: Esta forma consiste em manter a informação dividida em faixas ao longo de uma matriz RAID 3 ou 5. Estes híbridos providenciam os mesmos benefícios de arrays acesso paralelo (altas taxas de transferência) ou as vantagens de arrays de acesso independente baseados em paridade (alto throughput).

51.1.9 Comparativo de Desempenho entre as diversas configurações RAID

51.2 SAN - Storage Area Network

A SAN poderia ser definida como uma rede de alta velocidade, comparada à LAN (Local Area Network), que permite o estabelecimento de conexões diretas entre os dispositivos de armazenamento e processadores (servidores) centralizados à extensão suportada pela distância das fibras óticas. A SAN pode ser vista como uma extensão do conceito que permite que os dispositivos de armazenamento sejam compartilhados entre servidores e interconectados entre si. Uma SAN pode ser compartilhada entre servidores ou dedicada a um servidor local ou remoto.

Outra definição diz que SAN são dois ou mais dispositivos se comunicando via protocolo serial SCSI, tal como Fibre Channel ou iSCSI. Segundo essa definição, uma LAN que trafega nada mais do que tráfego de storage não pode ser considerada uma SAN. O que diferencia uma LAN de uma SAN (ou de uma NAS) é o protocolo que é usado. Assim, se o tráfego de storage trafega em uma LAN através do protocolo iSCSI, então essa LAN pode ser considerada uma SAN. Entretanto, simplesmente enviar dados de backup através de uma LAN dedicada isso não a torna uma SAN. Enfim, uma SAN é uma rede que usa um protocolo serial SCSI para transferir dados.

A figura 51.10 abaixo mostra uma visão geral de uma SAN conectando vários servidores a vários sistemas de armazenamento:

SANs criam novos métodos de conexão de armazenamento aos servidores. Estes novos métodos prometem uma notável melhora em performance e disponibilidade. SANs são usadas para conectar conjuntos de armazenamento compartilhados a vários servidores, e são usadas por servidores em ambiente de cluster para fail over. Elas podem interconectar discos ou fitas de mainframe a servidores ou clientes de rede e podem criar caminhos paralelos de dados para ambientes de computação e de alta largura de banda. A SAN é uma outra rede que difere das redes tradicionais, porque foi concebida a partir de interfaces de armazenamento.

Além disso, a SAN pode ser usada para contornar os conhecidos gargalos de rede, pois suporta diretamente altas velocidades de transferência de dados entre

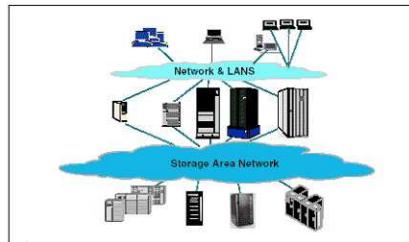


Figura 51.10: Exemplo de SAN

os servidores e dispositivos de armazenamento nas seguintes formas:

- Servidor para storage: é o modelo tradicional de interação com os dispositivos de armazenamento e cuja vantagem é que um mesmo dispositivo de armazenamento pode ser acessado serial ou concorrentemente por múltiplos servidores;
- Servidor para servidor: no qual a SAN pode ser usada para alta velocidade e comunicações de alto volume entre servidores;
- Storage para storage: permite a movimentação dos dados sem intervenção do servidor, liberando o processador para outras atividades. Por exemplo: um disco pode realizar o backup de dados para uma fita sem a intervenção do servidor ou um espelhamento de dispositivo remoto através da SAN.

O uso de SANs gera uma melhora de performance das aplicações, por exemplo, permitindo que o dado enviado diretamente do dispositivo de origem ao dispositivo de destino não requeira intervenção do servidor. As SANs também habilitam novas arquiteturas de rede nas quais vários computadores (hosts) acessam vários dispositivos de armazenamento conectados na mesma rede. Conheça outros benefícios que o uso da SAN pode oferecer às empresas:

- Mais disponibilidade: armazenamento independente de aplicações acessíveis através de caminhos alternativos de dados;
- Alta performance: os dados são descarregados do servidor e movidos para uma rede independente;
- Armazenamento centralizado e consolidado: gerenciamento mais simples, escalabilidade, flexibilidade e disponibilidade;
- Transferência e armazenamento: cópia remota de dados habilitada para proteção contra falhas e desastres;
- Gerenciamento centralizado mais simplificado: a imagem simples do meio de armazenamento simplifica o gerenciamento.

51.2.1 Hardware para SAN

Talvez a parte mais importante para a implantação de uma SAN seja referente ao hardware. Os principais componentes são:

- Servidor de Disco: Servidores de discos, também chamados de storages são dispositivos que armazenam discos compartilhados pelos hosts da rede. Eles possuem, em geral, diversas áreas diferentes, com esquemas de RAID diferentes ou discos específicos para a realização de espelhamentos para backup, os chamados BCV (business continuance volumes). Os BCVs facilitam muito tanto o backup quanto a restauração dos dados. O seu conteúdo é sincronizado com o conteúdo do disco principal, até que se faça uma quebra do sincronismo, o chamado split. Neste momento, o BCV guarda uma imagem do disco antes do split, e pode ser usado para backup, enquanto o servidor continua trabalhando, sem impactos na produção. Este procedimento pode ser feito com a freqüência mais conveniente para o usuário e, em muitos casos, o BCV é utilizado também para restauração de dados perdidos, que é muito mais rápido do que acessar fitas de backup.
- Switches fibre channel: Switches fibre channel são bem mais complexos que os hubs, tanto em seu projeto quanto em funcionalidade. Enquanto os hubs são apenas um concentrador de cabos para um segmento compartilhado, um switch é um dispositivo de rápido roteamento dos dados e possui uma taxa de transferência de dados exclusiva para cada porta. As taxas de transferência variam bastante dependendo do switch, que vêm evoluindo rapidamente. Atualmente, a velocidade máxima está em 400 MB/s para cada porta. Enquanto os hubs não participam de atividades no nível do protocolo Fibre Channel, os switches participam ativamente, tanto para fornecer serviços quanto para supervisionar o fluxo de frames entre a origem e o destino.
- HBA (Host Bus Adapter): Uma HBA é um dispositivo capaz de conectar dispositivos externos a um servidor. Por exemplo: para conectarmos um disco SCSI a um micro (barramento interno PCI), será necessário utilizar uma HBA SCSI-PCI. No caso da SAN, é necessário instalar em todos os servidores participantes dela uma HBA Fibre Channel, que se encarregará de fazer as conversões dos diferentes meios internos e externos ao servidor. As HBAs FC possuem, ainda, uma espécie de processador (um chip) capaz de fazer a conversão de protocolos para poupar a CPU do servidor deste trabalho.

51.2.2 Topologias de SAN

As SANs atuais são todas construídas em uma topologia física de estrela. Um switch é conectado ao storage e neles conectamos todos os outros servidores da rede. A exceção fica com a ligação ponto-a-ponto, como veremos a seguir.

Ligação ponto-a-ponto (point-to-point)

Para alguns não é considerada uma topologia de SAN, uma vez que não possui escalabilidade alguma. Neste tipo de ligação, os servidores são ligados diretamente ao storage, sem nenhum equipamento intermediário, como switch. O número máximo de servidores envolvidos é igual ao número de portas que o storage possui. Comparado com a ligação SCSI, no entanto, esta topologia, que utiliza Fibre Channel, apresenta uma performance muito melhor e atinge distâncias maiores.

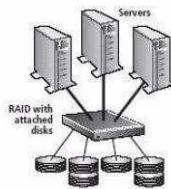


Figura 51.11: SAN ponto a ponto

Loops ou anéis: Fibre Channel Arbitrated Loop (FC-AL)

Nesta topologia utiliza-se hub, cuja largura de banda de no máximo de 100 MB/s é compartilhada por todos os seus membros. Teoricamente suporta loops com até 126 dispositivos, mas na prática este número é bem menor. Esta topologia está sendo muito pouco utilizada nas SANs modernas. Algumas pessoas compararam esta topologia como uma rede Token Ring, em que a largura de banda é dividida por todos os dispositivos do loop. Entretanto há equipamentos antigos que não suportam o modo fabric e, assim, é utilizado o modo em loop.

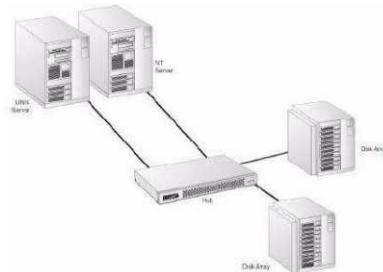


Figura 51.12: SAN FC-AL - Fibre Channel Arbitrated Loop

Malha (Switched Fabric ou Fabric)

Nesta topologia, utiliza-se switches Fibre Channel, o que permite um grande número e dispositivos interconectados. Dedicada largura de banda integral a cada uma das portas e permite transferências de dados simultaneamente para um único nó. É a topologia que permite a escalabilidade e crescimento. Teoricamente, pode conter mais de 7,7 milhões de nós.

51.3 NAS - Network Attached Storage

O compartilhamento de arquivos na rede já era possível mesmo antes da chegada do sistema NAS. Isso era possível graças aos protocolos NFS (Network File System) do Unix e CIFS (Common Internet File System) do Windows. Entretanto, ambos possuíam limitações de desempenho e gerenciamento. Além disso, servidores deviam ser dedicados a usar ou NFS ou CIFS. O sucesso do NAS, portanto, foi corrigir algumas limitações de cada protocolo e permitir que ambos pudessem operar em um mesmo servidor. Assim, pode-se dizer que NAS

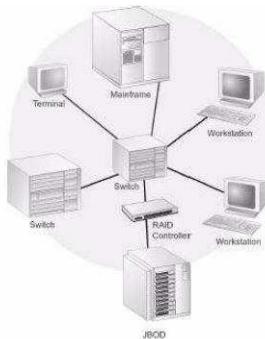


Figura 51.13: SAN Switched Fabric

(Network attached storage) é um computador dedicado a compartilhamento de arquivos através de NFS, CIFS ou DAFS(Direct Access Files). Cabe ressaltar que é um dispositivo dedicado exclusivamente ao compartilhamento de arquivos, centralizando a responsabilidade de servir os arquivos em uma rede e desse modo libera recursos de outros servidores.

Um dispositivo NAS combina a tecnologia dos arrays de discos com a inteligência de uma pequena unidade de processamento. Nesse sentido, é possível adicionar armazenamento na rede sem ser necessário desligar o servidor. É comum a familiaridade com o conceito de "impressora de rede", ou seja, aquela onde o usuário pode imprimir. Do mesmo modo o NAS é uma unidade partilhada através da LAN e todos os usuários com os direitos e permissões de acessos adequados podem montar sistemas de arquivos diretamente sem ter que veicular os dados através do servidor. Tipicamente, cada interação entre aplicações e o NAS envolve a transferência de um volume de dados relativamente pequeno e de curta duração. Além disso o NAS pode estar ligado em qualquer parte da LAN.

51.4 Comparativo entre SAN e NAS

Vistas como concorrentes, as SANs e as são tecnologias complementares. Enquanto as primeiras são otimizadas para transferências de altos volumes de dados orientadas em bloco, as NAS oferecem acesso a dados em nível de arquivo. Baseadas em protocolos abertos, como Fibre Channel e TCP/IP, oferecem varias vantagens em relação ao sistema de armazenamento ligado ao servidor. A seguir um resumo das principais diferenças entre cada tecnologia.

- Protocolo:
 - SAN: Serial SCSI-3
 - NAS: TCP/IP, NFS/CIFS
- Compartilhamento:
 - SAN: Drives de discos e fitas
 - NAS: Sistemas de arquivos

- Permite:
 - SAN: Diferentes servidores acessarem o mesmo drive de disco ou fita de forma transparente para o usuário final.
 - NAS: Diferentes usuários acessarem o mesmo sistema de arquivos ou até um mesmo arquivo.
- Substitui:
 - SAN: Os drivers de discos e fitas locais, ou seja, drivers conectados diretamente ao servidor. Com SAN, centenas de sistemas podem compartilhar o mesmo drive de disco ou fita.
 - NAS: Servidores Unix NFS e servidores NT CIFS que possibilitavam sistemas de arquivos compartilhados em rede.
- Aplicações:
 - SAN: Processamento de aplicações de banco de dados de missão crítica; Backup de dados centralizado; Operações de recuperação de desastres; Consolidação de armazenamento.
 - NAS: Compartilhamento de arquivo em NFS e CIFS; Transferência de pequenos blocos de dados a longas distâncias; Acesso ao banco de dados limitado somente a leitura.
- Vantagens:
 - SAN: Alta disponibilidade; Confiabilidade na transferência de dados; Trafego reduzido na rede primária; Flexibilidade de configuração; Alto desempenho; Alta escalabilidade; Gerenciamento centralizado; Oferta de múltiplos fornecedores.
 - NAS: Poucas limitações de distância; Adição simplificada da capacidade de compartilhamento de arquivo; Fácil instalação e manutenção.

Muitos poderiam argumentar que SAN é mais poderoso que NAS ou que NFS e CIFS sendo executado sobre TCP/IP gera muita mais sobrecarga para o cliente que o iSCSI sendo executado sobre um Fibre Channel. Estas questões poderiam dar a entender que sistemas SAN resultam em maior throughput para o cliente que os sistemas NAS. Essa observação é verdadeira se considerarmos como clientes os grandes servidores, entretanto grande parte das aplicações são menos exigentes sob esse ponto de vista sendo o desempenho oferecido pelo NAS bastante suficiente.

Enquanto SAN oferece acessos concorrentes em nível de dispositivos, NAS oferece em nível de arquivos e essa é uma exigência de muitas aplicações. Outra vantagem do NAS é que ele é mais simples de se entender e aprender. Em geral, os protocolos e tecnologias que envolvem os sistemas SAN são relativamente complexos. Além disso, SAN é composto de diversas partes de hardware, que são especializados para tal sistema, de diferentes fabricantes. Se o ambiente onde será implantado a SAN nunca recebeu um sistema assim, todos os equipamentos devem ser adquiridos. No caso do NAS isso não acontece, pois o NAS permite usar a infra-estrutura de rede já presente no ambiente. Enfim, o sistema NAS é mais fácil de se manter que o sistema SAN. Outra consequência imediata é que

sistemas SAN são mais caros que os NAS. Outro destaque é que o NAS permite que arquivos sejam replicados automaticamente para outros pontos, tornando mais fácil protegê-los contra falhas.

Entretanto, NAS tem limitações também. Embora os esquemas de replicação oferecidos por alguns NAS propiciem excelentes formas de recuperação, característica mais difícil de ser suportada em SANs, backups terão de serem feitos em fitas em algum momento e fazer backup de um arquivo para uma fita pode ser um desafio. Uma das razões é que fazer um backup completo para fita irá exigir muito mais tarefas de I/O que qualquer outra aplicação. Isso significa que fazer backup de um arquivo relativamente grande irá sobrecarregar bastante o sistema. Nesse sentido, sistemas SANs ainda são mais eficientes. Em complemento, realizar backup de um determinado arquivo significa que o sistema de arquivo terá que ser atravessado da mesma forma como se um usuário estivesse usando-o e isso é outro motivo de sobrecarga no sistema. Por fim, embora seja plausível que a maioria das aplicações não seja limitada pelo throughput de um sistema de arquivos é importante ressaltar que teoricamente um sistema NAS é capaz de transferir mais dados que o NAS. Assim se a aplicação necessitar de grandes transferências de dados é preciso fazer uma avaliação de qual opção melhor se adapta ao perfil da aplicação.

Assim como o NAS possuem vantagens e desvantagens o mesmo ocorre com SAN. A primeira vantagem do SAN em relação ao NAS é que o último só é capaz de atender a requisições em nível de arquivos via NFS e CIFS enquanto o SAN é capaz de atender requisições de acesso a dispositivos. Além disso, enquanto alguns vêm complexidade no SAN outros vêm como flexibilidade, ou seja, os recursos de sistema de arquivos e gerenciamento de volumes disponíveis no SAN não podem ser oferecidos pelo NAS. Outro ponto diz respeito ao fato que SAN pode ser mais rápido que NAS, conforme discutido anteriormente. A capacidade em throughput da SAN possibilita backups em maior escala e recuperação mais fácil. Várias desvantagens da SAN já foram citadas, mas a principais são seu custo e complexidade.

Capítulo 52

Clusters de servidores

Um cluster é um conjunto de máquinas independentes, chamadas nós, que cooperamumas com as outras para atingir um determinado objetivo comum. Por serem fracamente agrupadas, para atingir este objetivo elas devem comunicar umas com as outras a fim de coordenar e organizar todas as ações a serem tomadas. Deste modo, para um usuário externo, o cluster é visto como sendo um único sistema.

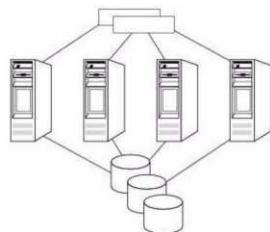


Figura 52.1: Cluster de 4 nós

O objetivo desejado em um cluster resume-se em:

- Alta Disponibilidade, quando se deseja que o cluster forneça determinados serviços, sendo que estes devem estar sempre (ou quase sempre) disponíveis para receber solicitações. Esta probabilidade do serviço estar apto a receber solicitações é um fator dependente do cluster.
- Alto Processamento, quando se deseja que o cluster execute determinadas tarefas, sendo que elas são divididas (na sua íntegra ou em frações de uma mesma tarefa) e processadas separadamente em vários nós, a fim de a velocidade de processamento seja incrementada.

É possível ainda ter uma situação onde o cluster deve atingir os dois objetivos juntos; às vezes, por razões de simplicidade, tal objetivo é atingido eliminando-se alguns rigores das definições acima.

52.0.1 Princípios de um Cluster

Para ser útil, um cluster precisa seguir alguns princípios básicos:

- Comodidade: Os nós em um cluster devem ser máquinas normais interconectadas por uma rede genérica. O sistema operacional também deve ser padrão, sendo que o software de gerenciamento deve ir acima dele como uma aplicação qualquer.
- Escalabilidade: Adicionar aplicações, nós, periféricos e interconexões de rede deve ser possível sem interromper a disponibilidade dos serviços do cluster.
- Transparência: O cluster, que é construído com um grupo de nós independentes fracamente agrupados, deve apresentar-se como um único sistema a clientes externos ao cluster. Aplicações clientes interagem com o cluster como se fosse um único servidor com alta performance e/ou alta disponibilidade.
- Confiabilidade: o cluster deve ter capacidade de detectar falhas internas ao grupo, assim como de tomar atitudes para que estas não comprometam os serviços oferecidos.
- Gerenciamento e Manutenção: Um dos problemas dos clusters é sua manutenção e configuração, que muitas vezes são tarefas complexas e propensas a gerar erros. Um mecanismo fácil de configuração e manutenção do ambiente deve existir a fim de que o cluster não seja um grande sistema complexo com um árduo trabalho de administração.

52.0.2 Abstrações em um Cluster

Um cluster possui vários elementos que, juntos com sua arquitetura, fornecem a funcionalidade desejada. Uma abstração dos elementos é necessária para se poder compreender qual o comportamento de cada um deles.

- Nό: Como visto anteriormente, o nó é a unidade básica do cluster; grupos de nós formam um cluster. Em um cluster, um nó comunica-se com os outros através de mensagens sobre conexões de rede, e falhas em nós podem ser detectadas através de timeouts de comunicação. Um nó é um sistema computacional unicamente identificado conectado a um ou mais computadores em uma rede. Assim, um nó tem quatro componentes principais:
 - CPU;
 - Memória;
 - Reppositório de Armazenamento;
 - Interconexão.
- Recurso: Um recurso representa certa funcionalidade oferecida em um nó. Ele pode ser físico, como por exemplo uma impressora, ou lógico, como por exemplo um endereço IP. Recursos são a unidade básica de gerenciamento, e podem migrar de um nó a outro. Independentemente se for resultado de uma falha ou intervenção humana, o processo de migração de um recurso de um nó para outro é chamado de failover. Como um recurso representa uma funcionalidade, ele pode falhar. Por isso, monitores devem observar

o status dos recursos, a fim de tomar atitudes desejadas em caso de falha (por exemplo, iniciar o serviço em outro nó). Um recurso tem associado a si um tipo, que descreve seus atributos genéricos e seu comportamento. Cada tipo deve ter associado a si mecanismos de iniciação/parada, a fim de que possam ser manipulados corretamente pelo gerenciador do cluster. Dizemos que um nó p hospeda um recurso r se em determinado momento o serviço associado a r está sendo fornecido a partir de p.

- Dependências de Recursos: Freqüentemente, recursos dependem da disponibilidade de outros recursos. Por exemplo, um servidor HTTP depende da presença de uma interface de rede online e de um sistema de arquivos operacional para fornecer os arquivos. Por outro lado, um sistema de arquivos pode depender de um volume manager. Estas dependências são descritas em um grafo de dependências de recursos. Este grafo de dependências descreve a seqüência na qual os recursos devem ser iniciados. Ainda, durante uma migração de recursos ele descreve quais recursos devem ser migrados juntos.
- Grupos de Recursos: Um grupo de recursos é a unidade de migração. Apesar de um grafo de dependências descrever os recursos que devem ser migrados juntos, pode haver considerações adicionais para agrupar recursos. O administrador de sistemas pode associar uma coleção de recursos independentes em um único grupo de recursos, a fim de garantir que todo o grupo seja migrado junto. Políticas de migração são definidas baseadas em grupos. As dependências de recursos vistas acima não podem ultrapassar as barreiras de um grupo, ou seja, recursos de um grupo devem depender somente de recursos contidos em seu próprio grupo.

52.0.3 Arquitetura de um Cluster

Isoladamente, os elementos descritos acima não provêem a funcionalidade desejada do cluster. É necessário que haja uma arquitetura que une todos os elementos e forneça o funcionamento que cada um espera do outro. Devido à natureza fracamente agrupada do cluster, uma abstração ideal de arquitetura deve ser baseada em componentes. Tal estrutura seria formada pelas seguintes camadas básicas:

- Camada de Comunicação: trata das comunicações ponto-a-ponto entre os nós.
- Camada de Ligação: agrupa canais de comunicação em uma única ligação entre dois nós.
- Camada de Integração: forma a topologia do cluster.
- Camada de Recuperação: executa a recuperação (failover) e a inicialização/parada controlada de serviços depois de uma transição do cluster.

Existem ainda quatro serviços chaves para um cluster:

- JDB: armazena estados persistentes internos ao cluster (e usados para o banco de dados do quorum). Nada mais é do que um repositório de informações local a cada nó do cluster.

- Camada de Quórum: determina qual partição do cluster possui autorização para prosseguir com sua execução.
- Serviço de Barreiras: provê um serviço de sincronização global ao cluster.
- Serviço de Nomes: provê um serviço de nomes global ao cluster.

52.0.4 Cluster X Sistemas Distribuídos

Um sistema distribuído consiste em um conjunto de processos concorrentes que cooperam uns com os outros para executar determinada tarefa. Ele é formado por vários nós, que se utilizam da rede de comunicação para trocarem mensagens. As máquinas são autônomas, formando um sistema fracamente agrupado, onde não existe memória compartilhada entre diferentes nós. Entre as aplicações típicas de um sistema distribuído estão serviços de arquivos de/para máquinas distribuídas e distribuição de carga de processamento entre diversas máquinas. A figura 52.2 ilustra essa organização.

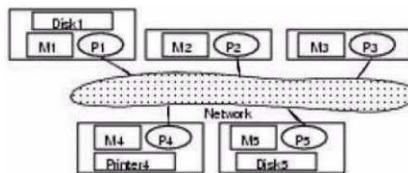


Figura 52.2: Sistema Distribuído

Clusters são uma subclasse de sistemas distribuídos. Geralmente eles são grupos de computadores homogêneos em menor escala, dedicados a um número pequeno e bem definido de tarefas, nas quais o cluster atua como uma única máquina. Entre os usos típicos de um cluster estão adicionar alta disponibilidade a serviços e aumentar a capacidade de processamento do grupo (fazenda de servidores).

A tabela abaixo enumera as principais diferenças entre cluster e sistemas distribuídos. Analisando a tabela pode-se concluir que um cluster nada mais é do que um sistema distribuído especializado para uma determinada tarefa.

- Estrutura:
 - Cluster: homogênea - adquirido para realizar determinada tarefa
 - Sist. Distribuído: heterogênea - montado a partir do hardware disponível.
- Tarefa:
 - Cluster: especializada - feita para executar um pequeno conjunto de tarefas bem definido.
 - Sist. Distribuído: generalizada - usualmente precisam ser ambientes de computação genéricos
- Preço:
 - Cluster: relativamente baixo.

- Sist. Distribuído: barato / caro.
- Confiabilidade:
 - Cluster: tão confiável quanto precisa ser.
 - Sist. Distribuído: pouco / muito confiável.
- Segurança:
 - Cluster: os nós confiam uns nos outros.
 - Sist. Distribuído: os nós não confiam uns nos outros.

52.0.5 Cluster de Alta Disponibilidade

Pelo fato do termo cluster abranger uma grande quantidade de implementações, convencionou-se o uso de termos específicos para cada tipo de implementação. Clusters Beowulf são usados para oferecer escalabilidade e processamento paralelo para execução de funções computacionais. Soluções de clusters de alta disponibilidade, por sua vez, são usados para fornecer alta disponibilidade para serviços ou aplicações. Clusters de alta disponibilidade são comumente conhecidos como Failover Cluster.

Failover é o processo no qual uma máquina assume os serviços de outra, quando esta apresenta alguma falha. O failover pode ser automático ou manual, sendo o automático normalmente esperado em uma solução de alta disponibilidade.

Para se executar o failover de um serviço, é necessário que as duas máquinas envolvidas possuam recursos equivalentes. Um recurso pode ser uma placa de rede, um disco rígido, os dados neste disco ou qualquer elemento necessário à prestação de um determinado serviço. É importante que a solução de alta disponibilidade mantenha recursos redundantes com o mesmo estado, de forma que o serviço possa ser retomado sem perdas. Para isto, pode-se usar técnicas de espelhamento de disco, replicação de dados ou sistemas de armazenamento compartilhados.

Ao ser resolvida a falha, o servidor será recolocado em serviço, e então tem-se a opção de realizar o processo inverso do failover, chamado fallback. O fallback é o processo de retorno de um determinado serviço de uma outra máquina para sua máquina de origem. Este processo também pode ser automático, manual ou até mesmo indesejado. Em alguns casos, em função da possível nova interrupção na prestação de serviços, o fallback pode não ser atraente.

Tipos de Clusters de Alta Disponibilidade

Existem dois tipos básicos de clusters de alta disponibilidade, os simétricos e os assimétricos. Clusters de alta disponibilidade assimétricos são os mais simples de se implementar. Eles se baseiam no conceito de servidores primário e secundário. Servidores que estão provendo algum tipo de serviço são considerados como primários. Servidores que aguardam a ocorrência de alguma falha no servidor primário são denominados servidores secundários. A detecção da falha ocorre através do monitoramento dos servidores primários por parte dos secundários.

A principal diferença entre clusters assimétricos e simétricos é que em um cluster simétrico não há a figura de um servidor inativo aguardando uma falha de

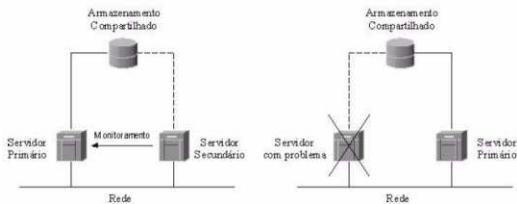


Figura 52.3: Cluster de Alta Disponibilidade Assimétrico

outro, primário, para então servir a aplicação. Neste modelo, ambos servidores servem alguma aplicação. Assim, os termos primário e secundário deixam de ser usados para designar o servidor, passando a ser usado para a aplicação.

De forma semelhante ao que ocorre em clusters assimétricos, na ocorrência de uma falha em um dos servidores, dá-se início ao processo de failover. Neste caso, o servidor ainda ativo assume o controle da aplicação do servidor problemático. Ao contrário do modelo anterior, neste o monitoramento deverá ocorrer entre todos os nós.

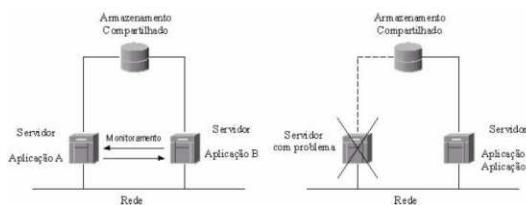


Figura 52.4: Cluster de Alta Disponibilidade Simétrico

Tipos de Armazenamento

Dependendo da escala, performance, funcionalidade e flexibilidade do cluster, são necessárias diferentes implementações de armazenamento. Os principais métodos de armazenamento são o compartilhamento total, sem compartilhamento e espelhamento de disco.

No compartilhamento total, todos os servidores acessam o mesmo meio de armazenamento. Este modelo apresenta potenciais problemas de corrupção de dados quando mais de um servidor está ativo. Para solução deste problema, características de locking são desejadas. Esta implementação é bastante comum em clusters assimétricos.

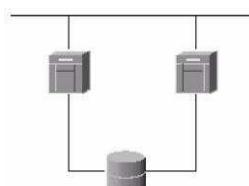


Figura 52.5: Cluster com compartilhamento total de armazenamento

No armazenamento sem compartilhamento, dois ou mais servidores possuem seu próprio meio de armazenamento. No evento de uma falha em um dos servidores, o servidor responsável pela substituição passa a ter acesso total ao meio de armazenamento original do nó defeituoso. Esta implementação é comum em clusters simétricos.

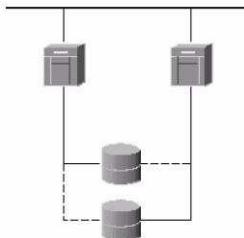


Figura 52.6: Cluster sem compartilhamento de armazenamento

Em um ambiente de espelhamento de discos, não há nenhum tipo de compartilhamento entre os nós servidores. Através do uso de software, os dados são espelhados ou replicados de um servidor para o outro através da rede. O princípio deste modelo é que todos servidores potenciais substitutos devem possuir seu próprio meio de armazenamento e, ao mesmo tempo, uma réplica do armazenamento do servidor a ser substituído.

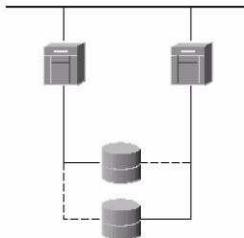


Figura 52.7: Cluster sem compartilhamento de armazenamento

Em um ambiente de espelhamento de discos, não há nenhum tipo de compartilhamento entre os nós servidores. Através do uso de software, os dados são espelhados ou replicados de um servidor para o outro através da rede. O princípio deste modelo é que todos servidores potenciais substitutos devem possuir seu próprio meio de armazenamento e, ao mesmo tempo, uma réplica do armazenamento do servidor a ser substituído.

52.0.6 Cluster de Alto Desempenho

Sistemas de processamento paralelo vêm se tornando cada vez mais populares em função da demanda por processamento de alto desempenho, exigido pelas diversas áreas da ciência (ex.: química, biologia, meteorologia). Infelizmente, os sistemas que oferecem a capacidade de processamento para satisfazer a essa demanda, representados pelas máquinas de arquiteturas maciçamente paralelas ou tem um custo elevado, ou são difíceis de programar, ou ambos. Em função

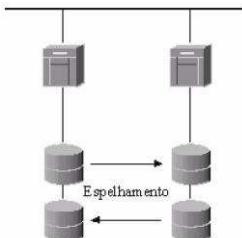


Figura 52.8: Cluster com espelhamento de dados

disso, nos últimos anos, têm-se investido na pesquisa de máquinas paralelas baseadas em clusters de multiprocessadores simétricos por possuírem um custo relativamente mais baixo que as máquinas de arquitetura maciçamente paralelas além de serem mais flexíveis que essas.

Atualmente, existem diferentes tipos de arquiteturas dedicadas à execução de aplicações paralelas, sendo que essas podem ser classificadas em três tipos:

- Arquiteturas maciçamente paralelas (MPP): são arquiteturas que possuem processadores altamente poderosos e links de comunicação dedicados. Este tipo de arquitetura, chamada de supercomputadores ou arquiteturas dedicadas, apresentam um alto custo, devido aos recursos que oferecem. Como exemplo, pode-se citar o Intel Paragon e o IBM SP2.
- Multiprocessadores simétricos (SMP): são arquiteturas compostas por um conjunto de processadores iguais, que se comunicam, geralmente, através de uma mesma memória. O termo simétrico significa que todos os processadores são idênticos em termos de arquitetura interna e poder de processamento. Exemplos dessa arquitetura são os processadores Dual Pentium.
- Redes de estações (NOW): são arquiteturas que correspondem a um conjunto de estações de trabalho interligadas através de uma rede local (LAN) e que servem como plataforma de execução de aplicações distribuídas. Nesse tipo de arquitetura, a comunicação é feita por troca de mensagens entre as diversas aplicações que executam na rede. Esse tipo de arquitetura é largamente utilizado, tanto comercialmente como academicamente. Como exemplo, podemos citar Estações Sun interligadas por rede Ethernet.

Nesse contexto, um cluster pode ser caracterizado como uma plataforma alternativa, aliando o poder e a velocidade de processamento das arquiteturas dedicadas (MPPs) com a disponibilidade de recursos (hardware e software baratos) das redes de estações.

Quando comparados com arquiteturas dedicadas, os clusters de multiprocessadores simétricos apresentam um grande número de vantagens. Eles são relativamente baratos (seus custos são menores que o custo de um supercomputador paralelo), eles oferecem uma boa relação custo/desempenho (porque todo o hardware e o software necessários estão à disposição), e, da mesma forma, suas volumosas vendas atraem investimentos diretos para o seu rápido melhoramento. Eles também permitem um desenvolvimento progressivo de aplicações,

começando com apenas um processador, passando para multiprocessadores e, finalmente, usando um conjunto de estações de trabalho multiprocessadoras interconectadas por alguma rede de comunicação de dados local.

Em complemento, podem-se caracterizar basicamente, duas classes de arquiteturas baseadas em clusters:

- Arquiteturas homogêneas: onde os nodos que compõem o cluster possuem a mesma arquitetura e sistema operacional, logo entendem as mesmas instruções sem a necessidade de conversão de dados a fim de possibilitar o processamento dos mesmos, em diferentes processadores. As arquiteturas homogêneas estão se tornando um padrão na área de clusters de alto desempenho, por serem mais simples de operar e por não apresentarem problemas ligados à conversão de dados entre diferentes sistemas operacionais e ou arquiteturas;
- Arquiteturas heterogêneas: onde os nodos que formam o cluster possuem processadores diferentes e, possivelmente, diferentes sistemas operacionais. Exigem a conversão de dados para que uma instrução possa processar em diferentes processadores. Apresentam problemas ligados à conversão de dados entre diferentes sistemas operacionais e ou arquiteturas.

Além dessas classes de arquiteturas cluster, pode-se distinguir dois tipos de classificação quanto aos nodos que fazem parte do cluster:

- Arquitetura simétrica: possuem todos os nodos homogêneos, sendo que todos os nodos possuem a mesma velocidade e capacidade de processamento, além de possuírem a mesma quantidade de recursos computacionais (ex.: memória). Somente clusters com esse tipo de arquitetura possibilitam uma verdadeira análise de desempenho.
- Arquiteturas assimétricas: possuem nodos diferentes. Podem possuir nodos homogêneos, mas com diferentes velocidades e capacidades de processamento ou nodos homogêneos com diferentes recursos de computação (ex.: memória). Arquiteturas dessa classe dificultam possíveis análises de desempenho.

Capítulo 53

Balanceamento de Carga

Todo hardware tem o seu limite, e muitas vezes o mesmo serviço tem que ser repartido por várias máquinas, sob pena de se tornar congestionado. Estas soluções podem-se especializar em pequenos grupos sobre os quais se faz um balanceamento de carga: utilização do CPU, de armazenamento ou de rede. Qualquer uma delas introduz o conceito de clustering, ou server farm, já que o balanceamento será, provavelmente, feito para vários servidores.

53.1 Balanceamento de armazenamento (storage)

O balanceamento do suporte de armazenamento permite distribuir o acesso a sistemas de arquivos por vários discos, pelo que derivam ganhos óbvios em tempos de acesso. Estas soluções podem ser dedicadas ou existir em cada um dos servidores do cluster. As soluções existentes hoje são implementadas em hardware, dentre elas podemos citar as técnicas RAID (Redundant Arrays of Independent Disks) , SAN (Storage Area Network) e NAS (Network-Attached Storage).

53.2 Balanceamento de rede

O balanceamento da utilização da rede passa sobretudo por reencaminhar o tráfego por caminhos alternativos a fim de descongestionar os acessos aos servidores. O mecanismo/dispositivo responsável pelo balanceamento é chamado de director. Na verdade, ele pode existir sob várias formas, dependendo do(s) serviço(s) que se pretende balancear. Este director serve também de interface entre o cluster de servidores e os clientes do(s) serviço(s) - tudo o que os clientes conhecem é o endereço semi-público deste servidor. Esta abordagem (clássica) é algo limitada, em termos de escalabilidade e número de tramas que o director consegue redirecionar, principalmente devido à velocidade dos barramentos das placas de rede. Existem, no entanto, outras soluções mais complexas que tiram melhor partido das características do protocolo TCP/IP em conjunto com um roteamento de pacotes especializado, tais como NAT, IP Tunneling e Direct

Routing. Outras soluções operam exclusivamente em apenas algumas camadas (níveis) do modelo OSI no nível 4, 5, 6 e 7.

53.2.1 NAT

Em redes de computadores, NAT, Network Address Translation, também conhecido como masquerading é uma técnica que consiste em reescrever os endereços IP de origem de um pacote que passam sobre um router ou firewall de maneira que um computador de uma rede interna tenha acesso ao exterior (rede pública).

Utilizamoe s seguinte situação como exemplo. A estação com IP 192.168.1.13 faz uma requisição, por exemplo, para um endereço externo. O pacote sai com o IP da estação e corre em direção ao intermediador entre o ambiente interno e externo, o gateway. O gateway, através do protocolo NAT mascara o IP da estação com seu IP (200.158.112.126 - que é válido na internet) assim fazendo com que o pacote seja entregue no destino solicitado pela estação. No retorno do pacote, ele parte do endereço externo, chega a nossa rede no servidor NAT (200.158.112.126) e lá volta até o IP da estação assim chegando à estação (192.168.1.13).

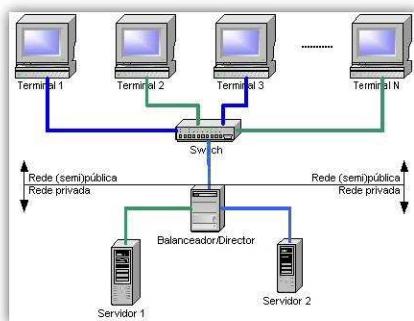


Figura 53.1: Balanceamento de carga NAT

53.2.2 IP Tunneling

Um IP Tunnel é um termo técnico para designar o encapsulamento de um pacote IP dentro de outro, com o propósito de simular uma conexão física entre duas redes remotas através de uma outra rede.

Este processo é frequentemente usado com o protocolo IPsec para criar um meio de conectar duas redes usando uma VPN. O meio para que essas duas redes se vejam é, tipicamente, a Internet. Dessa forma o fator limitador que é a distância é praticamente eliminado, permitindo a utilizadores de uma rede dispôr dos recursos de outra rede remota como se fossem locais.

53.2.3 Direct Routing

A figura 53.2 mostra como o Direct Routing funciona.

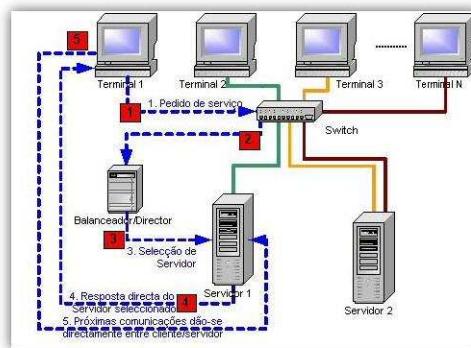


Figura 53.2: Balanceamento de carga com Direct Routing.

53.3 Algoritmos de balanceamento

- Round-Robin: Distribui cada requisição seqüencialmente entre servidores reais. Usando este algoritmo, todos os servidores são tratados igualmente sem considerar sua capacidade ou carga;
- Weighted Round-Robin Scheduling: Distribui cada requisição seqüencialmente entre servidores reais, mas dá mais requisições para servidores com mais capacidade;
- Least-Connection: Distribui mais requisições para servidores com menos conexões ativas. Por ele manter um histórico de conexões ativas através da tabela IPVS, least-connection é um tipo de algoritmo de escalonamento dinâmico, sendo a melhor escolha se há um alto grau de variação na carga do pedido. Se há um grupo de servidores com diferentes capacidades, weighed least-connection é a melhor escolha;
- Weighted Least-Connections (default): Distribui mais requisições para servidores com poucas conexões ativas em relação à sua capacidade. Este algoritmo é ideal quando os servidores contêm hardware de capacidade variedade;
- Destination Hash Scheduling: Distribui requisições para os servidores procurando o endereço IP de destino numa tabela hash estática. Este algoritmo é projetado para usá-lo em um servidor de proxy-cache;
- Source Hash Scheduling: Distribui requisições para os servidores procurando o endereço IP de origem numa tabela hash estática. Este algoritmo é projetado para roteadores LVS (Linux Virtual Server) com múltiplos

firewalls. O LVS é uma solução de balanceamento de carga para sistemas Linux. Basicamente, conta com um software para balancemaneto IP (IPVS) e um para balanceamento na camada de aplicação (KTCPVS).

53.4 Balanceamento de CPU

Este tipo de balanceamento é efectuado pelos sistemas de processamento distribuído e consiste, basicamente, em dividir a carga total de processamento pelos vários processadores no sistema (sejam eles locais ou remotos).

53.4.1 Sistema de processamento distribuído

Um sistema de processamento distribuído ou paralelo é um sistema que interliga vários nós de processamento (computadores individuais, não necessariamente homogéneos) de maneira que um processo de grande consumo seja executado no nó mais disponível, ou mesmo subdividido por vários nós. Adivinham-se, portanto, ganhos óbvios nestas soluções: uma tarefa qualquer, se divisível em várias subtarefas pode ser realizada em paralelo.

A nomenclatura geralmente utilizada neste contexto é **HPC (High Performance Computing)** e/ou **DPC (Distributed/Parallel Computing)**.// Com os desenvolvimentos nesta área, surgiram soluções por software que fazem, geralmente (mas não necessariamente), alterações no núcleo do sistema operacional e que, na maioria dos casos, não são compatíveis entre elas, e dificilmente entre versões diferentes da mesma solução. Assentam, no entanto, em arquiteturas de comunicação standard, como é o caso da **Parallel Virtual Machine** e **Message Passing Interface**. Resumidamente, estas arquiteturas conseguem transportar um processo (tarefa) e o seu contexto (arquivos abertos, etc.) pela rede até outro nó. O nó que originou o processo passa, assim, a ser apenas um receptor dos resultados desse processo.

Atualmente, a principal barreira destes sistemas é implementar mecanismos de **Inter-Process Communication (IPC)**, os **Distributed IPC**, dada a sua extrema complexidade.

A figura 53.3 ilustra as várias camadas de interoperabilidade de um sistema distribuído. Através do gateway a rede pública tem acesso a um supercomputador, sem ter conhecimento disso, dado que só conhece o gateway. Qualquer aplicação executada no gateway (preparada para ser paralelizada) pode ser distribuída por vários nós, entregando os resultados mais rápido do que se fosse processada por apenas um nó.

Parallel Virtual Machine

PVM é a abreviação de Parallel Virtual Machine. Este é um pacote de software que permite que uma rede heterogênea de computadores de todos os tipos seja programada como se fosse apenas uma única Máquina Paralela Virtual. A programação é baseada no paradigma de troca de mensagens. O PVM é

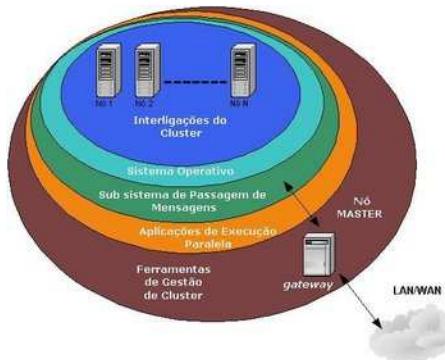


Figura 53.3: Sistema de processamento distribuído.

composto, basicamente, de três partes: uma biblioteca de funções (em C e em FORTRAN77) que implementam para o usuário as diretivas de programação da Máquina Virtual (MV); um processo daemon que rodará em cada host participante da MV e um console de onde podem ser executadas algumas funções básicas de controle (configuração da MV, gerenciamento de tarefas, etc.).

A implementação do PVM é baseada em processos do Unix. Na verdade, cada tarefa PVM é um processo Unix. Isto explica parcialmente a alta portabilidade do sistema para computadores de arquiteturas tão diferentes. Tarefas são as unidades básicas de execução do PVM. É verdade que, em algumas implementações, as tarefas podem não ser processos. Nestes casos, caberá ao implementador garantir a compatibilidade (tais exceções são mais comuns em sistemas de natureza mais complexa como computadores paralelos).

Os programas em PVM podem rodar espalhados por uma rede de natureza heterogênea. Mais particularmente, é possível disparar tarefas em computadores em qualquer parte da Internet (desde que o usuário tenha acesso a esta máquina, obviamente). Esta independência em relação à rede de comunicações que liga os hosts que participam da MV é garantida pelo uso do protocolo TCP/IP para comunicação entre as tarefas através da rede.

O pacote PVM é relativamente pequeno (cerca de 4.5 MB de código fonte em C), e necessita ser instalado apenas uma vez em cada máquina para ser acessível à todos os usuários. Além disso, a instalação não requer privilégios especiais e pode ser efetuada por qualquer usuário.

Message Passing Interface

Message Passing Interface (MPI) é um padrão para comunicação de dados em computação paralela. Existem várias modalidades de computação paralela, e dependendo do problema que se está tentando resolver, pode ser necessário passar informações entre os vários processadores ou nodos de um cluster, e o MPI oferece uma infraestrutura para essa tarefa.

No padrão MPI, uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado. Porém, esses processos podem executar diferentes programas. Por isso, o padrão MPI é algumas vezes referido como MPMD (multiple program multiple data). Elementos importantes em implementações paralelas são a comunicação de dados entre processos paralelos e o balanceamento da carga. Dado o fato do número de processos no MPI ser normalmente fixo, neste texto é enfocado o mecanismo usado para comunicação de dados entre processos. Os processos podem usar mecanismos de comunicação ponto a ponto (operações para enviar mensagens de um determinado processo a outro). Um grupo de processos pode invocar operações coletivas (collective) de comunicação para executar operações globais. O MPI é capaz de suportar comunicação assíncrona e programação modular, através de mecanismos de comunicadores (communicator) que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna.

Parte X

Sistemas Operacionais

Capítulo 54

Ambiente Microsoft Windows 2000/2003

54.1 DHCP - Dynamic Host Configuration Protocol

A implementação do DHCP no Windows 2000/2003 Server é baseada em padrões definidos pelo IETF - Internet Engineering Task Force. Por padrão, este serviço NÃO é instalado durante a instalação típica do Windows 2000/2003 Server. É importante ressaltar que este serviço não pode ser instalado nos Windows 98, Me, 2000 Professional, XP Professional, XP Home, Vista, etc. Já os clientes DHCP podem ser quaisquer hosts baseados no Microsoft Windows.

A maioria das tarefas de administração do DHCP pode ser executada no console DHCP, que é acessado através de: Iniciar >> Programas >> Ferramentas Administrativas >> DHCP. Inclusive, é possível utilizar um único console DHCP para se conectar, remotamente através da rede, com diferentes provedores DHCP. Por padrão, a maior parte das informações (log e banco de dados) deste servidor é armazenada na pasta \windir%\System32\dhcp, onde windir se refere à pasta onde o Windows foi instalado.

54.1.1 Processo de Instalação/Configuração

O processo de instalação se dá seguindo os passos: Iniciar >> Configurações >> Painel de Controle; Adicionar ou Remover Programas; Adicionar ou Remover Componentes do Windows. O DHCP é classificado como um SERVIÇO DE REDE (Networking Services). Após a instalação, não é preciso reinicializar o servidor para que o serviço possa ser disponibilizado, basta seguir alguns procedimentos obrigatórios que serão mencionados abaixo. Lembrando que este serviço é configurado para executar no contexto da conta LOCAL SYSTEM.

Após ter instalado o servidor DHCP, o PRÓXIMO PASSO é autorizar o servidor DHCP no Active Directory. Somente membros do grupo Enterprise Admins (Administradores de empresa) têm permissão para autorizar um servidor DHCP no Active Directory. A autorização obrigatória no Active Directory

é uma medida de segurança para evitar que servidores DHCP secundários sejam introduzidos na rede sem o conhecimento do administrador. O caminho a ser seguido é o: Iniciar >> Programas >> Ferramentas Administrativas >> DHCP; clique no nome do servidor DHCP; Selecione o comando Ação >> Autorizar.

Agora que o servidor DHCP está devidamente instalado e autorizado no Active Directory será necessário a criação de um ou mais ESCOPOS. Para criar e configurar um escopo é utilizado o console de administração do DHCP: Iniciar >> Programas >> Ferramentas Administrativas >> DHCP. Algumas propriedades importantes que devem ser configuradas neste passo são:

- Escopo DHCP: uma faixa específica de endereços IP que deve estar dentro da faixa de endereços da rede onde o servidor DHCP será utilizado. Mais de um escopo pode ser definido para uma rede (10.10.20.0/255.255.255.0), por exemplo: 10.10.10.20.30 a 10.10.20.100; 10.10.10.20.120 a 10.10.20.150; e 10.10.10.20.200 a 10.10.20.250. Cada escopo definirá uma sub-rede. Para cada escopo, é necessário configurar uma série de parâmetros: máscara de sub-rede, default gateway, endereço IP de um ou mais servidores DNS, endereço IP do servidor WINS e assim por diante;
- Intervalos de Exclusão: uma faixa específica de endereços IP para retirar de um escopo, ou seja, endereços que você não quer que sejam concedidos aos clientes da rede. Lembrando que endereços IPs excluídos podem estar ativos na sua rede, como por exemplo, em computadores ou outros dispositivos de rede configurados manualmente (IP fixo);
- Pool de Endereços: conjunto de endereços especificados no escopo DHCP descontando os eventuais intervalos de exclusão. Enfim, os endereços que serão cedidos automaticamente aos clientes do serviço;
- Intervalo de Duração da Concessão: especifica por quanto tempo um cliente DHCP pode usar um endereço IP atribuído antes que seja necessário renovar a configuração com o servidor DHCP;
- Endereços Reservados: concessão permanente a hosts especificados na sua rede. Cada reserva de IP é associada ao MAC Address da interface de rede do host em questão. Vale ressaltar que os endereços reservados devem estar fora do intervalo de exclusão.

Após definir e configurar um escopo, ele deve SER ATIVADO antes que o servidor DHCP comece a fazer concessões aos clientes. No entanto, você não deve ativar um novo escopo até ter especificado as opções DHCP para ele (default gateway, número IP do servidor DNS e assim por diante).

Então, o processo de instalação como um todo pode ser resumido em: instalar servidor DHCP; ativá-lo no Active Directory; criar e configurar o Escopo; e por fim ativar o Escopo.

É importante notar que as opções do DHCP (default gateway, endereço IP de um ou mais servidor DNS e assim por diante), podem ser configuradas em três níveis: servidor (são válidas para todos os escopos configurados no servidor

DHCP); escopo (se aplicam a um escopo especificamente) e opções de uma reserva (se aplicam a reserva e, por padrão, são herdadas do escopo onde a reserva foi criada).

54.1.2 Integração do DHCP com o DNS

O servidor DHCP pode ser utilizado para fazer atualizações dinâmicas no DNS, em nome dos clientes DHCP. Ou seja, quando um cliente DHCP é inicializado e obtém uma concessão de endereço, o DHCP pode atualizar os registros de recurso de endereço (A) e ponteiro (PTR) do cliente, na base de dados do DNS. Esse processo exige o uso de uma opção de DHCP adicional, a opção FQDN do Windows. Essa opção permite ao cliente fornecer seu nome de domínio totalmente qualificado (FQDN) e instruções para o servidor DHCP sobre como deseja que o servidor processe atualizações dinâmicas de DNS (se houver) em seu nome.

Clientes baseados em Windows XP ou Windows Server 2000/2003 atualizam automaticamente os registros do tipo A na zona direta do DNS. Ou seja, não é necessária nenhuma integração entre DNS e DHCP para isso acontecer. Já para os registros da zona reversa (registros PTR), a história é diferente. Esses clientes não conseguem atualizar seus registros PTR diretamente, esta atualização tem que ser feita pelo servidor DHCP.

Clientes mais antigos, baseados em Windows 95, 98, Me ou NT 4.0, não são capazes de atualizar dinamicamente nem seus registros A, nem seus registros PTR no DNS. Para estes clientes, o servidor DHCP atualiza ambos os registros no servidor DNS.

54.1.3 APIPA - Automatic Private IP Addressing

Esta é uma funcionalidade que foi introduzida no Windows 98. Ela também está presente no Windows XP, Vista, Longhorn Server e no Windows Server 2000/2003. Se os clientes estiverem configurados para usar um servidor DHCP (em vez de serem configurados manualmente com um endereço IP e outros parâmetros), o serviço do cliente DHCP entrará em funcionamento a cada vez que o computador for inicializado. O serviço do cliente DHCP usa um processo de três etapas para configurar o cliente com um endereço IP e outras informações de configuração:

- O cliente DHCP tenta localizar um servidor DHCP e obter as configurações do protocolo TCP/IP, a partir desse servidor;
- Se um servidor DHCP não puder ser encontrado, o cliente DHCP configura automaticamente seu endereço IP e máscara de sub-rede usando um endereço selecionado da rede classe B reservada, 169.254.0.0, com a máscara de sub-rede, 255.255.0.0 (RECURSO APIPA). O cliente DHCP irá fazer uma verificação na rede, para ver se o endereço que ele está se auto-atribuindo já não está em uso na rede. Se o endereço já estiver em uso será caracterizado um conflito de endereços. Se um conflito for encontrado, o cliente selecionará outro endereço IP. A cada conflito de endereço, o cliente irá tentar novamente a configuração automática escolhendo um

novo endereço IP. Após 10 tentativas o cliente tenta novamente localizar um servidor DHCP;

- Depois de selecionar um endereço no intervalo de rede 169.254.0.0 que não está em uso, o cliente DHCP irá configurar a interface com esse endereço. O cliente continua a verificar se um servidor DHCP não está disponível. Esta verificação é feita a cada cinco minutos. Se um servidor DHCP for encontrado, o cliente abandonará as informações configuradas automaticamente e usará um endereço oferecido pelo servidor DHCP (e quaisquer outras informações de opções de DHCP fornecidas).

54.1.4 Comandos ipconfig Relacionados ao DHCP

Os principais comandos relacionados ao DHCP e suas funcionalidades são: (1) ipconfig /release: para liberar a concessão (release) do cliente; (2) ipconfig /renew: para renovar a concessão do cliente; (3) ipconfig: para obter informações básicas sobre as configurações do protocolo TCP/IP; (4) ipconfig /all: exibe informações detalhadas sobre as configurações do protocolo TCP/IP em todas as interfaces do computador.

Clientes mais antigos como o Windows 95, 98 ou Me, não disponibilizam o comando ipconfig. Nestes clientes, você pode utilizar o comando WINIPCFG para verificar as configurações do protocolo TCP/IP e para liberar ou renovar uma concessão do servidor DHCP.

54.1.5 Regra "80/20"

Para equilibrar o uso do servidor DHCP, uma boa prática é usar a REGRA "80/20" para dividir os endereços do escopo entre dois servidores DHCP. Se o Servidor 1 estiver configurado para disponibilizar a maioria (aproximadamente 80%) dos endereços, o Servidor 2 pode ser configurado para disponibilizar os outros endereços (aproximadamente 20%) para os clientes.

54.2 DNS - Domain Name System

A implementação do DNS no Windows 2000/2003 Server é baseada em padrões definidos pelo IETF - Internet Engineering Task Force. Por padrão, este serviço NÃO é instalado durante a instalação típica do Windows 2000/2003 Server. É importante ressaltar que este serviço não pode ser instalado nos Windows 98, Me, 2000 Professional, XP Professional, XP Home, Vista, etc. Já os clientes DNS podem ser quaisquer hosts baseados no Microsoft Windows.

A maioria das tarefas de administração do DNS pode ser executada no console DNS, que é acessado através de: Iniciar >> Programas >> Ferramentas Administrativas >> DNS. Inclusive, é possível utilizar um único console DNS para se conectar, remotamente através da rede, com diferentes servidores DNS. Por padrão, a maior parte das informações (lista de servidores ROOT HINTS e informações sobre zonas) deste servidor é armazenada na pasta \%windir%\System32\DNS, onde windir se refere à pasta onde o Windows foi instalado.

54.2.1 Processo de Instalação/Configuração

O processo de instalação se dá seguindo os passos: Iniciar >> Configurações >> Painel de Controle; Adicionar ou remover programas; Adicionar ou Remover Componentes do Windows. O DNS é classificado como um SERVIÇO DE REDE (Networking Services). Após a instalação, não é preciso reinicializar o servidor para que o serviço possa ser disponibilizado. Lembrando que este serviço é configurado para executar no contexto da conta LOCAL SYSTEM.

Após a instalação do servidor DNS, o PRÓXIMO PASSO é criar uma ZONA PRIMÁRIA DIRETA. Por exemplo, vamos supor que você está implementando a estrutura de DNS da rede da sua empresa, cujo domínio root é abc.com.br. A zona é chamada primária porque ela ainda não existe e está sendo criada para conter as informações do domínio. Ela é chamada direta, porque conterá informações para resolução de nomes para endereço IP, ou seja, fornecido um nome no domínio abc.com.br, esta zona conterá informações para retornar o endereço IP associado a este nome.

A maioria das consultas realizadas são as chamadas consultas diretas (Forward Lookup). Neste tipo de consulta, o cliente fornece um nome e o servidor DNS retorna um endereço IP associado a esse nome. O DNS também dá suporte às chamadas consultas inversas (Reverse Lookup), na qual o cliente fornece um endereço IP válido e o servidor DNS retorna o nome associado a esse endereço IP. Originalmente o DNS não foi projetado para dar suporte às consultas reversas. Pela maneira hierárquica como o DNS é organizado, a única maneira para responder este tipo de consulta seria pesquisar todos os servidores DNS existentes no mundo. O que faria com que o tempo de consulta fosse muito elevado, inviabilizando o uso de pesquisas reversas.

Para resolver esta questão foi criado um domínio especial chamado de IN-ADDR.ARPA. Este domínio faz parte das definições atuais do DNS e foi a maneira encontrada para fornecer a resolução reversa de nomes sem que houvesse a necessidade de pesquisar em todos os servidores DNS. Para criar o espaço de nomes reverso, são criados subdomínios do domínio especial in-addr.arpa. Os nomes destes subdomínios são formados pela ordem inversa do número IP da rede. Por exemplo, considere a rede 100.20.50.0/255.255.255.0. A zona para resolução reversa desta rede seria 50.20.100.in-addr.arpa. É importante ressaltar que o uso de domínios in-addr.arpa é aplicado a todas as redes baseadas no protocolo IPv4. Ou seja, este uso não é uma particularidade dos sistemas Windows. Outro aspecto relevante é o fato dos mecanismos de recursão e interação também serem utilizados para a resolução reversa.

Quando você cria uma zona pela primeira vez, ela é uma zona primária. Somente na zona primária é permitido fazer alterações nos registros da zona. Além da zona primária, o DNS permite que sejam feitas cópias da zona em outros servidores DNS. Estas cópias são as ZONAS SECUNDÁRIAS. Por exemplo, você pode ter uma zona primária no servidor DNS da matriz da empresa e criar uma zona secundária (cópia da zona primária), nos servidores DNS de cada filial, para reduzir o tráfego, devido à resolução de nomes DNS, nos links de WAN. As zonas secundárias também são reconhecidas como AUTORIDADES para o

domínio em questão e podem responder às consultas enviadas pelos clientes. A única diferença, em relação à zona primária, é que nas zonas secundárias não podem ser feitas alterações, adições e exclusões de registros de recursos. Sempre que houver alterações, o servidor DNS onde está a zona primária, notifica os servidores DNS onde existem zonas secundárias e, então, os servidores DNS da zona secundário solicitam que as alterações sejam enviadas a partir da zona primária. Com isso o DNS mantém as zonas sincronizadas, ou seja, alterações feitas nas zonas primárias são repassadas para as zonas secundárias. O DNS usa um mecanismo de **REPLICAÇÃO INCREMENTAL**, ou seja, somente as alterações são replicadas e não todo o conteúdo da zona.

Por padrão, quando o servidor DNS é instalado, uma lista chamada ROOT HINTS é criada e gravada em um arquivo chamado Cache.dns localizado na pasta `\%windir%\System32\DNS`. Esta lista armazena informações dos servidores para o domínio root (representado pelo ponto .) e para os domínios de nível superior (.com, .net, .gov, etc.). Este recurso garante a interação entre servidores DNS ao redor do mundo de forma a ser possível a resolução de qualquer FQDN (Nome de Domínio Totalmente Qualificado).

Após a instalação e configuração do servidor DNS é necessária a criação de novos registros de recursos. No Windows 2000/2003 Server, foram dados os seguintes nomes para os campos do registro de recursos: Owner; Time_to_Live; Class; Type; Record-specific data. Além dos tipos de registros padrão IPv4, também são suportados outros tipos, como por exemplo, o tipo AAAA que define um mapeamento de um nome DNS em um endereço de host no padrão IPv6. (ipv6.host1.example.microsoft.com. IN AAAA 4321:0:1:2:3:4:567:89ab). Nos CLIENTES Windows 2000/2003 Server, a configuração do DNS envolve as seguintes tarefas (ao configurar as propriedades do TCP/IP do cliente):

- Configurar um nome (de host) DNS para cada computador;
- Configurar um sufixo DNS primário para o computador, que é posicionado após o nome de host para formar o FQDN. Por exemplo, abc.com.br;
- Configurar uma lista de servidores DNS que serão usados pelos clientes para resolver nomes DNS. Vale ressaltar que a lista sempre é utilizada de forma seqüencial, ou seja, primeiramente o primeiro servidor da lista é consultado. Caso este esteja offline ou haja problemas na comunicação, o cliente consultará o próximo servidor da lista e assim por diante;
- Configurar a lista ou método de pesquisa de sufixo DNS a ser usado pelo cliente quando ele executar pesquisas de consultas DNS para nomes de domínio curtos não qualificados. Além do sufixo DNS primário, podem ser adicionados sufixos específicos, por exemplo, vendas.abc.com. Desta forma, ao se fazer uma consulta do tipo xyz (nome não totalmente qualificado), na verdade, será consultado xyz.vendas.abc.com (sufixo específico) e se não houver êxito também será consultado xyz.abc.com (sufixo primário);

54.2.2 Segurança de Acesso

Você pode definir configurações de segurança para limitar o acesso às configurações dos servidores DNS. Por padrão, os grupos Domain Admins (Administradores do domínio), Enterprise Admins (Administradores de empresa), DNS Admins (Administradores do DNS) e o grupo Administrators têm permissão de controle total nos servidores DNS. Você pode retirar as permissões de todos os grupos deixando apenas as permissões do grupo DNS Admins (e talvez também a do grupo Administrators). Com isso, você pode controlar quais usuários podem fazer alterações nos servidores DNS por meio da inclusão destes usuários no grupo DNS Admins. É muito IMPORTANTE notar que é necessário manter as permissões padrão de acesso a servidor DNS atribuídas ao grupo Authenticated Users (Usuários Autenticados). Se você retirar este grupo, os usuários da rede não conseguirão ler informações no servidor DNS, ou seja, as consultas dos clientes não serão respondidas. Na prática é como se o servidor DNS estivesse desligado.

54.2.3 Integração do DNS com o Active Directory

Outro controle que é muito importante é em relação às quais clientes terão permissão para incluir registros dinamicamente no DNS. Sem um controle de quem pode fazer atualizações dinâmicas, um usuário mal intencionado poderá registrar uma série de registros falsos ou até mesmo registrar milhares de registros em um ataque do tipo "Denial of Service", apenas com o objetivo de paralisar o servidor DNS e com isso os serviços que dependem do DNS. No Windows 2000/2003 Server você pode fazer com que o DNS somente aceite atualizações dinâmicas enviadas por hosts autenticados no domínio, proporcionando assim uma maior segurança no acesso às informações das zonas do DNS. Este tipo de atualização é conhecido como Atualização Dinâmica Segura (Security Dynamic Update). Porém, este tipo de atualização somente está disponível em zonas DNS integradas com o Active Directory, onde você pode definir uma lista de permissão de acesso (semelhante a uma lista de permissões NTFS em uma pasta).

Existem dois pontos fundamentais a serem considerados na integração do DNS com o Active Directory:

- O DNS é necessário, obrigatório, para localização dos DCs (Controladores de Domínio) do domínio;
- O serviço Netlogon usa o novo suporte ao servidor DNS para fornecer registro de controladores de domínio no seu espaço de nomes de domínio DNS. Enfim, sua função básica é executar registros a cerca do AD no DNS.

A integração inicia no momento da instalação do Active Directory em um member server para torná-lo um DC. O assistente de instalação do Active Directory solicita que você informe o nome DNS do domínio para o qual está sendo criado um novo DC. Durante a instalação o assistente deve ser capaz de se conectar a um servidor DNS que seja AUTORIDADE para o domínio informado. Se isso não for possível, o assistente irá se oferecer para instalar e configurar o DNS no próprio servidor que está sendo promovido a DC. Se isso também não for

possível, o Active Directory não poderá ser instalado.

Depois que o Active Directory estiver instalado, existem duas opções para o tipo de armazenamento e duplicação de zonas do DNS:

- Armazenamento de zona padrão usando um arquivo baseado em texto: as zonas armazenadas dessa maneira estão localizadas em arquivos de texto, com a extensão .dns, os quais são armazenados na pasta `\%windir%\System32\DNS` em cada computador que opera um servidor DNS. Os nomes de arquivo de zona correspondem ao nome que você escolhe para a zona durante a sua criação, como Exemplo. `abc.com.br.dns` é o arquivo que armazena informações para a zona `abc.com.br`;
- Armazenamento de zona integrada ao diretório usando o banco de dados do Active Directory: as zonas armazenadas dessa maneira estão localizadas na árvore do Active Directory. Cada zona integrada ao diretório é armazenada em um objeto do tipo `dnsZone` identificado pelo nome que você escolhe para a zona durante a sua criação.

Em redes que distribuem o DNS para oferecer suporte ao Active Directory, as zonas primárias integradas ao diretório são especialmente recomendadas e proporcionam os seguintes benefícios:

- Atualizações multi-master e recursos de segurança avançada baseados nos recursos do Active Directory. Para ZONAS NÃO INTEGRADAS, o modelo de atualização é do tipo single-master. Neste último, somente a zona primária sofre alterações e repassa estas alterações para as zonas secundárias. Se o servidor onde está a zona primária apresentar problemas, novas atualizações dinâmicas não poderão ser processadas enquanto este servidor não for recuperado. Já com ZONAS INTEGRADAS, podem ser feitas alterações em qualquer cópia da zona e existe uma cópia em todos os DCs do domínio onde o DNS estiver instalado. O mecanismo de replicação do Active Directory se encarrega de manter as várias cópias sincronizadas;
- Com esse modelo, não haverá um ponto único de falha (como no caso do modelo baseado em zonas padrão), pois qualquer servidor DNS poderá receber atualizações dinâmicas enviadas pelos clientes;
- Outra vantagem das zonas integradas é que todo objeto do Active Directory possui uma ACL - Access Control List (idêntica à lista de permissões NTFS para uma pasta ou arquivo). Esta lista pode ser editada para ter um controle mais refinado sobre quem tem acesso e qual o nível de acesso;
- A replicação do Active Directory é mais rápida, mais eficiente e mais segura do que o mecanismo de transferência de zonas padrão do DNS.

Nota: Apenas as zonas primárias podem ser armazenadas no Active Directory. Um servidor DNS não pode armazenar zonas secundárias no diretório. Ele deverá armazená-las em arquivos de texto padrão.

54.2.4 Servidor DNS somente Cache

Um servidor DNS somente cache é um servidor que não tem nenhuma zona configurada. A função deste servidor é resolver consultas utilizando os métodos de recursão e/ou interação e armazenar os resultados obtidos no cache do servidor DNS. O cliente envia a consulta para o servidor DNS somente cache, este servidor se utiliza de outros servidores DNS para resolver o nome. O nome é armazenado no cache do servidor DNS somente cache e a resposta é retornada para o cliente que fez a consulta. Futuras consultas a este mesmo nome, dentro do período de expiração, serão respondidas com base nas informações do cache do servidor DNS. Este tipo de servidor deve ter quantidade suficiente de memória RAM para exercer esta função, pois toda a informação do cache do DNS é criada e mantida na memória RAM do servidor. Portanto, o servidor DNS somente cache NÃO ARMAZENA NENHUMA ZONA E NÃO EH AUTORIDADE para nenhum domínio.

54.2.5 Arquivo Hosts

Se não for encontrada a resposta no cache local do DNS, o programa resolvedor consulta as entradas do arquivo hosts. Este é um arquivo texto que fica na pasta onde o Windows 2000/2003 Server foi instalado, dentro do seguinte caminho: \system32\drivers\etc. Este arquivo possui entradas do tipo: um número IP por linha associado a um ou mais nomes:

```
10.200.200.3 www.abc.com.br intranet.abc.com.br  
10.200.200.4 ftp.abc.com.br arquivos.abc.com.br
```

54.2.6 Distribuição de Carga

Por padrão, o serviço DNS usa a atribuição de prioridades a sub-redes locais como o método para dar preferência a endereços IP da mesma rede quando uma consulta de cliente é resolvida para um nome de host que está mapeado a mais de um endereço IP. Ou seja, se o cliente enviou uma consulta para um nome e existem, por exemplo, dois endereços IP associados a este nome, o servidor DNS dará preferência ao endereço IP que for da mesma rede do cliente que enviou a consulta. Esse recurso permite que o aplicativo do cliente se conecte ao host que esteja usando seu endereço IP mais próximo (e normalmente o mais rápido).

Round Robin é o algoritmo de rotação utilizado pelo Windows 2000/2003 Server para a realização da distribuição de carga entre dois ou mais servidores DNS da rede. Vale ressaltar que a atribuição de prioridade a sub-rede locais substitui o uso da rotação Round Robin para nomes com vários endereços IP associados. Entretanto, quando o recurso Round Robin está ativado, ele continua sendo utilizado como método de classificação secundária. Ou seja, caso haja um nome associado a vários endereços IP não pertencentes à rede local, o algoritmo de Round Robin é utilizado para determinar qual endereço será retornado.

54.2.7 Comando ipconfig/dnscmd Relacionadas ao DNS

Os principais comandos relacionados ao DNS e suas funcionalidades são:

- ipconfig /flushdns: limpar o cache local;
- ipconfig /displaydns: exibir o cache local;
- ipconfig /registerdns: força o registro das informações do host no servidor;
- dnscmd /clearcache: limpar o cache do servidor DNS (executado no servidor).

54.3 Active Directory

Antes de falar em Active Directory, é necessário discursar sobre o LDAP (Lightweight Directory Access Protocol). Este é um protocolo padrão inicialmente projetado para o acesso a serviços de diretório X.500 da International Standards Organization (ISO). O LDAP é a versão reduzida de um protocolo chamado DAP (Directory Access Protocol). A principal função do DAP era a de estabelecer, de forma padrão, regras de comunicação de acesso com um diretório baseado no padrão X.500, mas por ser complexo permitiu o surgimento do LDAP que implementa apenas as operações básicas do DAP. A saber: Bind, Read, List, Search, Compare, Modify, Add, Delete e ModifyRDN.

Em outras palavras, o LDAP é simplesmente um protocolo encarregado por definir a maneira através da qual se realizam as pesquisas em uma base de dados. Inicialmente, o LDAP foi criado somente para servir de interface entre os clientes que queriam fazer consultas ao servidor X.500. Isto foi decorrente das deficiências que este protocolo tinha e de sua extrema lentidão. Mais adiante se concluiu que, como a maioria das consultas chegava normalmente através da interface LDAP, seria muito mais vantajoso utilizar o LDAP como um serviço independente de diretório sem a necessidade de utilizar o X.500.

Um diretório é uma base de dados estruturada hierárquica contendo diferentes tipos de informações e oferece uma versatilidade muito grande na hora de buscar a informação desejada. Um serviço de diretório contém informações em forma de entradas. Cada entrada contém uma série de dados (atributos). Uma boa analogia seria uma lista telefônica, que contém entradas: nomes de pessoas, nomes de empresas, etc. Cada entrada contém uma série de atributos: telefone, endereço, etc.

Active Directory é uma implementação do protocolo LDAP feita pela Microsoft. Isto somente é possível porque o protocolo LDAP é um padrão aberto. Existem muitas outras implementações, por exemplo, OpenLDAP (uma versão open source). O Active Directory implementa, além do protocolo LDAP, outras funcionalidades como o DNS e o Kerberos. Ele é utilizado como base de dados centralizada de informações necessárias para a operação de uma rede distribuída de computadores. Isto porque ele armazena informações de usuários e suas permissões, grupos, máquinas existentes na rede, recursos comuns, seja de disco ou físicos, como impressoras e outros periféricos. Além de armazenar vários objetos em seu banco de dados, o AD disponibiliza vários serviços, como: autenticação dos usuários, replicação do seu banco de dados, pesquisa dos objetos disponíveis na rede, administração centralizada da segurança utilizando

GPO (Group Policy). Com a utilização do AD, os usuários poderão ter apenas uma senha para acessar todos os recursos disponíveis na rede. O que representa um grande avanço se comparado a um cenário onde é necessário ter diferentes senhas para diferentes recursos (acessar o sistema principal da empresa, ler seus e-mails, se logar no computador, etc.).

O AD surgiu juntamente com o Windows 2000 Server e pode ser instalado em servidores que executem o Windows Server 2003 (Standard Edition, Enterprise Edition e Datacenter Edition). Vale ressaltar que não se pode instalar o AD em um servidor que execute o Windows Server 2003 - Web Edition, mas se pode ingressá-lo em um domínio do Active Directory como servidor membro.

54.3.1 Tipos de Servidores

Existem dois tipos de servidores:

- Controlador de Domínio (DC - Domain Controller): é o computador que possui o AD instalado, ou seja, é um servidor que possui uma cópia da base de dados do AD. Em um mesmo domínio podemos ter mais de um Controlador de Domínio. As alterações efetuadas em um DC são replicadas para todos os outros DC's. São os DC's quem fazem a autenticação dos usuários de um domínio;
- Servidor Membro (Member Server): é um servidor que não possui uma cópia do AD, porém tem acesso aos objetos do AD. Não fazem a autenticação dos usuários.

54.3.2 Definições de Floresta, Domínio, Site e Unidade Organizacional

Um domínio pode ser definido como um limite administrativo e de segurança. Ele é um limite administrativo, pois as contas de Administrador têm permissões de acesso em todos os recursos do domínio, mas não em recursos de outros domínios. Ele é um limite de segurança porque cada domínio tem definições de políticas de segurança que se aplicam às contas de usuários e demais recursos dentro do domínio e não a outros domínios. Eles também podem ser definidos como unidades de replicação. Todos os controladores de domínio podem receber alterações e replicá-las em todos os outros controladores deste domínio. Cada domínio do Active Directory é identificado por um sistema de nomes de domínios (DNS) e requer um ou mais controladores. Se a rede precisar de mais de um domínio, você poderá criar facilmente vários domínios. Um ou mais domínios que compartilham um esquema e um CATÁLOGO GLOBAL (controlador de domínio que armazena uma cópia de todos os objetos do AD) são chamados de FLORESTA. O primeiro domínio em uma floresta é chamado de DOMÍNIO RAIZ da floresta. Se vários domínios na floresta tiverem nomes de domínio DNS contíguos, essa estrutura será chamada de ÁRVORE DE DOMÍNIO. No AD, os SITES mapeiam a estrutura física da rede, enquanto os domínios mapeiam a estrutura lógica ou administrativa da organização. A estrutura do site e a estrutura do domínio são separadas e flexíveis. Portanto, é possível ter dois cenários distintos: um único site com mais de um domínio; e um único domínio

com mais de um site.

Com a utilização de domínios, podemos fazer com que nossa rede reflita a estrutura de uma empresa. Desta forma, permite-se que usuários de um domínio acessem recursos localizados em outros domínios. Algumas características próprias de cada domínio:

- Um domínio armazena informações somente dos objetos do próprio domínio.
- Um domínio possui suas próprias diretivas de segurança.

Os domínios do Windows 2000/2003 podem estar nos seguintes modos:

- Native (Nativo): utilizado em domínios que possuem somente Controladores de Domínio (DC) Windows 2000/2003;
- Mixed (Misto) (padrão de instalação): utilizado em domínios que possuem Controladores de Domínio (DC) Windows 2000/2003 e Windows NT. Este modo existe para manter compatibilidade com o Windows NT geralmente durante processos de upgrade.

Agora que foram definidos os conceitos de árvore e domínio, podemos então definir UNIDADE ORGANIZACIONAL (Organization Unit - OU). Uma OU é uma subdivisão de um domínio que pode ser utilizada para organizar os objetos deste domínio em um agrupamento lógico para efeitos de administração. Com a utilização de unidades organizacionais, é possível restringir os direitos administrativos apenas em nível da Unidade Organizacional sem que, com isso, o administrador tenha poderes sobre todos os demais objetos do Domínio. Vale ressaltar que a infra-estrutura das OU's não deve ser baseada na estrutura organizacional da companhia, mas sim na infra-estrutura da política da rede.

54.3.3 Recursos do Active Directory

Ao utilizar os domínios baseados no AD, temos os seguintes recursos:

- Logon único: com esse recurso, o usuário necessita fazer apenas um logon para acessar os recursos em diversos servidores da rede, inclusive e-mail e banco de dados;
- Conta de usuário única: os usuários possuem apenas um nome de usuário para acessar os recursos da rede. As contas de usuários ficam armazenadas no banco de dados do AD;
- Gerenciamento centralizado: com os domínios baseados no AD, temos uma administração centralizada. Todas as informações sobre contas de usuários, grupos e recursos da rede, podem ser administradas a partir de um único local no domínio;
- Escalabilidade: os domínios podem crescer a qualquer momento, sem limite de tamanho. A forma de administração é a mesma para uma rede pequena ou grande.

54.3.4 Segurança com o Active Directory

O Active Directory garante um ambiente de diretório seguro para a organização, pois usa os recursos internos de autenticação de logon e autorização de usuários, que são os principais recursos da LSA (Autoridade de Segurança Local). O AD oferece suporte a vários protocolos padrão da Internet e mecanismos de autenticação usados para comprovar a identidade no logon, incluindo Kerberos V5, certificados X.509 v3, cartões inteligentes, infra-estrutura de KPI (chave pública) e protocolo LDAP usando a SSL (Secure Sockets Layer).

A autenticação entre domínios ocorre por meio de relações de confiança. Confiança é uma relação estabelecida entre dois ou mais domínios que permite aos usuários em um domínio serem autenticados por um controlador de outro domínio. As relações de confiança podem ser transitivas ou intransitivas, mas devem sempre estar presentes para que os usuários de um domínio tenham acesso aos recursos compartilhados de outro domínio. As relações de confiança no Windows NT são diferentes das relações de confiança em sistemas operacionais Windows 2000/2003. No Windows NT 4.0 e versões anteriores, as relações de confiança são limitadas a dois domínios e a relação de confiança é unidirecional e intransitiva. Todas as relações de confiança em florestas do Windows 2000/2003 são relações de confiança transitivas bidirecionais. Desta forma, se o domínio X confia no domínio Y, e Y confia no domínio W, então X também confia em W (propriedade transitiva) e W confia em Y que confia em X (propriedade bidirecional).

De uma forma geral, um PROTOCOLO DE CONFIANÇA funciona da seguinte maneira. Um controlador de domínio autentica usuários e aplicativos usando um destes dois protocolos: Kerberos V5 ou NTLM. O protocolo Kerberos V5 é o protocolo padrão para computadores que executam o Windows 2000/2003. Se algum computador envolvido em uma transação não fornecer suporte a Kerberos V5, o protocolo NTLM será usado. Com o protocolo Kerberos V5, após a autenticação, o cliente solicita a um controlador no domínio de sua conta uma permissão para o servidor no domínio confiante. Essa permissão é emitida por um intermediário de confiança do cliente e do servidor. O cliente apresenta essa permissão confiável ao servidor do domínio confiante para que seja autenticada. Quando um cliente tenta acessar recursos de um servidor em outro domínio usando a autenticação NTLM, o servidor que contém o recurso deve entrar em contato com um controlador de domínio no domínio da conta do cliente para verificar as credenciais dessa conta.

Além de proteger o acesso à rede por meio de autenticação, o AD ajuda a proteger os recursos compartilhados, pois facilita a autorização de usuário. Depois que um logon de usuário é autenticado, os direitos atribuídos ao usuário através de grupos de segurança e das permissões concedidas em relação ao recurso compartilhado determinarão se o usuário terá autorização de acesso ao recurso. Ou seja, o AD também implementa CONTROLE DE ACESSO (similar ao do NTFS). Lembrando que o controle de acesso é administrado no nível do objeto, por meio da configuração de diversos níveis de acesso, ou permissões, aos objetos, como: Controle Total, Gravação, Leitura ou Sem Acesso.

54.3.5 Ferramentas de Controle

Várias ferramentas adicionais que podem ser usadas para configurar, gerenciar e depurar o Active Directory estão disponíveis como ferramentas de linha de comando. Essas ferramentas são conhecidas como ferramentas de suporte. As mais usadas são: Movetree, SIDWalk, LDP, Dnscmd, DSACLS, Netdom, NET-Diag, NLTest, Repadmin, Replmon, DSASStat, ADSI Edit, SDCheck, ACLDiag, DFSUtil, Dcdiag e ADMT.

54.4 IIS - Internet Information Services

O IIS (Internet Information Services) é um SERVIDOR WEB criado pela Microsoft para seus sistemas operacionais para servidores. Na verdade, para ser mais preciso, ele é um conjunto integrado de serviços de rede que permite a publicação de conteúdo, disponibilização de arquivos, e execução de aplicações em ambientes Intranet, Extranet e Internet.

Por padrão, o IIS NÃO É instalado durante a instalação típica do Windows 2000/2003 Server. Para instalar o IIS, clique no botão Iniciar >> Painel de Controle >> Ferramentas Administrativas >> clique em Assistente para configurar o Servidor >> clique em Avançar e Avançar. Para você começar a configurar o IIS, clique no botão Iniciar >> Painel de Controle >> Ferramentas Administrativas >> clique em Gerenciador dos Serviços de informações da Internet.

54.4.1 IIS versus Apache HTTP Server

Eles são servidores concorrentes diretos entre si, então, de certa forma, possuem algumas características convergentes e outras divergentes. Cada um possui suas fraquezas e forças. Ao longo desta seção, serão feitas as principais comparações entre os dois servidores.

Origens

O Apache HTTP Server, em sua primeira versão, foi baseado no UNIX httpd Server. Este desenvolvimento ocorreu durante a primeira geração da Web. Por muitos anos, este servidor esteve bastante atrelado à plataforma UNIX. A versão 1.3, que ainda é bastante utilizada, foi a última versão que assumia que o sistema operacional fosse UNIX ou UNIX-LIKE (Linux, OpenBSD, FreeBSD, Sun Solaris, etc.). Sua versão 2.0 foi reprojetada para abstrair o sistema operacional utilizado, passando a ser, portanto, um SISTEMA MULTIPLATAFORMA. Isto é feito utilizando-se basicamente do Apache Portable Runtime (APR) e do Multi-Processing Modules (MPMs). Os MPMs são específicos para cada plataforma. Por exemplo, para a plataforma Windows, este módulo é baseado fortemente no conceito de threads, pois Windows favoresse esse tipo de aplicativo. Já no caso da plataforma Linux, seu desenvolvimento é baseado fortemente em forking process. Sua versão atual é a 2.2.

A Microsoft iniciou a distribuição do IIS a partir do Windows NT 3.51. Desde então, este servidor sempre esteve disponível (não como padrão de in-

stalação). Seu desenvolvimento sempre foi e continua sendo focado, único e exclusivamente, na plataforma Windows. Portanto, ele NÃO É MULTI-PLATAFORMA. Um benefício direto desta abordagem é a vantagem de se poder implementar uma forte cooperação entre o IIS e o próprio kernel do Windows, o que influencia e muito em sua performance. IIS é na verdade uma coleção de serviços, incluindo servidores para: Web, FTP, SMTP e Network News Transfer Protocol (NNTP). Sua versão atual é a 6.0, mas a Microsoft incluirá a versão 7.0 do IIS em sua nova plataforma chamada Windows Longhorn Server.

Licenças

O Apache HTTP Server sempre teve e continua tendo seu código aberto. Sua licença permite seu desenvolvimento e seu uso de forma irrestrita.

Embora o IIS não tenha seu código aberto, ele possui uma licença liberal que garante seu desenvolvimento e uso.

Aplicações Web

O Apache usa por padrão a Common Gateway Interface (CGI) para suportar aplicações web. Foram criados também diversos módulos para suportar outras linguagens de programação. Portanto, o Apache suporta diversas linguagens, tais como: PHP, ASP, Perl, Python, Ruby, C++, etc. Vale ressaltar que devido a sua natureza modular, o Apache pode suportar novas linguagens se necessário.

O IIS suporta por padrão o Active Server Pages (ASP). Pouco se comenta, mas é possível habilitar no IIS a utilização de outras linguagens como Perl ou PHP. Isto se deve ao fato do ASP ser a linguagem preferida pela Microsoft. Depois do lançamento da plataforma .NET em 2002 o IIS ganhou também a função de gerenciar o ASP.NET. Além disso, o IIS passou a suportar novos serviços, como Universal Description, Discovery, and Integration (UDDI) Services, Simple Object Access Protocol (SOAP) e Web Services Description Language (WSDL). De qualquer forma, as opções de linguagens suportadas pelo IIS são menores que as suportadas pelo Apache.

UDDI é uma especificação que define um serviço de registro para Web Services. Um serviço de registro UDDI é um Web Service que gerencia informação sobre provedores, implementações e metadados de serviços. Provedores de serviços podem utilizar UDDI para publicar os serviços que eles oferecem. Usuários de serviços podem usar UDDI para descobrir serviços que lhes interessem e obter os metadados necessários para utilizar esses serviços. A especificação UDDI define: APIs SOAP utilizadas para publicar e obter informações de um registro UDDI; Esquemas XML do modelo de dados do registro e do formato das mensagens SOAP; Definições WSDL das APIs SOAP; Definições de registro UDDI (modelos técnicos - tModels) de diversos sistemas de identificação e categorização, que podem ser utilizados para identificar e categorizar registros UDDI.

WSDL é uma linguagem baseada em XML utilizada para descrever Web Services. Na verdade, trata-se de um documento escrito em XML que além

de descrever o serviço, especifica como acessá-lo e quais são as operações ou métodos disponíveis.

SOAP é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída, utilizando tecnologias baseadas em XML. Sua especificação define uma framework que provê maneiras para se construir mensagens que podem trafegar através de diversos protocolos e que foi especificado de forma a ser independente de qualquer modelo de programação ou outra implementação específica.

Performance

Muitos estudos sobre performance têm sido feitos, contudo, seus resultados não apontam com clareza qual servidor tem maior performance. Na verdade, tem-se mostrado que o que mais influencia na performance é a qualidade da implementação da aplicação Web, indiferentemente se esta sendo executada no IIS ou Apache. Ambos os servidores oferecem alguns recursos importantes para o quesito performance, a saber:

- Uso extensivo de cache (o Apache exige um módulo à parte para isso);
- Suporte ao GNU zip (gzip), o que reduz significativamente a largura de banda necessária entre o servidor e o usuário final;
- Suporte a filtros, o que anula em partes a redução da performance devido ao fato do uso de linguagens interpretadas como ASP, PHP ou Perl;
- Uso de kernel-level listener para aumentar a performance e escalabilidade. Por padrão, o IIS utiliza-se do HTTP.sys que é executado em modo super usuário. De forma análoga, contudo não por padrão e sim de forma opcional, o Apache utiliza-se do phptpd no intuito de obter uma maior performance.

Segurança

Com relação ao canal de comunicação, ambos os servidores utilizam o HTTPS (HTTP over SSL) para a realização da encriptação dos dados. Portanto, não existem grandes diferenças entre os dois servidores com relação à segurança no canal de comunicação.

Um outro aspecto da segurança que deve ser analisado é a vulnerabilidade do próprio host e do Web server. Como o Apache é um sistema multiplataforma, sua análise quanto à segurança não é tão trivial. Esta responsabilidade fica muito mais associada ao sistema operacional que propriamente ao servidor Web. O Apache geralmente é utilizado com os SO UNIX-LIKE, que praticamente em todos os casos executam o servidor Web em modo usuário. Desta forma, falhas de segurança no servidor Web acabam não sendo tão significativas ao sistema como um todo. Por outro lado, o IIS só pode ser utilizado com o Windows, que desde suas primeiras versões é bastante conhecido pelas suas falhas de segurança. Mesmo em suas versões mais atuais, devido à forte colaboração entre o Windows e o IIS, via HTTP.sys, seu histórico de vulnerabilidade é bastante

considerável.

Um terceiro aspecto da segurança a ser analisado é o controle de acesso a conteúdo. Com relação a este aspecto, ambos os servidores são eficientes. O Apache tipicamente utiliza o HTTP Basic Authentication (geralmente sobre o SSL) para a realização das autenticações. É possível também este controle ser gerido pelo próprio administrador local via arquivos de usernames/password, o que não seria nada modular. Como o Apache é altamente modular, uma alternativa a isto é a sua integração com o LDAP ou Active Directory (AD). Já o IIS, que é altamente integrado ao Windows, oferece autenticação e autorização tipicamente via AD. Autorizações para URLs específicas tipicamente ocorrem sobre o sistema ACLs. Cabe ressaltar que em sites dinâmicos, a realização de autorização e autenticação pode se tornar extremamente oneroso. Sendo assim, geralmente esta responsabilidade é passada para a aplicação.

Administração

Sem dúvida a maior diferença entre os dois servidores é com relação aos seus recursos de configuração e administração.

A configuração do Apache é geralmente feita na mão, editando arquivos em formato texto. Isto é feito tanto em pequenos servidores quanto grandes. A principal vantagem existente é a possibilidade de se utilizar de templates que automatizam o processo de configuração em vários servidores em uma Web Farm. Por outro lado, a configuração manual pode se tornar difícil se estiverem sendo configurados múltiplos Web sites. Uma alternativa é a utilização de ferramentas (Web-based ou GUI-based). Webmin é um exemplo quando se estiver utilizando sistemas UNIX-LIKE.

Com relação ao IIS, a administração é sempre feita via console (GUI-based Microsoft Management Console snap-in). Este console prove um ambiente comum para a administração tanto do IIS quanto dos diversos servidores (Web, FTP, SMTP e NNTP) que o compõem.

54.4.2 Principais Componentes do IIS

O IIS tem muitos subcomponentes que podem ser adicionados ou removidos a qualquer momento. A saber:

- Common Files: instala arquivos comuns requeridos pelo programas IIS;
- Documentation: instala a documentação que contempla administração do servidor e publicação de conteúdos;
- File Transfer Protocol (FTP) Server: instala o FTP;
- FrontPage 2000 Server Extensions: instala extensões que possibilitam administração de Web sites usando o FrontPage e o Microsoft Visual InterDev;
- Internet Information Services Snap-In: instala a ferramenta de administração principal;

- Internet Services Manager: instala uma versão baseada em browser da ferramenta de administração. Por padrão, o acesso a este site é somente local;
- NNTP Service: instala o Network News Transfer Protocol (NNTP). Este serviço é utilizado na criação e manutenção de newsgroups;
- SMTP Service: instala o SMTP;
- Visual InterDev RAD Remote Deployment Support: permite que aplicações sejam distribuídas remotamente;
- World Wide Web Server: instala o servidor Web.

54.4.3 Principais Recursos do IIS

Um serviço que não faz parte do IIS, mas que estão altamente relacionados é o INDEXING SERVICE. Este servidor é utilizado na criação de catálogos de documentos que podem ser buscados dentro de um sistema. Quando este serviço é adicionado a um Web site, usuários são capazes de fazer buscas por tópicos de interesse via formulários do próprio HTML.

Exclusivamente com relação ao IIS, os principais recursos disponibilizados são:

- HTTP 1.1 and HTTP Compression;
- Host Headers: permite que diversos Web sites possam ser hospedados em um mesmo servidor. O IIS usa o host name passado no cabeçalho HTTP para determinar o site (pasta no servidor) que o usuário está acessando;
- FTP Restart: permite que downloads via FTP possam ser restabelecidos, em caso de uma eventual desconexão, a partir do ponto anterior;
- Active Server Pages (ASP): utilizada juntamente com HTML e Component Object Model (COM) para criação de aplicações dinâmicas e interativas baseadas em Web;
- Application Protection: permite que aplicações ASP sejam executadas em espaços separados de memória. Existem três níveis de proteção: BAIXA (in-process), MÉDIA (pooled out-of-process) e ALTA (out-of-process);
- Process Accounting and Throttling: Process Accounting prove informações sobre como certo Web site usa a CPU. Process Throttling permite que o uso da CPU;
- WebDAV: possibilita que usuários remotos possam publicar, trancar e administrar recursos usando uma conexão HTTP;
- SSL 3.0 and TLS: provem métodos seguros de troca de informações entre cliente e servidor;
- Digest Authentication: um dos muitos mecanismos de autenticação suportados pelo IIS. Ele é o mecanismo que trabalha melhor entre servidores proxy e firewalls.

54.4.4 Principais Diferenças entre IIS4, IIS5 e IIS6

	IIS4	IIS5	IIS6
Plataforma	Windows NT 4.0	Windows 2000	Windows Server 2003
Arquitetura	32-bit	32-bit	32-bit e 64-bit
Metabase configuration	Binary	Binary	XML
Segurança	Windows authentication; SSL	Windows authentication; SSL; Kerberos	Windows authentication; SSL; Kerberos; Passport support
Administração remota	HTMLA	HTMLA; Terminal Services	Remote Administration Tool (HTML); Remote Desktop
Suporte a Mail	SMTP	SMTP	SMTP e POP3
Suporte a IP	IPv4	IPv4	IPv4 e IPv6

Tabela 54.1: Principais Diferenças entre IIS4, IIS5 e IIS6

54.5 Terminal Services

Um SERVIDOR configurado para funcionar como terminal services pode suprir diversas necessidades em uma rede de uma organização. Os benefícios de se ter um terminal services serão descritos na próxima subseção. Fundamentalmente, o princípio de funcionamento é o seguinte. Diversos CLIENTES estabelecem conexões simultâneas com o TERMINAL SERVICES e a partir daí enviam somente sinais de teclado e mouse. O terminal services é responsável por todo o processamento de aplicações Windows e por enviar ao cliente, via rede, o sinal de vídeo que permitirá a visualização dos resultados do processamento.

Como consequência dessa configuração, não há necessidade dos clientes executarem o mesmo sistema operacional utilizado no terminal services (Windows). Ou seja, CLIENTES podem estar executando qualquer SO, inclusive os não baseados em Windows. Já o terminal services pode ser instalado basicamente em Windows NT e Windows 2000/2003 Server (Standard Server, Enterprise Server, e Datacenter Server). Somente o Windows 2000/2003 Web Server não pode ser configurado como SERVIDOR DE APLICAÇÃO, mas pode ser configurado para ADMINISTRAÇÃO REMOTA.

Durante a instalação do sistema operacional no servidor que funcionará como SERVIDOR DE APLICAÇÃO, é fortemente recomendado a utilização do sistema de arquivo NTFS. Este sistema provê basicamente os seguintes benefícios: estabelecimento de cotas para usuários; compressão de partições e arquivos; controle de acesso em nível de diretório e/ou arquivo.

As configurações do terminal services com relação aos parâmetros de comunicação podem ser realizadas em quatro níveis hierárquicos (descendente): Group Policies (GPO); Terminal Services configuration; User configuration; Client configuration.

Cabe também ressaltar que esta tecnologia não é somente utilizada pela Microsoft. Por exemplo, o Linux Terminal Server Project trabalha exatamente na questão de possibilitar servidores Linux funcionarem como terminal services.

54.5.1 Principais Benefícios

É importante notar que, na realidade, esta tecnologia pode ser aplicada e utilizada tanto em clientes quanto em servidores. A seguir, são sintetizados os mais relevantes benefícios ao se utilizar esta tecnologia. Alguns tópicos dizem respeito à utilização em servidores e outros em clientes.

- Administração Remota versus Servidor de Aplicação. O terminal server (SERVIDOR) pode funcionar nestes dois modos (simultaneamente ou não). No primeiro modo, será possível acesso remoto ao servidor (similar ao Remote Desktop usado em clientes). O segundo modo permite que aplicações sejam executadas no servidor e que diversos clientes accedam essa aplicação de forma simultânea através da rede (este modo é instalado como um módulo a parte do SO). Este recurso é especialmente interessante para compartilhamento de aplicações em uma intranet. Uma observação importante: algumas aplicações instaladas antes da configuração e ativação do servidor como servidor de aplicação podem não funcionar corretamente. Isto ocorre devido ao fato de aplicações multiusuários terem requisitos diferentes de configurações;
- Construção de redes com clientes sem SO próprio. Geralmente estes clientes são equipamentos "pobres", em relação ao hardware. São geralmente computadores reaproveitados. Eles também são chamados de thin clients, pois suas configurações são básicas: HD AUSENTE, um mínimo de RAM, uma placa gráfica simples, uma placa de rede simples e interfaces de entrada/saída (teclado, mouse e monitor). Como estes equipamentos não possuem HD, eles utilizam via rede um único SO, o do terminal services. São estabelecidas ligações necessárias para criações de sessões de trabalho no servidor que são visíveis e controláveis pelos thin clients;
- Esta tecnologia também pode ser utilizada para redirecionamento de diversos recursos de clientes, tais como: drives de CD, disquete, etc.; clipboard; portas COM; portas LPT; impressoras; saída de áudio; etc.;
- Terminal Server Client. Disponível para Windows XP Home Edition e Professional Edition instalados em clientes. O Remote Desktop Protocol (RDP) permite acesso a terminal services. Em ambos os SOs, há disponível também o Remote Assistance, que possibilita um usuário solicitar ajuda a um especialista, que pode assumir o controle da máquina cliente. Para o Windows XP Professional, há um módulo adicional, o Remote Desktop, que possibilita o uso de máquinas clientes de forma remota.

54.5.2 Protocolos de Comunicação

A tecnologia terminal services suporta diversos protocolos de COMUNICAÇÃO e protocolos de TRANSPORTE. Protocolos podem ser instalados ou desinstalados a qualquer momento.

Os protocolos de transporte são aqueles que implementam funcionalidades das quatro camadas inferiores do modelo OSI (física, enlace, rede e transporte). Os principais protocolos suportados são: TCP/IP; NWLink; IPX/SPX; NetBEUI.

Os protocolos de comunicação são aqueles que implementam funcionalidades da camada de aplicação do modelo OSI. Algumas dessas funcionalidades são providas pelos processos de comunicação. Os processos de comunicação suportados pelo terminal services são: named pipes; mail slots; Windows sockets; remote procedure calls; NetBIOS; NetDDE, server message blocks, DCOM (COM+); SOAP.

Um protocolo de comunicação muito importante para a tecnologia terminal services é o Remote Desktop Protocol (RDP). Ele regulamenta as conexões entre terminal services e clientes utilizando como protocolo de transporte somente o TCP/IP. Como o terminal services trabalha de forma flexível, ou seja, se comunicando através de diversas versões de protocolos com clientes que utilizam mais variados SOs, se faz necessário uma troca prévia de parâmetros que caracteriza o cliente. Esta troca ocorre no início da conexão e os parâmetros descrevem as capacidades do cliente. Estas capacidades são divididas em grupos: General Abilities (plataforma e SO do cliente; versão do protocolo; compressão de dados suportada); Bitmaps (tamanho da tela; qualidade da cor; compressão bitmap); Character Commands; Bitmap Cache; Color Table; Panel Activation; Remote Control (habilita a possibilidade do cliente ser controlado remotamente); Cursor (define propriedades de cor do cursor do mouse). A tabela a seguir resume a evolução deste protocolo.

	RDP 5.0	RDP 5.1	RDP 5.2
Surgiu com:	Windows 2000	Windows XP	Windows Server 2003
Características	256 cores (8 bits); Monitoramento Remoto de clientes; Criptografia; com 56 ou 128 bits; Suporte a compressão e caching (reduz o tráfego); Conexão com impressoras e clipboard no cliente	640 x 480 até 1600 x 1200 pixels (24 bits); Transmissão de áudio; Redirecionamento de portas COM; Suporte a Smart Cards; Administração Remota do servidor	Todas as propriedades anteriores com poucas adições, como reconexão automática; transmissão de Windows key (ALT+TAB, etc.)

Tabela 54.2: Evolução do RDP

Atualmente, existem dois aplicativos que utilizam o protocolo RDP, o Re-

mote Desktop Connection (RDC) e o Remote Desktop MMC Snap-in. O RDC, aplicativo padrão para usuários, substituiu o Windows 2000 Terminal Services client e também o Client Connection Manager. O Remote Desktop MMC Snap-in não é um aplicativo para usuário convencionais e sim uma ferramenta para administradores (login de administrador necessário). Esta ferramenta geralmente é utilizada para manter e administrar diversas conexões com múltiplos servidores.

54.5.3 Licenças

Para que um servidor funcione corretamente no modo servidor de aplicação, algumas licenças devem ser adquiridas. São elas:

- Server License: necessária para a instalação do sistema operacional do servidor;
- Client Access License (CAL): necessária para que clientes possam se conectar ao servidor e utilizarem serviços como: compartilhamento de arquivos e impressoras; e outros serviços de redes. Essa licença pode ser obtida por usuário ou por dispositivo, o que for mais vantajoso para quem estiver comprando as licenças;
- Terminal Server Client Access License (TS-CAL): necessária para que clientes possam se conectar ao servidor e utilizarem aplicativos Windows. Essa licença também pode ser obtida por usuário ou por dispositivo.

Lembrando que a cada conexão, é verificado se o usuário ou dispositivo em questão tem sua licença ou não. Esta verificação é feita por um servidor de licenças, que pode ser instalado no próprio equipamento que funcionará o servidor de aplicação ou outros equipamentos da organização, por exemplo, controladores de domínio (DCs).

Parte XI

Banco de Dados

Capítulo 55

Conceitos Básicos

Um sistema de gerência de banco de dados (SGBD) é uma coleção de arquivos e programas inter-relacionados que permitem ao usuário o acesso para consultas e alterações desses dados. O maior benefício do banco de dados é proporcionar ao usuário uma visão *abstrata* dos dados.

A *abstração de dados* se dá em três níveis: nível conceitual, nível lógico e nível físico e serão explicados abaixo.

A capacidade de modificar a definição dos esquemas em determinado nível sem afetar o esquema do nível superior, é chamado *independência dos dados*. A independência de dados lógica (alteração no nível lógico) é mais difícil de ser alcançada que a independência física, uma vez que os programas de aplicação são mais fortemente dependentes da estrutura lógica dos dados do que de seu acesso.

Como o modelo E-R, explicado com mais detalhes, à frente, o modelo orientado a objetos tem por base um conjunto de objetos. Aqui, utilizamos os conceitos de classes, métodos, etc. Ao contrário do modelo E-R, cada objeto, possui uma única identidade, independente dos valores neles contidos.

Existem três modelos lógicos com base em registros:

- Modelo Relacional: Usa um conjunto de tabelas para representar tanto os dados como a relação entre eles;
- Modelo de Rede: Os dados do modelo de rede são representados por um conjunto de registros e as relações entre estes registros são representadas por links (ligações), as quais podem ser vistas pelos ponteiros. Os registros são organizados no banco de dados por um conjunto arbitrário de grafos;
- Modelo Hierárquico: É similar ao modelo de rede, pois os dados e suas relações são representados, respectivamente, por registros e links. A diferença é que no modelo hierárquico, os registros estão organizados em árvores.

Uma *transação* é uma coleção de operações que desempenha uma função lógica única dentro de uma aplicação do sistema de banco de dados. Exigimos que as transações não violem nenhuma das regras de consistência do sistema

de banco de dados. É necessário aceitar inconsistências temporárias para que as transações possam ocorrer sem problemas, mas deverá ser possível retornar caso ocorra uma falha, por isso todo SGDB deve conter um sistema de recuperação.

O *arquivo de dados* armazena o próprio banco de dados, o *dicionário de dados* armazena os metadados relativos à estrutura do banco de dados. É muito usado, portanto uma implementação eficiente é importante.

Um projeto de banco de dados envolve as seguintes etapas:

- Análise de Requisitos: Processo informal que envolve discussões entre grupos de usuários;
- Projeto Conceitual: Descrição de alto nível dos dados a serem armazenados no BD (modelo ER);
- Projeto Lógico: Escolher um SGDB e converter o projeto conceitual em um esquema do modelo de dados do SGDB, exemplo: modelo relacional;
- Refinamento do esquema: Identificar potenciais problemas, usando teorias como a normalização;
- Projeto Físico: Garantir critérios de desempenho, envolve, por exemplo, a construção de índices para tabelas;
- Projeto de Segurança: Identificar grupos de usuários e regras entre esses grupos e seus acessos às tabelas.

Capítulo 56

Abordagem Relacional

56.1 Conceitos

O abordage relacional apresenta basicamente cinco conceitos que são:

- Domínio: Conjunto de valores permitidos para um dado;
- Atributo: Um item de dado do BD (possui um nome e um domínio);
- Tupla: Um conjunto de pares (atributo, valor). Exemplo: (idade, 34);
- Relação: É um conjunto de tuplas. Composto por um cabeçalho e um corpo. O cabeçalho possui um número fixo de atributos não-ambíguos (grau da relação). Corpo é um número variável de tuplas (cardinalidade da relação) em que a ordem não relevante.
- Chave: Conjunto de um ou mais atributos de uma relação;

56.2 Esquemas e Restrições de Integridade

O esquema de um BD relacional é um conjunto de esquemas de relação, $S = R_1, \dots, R_m$. Uma instância de S é um conjunto de relações $BD = r_1, \dots, r_m$, onde cada r_i é uma instância do esquema de relação R_i .

A *restrição de domínio* é a condição em que os atributos devem ser aceitos somente dentro de um conjunto especificado, por exemplo: um valor maior que quatro ou um valor diferente de nulo.

A *restrição de integridade referencial* é a condição em que desejamos garantir que um valor que aparece em uma relação para um dado conjunto de atributos também apareça para um certo conjunto de atributos de outra relação. Um conjunto de atributos FK no esquema da relação R_1 é uma chave estrangeira de R_1 que referencia R_2 se:

1. os atributos em FK possuem os mesmos domínios que os atributos da chave primária PK de R_2 ; diz-se que os atributos FK fazem referência à relação R_2 .

2. para qualquer tupla t_1 de $r_1(R_1)$, ou existe uma tupla t_2 em $r_2(R_2)$ tal que $t_1[FK] = t_2[PK]$, ou $t_1[FK]$ é nulo.

Violações de restrições que podem ocorrer:

1. inserir(v1,...,vn) pode causar:
 - violação de integridade de identidade (chave prim. nula);
 - violação de restrição de chave;
 - violação de integridade referencial.
2. excluir(PK) pode causa violação de integridade refencial. Podemos lidar com isso das seguintes maneiras:
 - rejeitar;
 - propagar a exclusão; .
 - modificar os valores dos atributos referenciados para nulo.
3. atualização(PK, atributo, novo_valor): podem causar as mesmas violações vistas anteriormente quando ou a chave primária ou chave estrangeira são atualizadas.

Uma *asserção* é um predicado que expressa uma condição que desejamos que seja sempre satisfeita no banco de dados. Restrições de domínio e de integridade são casos particulares de asserções. Aserções complexas podem prejudicar o desempenho do banco de dados.

Um *gatilho(trigger)* é um comando que é executado pelo sistema automaticamente, em consequência de uma modificação no banco de dados. O padrão SQL-92 não dispõe da gatilhos.

A noção de *dependência funcional* generaliza o conceito de superchave. Quando dizemos que uma determinada relação possui a dependência funcional $\alpha \rightarrow \beta$ queremos dizer que para todos os pares de tuplas t_1 e t_2 , se $t_1[\alpha] = t_2[\alpha]$ então $t_1[\beta] = t_2[\beta]$. Ou seja, se duas tuplas assumem os mesmos valores para o conjunto de atributos α então também deve assumir os mesmos valores para o conjunto de atributos β . A *clausura* do conjunto dependências funcionais F é denotada por F^+ e inclui as dependências funcionais logicamente implícitas.

Capítulo 57

Modelagem Entidade Relacionamento

57.1 Conceitos

Nesta etapa, nos movemos de uma descrição informal do que os usuários desejam para uma descrição formal. Os conceitos mais importantes são:

- Entidade: Objeto do mundo real distingüível de outros objetos, é descrita utilizando um conjunto de atributos;
- Conjunto de Entidades: Uma coleção de entidades similares. Exemplo: todos os funcionários;
- Chave
 - Superchave: é o conjunto de um ou mais atributos que, tomados coletivamente, nos permite identificar de maneira unívoca uma entidade em um conjunto de entidades;
 - Candidata: superchave em que nenhum subconjunto é superchave;
 - Primária: chave candidata definida pelo projetista do BD para identificar as entidades;
 - Estrangeira: atributo de um conjunto de entidades que é chave primária de outro conjunto de entidades;
- Relacionamento: Associação entre duas ou mais entidades;
- Conjunto de Relacionamentos: Coleção de relacionamentos similares;
- Atributo Descritivo: registram informação sobre o relacionamento;
- Atributo Multivalorado: quando mais de um valor pode ser inserido, por exemplo, um funcionário pode ter vários dependentes e pode-se criar um atributo multivalorado para colocar o nome de cada um desses dependentes.

57.2 Cardinalidade

- Um para um: uma entidade em A está associada a no máximo uma entidade em B.
- Um para muitos: uma entidade em A está associada a qualquer número de entidades em B.
- Muitos para muitos: uma entidade em A está associada a qualquer número de entidades em B e uma entidade em B está associada a qualquer número de entidades em A.

57.3 Representação Gráfica

- Conjunto de entidades: retângulos;
- Atributos: elipses;
- Conjunto de relacionamentos: losangos;
- Atributos multivalorados: elipses duplas;

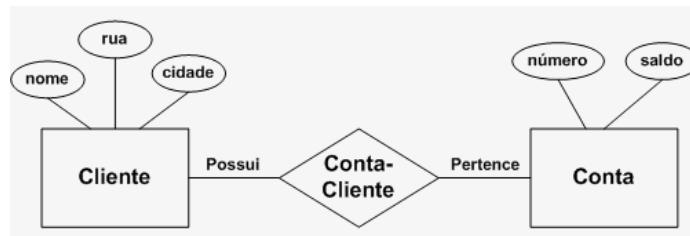


Figura 57.1: Um Diagrama Entidade Relacionamento

57.4 Recursos do Modelo Entidade Relacionamento

- Conjunto de Entidades Fracas: não possui chave primária, mas o identificador é composto juntamente com a chave primária de um conjunto de entidades dominante (forte). Esse identificador é chamado de *chave parcial (pname)*. Essa relação é feita através de um *relacionamento identificador*. Por exemplo, um pedido de compra pode possuir vários itens, mas cada um desses itens está associado a somente um pedido de compra. Poderia-se associar um identificador parcial para cada item em relação ao seu pedido. A chave parcial poderá ser formada entre esse identificador parcial e a chave primária do conjunto de entidades que representa o pedido. O relacionamento é um para muitos e conjunto de entidades fracas tem participação total (todo item está associado a um pedido).

- Especialização: no caso de um conjunto de entidades que possuem subgrupos de entidades, pode-se realizar a especialização fazendo que esses subconjuntos tenham os mesmos atributos do conjunto de entidades principal mais atributos específicos para o subconjunto que não é compartilhado com outros subconjuntos. É como se fosse uma herança.
- Generalização: Difere da especialização no sentido de como é feito o projeto. Na generalização, o projetista procura atributos em comum para formar um conjunto de entidades "pai". A representação no diagrama é a mesma (uso do losango ISA).
- Agregação: Permite-nos tratar um conjunto de relacionamento como um conjunto de entidades com o propósito de permitir a participação em outros relacionamentos.

Capítulo 58

Normalização

58.1 Aspectos desejáveis em um bom projeto

Considere o seguinte caso: queremos fazer um relatório que representa um pedido de compra, gostaríamos de obter os nomes dos produtos, seu volume, seu peso e os seus preços e suponha que alguém tenha pensado num esquema de item de um pedido da seguinte maneira: $item(quantidade, nome_do_produto, volume, peso, preço)$. Sabemos que um determinado produto terá sempre o mesmo volume e o mesmo peso. Dizemos que há *redundância* nesse caso, já que poderíamos criar outro esquema $produto(id_produto, volume, peso)$ e associarmos $id_produto$ como chaves estrangeira em $item$. Acabamos de realizar o que é chamado de *decomposição*. Decomposições descuidadas, entretanto, podem gerar outro tipo de projeto de má qualidade. Podemos recuperar a relação desejada por meio da operação de junção natural (*natural join*), mas pode ocorrer que a resposta alcançada tenha mais tuplas do que realmente deveria ter, devido a uma má decomposição (*decomposição com perda de junção*).

Um conjunto de esquemas de relações R_1, R_2, \dots, R_n é uma decomposição de R se $R = R_1 \cup R_2 \cup \dots \cup R_n$. Assim, é sempre válido que $r \subseteq r_1 \bowtie r_2 \bowtie \dots \bowtie r_n$.

Seja R um esquema de relação e F um conjunto de dependências funcionais sobre R . Sejam R_1 e R_2 formas de decomposição de R . Essa decomposição é uma decomposição sem perda de junção de R se ao menos uma das seguintes dependências funcionais está em F^+ :

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$.

Outro aspecto que desejamos para o banco de dados é a preservação de dependência. O sistema deve checar se uma atualização no banco de dados criará uma relação ilegal (que não satisfaça todas as dependências funcionais).

58.2 Forma normal de Boyce-Codd

Uma relação do esquema R está na FNBC (forma normal de Boyce-Codd) com respeito a conjunto F de dependências funcionais se todas as dependências fun-

cionais em F^+ da forma $\alpha \rightarrow \beta$, em que $\alpha \subseteq R$ e $\beta \subseteq R$ atendem ao menos uma das exigências abaixo:

- $\alpha \rightarrow \beta$ é uma dependência funcional trivial (isto é, $\beta \subseteq \alpha$).
- α é uma superchave para o esquema R .

58.3 Terceira forma normal

Podemos aceitar uma forma normal mais fraca chamada *terceira forma normal* ($3FN$). Essa forma normal permite dependências funcionais não-triviais.

Uma relação do esquema R está na $3FN$ com respeito a conjunto F de dependências funcionais se todas as dependências funcionais em F^+ da forma $\alpha \rightarrow \beta$, em que $\alpha \subseteq R$ e $\beta \subseteq R$ atendem ao menos uma das exigências abaixo:

- $\alpha \rightarrow \beta$ é uma dependência funcional trivial.
- α é uma superchave para o esquema R .
- Cada atributo de A em $\beta - \alpha$ está contido em uma chave candidata de R .

Todo esquema $FNBC$ é também $3FN$. Nem todo $FNBC$ preserva as dependências funcionais, já em um projeto $3FN$ é sempre possível garantir as dependências funcionais e que as decomposições são sem perda de junção. Entretanto, na $3FN$ pode haver repetição de informação e uso de valores nulos para representarmos um relacionamento entre itens de dados, mas mesmo assim é menos pior do que não garantir a preservação de dependência.

Capítulo 59

Transformação do Modelo Conceitual

- Conjunto de entidades fortes: uma coluna para cada um de seus atributos;
- Conjunto de entidades fracas: uma coluna para cada um de seus atributos mais as colunas que compreendem os atributos que formam a chave primária do conjunto de entidades dominantes;
- Conjunto de Relacionamentos: Formado pelos seus atributos descritivos e pelas chaves primárias de cada uma das entidades participantes;

Um conjunto de Relacionamento que possui a cardinalidade muitos para um ou um para um e não possui atributos descritivos não precisa ser representado em tabela. Por exemplo, no caso de um relacionamento entre um conjunto de entidades fraca e um conjunto de entidades forte, a chave primária do conjunto de entidades forte funciona como um atributo no conjunto de entidades fraca (chave estrangeira).

Capítulo 60

Linguagem SQL

60.1 Criação de tabela

Uma nova tabela pode ser criada especificando o seu nome juntamente com os nomes das colunas e seus tipos de dado:

```
CREATE TABLE clima (
    cidade      varchar(80),
    temp_min    int,           -- temperatura mínima
    temp_max    int,           -- temperatura máxima
    prcp        real,          -- precipitação
    data        date
);
```

O comando INSERT é utilizado para carregar as linhas da tabela:

```
INSERT INTO clima VALUES ('São Francisco', 46, 50, 0.25, '1994-11-27');
```

A sintaxe usada anteriormente requer que seja lembrada a ordem das colunas.
Uma sintaxe alternativa permite declarar as colunas explicitamente:

```
INSERT INTO clima (cidade, temp_min, temp_max, prcp, data)
    VALUES ('São Francisco', 43, 57, 0.0, '1994-11-29');
```

60.2 Consultas

Para ver os dados de uma tabela, a tabela deve ser consultada. O comando SELECT do SQL é utilizado para esta função. Por exemplo, para ver todas as linhas da tabela clima digite:

```
SELECT * FROM clima
```

Pode ser especificada qualquer expressão arbitrária na lista de seleção. Por exemplo, pode ser escrito

```
SELECT cidade, (temp_max+temp_min)/2 AS temp_media, data FROM clima
```

Operadores booleanos arbitrários (AND, OR e NOT) são permitidos na qualificação da consulta. Por exemplo, o comando abaixo obtém o clima de São Francisco nos dias de chuva:

```
SELECT * FROM clima
  WHERE cidade = 'São Francisco'
    AND prcp > 0.0;
```

Pode ser desejado que os resultados da seleção retornem em uma determinada ordem, ou com as linhas duplicadas removidas:

```
SELECT DISTINCT cidade
  FROM clima
  ORDER BY cidade;
```

A consulta da forma:

```
SELECT clima.cidade, clima.temp_min, clima.temp_max,
      clima.prcp, clima.data, cidades.localizacao
    FROM clima, cidades
   WHERE cidades.nome = clima.cidade;
```

Pode ser escrita na forma alternativa:

```
SELECT *
  FROM clima INNER JOIN cidades
  ON (clima.cidade = cidades.nome);
```

Desejamos fazer a varredura da tabela clima e, para cada uma de suas linhas, encontrar a linha correspondente em cidades. Se nenhuma linha for encontrada, desejamos que algum "valor vazio" seja colocado nas colunas da tabela cidades. Este tipo de consulta é chamado de junção externa (outer join). As consultas vistas anteriormente são junções internas (inner join). O comando então fica assim:

```
SELECT *
  FROM clima LEFT OUTER JOIN cidades
  ON (clima.cidade = cidades.nome);
```

Também é possível fazer a junção da tabela consigo mesmo. Isto é chamado de autojunção (*self join*).

60.3 Funções de agregação

Existem funções de agregação para contar (count), somar (sum), calcular a média (avg), o valor máximo (max) e o valor mínimo (min) para um conjunto de linhas.

Como exemplo, podemos obter a maior temperatura mínima ocorrida em qualquer lugar com

```
SELECT max(temp_min) FROM clima;
```

Se desejarmos saber a cidade (ou cidades) onde esta leitura ocorreu, podemos tentar

```
SELECT cidade FROM clima WHERE temp_min = max(temp_min);  
ERRADO!
```

mas não funciona porque a função de agregação max não pode ser usada na cláusula WHERE (esta restrição existe porque a cláusula WHERE determina as linhas que vão passar para o estágio de agregação e, portanto, precisa ser avaliada antes das funções de agregação serem computadas). Entretanto, uma forma correta é a subconsulta abaixo:

```
SELECT cidade FROM clima  
WHERE temp_min = (SELECT max(temp_min) FROM clima);
```

As agregações também são muito úteis quando combinadas com a cláusula GROUP BY. Por exemplo, pode ser obtida a maior temperatura mínima observada em cada cidade com

```
SELECT cidade, max(temp_min)  
FROM clima  
GROUP BY cidade;
```

Cada resultado da agregação é calculado sobre as linhas da tabela correspondendo a uma cidade. Estas linhas agrupadas podem ser filtradas utilizando a cláusula HAVING

```
SELECT cidade, max(temp_min)  
FROM clima  
GROUP BY cidade  
HAVING max(temp_min) < 40;
```

É importante compreender a interação entre as agregações e as cláusulas WHERE e HAVING do SQL. A diferença fundamental entre WHERE e HAVING é esta: o WHERE seleciona as linhas de entrada antes dos grupos e agregações serem computados (portanto, controla quais linhas irão para o computo da agregação), enquanto o HAVING seleciona grupos de linhas após os grupos e agregações serem computados. Portanto, a cláusula WHERE não pode conter funções de agregação; não faz sentido tentar utilizar uma agregação para determinar quais linhas serão a entrada da agregação.

60.4 Atualizações e exclusões

As linhas existentes podem ser atualizadas utilizando o comando UPDATE. Suponha que foi descoberto que as leituras de temperatura estão todas mais altas 2 graus após 28 de novembro de 1994. Estes dados podem ser corrigidos da seguinte maneira:

```
UPDATE clima  
SET temp_max = temp_max - 2, temp_min = temp_min - 2  
WHERE data > '1994-11-28';
```

Suponha que não estamos mais interessados nos dados do clima em Hayward. Então precisamos excluir suas linhas da tabela. As exclusões são realizadas utilizando o comando DELETE:

```
DELETE FROM clima WHERE cidade = 'Hayward';
```

60.5 Visões

Suponha que a consulta combinando os registros de clima e de localização das cidades seja de particular interesse para sua aplicação, mas que você não deseja digitar esta consulta toda vez que necessitar dela. Você pode, então, criar uma *visão* baseada na consulta, atribuindo um nome a esta consulta pelo qual é possível referenciá-la como se fosse uma tabela comum.

```
CREATE VIEW minha_visao AS
    SELECT cidade, temp_min, temp_max, prcp, data, localizacao
        FROM clima, cidades
        WHERE cidade = nome;
```

60.6 Chaves estrangeiras

Desejamos ter certeza que não serão inseridas linhas na tabela clima sem que haja uma entrada correspondente na tabela cidades. Isto é chamado de manter a integridade referencial dos dados. As declarações para as tabelas ficariam assim:

```
CREATE TABLE cidades (
    cidade      varchar(80) primary key,
    localizacao point
);

CREATE TABLE clima (
    cidade      varchar(80) references cidades,
    temp_min    int,
    temp_max    int,
    prcp       real,
    data        date
);
```

Capítulo 61

Conceitos de Datawarehousing e Business Intelligence

61.1 Banco de Dados Multidimensionais

Os BDs Multidimensionais são altamente otimizados para minimizar o tempo de consulta e apresentação. Os dados extraídos dos sistemas fontes são gravados diversas vezes em vetores ou "arrays" ordenadamente de forma que a realização da consulta seja implementada como uma simples varredura de uma parte de um vetor. O BD Multidimensional também possui um conjunto de funções matemáticas que podem ser estendidas e que são executadas dinamicamente à varredura dos registros. Com isso, o sistema trabalha com arquivos temporários menores, reduzindo drasticamente o tempo de I/O.

Claro que a manutenção destas estruturas ordenadas são muito mais demoradas que num BD convencional. Mas como as atualizações são realizadas periodicamente (de mês em mês, geralmente) e o seu tempo não é crítico ao nível de aplicação (não importa para o usuário se a carga levou 24h, mas sim que a consulta levou menos de 10 min), não existe grande preocupação com o tempo de atualização dos dados.

Além de já guardarem os dados de forma ordenada, os BDs Multidimensionais, também se utilizam de estruturas auxiliares de consulta que guardam o resultado de consultas anteriores armazenados em disco e apenas consultam os registros novos que não foram consultados na pesquisa anterior, gerando um incremental. Exemplo: em 01/02/2001 foi realizada uma consulta que demonstrava o número de produtos vendidos por cidade e mês da venda. Para fazer esta consulta o banco de dados teve que varrer todos os meses e todas as cidades que tiveram vendas. Quando esta consulta é realizada novamente em 01/04/2001. O BD em vez de varrer tudo de novo, apenas procura pelos registros de vendas de Fevereiro e Março e utiliza os valores gerados na consulta de 01/02/2001 que foram armazenadas em um arquivo de consulta.

Outra estrutura de consulta muito utilizada é a geração de diversos níveis de granularidade das informações. Isto é, quando o usuário faz uma consulta como a citada no exemplo acima, o BD em vez de fazer os agrupamentos somente ao nível de Cidade e Mês ele dispara simultaneamente agrupamentos por Bairro, Estado, Semana e Trimestre, supondo que a próxima consulta a ser disparada pelo usuário seja uma operação de "Drill Down ou Roll UP".

Varias outras estratégias de minimizar o tempo de consulta existem, mas os dois tipos de estratégias apresentadas: consulta incremental e em vários níveis de granularidade são as mais comumente utilizadas hoje em dia.

Bancos de dados multidimensionais são sistemas proprietários que não seguem padrões (linguagem, API) estabelecidos pela indústria de banco de dados. Isto se torna uma desvantagem para tais sistemas, uma vez que a arquitetura não é aberta.

61.1.1 Modelagem Multidimensional

O modelo multidimensional visa facilitar a compreensão do estruturamento dos dados armazenados tanto para desenvolvedores quanto para os usuários do sistema. Neste tipo de modelo existem três elementos básicos: os fatos, as dimensões e as medidas ou variáveis.

Fato é uma coleção de itens de dados compostos de dados de medidas e de contexto. O fato reflete a evolução dos negócios do dia-a-dia e é representado por valores numéricos. É implementado em tabelas denominadas tabelas de fato (fact tables).

Dimensões são os elementos que participam de um fato. Elas determinam o contexto de um assunto de negócios. As dimensões normalmente não possuem valores numéricos, pois são somente descritivas e classificatórias dos elementos que participam de um fato. Um Datawarehouse que analisa vendas de um produto (fato) poderia ter as seguintes dimensões: tempo, localização, clientes, vendedores.

As dimensões podem ser compostas por membros que podem conter hierarquias. Membros (atributos) são as possíveis divisões ou classificações de uma dimensão. A dimensão tempo poderia ser dividida nos seguintes membros: ano, trimestre e mês e a dimensão localização em: cidade, estado e país.

As Medidas (variáveis) são os atributos numéricos que representam um fato e é sobre eles que são feitas as análises do negócio. São por exemplo, o valor em reais das vendas, o número de unidades vendidas, o custo da venda. Uma medida é determinada pela combinação das dimensões que participam de um fato, e estão localizadas como atributos de um fato.

Para facilitar o entendimento, o modelo multidimensional é representado pelo desenho de um cubo. Todavia, o número de dimensões geralmente é maior

que três, o que sugere um hipercubo. Como um hipercubo é algo difícil de se representar à literatura utiliza geralmente como referência somente o cubo. A figura 61.1 mostra um cubo para a medida vendas.

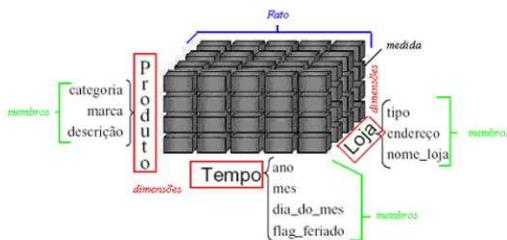


Figura 61.1: Abordagem Dimensional representada por cubos

Utilizando a abstração da organização da informação em um cubo, podemos definir mais facilmente as quatro operações básicas OLAP:

- Drill Down: consiste em detalhar o nível dos dados, navegar do mais alto nível até o dado detalhado. Exemplo: Em vez de ver as vendas anuais, passar a ver as vendas trimestrais. Ou em vez de ver as vendas por estado, ver por cidade;
- Roll UP: operação inversa da Drill Down. Navegação do nível de detalhe para o mais alto nível de sumarização de dados;
- Slice: operação de corte do cubo, mantendo a mesma perspectiva de visão. A idéia é selecionar apenas algumas dimensões ou membros de dimensões. Exemplo: analisar somente as vendas do Estado de São Paulo, no intervalo de 1998 até 2001;
- Dice: mudança da perspectiva de visão. Pode simplesmente ser a alteração da posição das linhas pelas colunas numa tabela, ou a apresentação dos dados de trás para frente (2001, 2000, 1999 em vez de 1999, 2000, 2001). Ou operações mais complexas que mostram a variabilidade de uma medida ao longo das diferentes instâncias de uma ou mais dimensões.

Dois modelos multidimensionais comuns são: o esquema estrela e o flocos de neve (snowflake). No esquema estrela existe uma tabela dominante no centro do esquema. Esta é a única tabela com múltiplos relacionamentos para as outras tabelas. As outras tabelas possuem um único relacionamento para a tabela central. A tabela central é chamada fato (fact table) e as outras tabelas são chamados de dimensão (dimension table). A figura 61.2 ilustra o esquema estrela.

Como já dissemos uma dimensão pode conter uma ou mais hierarquias que podem ser decompostas em tabelas separadas, gerando uma estrutura conhecida por snowflake. A figura 61.3 ilustra uma estrutura snowflake.

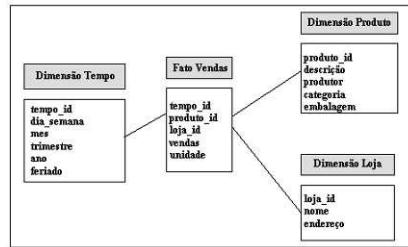


Figura 61.2: Modelo Estrela

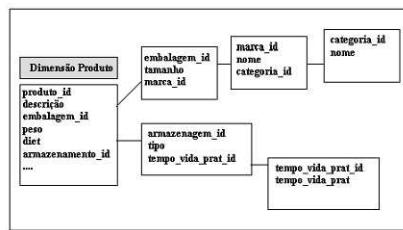


Figura 61.3: Modelo Snowflake

Na verdade, snowflake significa a normalização da tabela. Isto elimina redundância e diminui o espaço em disco. Mas, em um datawarehouse, redundância não é importante porque não é um ambiente transacional, operações de update não ocorrem com freqüência. Espaço físico também é irrelevante porque a tabela de fato é que ocupará a maior parte deste espaço.

Para uma boa performance do esquema estrela, é importante determinar o nível de consolidação, ou a granularidade, dos fatos. O fato pode estar no nível de transação, por exemplo, a venda individual de determinado produto, ou o fato pode ser armazenado com uma consolidação maior, como por exemplo, a venda de determinada linha de produtos em um dia. Armazenando o fato ao nível de transação faz com que o tamanho da tabela fato se torne excessivo e, além disso, este nível de detalhamento pode ser de pouca utilidade.

61.2 Datawarehousing

Um datawarehouse (ou armazém de dados, ou depósito de dados) é um sistema de computação utilizado para armazenar informações relativas às atividades de uma organização em bancos de dados, de forma consolidada. O desenho da base de dados favorece os relatórios, a análise de grandes volumes de dados e a obtenção de informações estratégicas que podem facilitar a tomada de decisão.

O datawarehouse possibilita a análise de grandes volumes de dados, coletados dos sistemas transacionais (OLTP). São as chamadas séries históricas que possibilitam uma melhor análise de eventos passados, oferecendo suporte às

tomadas de decisões presentes e a previsão de eventos futuros. Por definição, os dados em um datawarehouse não são voláteis, ou seja, eles não mudam, salvo quando é necessário fazer correções de dados previamente carregados. Os dados estão disponíveis somente para leitura e não podem ser alterados. A ferramenta mais popular para exploração de um datawarehouse é a Online Analytical Processing OLAP ou Processo Analítico em Tempo Real, mas muitas outras podem ser usadas.

Os datawarehouse surgiram como conceito acadêmico na década de 80. Com o amadurecimento dos sistemas de informação empresariais, as necessidades de análise dos dados cresceram paralelamente. Os sistemas OLTP não conseguiam cumprir a tarefa de análise com a simples geração de relatórios. Nesse contexto, a implementação do datawarehouse passou a se tornar realidade nas grandes corporações. O mercado de ferramentas de datawarehouse, que faz parte do mercado de Business Intelligence, cresceu então, e ferramentas melhores e mais sofisticadas foram desenvolvidas para apoiar a estrutura do data warehouse e sua utilização.

Atualmente, por sua capacidade de sumarizar e analisar grandes volumes de dados, o datawarehouse é o núcleo dos sistemas de informações gerenciais e apoio à decisão das principais soluções de business intelligence do mercado.

Um Datawarehouse é caracterizado como uma coleção de dados orientada a assunto, integrada, não-volátil e variante no tempo que auxiliam nas decisões de gerenciamento. Datawarehouses fornecem acesso aos dados para análises complexas, descoberta de conhecimento e tomada de decisões. Algumas características importantes dos datawarehouses são:

- Orientada ao assunto: pois o objetivo é tomar decisões sobre o "assunto-tema dos dados;
- Integrada: pois deve consolidar dados de diferentes origens ou fontes;
- Não volátil: pois as informações já presentes no banco são raramente modificadas (sobrescrita). Os novos dados são absorvidos pelo banco, integrando-se com as informações previamente armazenadas;
- Variante no tempo: pois deve-se manter um histórico dos dados, permitindo comparações ao longo do tempo.

Tipicamente o datawarehouse é mantido separadamente dos bancos de dados operacionais de uma organização. Existem muitas razões para fazer isto. Um datawarehouse suporta requerimentos funcionais e de performance para consultas OLAP (abordagem dimensional para o suporte à decisão), os quais são bastante diferentes de aplicações online transaction processing (OLTP) tradicionalmente suportadas por banco de dados operacionais. Aplicações OLTP tipicamente automatizam tarefas de processamento de dados do dia a dia tais como ordem de entrada de produtos e transações bancárias que são processadas durante todo o dia, dia após dia. Estas tarefas são estruturadas e repetitivas, e consistem de transações curtas, atômicas e isoladas.

Estas transações requerem detalhes, dados atualizados, leitura e atualização de poucos (dezenas de) registros que são acessados tipicamente por suas chaves primárias. Bancos de dados operacionais tendem a ter tamanho de centenas de megabytes a gigabytes. Consistência e capacidade de recuperação do banco de dados são críticas e a maximização da vazão de transações são métricas chaves de performance. Conseqüentemente, o banco de dados é projetado para refletir a Semântica operacional de aplicações conhecidas e, em particular, minimizar conflitos de concorrência.

Datawarehouse, em contraste, é dirigido a suporte a decisão. Dados históricos, sumariados e consolidados são mais importantes que dados detalhados em registros individuais. Já que o datawarehouse contém dados consolidados geralmente a partir de muitos banco de dados operacionais, sobre muitos períodos de tempo, eles tendem a serem maiores que os banco de dados operacionais. Datawarehouse de empresas são projetados para ter tamanho de centenas de gigabytes a terabytes. O trabalho pesado de um datawarehouse são as consultas intensivas na maioria das vezes ad hoc (consultas complexas que podem acessar milhões de registros e realizar muitas buscas por registros), uniões e agregações. Vazão de consulta e tempo de resposta são mais importantes que vazão de transações. Para facilitar análises e visualizações complexas, os dados em um datawarehouse são tipicamente modelados multidimensionalmente. Por exemplo, em um datawarehouse de vendas, tempo de vendas, lugar de vendas, vendedor e produto podem ser algumas dimensões de interesse. Freqüentemente, estas dimensões são hierárquicas, o tempo pode ser organizado na hierarquia dia-mês-trimestre-ano, produto como produto-categoria-indústria (marca) para facilitar e ampliar o domínio de consultas que podem ser feitas no Datawarehouse. O modelo estrela, floco de neves são dois exemplos de esquemas multidimensionais com os quais um datawarehouse pode ser modelado.

61.3 OLTP, OLAP, MOLAP, ROLAP e HOLAP

OLTP à É um acrônimo de Online Transaction Processing ou Processamento de transações em tempo-real. São sistemas que se encarregam de registrar todas as transações contidas em uma determinada operação organizacional, ou seja, é uma aplicação que tem como característica principal muitas atualizações em dados operacionais. Por exemplo: sistema de transações bancárias registra todas as operações efetuadas em um banco. Os ERPs (Enterprise Resource Planning) são sistemas que se enquadram nessa categoria.

OLAP à É um acrônimo de Online Analytical Processing ou Processamento analítico em tempo-real. É a categoria de tecnologia de software que capacita os analistas, gerentes e executivos a conseguir obter discernimento nos dados através de um acesso rápido, consistente e interativo, para uma larga variedade de possibilidades de visões da informação que vêm a ser transformada a partir de dados "crus" para refletir o real dimensionamento da corporação como entendido pelo usuário. A funcionalidade OLAP é caracterizada pela análise dinâmica multidimensional dos dados consolidados da corporação, dando suporte às atividades de análise e navegação do usuário final. A funcionalidade

OLAP é implementada em um modo cliente/servidor multi-usuário, e oferece consistentemente rápidas respostas para consultas, apesar do tamanho e complexidade do banco de dados. Ela ajuda o usuário a sintetizar as informações da corporação através de visões comparativas e personalizadas, assim como através de análises de históricos e projeções de dados em vários modelos de cenários do tipo "what-if". Alguns exemplos de consultas típicas de OLAP são: Quais os produtos mais bem vendidos no mês passado? Quais os 10 piores vendedores do departamento X? Qual a média salarial dos funcionários de informática na região sul nos últimos 5 anos? dentre outras.

Características	OLTP	OLAP
Operação Típica	Transação	Análise
Granularidade	Atômico	Agregado
Idade dos Dados	Presente	Histórico, Atual e Projetado
Recuperação	Poucos Registros	Muitos Registros
Orientação	Registro	Arrays
Modelagem	Processo	Assunto
Usuários	Muitos	Poucos

Tabela 61.1: OLTP vs. OLAP

Quanto à localização dos dados a serem utilizados na análise, atualmente existem duas abordagens:

- Um banco de dados multidimensional especializado;
- Um datawarehouse implementado com a tecnologia de banco de dados relacional, mas otimizado para a tarefa de análise.

Sistemas OLAP que implementam a primeira abordagem são chamados de MOLAP (Multidimensional OLAP) e aqueles que implementam a segunda são chamados ROLAP (Relational OLAP).

ROLAP à OLAP Relacional. Sistemas ROLAP fornecem análise multidimensional de dados armazenados em uma base de dados relacional. Atualmente existem duas maneiras de se fazer este trabalho:

- Fazer todo o processamento dos dados no servidor da base de dados; o servidor OLAP gera os comandos SQL em múltiplos passos e as tabelas temporárias necessárias para o devido processamento das consultas;
- Executar comandos SQL para recuperar os dados, mas fazer todo o processamento (incluindo joins e agregações) no servidor OLAP.

Além das características básicas de sistemas OLAP, servidores ROLAP devem também:

- Utilizar metadados para descrever o modelo dos dados e para auxiliar na construção das consultas. Desta maneira um analista pode executar suas análises utilizando seus próprios termos;

- Criar comandos SQL otimizados para os bancos de dados com o qual trabalha.

A principal vantagem de se adotar uma solução ROLAP reside na utilização de uma tecnologia estabelecida, de arquitetura aberta e padronizada como é a relacional, beneficiando-se da diversidade de plataformas, escalabilidade e paralelismo de hardware (SMP e MPP). Quanto às limitações, pode-se citar o pobre conjunto de funções para análise, a inadequação do esquema estrela para atualização dos dados e as soluções proprietárias para metadados que acaba por anular muitas das vantagens do uso da tecnologia relacional.

MOLAP à OLAP multidimensional. MOLAP é uma classe de sistemas que permite a execução de análises bastante sofisticadas usando como gerenciador de dados um banco de dados multidimensional. Em um banco de dados MOLAP, os dados são mantidos em arranjos e indexados de maneira a prover um ótimo desempenho no acesso a qualquer elemento. A indexação, a antecipação da maneira como os dados serão acessados e o alto grau de agregação dos dados fazem com que sistemas MOLAP tenham um excelente desempenho. Além de serem rápidos, outra grande vantagem desses sistemas é o rico e complexo conjunto de funções de análise que oferecem.

HOLAP à O armazenamento HOLAP (OLAP Híbrido) é uma combinação dos tipos de armazenamento MOLAP e ROLAP. Os dados de agregação são armazenados em MOLAP, enquanto os dados de base são deixados no banco de dados relacional.

61.4 Outros conceitos importantes

- Data Mart: é um subconjunto de um data warehouse. É um repositório de dados extraídos das fontes operacionais ou de um data warehouse que é projetado para servir as necessidades particulares de um grupo específico. Na prática, o data warehouse e data mart pode trabalhar juntos. O data warehouse atende as necessidades estratégicas da organização, enquanto o data mart atende as necessidades gerenciais mais a nível operacional. Quando os data mart atualiza seus dados através do data warehouse, ele é considerado de dependente. Quando ele atualiza os dados através das fontes operacionais, ele é considerado independente;
- Agregações: Grande parte dos usuários não está interessada em realizar uma consulta que retorne uma grande quantidade de linhas de uma tabela de fatos, mas sim em sumarizar os resultados usando operações de soma ou média por exemplo. Para evitar que as mesmas operações sejam realizadas a cada vez que um usuário execute uma consulta, é importante realizar um levantamento das agregações e sumários nos quais os usuários estão interessados;
- Funcionamento (Fluxo de dados). Os dados para serem usados para fins analíticos devem ser transformados e carregados dos sistemas OLTP para

o Data Warehouse. Durante essas transformações (realizadas pelas ferramentas ETL – Extrair, Transformar e Carregar) são criados resumos e agregações entre esses dados, transformando-os em informações de mais alto nível e mais signifitivas para os analistas, gerentes e executivos;

- Metadados: são os chamados dados sobre os dados. É um mapeamento dos dados do modo como foram extraídos das fontes operacionais e como estão inseridos no data warehouse. Os metadados definem e descrevem os dados de negócio (valor aos dados) e os tipos de dados (definições de tabelas, atributos, domínio e relações). Os metadados são geralmente armazenados em repositórios separados ao alcance dos usuários;
- Granularidade: é o nível de detalhe contido nas unidades de dados existentes no data warehouse. A granularidade é importante porque pode diminuir o tempo de acesso aos dados que realmente interessam e diminuem a quantidade de discos rápidos e caros necessários para armazenar uma grande quantidade de dados históricos.

Parte XII

Administração de Bancos de Dados Relacionais

Capítulo 62

Gerência de Transações

Uma *transação* é a unidade lógica de processamento, e pode incluir uma ou mais operações sobre o Banco de Dados. Para garantir a correção, uma transação deve ser executada até o final. As duas operações básicas fornecidas por um banco de dados são READ e WRITE. As fronteiras de uma transação podem ser definidas explicitamente na aplicação através das chamadas *BEGIN TRANSACTION* e *END TRANSACTION*.

Entende-se por *transações concorrentes* aquelas que são executadas ao mesmo tempo e pelo menos uma operação de WRITE é executada por alguma das transações. A possibilidade de existir transações sendo executadas de forma concorrente faz com que o SGBD utilize técnicas para realizar o controle de concorrência. Uma vez iniciado o processamento da transação, o SGBD deve garantir que ela será executada até o final e os resultados serão armazenados de forma permanente. No entanto, durante a execução de uma transação podem ocorrer falhas como crash do sistema, falhas no disco, catástrofes. Isso faz com que o SGBD tenha que utilizar técnicas de Recuperação em caso de falhas.

Para efeitos e recuperação são utilizadas as seguintes operações sobre as transações:

- BEGIN TRANSACTION - Marca o início da transação;
- READ/WRITE - Executa operações como parte da transação;
- END TRANSACTION - Especifica o fim das operações de leitura e escrita;
- COMMIT TRANSACTION - Confirma o sucesso da transação;
- ROLLBACK - Indica que ocorreram problemas com a transação.

Alterações devem ser desfeitas. As transações podem ainda se encontrar em cinco estados que são: (1) *Ativa*; (2) *Parcialmente Confirmada*; (3) *Confirmada*; (4) *Falha*; (5) *Terminada*.

As propriedades desejáveis de uma transação devem ser:

- *Atomicidade*: Garantir que serão executadas até o fim;
- *Consistência* - Devem levar o Banco de Dados de um estado consistente para outro também consistente;
- *Isolamento* - A execução de uma transação não deve ser prejudicada por transações concorrentes;
- *Durabilidade* - As alterações realizadas por uma transação confirmada devem ser persistidas no Banco de Dados.

Estas quatro propriedades são usualmente referenciadas como *ACID*.

Quando várias transações são executadas simultaneamente, as instruções de cada transação podem ser intercaladas. A atribuição da ordem dessas instruções é conhecido como *escala de execução*. É responsabilidade do SGDB escolher uma escala que deixe o banco de dados em estado inconsistente. As únicas operações significativas de uma transação, do ponto de vista da escala de execução, são suas instruções de leitura e escrita.

Considere uma escala de execução com duas instruções sucessivas I_i e I_j , das transações T_i e T_j , respectivamente. Se I_i e I_j referem-se a itens de dados diferentes, então podemos alternar I_i e I_j sem afetar os resultados de qualquer instrução da escala. Já quando se referem aos mesmos dados temos os seguintes casos:

1. $I_i = \text{read}(Q)$, $I_j = \text{read}(Q)$: a seqüência não importa;
2. $I_i = \text{read}(Q)$, $I_j = \text{write}(Q)$: a ordem importa, pois se I_i vier depois de I_j , I_i irá ler o valor escrito por I_j ;
3. $I_i = \text{write}(Q)$, $I_j = \text{read}(Q)$: a ordem importa, por razões semelhantes a anterior;
4. $I_i = \text{write}(Q)$, $I_j = \text{write}(Q)$: a ordem importa, não afeta as transações T_i e T_j , entretanto, o valor obtido pela próxima instrução de leitura sobre Q será afetado.

Dizemos que I_i e I_j entram em conflito caso a ordem importe. Se uma escala de execução puder ser transformada em outra por uma série de trocas de instruções não-conflitantes, dizemos que essas escalas de execução são *equivalentes em conflito*. Dizemos que uma escala é *conflito serializável* se ela é equivalente em conflito à uma escala seqüencial (uma escala que contém as instruções de cada transação agrupadas). É possível ter duas escalas de execução que produzam o mesmo resultado, mas que não sejam equivalentes em conflito.

Capítulo 63

Controle de Concorrência

Entende-se por transações concorrentes aquelas que desejam executar ao mesmo tempo e pelo menos uma operação de WRITE precisa ser realizada. O SGBD é responsável por garantir o acesso mutuamente exclusivo a um item de dados qualquer. Em outras palavras, enquanto uma transação acessa um item de dado, nenhuma outra poderá modificá-lo. A forma mais simples de se implementar o controle de concorrência é a utilização de mecanismos de bloqueio baseados em *locks binários*, através das operações básicas *lock* e *unlock*, que devem ser aplicadas antes e depois da realização de uma operação qualquer (READ ou WRITE) sobre qualquer item de dados.

Os locks binários tem o inconveniente de não permitir que duas ou mais transações que desejem realizar operações apenas de leitura acessem o item simultaneamente. Isso diminui o desempenho do sistema, uma vez que muitas transações tem que aguardar na fila de espera por acesso a um item de dados. Para contornar esse problema, um outro mecanismo de lock baseado em 3 operações foi criado. Quando um transação deseja apenas ler um item, ela realiza uma operação *read_lock*, permitindo que outras transações que desejem apenas ler o mesmo item também possam executar simultaneamente. Se uma operação deseja escrever, deve primeiro executar uma operação do tipo *write_lock*. Nenhuma operação de leitura ou escrita pode ser realizada por outra transação sobre um item que esteja bloqueado para escrita. A operação *unlock* deve ser aplicada após o término das operações de leitura ou escrita por parte das transações que detêm o lock de um item.

É comum que uma transação deseje mudar o tipo de lock que possui sobre um determinado item. Esta operação é chamada de *conversão de lock*. Se apenas uma transação possuir o *read_lock*, ela pode atualizá-lo para *write_lock*. Uma transação que possua um *write_lock* pode convertê-lo para *read_lock* após terminar suas operações de escrita. Essas operações são também chamadas *upgrade* e *downgrade*.

Os locks binários e os locks read/write embora garantam a exclusividade mútua, não podem garantir a *serialização* na execução de transações concorrentes. Um outro protocolo que visa solucionar os dois problemas é o chamado *Two-Phase Locking (2PL)*. As fases são: (i) *Crescimento (Growing)* e (ii) *En-*

colhimento (Shrinking). Todos os *read_lock*, *write_lock* e *upgrdrades* ocorrem na primeira fase, enquanto *unlock* e *downgrade* ocorrem na segunda.

Um problema dos protocolos de bloqueio é a possibilidade de ocorrência de *deadlocks*. Um protocolo de prevenção de deadlocks é baseado em *timestamps*, que são identificadores únicos associados a cada transação. Outras duas técnicas para identificação e prevenção de deadlocks são *timeouts* e grafos (*Wait-For Graphs*). Protocolos baseados em *timestamps* também podem ser utilizados para controle de concorrência. Os valores do *timestamp* para um dado item X são: *readTS(X)* e *writeTS(X)*. Eles indicam a última leitura e escrita realizadas sobre o item X respectivamente.

Uma outra técnica para controle de Concorrência basea-se em guardar valores antigos de um determinado item. Procolos que se utilizam dessa técnica são chamados *Multiversão*. Operações de leitura que antes em outros protocolos não seriam permitidas, podem ser realizadas fornecendo-se a versão antiga de um item de dado. O principal encoveniente desta técnica é a necessidade de mais espaço em disco.

A granularidade de um item de dado também exerce grande influência no desempenho de operações de controle de concorrência. Por exemplo, uma transação que deseja realizar operações sobre um resgistro, pode tornar ineficiente o controle de transações se bloquear inadequadamente um bloco físico inteiro. Procolos que determinam tipo de bloqueio de acordo com a granularidade são chamados *Multiple Granularity Locking (MGL)*.

Capítulo 64

Gerência de Desempenho

O Design físico do Banco de Dados é o processo de escolha das estruturas de armazenamento e métodos de acesso de forma que um bom desempenho possa ser atingido pelas diversas aplicações. Cada SGBD oferece uma diversidade de opções de organização de arquivos e métodos de acessos especiais como índices, *clustering* de registros relacionados, encadeamento de registros relacionados usando *ponteiros*, e funções *hash*. Os critérios utilizados que guiam a escolha das opções de design físico são o *Tempo de Rsposta*, a *Utilização do Espaço* e o *Throughput de Transações*. O desempenho das estruturas e métodos de acesso depende fundamentalmente do tamanho dos arquivos, portanto durante a fase de design devem ser realizadas estimativas de crescimento dos arquivos. O crescimento pode se dar tanto no tamanho dos registros assim como no número de registros por arquivo.

A maioria dos SGBDs possuem ferramentas de monitoração que coletam estatísticas de desempenho, que são mantidas no Catálogo do Sistema (*System Catalog*). As informações dizem respeito, por exemplo, à número de chamadas de transações ou *queries*, atividade de I/O por arquivo, número de páginas por arquivo, número de registros por índice e frequência de utilização dos índices. Assim que os requisitos do sistema mudam, pode se tornar necessário adicionar ou remover tabelas, reorganizar arquivos (mudar método de acesso primário), dropar índices antigos e criar novos.

Exemplos de atividades importantes no design físico de um Banco de Dados: (i)Análise de Transações de de Queries - Envolve determinar arquivos envolvidos, atributos mais acessados, atributos utilizados em operações de JOIN, e determinação de operações predominantes (SELECT, UPDATE, INSERT, DELETE); (ii)Estimativa da Frequência das Chamadas de Transações e *Queries* - Geralmente para um grande volume de processamento, é válida a regra informal que diz que 80% do processamento é gasto com 20% das transações e *queries*; (iii)Restrições de Tempo - Identificar das restrições de tempo de transações e *queries*; (iv)Restrições de Unicidade de Atributos - Devem ser estabelecidos métodos de acesso para todas as chaves candidatas ou conjunto de chaves candidatas (ex: PK ou UNIQUE), tendo por objetivo otimizar os testes de unicidade.

Os índices primários e de *clustering* estão diretamente relacionados com a

ordenação física dos registros no disco. Portanto, pode ser definido no máximo um índice deste tipo. Embora o desempenho das operações de consulta seja otimizado com a utilização de índices, é importante ressaltar que a existência de índices representa um *overhead* significativo nas operações de inserção, atualização e exclusão. Para consultas baseadas em múltiplos campos, podem ser definidos *índices múltiplos*, ou seja, baseado em 2 ou mais campos. Índices do tipo clustering são especialmente úteis em consultas envolvendo intervalo (*range queries*). Os índices baseados em *B+Trees* são úteis para pesquisas em intervalos e testes de igualdade, enquanto os índices baseados em funções *hash* são extremamente eficientes para testes de igualdade, podendo otimizar bastante operações de JOIN.

Uma outra estratégia utilizada para otimização em Bancos de Dados é a *Desnormalização*. Essa técnica consiste em armazenar o banco de dados em uma forma normal mais fraca como a 1NF ou 2NF, introduzindo algumas dependências funcionais que não existiriam nas formas BCNF ou 4NF. Essa técnica se baseia em eliminar a necessidade de operações de JOIN, de forma que consultas possam ser respondidas acessando o menor número de arquivos possíveis. Outras duas técnicas que podem ser utilizadas são o *Particionamento Vertical*, que consiste na divisão de um tabela e múltiplas tabelas de colunas iguais, porém com um número menor de linhas por tabela, e o *Particionamento Horizontal*, que consiste em transformar uma relação $R(K, A, B, C, D, \dots)$ em múltiplas relações $R_1(K, A, B), R_2(K, C, D), R_3(K, \dots)$. As técnicas de particionamento são interessantes pois não sacrificam a forma normal da relação original.

Alguns cuidados adicionais podem ser tomados para garantir o bom desempenho do sistema. A utilização da operação DISTINCT causa uma operação de ordenação, e portanto é muito custosa. Cláusulas DISTINCT redundantes devem ser eliminadas. Deve-se ter cuidado com a utilização de tabelas temporárias, porém em casos especiais elas podem ser úteis.

```
SELECT SSN FROM EMPLOYEE E  
WHERE SALARY = SELECT MAX (SALARY) FROM EMPLOYEE AS M  
WHERE M.DNO = E.DNO
```

No caso acima, a utilização de uma tabela temporária pode impedir que o cálculo do salário máximo por departamento seja realizado para todos os empregados da tabela E.

```
SELECT MAX (SALARY) AS HIGHSALARY, DNO INTO TEMP  
FROM EMPLOYEE GROUP BY DNO
```

```
SELECT SSN FROM EMPLOYEE E, TEMP T  
WHERE E.SALARY = T.HIGHSALARY AND E.DNO = T.DNO
```

Alguns otimizadores de consulta não são capazes de determinar em qual ordem as tabelas que aparecem na cláusula FROM devem ser varridas, e utilizam na execução a ordem em que elas aparecem na cláusula. Portanto é importante que se organize as tabelas de forma que as tabelas menores sejam varridas enquanto as tabelas maiores sejam indexadas de forma adequada. Condições OR na cláusula where podem inibir a utilização de índices, portanto a estratégia

ser tomada é separar a *querie* em duas, forçando a utilização do índice, e realizar uma operação UNION na sequência. As operações IN, =ALL, =ANY, =SOME devem ser substituídas por operações do tipo JOIN sempre que possível.

Processamento de consultas são atividades envolvidas em extrair dados de um banco de dados. O custo do processamento de uma consulta é principalmente determinado pelo acesso ao disco. Os passos envolvidos no processamento de consultas são:

1. Análise sintática e tradução;
2. Otimização;
3. Avaliação.

O processo de tradução é semelhante à tarefa de um analisador sintático em um compilador. A otimização, geralmente, fica a cargo do SGDB, no caso do modelo relacional. Já para os modelos de rede e hierárquico, a otimização fica a cargo do programador. Um *plano de execução de consulta* é um conjunto de operações primitivas que são usadas para avaliar uma consulta. Os diferentes planos de execução podem possuir diferentes custos. A *otimização de consultas* é o processo de selecionar o plano de execução mais eficiente para uma consulta. Um aspectos de otimização envolve a procura de expressão mais eficiente no nível de álgebra relacional, mas diversas outras estratégias são importantes como a escolha de um algoritmo para ser usado na execução de uma operação e a escolha de índices a usar.

Para escolher diferentes planos de execução, o otimizador deve estimar o custo de cada plano de avaliação. Os otimizadores de consulta usam informações estatísticas armazenadas em um catálogo e incluem:

- Número de tuplas de cada relação;
- Número de blocos que contêm as tuplas de cada relação;
- Número de bytes de uma tupla para uma certa relação;
- Número de valores distintos que aparecem numa relação para um determinado atributo;
- Número médio de registros que satisfazem uma condição de igualdade para um determinado atributo.

Podemos processar consultas que envolvem seleções simples por meio da execução de uma varredura linear, de uma procura binária ou do uso de índices. Podemos tratar as seleções complexas computando uniões e intersecções dos resultados de seleções simples. Podemos ordenar relações maiores que a memória usando o algoritmo do merge-sort externo. Uma junção pode ser calculada por diversos algoritmos e o sistema deve ser sagaz para escolher a melhor alternativa para cada caso.

Parte XIII

Oracle e Microsoft SQL Server

Capítulo 65

Administração de Bancos de Dados Oracle

65.1 Arquitetura de um Servidor Oracle

65.1.1 Estruturas em memória

Para operar, uma instância Oracle aloca uma série de áreas de cache em memória para armazenar três estruturas que são o SGA (*System Global Area*), o PGA (*Program Global Area*) e as *Sort Areas*. O SGA é uma área única de memória compartilhada composta por três componentes principais que são:

- Database Buffer Cache (DBBC): o DBBC é uma área de memória que armazena os blocos de dados lidos do disco. Quando um processo necessita de acessar dados do banco ele primeiro consulta essa área de memória. Todas as operações sobre os dados do banco são realizadas nessa área antes de serem persistidas em disco. No Oracle, a política de substituição de blocos utilizada é a LRU (*Last Recently Used*);
- Shared Pool: o shared pool é uma área compartilhada que armazena informações sobre as instruções SQL enviadas pelos processos usuário (texto do comando, planos de execução etc.) e informações sobre o dicionário de dados (nome das tabelas, colunas, tipos, privilégios etc.);
- Redo Log Buffer: buffer circular que contém informações sobre as modificações efetuadas no banco de dados. O redo log buffer é uma das estruturas utilizadas para realizar a recuperação do banco de dados. É importante ressaltar que o redo log buffer é utilizado apenas para recuperação e não para operações de rollback;

Já o PGA, em um sistema single-threaded, é uma área de memória não compartilhada utilizada para guardar informações de controle para um único processo. O PGA armazena informações como arrays, estado de cursores e informações sobre sessões de usuários. Quando trata-se de um sistema multi-threaded (MTS), as informações sobre sessões de usuários são guardadas em uma outra área chamada UGA (User Global Area), que é alocada no shared pool.

A *Sort Area* é uma área de memória compartilhada alocada especialmente para realização de operações que exigem ordenação, como SORT BY, GROUP BY e DISTINCT. Quando o tamanho da *Sort Area* não é suficiente, a operação de ordenação é realizada em disco, em uma tablespace chamada TEMP.

Duas outras estruturas de memória também têm muita importância na arquitetura Oracle. A primeira delas é o Java Pool, que define a quantidade de memória que pode ser alocada pela JVM (*Java Virtual Machine*). A segunda dela é o Large Pool, que é utilizada para operações de backup, recuperação, entre outras. Em alguns casos, pode ser utilizada para operações de ordenação.

65.1.2 Processos server

Os processos server são recebem as requisições dos processos user, realizam o parse das instruções SQL, verificam as permissões de acesso do usuário, traz os dados do disco para o DBBC, caso necessário, e retorna os dados para o usuário. Um processo server pode ser dedicado para um processo user (*dedicated server process*) ou compartilhado entre múltiplos processos user (*shared server process*). Os processos server compartilhados só são possíveis em sistemas multi-threaded.

65.1.3 Processos user

As funções desempenhadas pelos processos user são se conectar com os processos server, enviar instruções SQL e receber os resultados. Caso o servidor suporte processos server compartilhados, diversos processos server podem ser atendidos por um mesmo processo server.

65.1.4 Processos em Background

Ao contrário dos processos server, os processos em background não realizam nenhuma comunicação com os processos user. Os processos em background são responsáveis pelas tarefas de apoio para garantir o funcionamento do sistema de gerenciamento de banco de dados como um todo. Um sistema Oracle tem inúmeros processos em background, porém apenas 4 deles são obrigatórios que são:

- Process Monitor (PMON): Realiza a recuperação quando algum processo falha, além de liberar o cache, locks e demais recursos que o processo estava utilizando;
- System Monitor (SMON): Realiza o processo de recuperação da instância durante o processo de inicialização, limpa segmentos temporários que não estão mais em uso, recupera transações terminadas de forma anormal e realiza desfragmentação nos arquivos de dados para facilitar a alocação;
- Database Writer (DBWR): A função principal do DBWR é escrever os blocos de dados modificados (*dirty*) do DBBC para o disco. Isso é feito nas seguintes situações: (i) quando a lista de blocos modificados atinge um tamanho configurado; (ii) quando o processo percorre um quantidade configurada de blocos e não encontra nenhum bloco livre; (iii) quando ocorre

um timeout; (iv) quando ocorre um checkpoint ou (v) quando o DBWR recebe um sinal do processo LGWR. Dessa forma, o processo DBWR além de garantir que as alterações feitas em buffer serão persistidas, também gerencia o espaço em buffer para melhorar o desempenho do banco;

- Log Writer (LGWR): É o responsável por copiar as informações do redo log buffer para o redo log file (arquivo de log com a mesma estrutura do redo log buffer). O LGWR também é responsável por atualizar os headers dos arquivos de dados quando ocorre um checkpoint. O LGWR é disparado nas seguintes situações: (i) quando ocorre um commit; (ii) quando ocorre um checkpoint; (iii) quando ocorre um timeout ou (iv) quando o redo log buffer atinge um terço de sua capacidade.

Além desses quatro, o Oracle possui uma série de outros processos em background que são de instalação e uso opcionais. Os mais importantes são os seguintes:

- CKPT: Atualiza os headers dos arquivos de dados quando ocorre um checkpoint. A utilização desse processo pode ajudar a melhorar o desempenho do sistema permitindo que o processo LGWR se concentre apenas na cópia do redo log buffer para o disco;
- RECO: Responsável pela recuperação de falhas envolvendo transações distribuídas. Esse processo é necessário quando o Oracle está rodando de forma distribuída;
- ARCH: É o responsável por copiar o redo log file (que é um buffer circular) para um dispositivo de armazenamento offline para que os logs não sejam perdidos;
- Pnnn: É o processo responsável pela execução de consultas paralelas;
- SNPn: Controla a replicação de objetos dos banco de dados em outro site. Essas cópias são chamadas *snapshots*. Esse processo pode ser escalonado para executar periodicamente;
- LCKn: Realiza o controle de locks entre múltiplas instâncias;
- Dnnn: Esse processo funciona como um dispatcher quando o sistema está utilizando processos server compartilhados. É necessário um dispatcher para cada protocolo de comunicação utilizado.

65.1.5 Arquivos

Em conjunto, O SGA e os processos em background formam uma intância Oracle. O banco de dados propriamente dito é formado por três tipos de arquivos que são:

- Datafiles: Armazenam as tabelas e índices do banco;
- Redo Log Files: Armazenam os logs do sistema. Esse arquivo consiste de um buffer circular assim como o redo log buffer. O processo LGWR é o responsável por copiar os logs em memória para este arquivo;

- Control Files: Este é um arquivo binário que descreve a estrutura e o status do banco de dados. Contém a identificação dos arquivos de log e de dados, o nome do banco e informações de sincronismo entre os arquivos que o compõe.

Embora na arquitetura do Oracle o banco de dados seja composto somente por esses três arquivos, uma série de outros arquivos são importantes para colocar uma intância no ar. Exemplos desses arquivos são:

- Parameter File: Arquivo texto que contém todos os parâmetros necessários para colocar uma instância do Oracle no ar, por exemplo, quantidade de memória alocada para o SGA, nome e localização de arquivos de controle, tamanho de buffers e de blocos etc;
- Alert File: Arquivo de log onde são registrados os erros ocorridos (processo, blocos corrompidos, deadlock etc.), tarefas administrativas além dos valores de parâmetros de incialização no momento que a instância é posta no ar;
- Trace File: Arquivo de log onde são armazenadas informações delatalhadas sobre os problemas ocorridos. A utilização desse arquivo é opcional;

65.2 Arquitetura Oracle de Armazenamento de Dados

Um banco de dados é composto por uma ou mais *tablespaces*. Uma *tablespace* contém um ou mais arquivos no nível de sistema operacional. Cada *tablespace* pode ter um ou mais segmentos. Um segmento é adicionado na *tablespace* quando uma tabela ou um índice é criado, ou seja, um segmento é associado a cada objeto de banco de dados. É possível que um objeto seja atribuído a mais de um segmento, mas, para isso, o usuário deverá particionar explicitamente o objeto. Várias tabelas ou índices podem ser incluídos em um mesmo segmento através da criação de um objeto conhecido como *cluster*.

A medida que um objeto de banco de dados necessita de mais espaço, é necessário que o segmento aloque mais dados. O banco de dados procura, nos arquivos do *tablespace*, áreas contíguas de tamanho pré-determinado para o armazenamento de novos dados. Essa área contígua é conhecida como extensão (*extent*), que por sua vez é formada por diversos blocos de banco de dados. Cada bloco de banco de dados possui um tamanho fixo determinado nos parâmetros de configuração, esse tamanho fixo deve ser múltiplo do tamanho de um bloco do nível do sistema operacional. O tamanho *extent* também pode ser configurado pelo usuário ou determinado automaticamente pelo Oracle.

65.3 Tratamento de Transações no Oracle

65.3.1 Gerenciamento do *Redo Log*

O *Redo Log* é a estrutura mais crucial para as operações de recuperação. Cada instância possui um *Redo Log* para protegê-la em caso de falha. Como explicado anteriormente, o processo LGWR é responsável por copiar para o disco o

conteúdo do *redo log buffer* em determinadas situações, como em um COMMIT por exemplo. Essa escrita é feita de maneira circular. Quando não houver mais espaço para escrever no arquivo de *Redo Log* atual, o LGWR começa a escrever no próximo arquivo. Caso esteja no último arquivo do buffer, o LGWR voltará a escrever no primeiro arquivo. É possível configurar o banco de dados para arquivar as informações escritas nos arquivos de redo. Com essa configuração, o processo ARCH será ativado.

Um *log switch* é o ponto quando o banco de dados para de escrever em um arquivo de *Redo Log* e começa a escrever em outro. Um *log switch* também pode ser forçado pelo DBA e pode ser configurado para ocorrer em intervalos regulares. O Oracle atribui, para cada arquivo de *Redo Log*, um *log sequence number* toda vez que um *log switch* ocorre. A ordem atribuída pelos *log sequence numbers* é utilizada na recuperação do banco de dados em caso de falha.

65.3.2 Checkpoints

A um tempo específico, todos os dados do *database buffer cache* modificados são escritos em disco pelo processo DBWR; este evento é chamado de *checkpoint*. O processo CKPT é responsável para informar ao processo DBWR o momento de gravar os dados em disco. O DBWR também atualiza os arquivos de controle do banco de dados para indicar o mais recente *checkpoint*. O processo CKPT é opcional; se ele não estiver presente, o LGWR assume sua responsabilidade. No momento do checkpoint a falta de sincronismo entre o DBBC e os arquivos de *redo* é eliminada. Os eventos em que um *checkpoint* ocorre são:

- A cada *log switch*.
- Após ocorrido um certo tempo desde o último *checkpoint* (LOG_CHECKPOINT_TIMEOUT).
- Quando a instância sofre um *shutdown*, a menos que ela seja abortada.
- Quando forçada pelo DBA. (ALTER SYSTEM CHECKPOINT)
- Quando uma *tablespace* é colocada *offline* com pelo menos um arquivo (*datafile*) online.

65.3.3 Segmentos de rollback

Um segmento de rollback é uma porção de um banco de dados que registra as ações das transações dos usuários nos dados para que possam ser desfeitas sob certas circunstâncias.

Um segmento de rollback é usado para permitir a consistência da leitura, recuperar um comando quando ocorre o dead-lock, recuperar uma transação até uma certa marca identificada por um SAVEPOINT (marcador para instruções não confirmadas), recuperar uma transação terminada por uma falha de processo de um usuário e desfazer todas as transações pendentes durante a recuperação de uma instância.

Todas as operações de atualização de dados em um banco de dados envolvem os segmentos de *rollback* para permitir a consistência da leitura, a recuperação das informações e permitir que uma transação ou um comando sejam desconsiderados ou desfeitos.

65.3.4 Consistência de leitura

O *Database Buffer Cache* é organizado em duas listas: a *dirty list* e a *least recently used list* (LRU). A *dirty list* é uma lista que contém os blocos alterados que ainda não foram escritos em disco. A LRU é uma lista que contém blocos do ORACLE que foram alterados pelos comandos dos usuários mas ainda não foram gravados em disco. Contém ainda blocos livres e blocos em uso. Assim, quando um processo servidor precisa ler um bloco de dados do disco para a memória, ele:

1. Pesquisa nas listas LRU e dirty list pelo bloco de dados desejado.
2. Caso esse bloco de dados não seja localizado, o processo servidor pesquisa a lista LRU em busca de um bloco livre.
3. Em seguida, o processo servidor move os blocos alterados encontrados na lista LRU para a dirty list, ou seja, movimenta-os para a lista de blocos alterados ainda não gravados nos arquivos de dados, de acordo com a localização de cada um deles, durante o processo de pesquisa de um bloco livre.
4. Finalmente, o processo servidor efetua uma cópia do bloco de dados do disco para um bloco livre.
5. Esse procedimento termina quando o processo servidor localiza um bloco livre ou se um número específico de blocos forem pesquisados sem encontrar um único bloco livre.

Se nenhum bloco foi encontrado, o ORACLE deve gravar os blocos alterados da dirty list para os arquivos em disco, para liberar espaço em memória para os novos blocos de dados que precisam ser manipulados pelos comandos dos usuários.

65.4 Configuração do Servidor

A criação de um banco de dados pode ser realizada através de uma ferramenta gráfica conhecida como *Database Configuration Assistant*. Com essa ferramenta também é possível configurar opções de um banco de dados existente, deletar um banco de dados e gerenciar gabaritos de banco de dados. As etapas são as seguintes:

Etapa 1 Em uma tela, será possível escolher se é desejado criar um banco de dados, configurar ou deletar um já existente ou gerenciar gabaritos. Nesse caso, será criado um banco de dados.

Etapa 2 Será possível escolher sobre qual gabarito se deseja basear o banco de dados a ser criado. Há como escolher gabaritos otimizados para *data warehouse*, processamento de transações ou para uso geral.

Etapa 3 Será atribuído o Nome do Banco de Dados Global, pelo qual o banco de dados é identificado exclusivamente. Além disso, será necessário identificar o nome de uma instância para o banco de dados que está sendo criado.

Etapa 4 Será necessário selecionar se deseja que o banco esteja em Modo Servidor Dedicado ou em Modo de Servidor Compartilhado. No primeiro caso, cada conexão cliente é atendida um processo servidor para atender somente a esse cliente. No segundo caso, será possível para um processo servidor atender mais de uma conexão cliente.

Etapa 5 Nessa etapa, os parâmetros de inicialização relacionados à memória, à localização dos arquivos de configuração e ao conjunto de caracteres serão configurados.

Etapa 6 Será configurado o armazenamento do banco de dados, através de uma tela será possível configurar arquivos de controle, *tablespaces*, arquivos de dados, segmentos de *rollback* e grupos de *redo logs*.

Etapa 7 Na última etapa, haverá a opção de criar o banco de dados ou, simplesmente, utilizá-lo como gabarito.

65.5 Tipos de Usuários Oracle

Os tipos de usuários (papéis) variam de acordo com o lugar, mas podem ser divididos em administradores de banco de dados, responsáveis pela segurança, administradores de rede, desenvolvedores de aplicação, administradores de aplicação e usuários do banco de dados.

65.5.1 Administradores de banco de dados

Cada banco de dados requer no mínimo um DBA (*database administrator*). As suas responsabilidades incluem:

- Instalação e atualização dos servidores de banco de dados Oracle e das ferramentas de aplicação.
- Alocação de espaço para o sistema e planejamento de futuras necessidades.
- Criação de estruturas de armazenamento primárias do banco de dados (*tablespaces*).
- Criação e alteração de objetos primários: tabelas, visões etc. após os desenvolvedores terem projetado a aplicação.
- Administração de usuários, controle de acesso e segurança do sistema.
- Garantia do atendimento aos termos da licença Oracle e contato direto com o suporte técnico da Oracle.
- Monitoração e otimização do desempenho do banco de dados.
- Planejamento e execução de backup e de restauração das informações do banco de dados.

65.5.2 Outros papéis

Responsável pela segurança É um papel que existe somente em alguns casos. Pode assumir as responsabilidades de administração de usuários, controle de acesso e segurança do sistema.

Administradores de redes Responsáveis em administrar os produtos de rede da Oracle como os serviços de rede da Oracle.

Desenvolvedores de aplicação Entre as atividades desse papel, pode-se destacar: o projeto e desenvolvimento da aplicação e o projeto da estrutura do banco de dados bem como os requisitos necessários em relação ao espaço de armazenamento necessário.

Administradores de aplicação Cada aplicação deve ter um administrador de aplicação, que será responsável por ela.

Usuários do banco de dados Responsáveis pela criação, modificação e deleção de dados quando permitidos, além da obtenção de relatórios através dos dados.

Capítulo 66

Administração de Bancos de Dados SQL Server

66.1 Arquitetura de um Servidor SQL Server

Um banco de dados no SQL Server 2005 se refere a um agrupamento físico de um conjunto de objetos de esquema de um banco de dados em um ou mais arquivos físicos. Os bancos de dados podem ser classificados como bancos de dados de sistemas e bancos de dados dos usuários. Os banco de dados de sistema armazenam dados do sistema e também controlam o armazenamento temporário requerido para os dados de aplicação.

Cada instância do SQL Server pode suportar vários bancos de dados. Além disso, pode haver várias instâncias em um mesmo computador. Cada banco de dados pode suportar grupos de arquivos (*filegroups*), que provedem a funcionalidade de distribuir os dados fisicamente. Um banco de dados pode possuir vários *filegroups*, mas o contrário não é possível. Após a criação do banco de dados, os *filegroups* do banco de dados podem ser adicionados.

Por padrão, o SQL Server instala os seguintes bancos de dados:

- Modelo – um template para os outros banco de dados.
- Tempdb – para armazenamento temporário, como as operações de ordenação.
- Msdb – suporta o agente do SQL Server, com tarefas agendadas, alertas e informações sobre replicação.
- Adventure Works e AdventureWorksDW – opcionais, são banco de dados de exemplos.

66.1.1 Catálogos de sistema

Cada banco de dados Oracle executa em catálogo de sistema centralizado, ou dicionário de dados, que reside na *tablespace* SYSTEM. No caso do SQL Server 2005, cada banco de dados mantém seu catálogo de sistema, que possui informações relacionadas, por exemplo, à: objetos do banco de dados (tabelas,

índices, procedimentos, visões, *triggers* etc.), restrições, usuários, permissões e Tipos de dados definidos pelo usuário.

As informações relativas ao nível do sistema são armazenadas no *Master Database* (Msdb). Essas informações incluem, por exemplo: contas de login, valores de configuração, mensagens do sistema, nomes dos bancos de dados e a localização dos arquivos primários de cada banco de dados.

Atenção, no SQL Server 2005, os objetos do sistema não são armazenados no Msdb, mas em um banco de dados oculto conhecido como *resource database*.

66.1.2 Processos em *background*

O SQL Server possui diversos processos em *background*. Um exemplo são os *checkpoints*. O SQL Server periodicamente gera *checkpoints* automaticamente em cada banco de dados.

Outro processo em *background* importante é o *Lazywriter* que é único para cada instância. O *lazywriter* adormece um intervalo de tempo e então acorda para checar o espaço livre na lista do *buffer*. Caso esse espaço livre esteja abaixo de um certo ponto (dependente do tamanho do cache), o processo *lazywriter* procura páginas não utilizadas no *buffer cache* e escreve as páginas com *write dirty* que não foram recentemente referenciadas em disco.

Um processo conhecido como *ResourceMonitor* é utilizado para controlar o restante do caches, pois o *lazywriter* só controla as páginas de dados.

66.2 Arquitetura SQL Server de Armazenamento de Dados

O SQL Server utiliza a unidade básica de IO fixo (página) em 8KB. Para organizar melhor os dados, essas páginas são agrupadas em uma quantidade de oito contíguas entre si formando as chamadas extensões (*extents*). O espaço é gerenciado em termos de extensões. Cada página pertence a um objeto de um tipo (dado, índice etc), mas uma extensão pode pertencer a vários objetos.

Uma extensão em que as oito páginas pertencem ao mesmo objeto é considerada uma extensão uniforme, caso contrário, trata-se de uma extensão mista.

O SQL Server utiliza os *filegroups* para controlar a alocação das tabelas e dos índices. Os dados contidos em *filegroup* são proporcionalmente distribuídos entre os arquivos do *filegroup*. Se os *filegroups* não estiverem definidos, os objetos do banco de dados serão armazenados em um *filegroup* padrão que é implicitamente definido durante a criação do banco de dados.

Os segmentos utilizados no Oracle não são necessários no SQL Server. Em vez disso, o SQL Server distribui os dados através de RAID suportado em hardware ou software (solução presente no Windows ou em outro software).

66.3 Tratamento de Transações no SQL Server

O Oracle executa recuperação automática toda vez que é iniciado. Ele checa se os arquivos do *tablespace* estão sincronizados com o conteúdo dos arquivo de *redo log*. Caso não estejam, o Oracle aplica o conteúdo do arquivo de *redo log* no caso das transações comitadas e remove as transações não comitadas

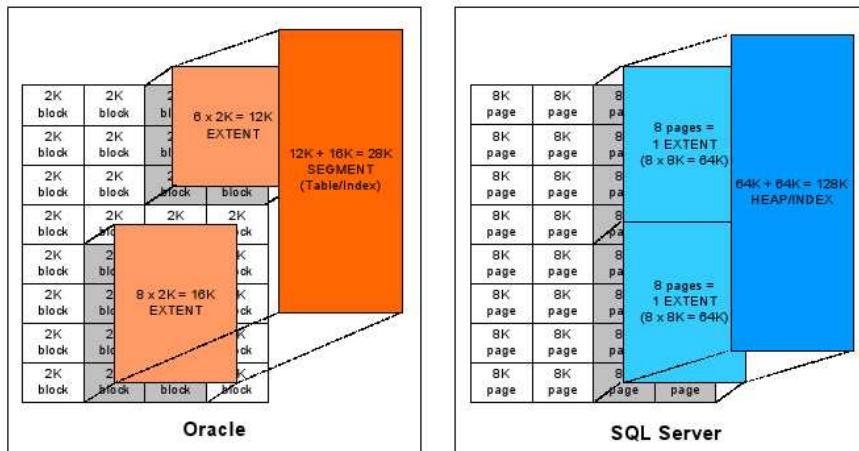


Figura 66.1: Comparação gráfica entre os esquemas de armazenamento do Oracle e do SQL Server

com o auxílio dos segmentos de *rollback*. Se o Oracle não possui a informação requerida, ele consulta os arquivos *redo log* arquivados pelo processo ARCH.

O SQL Server também executa recuperação automática através da checagem de cada banco de dados. Primeiramente o SQL Server checa o Msdb e então inicia *threads* para recuperação automática em todos os bancos de dados. Para cada banco de dados, o mecanismo de recuperação automática checa o *transaction log*. O tratamento é o mesmo do Oracle para transações comitadas e não comitadas.

Cada *transaction log* combina as funcionalidades de um segmento de *rollback* e de um *redo log* do Oracle. Cada banco de dados possui seu *transaction log* que armazena todas as alterações do banco de dados e é compartilhado com todos os usuários do banco de dados.

Internamente, os *transaction logs* são quebrados em pedaços menores conhecidos como *virtual log files* (VLFs). Quando um VLF não possui mais registros de logs para transações ativas, então o VLF pode ser dividido. O crescimento dos VLFs pode ser configurado para se ocorrer de forma automática e assim como no Oracle são preenchidos em forma circular.

Parte XIV

ITIL

Capítulo 67

Suporte a Serviços

67.1 Service Desk

67.1.1 Objetivos

- Atuar como ponto único de contato entre o usuário e o Gerenciamento de Serviços de TI;
- Tratar incidentes e requisição de serviços;
- Ser a principal interface operacional entre a TI e seus usuários;
- Promover a retenção e a satisfação do usuário;
- Aumentar o percentual de resolução de chamados no primeiro ponto de contato.

67.1.2 Responsabilidades

- Receber, registrar, priorizar e rastrear chamadas de serviço;
- Prover uma avaliação inicial de todos os incidentes reportados pelos usuários;
- Resolver incidentes com base em soluções já conhecidas ou identificar soluções de contorno para o reestabelecimento dos serviços;
- Escalar chamado para suporte em nível adequado;
- Monitorar e acompanhar as chamadas com base nos acordos de nível de serviços;
- Manter usuários informados;
- Produzir informações gerenciais;
- Fechar incidentes e confirmar com o cliente / usuário.

67.1.3 Vários Tipos de Central

Central de Atendimento *Call Center*. Apenas registra e encaminha para outras áreas da organização.

Central de Suporte *Call Center*. Gerencia, coordena e resolve incidentes o mais rápido possível. Utiliza o gerenciamento de comunicação e ferramentas de suporte ao conhecimento como tecnologia de apoio. Normalmente, trata apenas incidentes.

Central de Serviços *Service Desk*. Não trata somente incidentes, problemas e dúvidas, mas também fornece uma interface para outras atividades como requisições de mudança do usuário, contratos de manutenção, licenças de software, gerenciamento de nível de serviço etc.

67.2 Gerenciamento de Incidentes

67.2.1 Objetivos

Um incidente é qualquer evento que não faz parte do serviço combinado. A indisponibilidade de um serviço ou a diminuição de sua qualidade também pode ser considerado um incidente. Exemplos de incidente em um ambiente de TI são, por exemplo, uma aplicação com erro, um usuário que esqueceu a senha ou a indisponibilidade de um servidor, mesmo que os usuários não tenham sido impactados.

O objetivo do processo de gerenciamento de incidentes é restaurar a operação normal de um serviço minimizando o impacto no negócio. É importante enfatizar que o gerenciamento de incidentes trata o efeito e não a causa. A porta de entrada para o tratamento de incidentes é o Service Desk.

Os incidentes podem ser detectados pelos usuários ou por algum processo de monitoração, além de poderem surgir de dúvidas, manifestações ou requisições de serviço. As saídas do processo de gerenciamento de incidentes podem ser o reestabelecimento do serviço, o esclarecimento da dúvida, a atendimento da requisição de serviço, a identificação de uma necessidade de mudança, ou ainda, o registro de um problema.

67.2.2 Atividades do Processo

- Identificação e Registro do Incidente: Registro dos incidentes seja qual for a fonte de abertura, reportando sintomas, diagnóstico básico, itens de configuração envolvidos, perfil do usuário etc;
- Análise e Classificação: Especifica melhor os sintomas do incidente para determinar a solução;
- Priorização: Combina os fatores impacto e urgência para definir a prioridade do incidente em relação ao demais (Matriz Impacto x Urgência). Também é levado em consideração o acordo de nível de serviço (ANS ou SLA);

- Suporte Inicial: Caso exista uma solução previamente identificada na base de soluções (BDS), o analista deverá aplicá-la. Caso contrário, uma solução deverá ser implementada. Dependendo da complexidade, o registro pode ser escalado para níveis mais especializados;
- Investigação e Diagnóstico: Caso não seja encontrada uma solução definitiva, uma solução de contorno pode ser especificada. Nesse caso, deve ser aberto um registro de problema;
- Estratégia de Recuperação: Após definida a solução, esta deve ser implementada. Em algumas situações pode ser necessário abrir uma requisição de mudança (RDM);
- Fechamento: Garantir que o incidente foi devidamente tratado. Em alguns casos, isso significa checar se a RDM foi implementada. A validação do cliente também é importante para o fechamento;
- Informação e Comunicação: O Service Desk deve manter o histórico completo do registro e garantir que o usuário tenha acesso às informações seja qual for o status de atendimento do incidente ou requisição.

67.2.3 Papéis e Responsabilidades

- Gestor de Incidentes: Avaliar e garantir a eficiência e qualidade do serviço, auditar base de incidentes, garantir a avaliação da satisfação dos usuários, produzir relatórios gerenciais e realizar análise crítica do processo;
- 1º Nível de atendimento: Receber, categorizar e priorizar as requisições dos usuários. Realizar investigação inicial e aplicar soluções de contorno conhecidas. Caso necessário, encaminhar os incidentes para o nível adequado de atendimento;
- 2º Nível de atendimento: Desenvolver, quando possível, uma solução de contorno ou definitiva. Caso necessário, encaminhar incidente para o terceiro nível;
- 3º Nível de atendimento: Nível mais alto de atendimento de incidentes. Deve obrigatoriamente desenvolver uma solução de contorno;
- Equipes Especializadas: Atende às solicitações de serviço, esclarecimento de dúvidas e manifestações dos usuários encaminhadas pelo primeiro nível. Deve executar os serviços segundo procedimentos pré-definidos.

Os três níveis de atendimento e as equipes especializadas devem, quando necessário, emitir uma RDM, verificar as divergências encontradas na banco de dados de soluções e na banco de dados de gerenciamento de configuração (BDGC).

67.3 Gerenciamento de Problemas

67.3.1 Objetivos

Eliminar erros na infra-estrutura de TI, identificando e removendo a causa raiz, evitando assim, a recorrência de incidentes e problemas no ambiente operacional.

67.3.2 Definições Importantes

SC Soluções de Contorno. Eliminam o sintoma ou reduzem o impacto de um erro, sem eliminar as suas causas.

Problemas e Erros Conhecidos Um problema é a causa subjacente desconhecida de um ou mais Incidentes. Um Problema passa a Erro Conhecido quando a causa-raiz for conhecida e for identificada uma solução de contorno ou uma alternativa definitiva.

BDP Base de Dados de Problemas. Base de Dados que contém todos os registros de problemas.

BDS Base de Dados de Problemas. Base de Dados que contém informações referente aos Erros Conhecidos, suas respectivas Soluções de Contorno e demais soluções definitivas.

67.3.3 Atividades do Processo

- Controle de Problemas
 - Identificação e registro do Problema
 - Classificação do Problema
 - Investigação e diagnóstico do Problema
- Controle de Erros
 - Identificação e registro do Erro
 - Avaliação do Erro
 - Registro da resolução do Erro
 - Fechamento do Erro
 - Monitoramento da resolução do Erro
- Apoio no tratamento de Incidentes Graves
- Prevenção pro ativa de Problemas
 - Análise de tendências
 - Direcionamento das ações de suporte
 - Fornecimento de informações para a organização
- Obtenção de Informações gerenciais a partir dos dados de Problemas
- Completar as revisões dos principais Problemas

67.3.4 Papéis e Responsabilidades

Gestor de Problemas Analisar e garantir a eficiência do processo através da auditoria e da revisão. Responsável também pela produção de relatórios gerenciais, alocação dos Responsáveis Técnicos e pela identificação de tendências.

Responsável Técnico pelo Problema Identificar, registrar, categorizar e priorizar. Investigar e diagnosticar a causa do Problema.

67.4 Gerenciamento de Configuração

67.4.1 Objetivos

O objetivo do processo de gerenciamento de configuração é prover informações confiáveis sobre as configurações e documentações relativas a infra-estrutura de TI de forma suportar os demais processos de gerenciamento de serviços. O processo de gerenciamento de configuração é o principal responsável pela construção e manutenção do banco de dados de gerenciamento de configuração (BDGC), considerado o coração do modelo de gerenciamento de serviços do ITIL. Essa base de dados deve conter informações sobre todos os itens de configuração (IC's) do ambiente e os seus respectivos relacionamentos.

Um item de configuração é um componente que faz parte ou está associado a infra-estrutura, fazendo-se necessário para a prestação de um serviço. (Exemplo: servidor, software, procedimento etc.) Cada IC é composto por atributos como identificador único, nome, tipo etc.

O BDGC é fundamental para o gerenciamento de nível de serviço, uma vez que a partir dele é possível identificar todos os itens de configuração que fazem parte de um serviço específico. Além disso, o BDGC permite identificar se incidentes, problemas e mudanças em um determinado IC irão impactar nos demais. O BDGC só pode ser alterado mediante uma requisição de mudança (RDM).

Uma *baseline* é o estado do BDGC em determinado instante do tempo. Ela serve como referência para auditorias e também para estipular uma base de retorno após mudanças mal sucedidas, por exemplo.

67.4.2 Atividades

- Planejamento: Identificação do escopo, granularidade, objetivos, políticas e procedimentos para o processo de gerenciamento de configuração;
- Identificação e Denominação: Seleção e identificação dos IC's, definindo seus atributos, inter-relacionamento e identificação;
- Controle: Garante a autorização e identificação dos IC's nos processos de inclusão, alteração e descarte, além de não permitir alterações sem documentação de controle sejam feitas no BDGC;
- Registro e Histórico da Situação: Realiza a atualização no BDGC seguindo procedimentos pré-definidos, de forma não perder o controle do ciclo de vida dos IC's e vinculação às RDM's;
- Descrição da Situação: Geração de relatórios sobre status e ciclo de vida dos IC's, permitindo a localização de mudanças;
- Verificação e Auditoria: Verificar a consistência entre a situação real e o BDGC e demais bibliotecas controladas.

67.4.3 Papéis e Responsabilidades

- Gestor de Configuração: Implementar políticas de gerenciamento de configuração e assegurar sua disseminação e utilização por toda organização. Elaborar relatórios com indicadores do processo e análises de impacto. Deve ser membro do Grupo de Aconselhamento de Mudanças (GAMA);
- Administrador de Configuração: Administrar o BDGC, assegurando sua atualização a cada mudança. Deve ainda auditar o BDGC, garantindo sua consistência com o inventário físico.

67.5 Gerenciamento de Mudanças

67.5.1 Objetivos

Garantir a aplicação de métodos e procedimentos padronizados a fim de lidar eficientemente com todas as mudanças no ambiente operacional, minimizando os impactos na qualidade dos serviços e prevenindo a ocorrência de incidente provenientes das mudanças realizadas.

67.5.2 Responsabilidades

- Controle das alterações em todos os ICes;
- Levantamento e registro das mudanças;
- Avaliação de impacto, custo, benefício e risco associado às mudanças;
- Justificativa da mudança perante o negócio e obtenção de aprovação para as mudanças;
- Gerência e coordenação da implementação de mudanças;
- Monitoramento e informes sobre a implementação;
- Revisão e fechamento das RDM (requisições de mudanças).

67.5.3 Definições Importantes

Mudança Padrão É a mudança bem conhecida, rotineira e que comprovadamente não causa alto impacto.

Mudança Normal Planejada É a mudança planejada e programada respeitando o cronograma de atividades e o acordo com os clientes.

Mudança Emergencial É a mudança executada para recuperar serviços a fim de evitar danos à imagem do negócio, perdas financeiras e consequências legais.

RDM Requisição de mudança. *RFC - Requests for Change*. Documento que formaliza a necessidade da mudança e define detalhes de como ela acontecerá. Inclui identificação numérica, descrição e identificação dos ICs, motivo da mudança, efeito da não implementação da mudança, nome do solicitante da mudança, data proposta para a mudança, grau de prioridade, assinatura, plano de implementação, plano de retorno etc.

Gestor de Mudança *Change Manager* É responsável pelas mudanças que ocorrerão no ambiente de TI. Ele não detém o poder de autorizar qualquer mudança, isso cabe ao GAM.

GAM Grupo de Aconselhamento de Mudanças. *CAB - Change Advisory Board*. Tem a responsabilidade de planejar, avaliar e aprovar mudanças de médio e alto impacto. O grupo é multidisciplinar, pode ter representantes do Service Desk, desenvolvedores, área de relacionamento com o cliente etc.

GAM/CE Grupo de Aconselhamento de Mudanças/Comitê Emergencial. *CAB/EC - Change Advisory Board/Emergency Committee*. Tem a mesma função do GAM, porém para mudanças emergenciais de alto e médio impacto. Envolve níveis hierárquicos mais altos da TI.

Agenda de Mudanças (AM) Contém detalhes de todas as mudanças aprovadas e suas datas de implementação. É usada para que todos os grupos afetados por uma mudança possam se planejar.

Projeção de Indisponibilidade *PSA - Projected Services Availability*. Contém detalhes das mudanças e sua interferência nos Acordos de Nível de Serviços e na disponibilidade de serviços.

Revisão Pós-Implementação Revisão da Mudança realizada após a implementação. Tem por objetivo avaliar se a mudança atingiu e se houve imprevistos ou efeitos colaterais.

Priorização de Mudanças A priorização das mudanças é realizada de acordo com o seu impacto e a sua urgência.

67.6 Gerenciamento de Liberação

67.6.1 Objetivo

Gerenciar a liberação de componentes de software, hardware e documentos nas implementações e modificações dos serviços, garantindo liberações compatíveis, licenciadas e apropriadas para o ambiente de produção. Opera como interface entre o Gerenciamento de Mudanças e o Gerenciamento de Configurações.

67.6.2 Atividades do Processo

- Planejar e administrar um implementação bem sucedida de hardware e software.
- Projetar e implementar procedimentos eficientes para a distribuição e instalação de mudanças.
- Garantir a localização do HW e do SW e assegurar que apenas as versões corretas sejam instaladas.
- Gerenciar as expectativas dos clientes durante o planejamento e implementação (rollout) de novas liberações.
- Acordar o plano de implementação com o Gerenciamento de Mudanças.

- Testar o Plano de Retorno (fallback).
- Implementar as liberações no ambiente operacional, atualizando os ICs necessários.
- Garantir que as cópias originais de todos os softwares estão seguras em uma Biblioteca de Software Definitiva (BSD) e que o BDCG reflita as informações do BSD e das liberações realizadas no ambiente.
- Garantir a acurácia do DHD (Depósito de Hardware Definitivo) e que todo IC retirado para ser utilizado no ambiente operacional seja reposto.
- Garantir que todo o hardware a ser implantado ou alterado esteja protegido e seja localizável, utilizando os serviços de Gerenciamento de Configuração.

67.6.3 Definições Importantes

BSD Biblioteca de Software Definitiva. *DSL - Definitive Software Library*. Coleção de itens de configuração de software e documentação guardados em local seguro. Todas versões liberadas devem estar salvas na BSD.

DHD Depósito de Hardware Definitivo. *DHS - Definitive Hardware Store*. Depósito de componentes de reposição no mesmo nível dos sistemas correspondentes no ambiente de produção. Uma vez encerrada a utilização dos componentes, devem retornar ao DHD, ou adquirir substitutos. Assim como a BSD, o DHD é um repositório e quem contém as informações sobre os atributos dos itens é o BDGC.

Liberação completa Todos os componentes são construídos, testados, distribuídos em conjunto.

Liberação Delta São incluídos os ICs alterados desde a última alteração.

Liberação Pacote Liberações completas, deltas ou as duas são agrupadas em pacote para liberação.

67.6.4 Papéis e Responsabilidades

Gestor de Liberações Garantir que o processo seja seguido. Efetua auditoria no BSD e no DHD. Autoriza a liberação para o ambiente de produção.

Responsável Técnico pela Liberação Desenvolver o projeto, construção e configuração da liberação. Realizar testes. Realizar a instalação e distribuição de SW e HD. Armazenar o software na BSD.

Capítulo 68

Entrega de Serviços

68.1 Gerenciamento do Nível de Serviço

68.1.1 Objetivos

O gerenciamento do nível de serviço visa a melhoria da qualidade dos serviços de TI através de um ciclo contínuo de negociação e monitoração, promovendo ações para eliminar níveis de serviço inaceitáveis. A essência do processo em questão é o Acordo de Nível de Serviço (ANS) (*Service Level Agreement - SLA*), que funciona como um contrato entre a TI e os clientes. O ANS deve descrever em detalhes os serviços fornecidos, além especificá-los quanto a qualidade, quantidade, desempenho e disponibilidade. Os principais objetivos do processo de gerenciamento de nível de serviço são:

- Melhorar a especificação dos serviços;
- Reduzir demandas imprevistas;
- Avaliar o custo dos serviços;
- Controlar o nível de serviço;
- Promover a melhoria da qualidade do serviço;
- Manter alinhamento com o negócio.

Para alcançar tais objetivos, o processo de gerenciamento de nível de serviço tem as seguintes responsabilidades:

- Catálogo de Serviços: Define, do ponto de vista do cliente, os serviços fornecidos;
- Acordo de Nível de Serviço (ANS): Acordos formais entre a TI e seus clientes, onde são definidas as metas e as responsabilidades de ambas as partes. Um ANS deve conter informações como descrição do serviço, agenda (24x7,5x8), suporte (sobreaviso, prorrogação, tempo de reparo), procedimentos para mudanças, metas para disponibilidade, confiabilidade, segurança, capacidade etc;

- Requisições de Nível de Serviço: Documentos para fornecer visão detalhada das necessidades do cliente, usados para ajustar ou criar novos serviços (*Service Level Request - SLR*);
- Acordo de Nível Operacional (ANO): Formalização de acordos com fornecedores internos da organização, sobre manutenção de serviços ou componentes de serviços (*Operational Level Agreement - OLA*);
- Contratos de Apoio: Contratos de apoio com fornecedores externos (*Underpinning Contract - UC*);
- Programa de Aperfeiçoamento de Serviços (PAS): Identificar e implementar ações necessárias para superar dificuldades e restabelecer a qualidade do serviço;
- Gerenciamento do Relacionamento com o Cliente: Administrar o relacionamento com os clientes visando a sustentação dos acordos de nível de serviço e nivelamento das expectativas;
- Monitoração, Revisão e Informação: Garantir o andamento do processo e o cumprimento das metas estabelecidas nos acordos.

68.2 Gerenciamento Financeiro

68.2.1 Objetivos

Administrar os recursos monetários, garantindo o planejamento e execução dos objetivos de negócio. Identificar, calcular e gerenciar os custos de entrega de serviço. Fornecer dados de previsão orçamentária para a administração.

68.2.2 Responsabilidades

- Otimizar os recursos financeiros.
- Apoiar decisões de investimentos.
- Atribuir valores dos serviços e rateio dos custos por cliente.
- Influenciar o comportamento do cliente.
- Melhorar o controle dos contratos Externos/Fornecedores.

68.2.3 Atividades do Processo

Dentro de uma organização de TI existem três disciplinas principais para o Gerenciamento Financeiro:

Orçamento

Processo de previsão e controle dos gastos em dinheiro. As suas responsabilidades compreendem:

- Previsão do orçamento necessário para a realização de serviços de TI no período acordado.

- Garantia que a despesa real possa ser comparada com a despesa prevista a qualquer momento do período.
- Redução do risco de gastar além do orçado.
- Garantir a disponibilização de verba para cobrir despesas extras.
- Garantia que as receitas estarão disponíveis para cobrir despesas previstas.

Contabilidade

Conjunto de processos que permite a organização de TI prestar contas de como o dinheiro foi gasto em um determinado período de tempo, sendo capaz de identificar custo por clientes, serviços ou atividades. Deve ter focos nos custos para justificar investimentos e lucro zero para que a organização possa contabilizar as despesas de TI. As responsabilidades da disciplina compreendem:

- Contabilização do dinheiro gasto para provisão dos serviços de TI.
- Cálculo do custo de provimento dos serviços de TI para clientes externos e internos.
- Análise Custo x Benefício ou o retorno do investimento (ROI).
- Identificação do custo de mudanças.

Cobrança

Conjunto de processos necessários para faturar um Cliente (Oops...) pelos serviços prestados, otimizando a relação custo/benefício do ponto de vista do negócio. As responsabilidades da disciplina compreendem:

- Recuperação dos custos dos serviços de TI dos clientes dos serviços.
- Rateio dos custo entre os clientes da TI.
- Influência do comportamento do Cliente.
- Precificar os serviços.

68.2.4 Elementos de Custo

Os elementos de custo típicos são: hardware, software, pessoal, acomodações, serviço externo e transferência de custo (cobrados por outros centros de custo da empresa). Os elementos de custo podem ser classificados em fixos, variáveis, diretos, indiretos (que devem ser divididos entre vários clientes), de capital e operacional (do dia-a-dia).

68.3 Gerenciamento da Capacidade

68.3.1 Objetivos

O gerenciamento de capacidade visa garantir o atendimento das necessidades futuras do negócio estabelecendo sempre um equilíbrio entre demanda e custos. O principal produto do processo A principal saída desse processo é um documento chamado Plano de Capacidade. Nele são reportadas as previsões de carga, hardware e software necessários, custos e outras recomendações. Os principais modelos para elaboração de um plano de capacidade são os seguintes:

- Modelagem por Referência: Modelagem a partir de um modelo válido pré-existente;
- Análise de Tendências: Projeções futuras com base em dados históricos;
- Modelagem Analítica: Validação de um modelo matemático com a situação real;
- Modelagem por Simulação: Utilização de dados fictícios para dimensionamento de novas aplicações;
- Testes de Laboratório: Testes como dados reais em um ambiente real.

O gerenciamento da demanda tem por objetivo influenciar a demanda, direcionando o seu comportamento. Para isso pode ser necessário realizar cobrança diferencial de acordo com o horário, limitar tempo de uso em horários de pico etc. A tarefa de monitoração de desempenho dos recursos e serviços para garantir o estabelecido no plano de capacidade é chamada gerenciamento de performance. No contexto de gerenciamento de capacidade outra atividade muito importante é a modelagem e dimensionamento de aplicações (*Sizing de Aplicações*).

Um elemento importante no processo de gerenciamento de capacidade é o BDC (Banco de Dados de Capacidade), que armazena dados do negócio, dos serviços, financeiros e de utilização, sendo, portanto, base para elaboração de relatórios relacionados a questão de capacidade atuais e futuras.

68.3.2 Atividades

- Gerenciamento de Capacidade do Negócio: Garante que as capacidades futuras do negócio sejam consideradas e planejadas em tempo certo. Requer conhecimento do Plano de Negócio da organização e técnicas de modelagem;
- Gerenciamento da Capacidade do Serviço: Monitoração do desempenho dos serviços de TI;
- Gerenciamento da Capacidade dos Recursos: Gerencia os componentes de infra-estrutura de TI para garantir que desempenho e demanda estão de acordo com o planejado.

68.4 Gerenciamento de Disponibilidade

68.4.1 Objetivos

O processo de gerenciamento de disponibilidade visa garantir que os serviços de TI estarão disponíveis sempre que os clientes necessitarem deles, de acordo com os níveis de disponibilidade acordados. Os principais elementos de disponibilidade de um serviço são:

- Disponibilidade: Tempo durante o qual o serviço estará disponível;
- Confiabilidade: Capacidade de se manter operacional dentro de um determinado período de tempo;
- Sustentabilidade: Capacidade de retorno ao estado normal após algum incidente;
- Resiliência: Capacidade de se manter operacional mesmo na falta de um ou mais componentes;
- Oficiosidade: Obrigações contratuais com terceiros (Contratos de Apoio);
- Segurança: Confidencialidade, Integridade e Segurança.

Dizemos que um serviço está disponível quando o cliente o recebe dentro das condições do acordo de nível de serviço. Portanto, a taxa de disponibilidade é dada pela seguinte fórmula:

$$\text{Disponibilidade} = 100((TSA - TF)/TSA))$$

onde TSA é o tempo de serviço acordado e TF é o tempo durante o qual o serviço ficou indisponível. Exemplo: Se um serviço tem TSA 24x7 (precisa estar disponível 24 horas 7 dias por semana) e ficou indisponível por um total de 10 horas durante um mês, então a taxa de disponibilidade do serviço no período foi de $100x(((30x24x7)-10)/(30x24x7))$, ou seja, 99,8015873%.

68.4.2 Ciclo de vida do incidente

Suponhamos que um incidente ocorrido no instante t_1 tenha ocasionado a interrupção de um serviço. No instante t_2 o serviço foi restabelecido, e em um instante posterior t_3 , um outro incidente voltou a interromper o serviço. A partir dos valores de t_1 , t_2 e t_3 podemos definir as seguintes métricas:

- TMPR (Tempo Médio para Reparo) = $t_2 - t_1$;
- TMEF (Tempo Médio Entre as Falhas) = $t_3 - t_2$;
- TMEIS (Tempo Médio entre Incidentes de Sistema) = $t_3 - t_1$.

68.5 Gerenciamento de Continuidade

68.5.1 Objetivos

O processo de gerenciamento de continuidade tem como principal objetivo a continuidade operacional dos serviços de TI em ocasiões de desastre, a fim de suportar os requisitos mínimos do negócio. O principal produto desse processo é um documento chamado Plano de Continuidade dos Negócios (*Business Continuity Plan - BCP*). Os principais componentes de um plano de continuidade são:

- Administração: Quando e como invocar o plano;
- Infra-Estrutura de TI: Quais os itens envolvidos;
- Procedimentos Operacionais: Instruções de trabalho;
- Pessoal: Responsáveis, contatos e substitutos;
- Site de Contingência: Localização, contato e transporte;
- Retorno ao normal: Como, onde e quanto tempo para o retorno.

Algumas métricas importantes de retorno a normalidade são o RTO (*Recovery Time Objective*) e o RPO (*Recovery point Objective*). O RTO indica o último instante no tempo a partir do qual a operação deve ser restabelecida após um desastre. O RPO, por sua vez, determina o ponto a partir do qual os dados devem ser recuperados após um desastre.

No processo de gerenciamento de continuidade são realizadas análises detalhadas de risco e impacto, utilizando metodologias de análise de risco como CRAMM (CTTA *Risk Analysis and Management Methodology*), OCTAVE (*Operationally Critical Threat, Asset, and Vulnerability Evaluation*), CORAS etc. Todas essas metodologias são baseadas na norma ISO 27001. Uma elaboração de uma análise de risco e impacto, geralmente, se baseia na identificação de três elementos que são: Ativos, Ameaças e Vulnerabilidades.

E a sopa de letrinhas não tem fim. Algumas outras siglas que podem ser úteis na hora da prova são: BIA (Business Impact Analysis), OCP (Operational Continuity Plan), DRP (Disaster Recovery Plan). Todas essas análises e planos, na verdade, fazem parte do Plano de Continuidade de Negócios.

68.5.2 Estágios

- Início: Obtenção do aval gerencial e levantamento dos serviços de TI críticos para o negócio;
- Requisitos Estratégicos: Análise de Risco e de Impacto e elaboração da estratégia de continuidade;
- Implementação: Planejamento e implementação das instalações para recuperação, elaboração do plano de continuidade, medidas para redução de riscos e realização de testes iniciais;

- Gerenciamento Operacional: Educação, revisão, auditoria, testes, treinamento e gerenciamento de mudanças.

68.5.3 Tipos de Continuidade

- Soluções de contorno manuais: Medidas temporárias até que os serviços sejam restabelecidos;
- Acordos de reciprocidade: Organizações concordam auxiliar uma a outra em caso de desastres;
- Recuperação Gradual (*Cold Standby*): Ambiente computacional vazio, exceto por energia e telecomunicações. Mínimo de 72 horas para reestabelecimento dos serviços;
- Recuperação Intermediária (*Warm Standby*): Ambiente com equipamentos para recuperação dos serviços mais críticos entre 24 e 72 horas;
- Recuperação Imediata (*Hot Standby*): Instalação alternativa com espelhamento contínuo de equipamentos e dados.

Parte XV

Gerência de Projetos segundo PMBOK

Capítulo 69

Gerenciamento de Escopo

A primeira área de conhecimento definida pelo PMBOK é a de Gerenciamento de Escopo. Os principais processos respectivas atividades desta área são:

1. *Iniciação*: declaração formal do início do projeto, com a elaboração do *Project Charter*;
2. *Planejamento*: declaração do escopo, indicando justificativas para o projeto, além dos principais produtos, subprodutos e objetivos do projeto;
3. *Detalhamento*: elaboração da Estrutura Analítica do Projeto (*EAP*), também conhecida como Work Breakdown Structure (*WBS*), que tem por objetivo identificar fases, atividades e pacotes de trabalho. A *WBS* é de grande importância para realização de estimativas de tempo e custo;
4. *Verificação*: trata da aceitação do escopo;
5. *Controle de Mudanças*: identifica as mudanças de escopo assim como seus motivos e impactos, além de tratar dos ajustes necessários.

69.1 WBS e Definição do Escopo

No processo de Definição do Escopo, as entregas são subdivididas em componentes menores e gerenciáveis, para permitir o planejamento de tarefas e atividades. As ferramentas e técnicas desse processo são a decomposição e os modelos da estrutura analítica do projeto (*EAP* ou *WBS* - Work Breakdown Structure).

Segundo o PMBOK, a *EAP* é um conjunto de componentes do projeto, estruturados com base nas entregas, que organiza e define o escopo total do projeto; o trabalho não incluído na *EAP* está fora do escopo do projeto. Assim, com a declaração de escopo, a *EAP* equivale a um acordo básico entre os stakeholders e os integrantes da equipe com relação ao escopo do projeto. Enfim, a *EAP* deve especificar o escopo completo do trabalho necessário para concluir o projeto.

A decomposição é uma das ferramentas usadas na preparação da *EAP*, e envolve a subdivisão das entregas em componentes distintos o suficiente para

facilitar a realização das estimativas, a definição de baseline para controle de desempenho e a atribuição clara de responsabilidades. A estrutura mais comum usada para representar essa decomposição é árvore. O nível 1 (raiz) é considerado o próprio projeto, seguido pelas entregas, que podem ser seguidas por outras entregas, seguidas por atividades e assim por diante. Cada elemento de cada nível da EAP deve receber um identificador exclusivo.

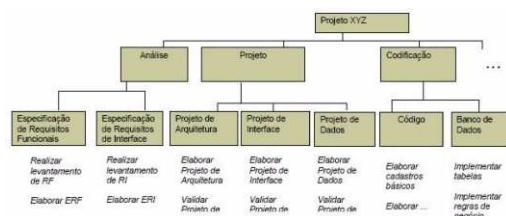


Figura 69.1: WBS - Work Breakdown Structure

O gerente de projeto pode determinar à vontade o número de níveis na EAP, com base na complexidade do projeto. Independentemente do número de níveis da EAP, o nível mais baixo de cada estrutura é denominado pacote de trabalho. Pacotes de trabalho são as atividades ou componentes que podem ser facilmente atribuídos a uma pessoa ou grupo, que assumirão a clara responsabilidade pelo seu cumprimento. É no nível de pacotes de trabalho que se fazem as estimativas de tempo, custos e recursos. Cabe ressaltar que não existe nenhuma regra no PMBOK determinando a presença de atividades na EAP. Entretanto, quando possível, considera-se uma boa prática.

Finalmente, o Dicionário da EAP contém uma descrição do nível dos pacotes de trabalho e os detalhes relativos às atividades dentro de cada pacote. Nele se encontram informações como custos, orçamentos, prazos, distribuição de recursos e descrição das atividades. Outro conceito importante são os marcos de projeto (*milestones*). Os marcos são realizações significativas do projeto que ajudam a verificar o andamento dos trabalhos.

Capítulo 70

Gerenciamento de Recursos Humanos

A fase de Gerenciamento de Recursos Humanos engloba os seguintes processos:

1. *Planejamento Organizacional*: definição de equipes, perfis e responsabilidades, além do formato da equipe (ex: democrática descentralizada, controlada centralizada, controlada descentralizada). Deve levar em consideração as características da organização (matricial forte ou fraca, orientada a projetos,etc);
2. *Montagem de Equipe*: alocação de pessoas ou equipes aos papéis definidos;
3. *Desenvolvimento da Equipe*: estratégias para aumentar capacidade das pessoas e equipes, gerenciar conflitos (ex: negociação,panos quentes,retirada,força,etc). Principais teóricos: McGregor (*Teoria XY*), Maslow (*Pirâmide de Necesidades*), Herzberg (*Fatores Motivacionais*).

70.1 Estruturas Organizacionais

70.1.1 Organização Funcional

As organizações funcionais giram em torno de especialidades e são agrupadas por função - daí o nome organização funcional. Nela pode haver, por exemplo, um departamento de recursos humanos, outro de marketing e etc. O trabalho executado nesses departamentos é especializado e requer que os funcionários tenham aptidões específicas e experiência nas funções para cumprir as responsabilidades. Esse tipo de organização é configurado de maneira hierárquica, ou seja, cada funcionário responde a um único gerente e, em última instância, existe um único responsável no topo. Cada departamento ou grupo é administrado separadamente e tem um âmbito de controle limitado. A seguir são apresentadas as vantagens e desvantagens dessa organização.

As principais vantagens e desvantagens dessa organização são:

- Vantagens:

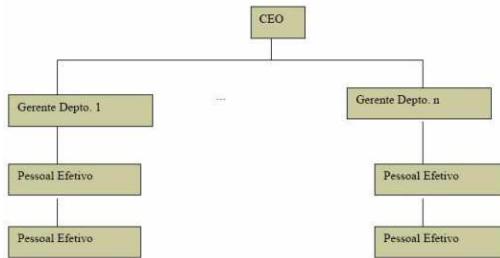


Figura 70.1: Organização Funcional

- Estrutura organizacional duradoura;
 - Carreira profissional transparente;
 - Cadeia de comando explícita.
- Desvantagens:
 - O gerente de projeto tem pouca ou nenhuma autoridade oficial;
 - Vários projetos disputam recursos limitados e prioridades;
 - Os integrantes do projeto são leais ao gerente funcional.

70.1.2 Organização por Projeto

As organizações por projeto são praticamente o oposto das organizações funcionais. O enfoque é o próprio projeto, assim cultiva-se a lealdade ao projeto e não a um gerente funcional. Os gerentes de projeto têm total autoridade sobre o projeto e respondem diretamente ao CEO. As equipes são formadas e seus membros com freqüência são co-alocados, ou seja, trabalham no mesmo local físico.

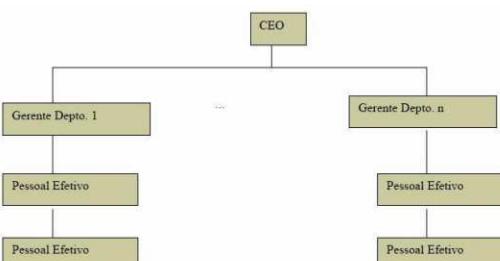


Figura 70.2: Organização por Projeto

As principais vantagens e desvantagens dessa organização são:

- Vantagens:
 - O gerente de projeto é a autoridade máxima;

- O enfoque da organização é o projeto;
 - Os recursos da organização são destinados ao projeto;
 - Lealdade ao gerente do projeto;
 - Comunicação efetiva.
- Desvantagens:
 - Quando o projeto é finalizado a equipe é desalocada;
 - Uso dos recursos pode não ser eficiente;
 - Duplicação de funções.

70.1.3 Organização Matricial

As organizações matriciais surgiram para minimizar as diferenças entre os pontos fortes e fracos das organizações funcionais e das estruturadas por projeto. e melhor explorá-los. Assim, os objetivos são atendidos, técnicas eficientes de gerenciamento de projetos são aplicadas, ao mesmo tempo que também se mantém a estrutura hierárquica da organização.

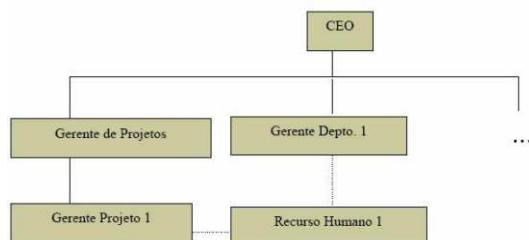


Figura 70.3: Organização Matricial

Os funcionários de uma organização matricial se reportam ao gerente funcional e a, no mínimo, um gerente de projeto - podendo ser subordinados a vários gerentes de projeto, caso trabalhem em vários projetos simultâneos. O gerente funcional responde por incumbências administrativas e aloca funcionários para os projetos, além de monitorar o trabalho de seus funcionários nos diversos projetos. O gerente de projeto, por sua vez, é responsável pela execução do projeto e distribuição das tarefas de acordo com as atividades previstas. Ambos dividem as responsabilidades pela avaliação de desempenho dos funcionários.

Como se pode perceber, há muita comunicação e negociação entre os gerentes de projeto e funcional. Isso requer um equilíbrio de poder entre os dois. Desse modo, são três os tipos de organização matricial:

- Matricial forte: gerente de projeto tem prioridade sobre gerente funcional;
- Matricial fraca: gerente funcional tem prioridade sobre gerente de projeto;
- Matricial Equilibrada: há negociação entre gerente funcional e gerente de projeto.

Em todo o caso, as vantagens e desvantagens são:

- Vantagens:
 - Quando o projeto é finalizado, a equipe é alocada em outras atividades na empresa;
 - A disseminação de informações ocorre na vertical e horizontal;
 - Os recursos escassos são bem utilizados.
- Desvantagens:
 - Mais de um gerente para a equipe do projeto se reportar;
 - Grande probabilidade de conflitos;
 - Precisa de pessoal administrativo extra para cumprir as necessidades dos projetos.

70.2 Planejamento Organizacional

O objetivo desse processo é documentar as funções e responsabilidades de indivíduos e grupos para cada elemento do projeto e, em seguida, registrar as relações de subordinação de cada um. Neste momento devem ser definidas todas as equipes que estarão envolvidas no projeto. Dessa forma, Marilyn Mantei propõe três tipos de estruturas para equipes:

- Democrática Descentralizada: nessa estrutura não há um líder ou gerente permanente. As decisões são tomadas por consenso. Na prática, não funciona muito bem para projetos;
- Controlada Descentralizada: nessa estrutura há um gerente geral para o projeto e gerentes secundários ou líderes locais. As decisões de problemas do projeto são tomadas por consenso entre o gerente do projeto e os gerentes secundários. A execução das ações são controladas pelos gerentes secundários. Há comunicação horizontal e vertical. Essa estrutura é comum em projetos que possuem vários módulos, principalmente, se estes são desenvolvidos em paralelo;
- Controlada Centralizada: nessa estrutura há um gerente geral para o projeto e os demais membros estão diretamente ligados a ele. Não há líderes secundários. Só há comunicação vertical.

70.3 Desenvolvimento da Equipe

O desenvolvimento da equipe implica criar um ambiente franco e estimulante para que os stakeholders contribuam, e transformar a equipe num grupo coordenado, funcional e eficiente. Nesse aspecto é desejável manter a equipe sempre motivada. São várias as teorias sobre a motivação. As principais são:

- Hierarquia de necessidades de Maslow;

- Teoria da Higiene ou teoria Motivação-Higiene;
- Teoria das Expectativas;
- Teoria da Realização.

Maslow defendia a hipótese de que os seres humanos possuem cinco necessidades básicas, distribuídas em ordem hierárquica. Eis um resumo de cada uma delas, do nível mais alto até o mais baixo:

- Realização pessoal - Trabalhar no seu potencial máximo;
- Necessidade de auto-estima - Realização, respeito por si mesmo, capacitação;
- Necessidades sociais - Senso de pertencimento, amor, aceitação, amizade;
- Necessidade de proteção e segurança - Seu bem-estar físico e seus pertences.

Quando todas as necessidades físicas, de proteção. Sociais e de auto-estima forem atendidas, a pessoa alcançará um estado de independência em que conseguirá se expressar e realizar ao máximo, e fará um bom trabalho apenas pelo fato de fazer um bom trabalho.

Herzberg desenvolveu a teoria da Higiene que postula a existência de duas categorias de fatores que contribuem para a motivação: fatores de higiene e motivadores. Os primeiros tratam de questões relacionadas ao ambiente de trabalho: evitam a insatisfação: salário, benefícios, as condições de ambiente de trabalho. Os motivadores dizem respeito à substância do trabalho em si e com a satisfação do sujeito ao exercer as funções do cargo: conduzem a satisfação.

Segundo a Teoria das expectativas, é a perspectiva de um resultado positivo que aciona a motivação: as pessoas vão apresentar um determinado comportamento de acharem que receberão uma boa recompensa para isso. Além disso, a força da expectativa influí no comportamento. A teoria alega ainda que as pessoas se tornam o que se esperam delas.

Segundo a Teoria da Realização somos motivados por três necessidades: realização, poder e afiliação. A motivação da realização é a necessidade de triunfar ou atingir um dado objetivo. A motivação do poder refere-se ao desejo de influenciar os comportamentos alheios. E a necessidade de afiliação diz respeito a relacionamentos.

Ao lado das teorias motivacionais estão as teorias de sobre a liderança. Destacam-se as teorias X e Y e a teoria da contingência.

McGregor estabeleceu dois modelos de comportamento dos trabalhadores, a Teoria X e a Teoria Y, que procuram explicar como cada gerente lida com os integrantes de suas equipes. Os gerentes adeptos da Teoria X acreditam que a maioria das pessoas não gosta de trabalhar, têm pouca ou nenhuma ambição, precisam de supervisão constante não cumprião suas obrigações, a menos que

ameaçadas. Em razão disso, agem como ditadores e impõem ao seu pessoal controles muito rígidos. Já os gerentes partidários da Teoria Y entendem que as pessoas vão se interessar por dar o melhor de si, se tiverem a motivação certa e as expectativas adequadas. Assim, proporcionam apoio às suas respectivas equipes, preocupam-se com seus membros e são bons ouvintes.

A teoria da contingência resulta de uma combinação entre as Teorias Y e da Higiene; em suma, argumenta que somos motivados a atingir níveis de competência e continuaremos motivados por essa necessidade mesmo depois de atingir a competência.

Capítulo 71

Gerenciamento do Tempo

Os processos da área de Gerenciamento do Tempo segundo o PMBOK são:

1. *Definição das Atividades*: definir as atividades e respectivos responsáveis.
2. *Sequenciamento das atividades*: estabelecer relações de precedência entre as atividades (Diagramas de Precedencia);
3. *Estimativas de Duração*: estimar a duração do projeto utilizando alguma abordagem (*TopDown*, *BottomUp*). As principais técnicas utilizadas para realização de estimativas são Modelos Matemáticos, Analogias e Consultoria de Especialistas;
4. *Desenvolvimento do Cronograma* : utilização de alguma técnica matemática como PERT, CPM (*Critical Path Method*) para formalizar o cronograma. Técnicas de *Nivelamento de Recursos* leva a alocação dos recursos em conta no estabelecimento de precedências das atividades. Para comprimir o cronograma as técnicas mais comuns são *Crashing* (adição de recursos) e *Fast-Tracking* (antecipação de inicio de atividade);
5. *Controle do Cronograma*: comparar, analisar e revisar o cronograma para permitir administrar desvios ao longo do projeto.

71.1 Técnicas de Desenvolvimento do Cronograma

Consiste em formalizar o cronograma, identificando suas datas, caminhos críticos e registrando os marcos e pontos de controle identificados. Há seis ferramentas e técnicas usadas no Desenvolvimento de Cronograma:

- Análise Matemática;
- Compressão de cronograma;
- Simulação;
- Heurística de nivelamento de recursos;
- Software de gerenciamento de projeto;
- Estrutura de Codificação.

71.1.1 Análise Matemática

A análise matemática consiste no cálculo das datas de início mais cedo e de término mais tarde das atividades do projeto - sem levar em conta as limitações dos recursos, o que acaba gerando prazos teóricos. A análise matemática abrange três técnicas:

- Método do caminho crítico (CPM - Critical Path Method);
- Técnica de Análise e Avaliação Gráfica (GERT - Graphical Evaluation and Review Technique);
- Técnica de Análise e Avaliação de Programa (PERT - Program Evaluation and Review Technique).

O método CPM calcula as datas de início mais cedo, de fim mais cedo, de início mais tarde e de fim mais tarde de cada atividade. Para isso, baseia-se em redes seqüenciais (em que uma atividade acontece depois da outra) e numa estimativa única de duração para cada atividade. O caminho crítico de todo o projeto é o caminho mais longo. Toda atividade com folga igual a zero é considerada uma tarefa do caminho crítico. Assim, para calcular a duração do caminho crítico basta somar a duração de cada atividade com folha igual a zero. Existem dois tipos de folga: folga total e folga livre. A primeira é o tempo total pelo qual é possível atrasar o início da tarefa sem prorrogar o término do projeto. Já a folga livre é o tempo durante o qual é possível atrasar o início da tarefa sem prorrogar o início de uma tarefa subsequente.

O que é preciso saber sobre a técnica GERT é que ela aceita desvio condicional, looping e tratamento probabilístico. Por exemplo, um projeto de um software pode exigir o teste de módulos individuais antes da verificação do programa como um todo, a cada módulo precisa ser aprovado em seu teste à parte. Cada teste constitui uma execução no laço.

A técnica PERT usa o valor esperado para definir a duração do projeto, portanto se baseia em ferramentas estatísticas para isso. As técnicas com e PERT são muito parecidas, entretanto CPM usa a duração mais provável e PERT o valor esperado (ou média ponderada).

71.1.2 Compressão do Cronograma

Trata-se de uma condensação do cronograma de modo a finalizar todas as atividades antes do estimado. Duas técnicas são usadas; compressão propriamente dita (crashing) e paralelismo (fast tracking). A compressão é uma técnica que leva em conta a relação entre custo e cronograma. Uma das possibilidades é adicionar recursos às tarefas do caminho crítico, já que estas atividades possuem folga zero. Podem-se também limitar ou reduzir os requisitos do projeto. Nem sempre produz os resultados esperados e geralmente acaba aumentando os custos. O paralelismo consiste em iniciar ao mesmo tempo duas atividades que a princípio estavam programadas para serem seqüenciais. A técnica pode aumentar os riscos do projeto e acarretar retrabalho.

71.1.3 Simulação

Na simulação parte-se de diferentes conjuntos de premissas para chegar a diversas durações para ao projeto. O principal representante é a Simulação de Monte Carlo que mostra a probabilidade de todas as possíveis datas de término do projeto. Outra técnica chamada what-if (análise de hipóteses) permite examinar o cronograma sob diversas circunstâncias.

71.1.4 Heurística do nivelamento de recursos

A análise matemática não leva em consideração a disponibilidade de recursos. Entretanto, com freqüência o cronograma inicial tem períodos de tempo com mais atividades do que recursos disponíveis para elas e que nem sempre é possível alocar às tarefas 100 % do tempo dos membros da equipe. Nesse contexto, o nivelamento de recursos - também chamado método baseado em recursos - visa a aplaciar a distribuição de recursos para que as tarefas sejam concluídas sem sobreregar ninguém, ao mesmo tempo procurando manter o projeto dentro do cronograma.

71.1.5 Estrutura de Codificação

A estrutura de codificação é uma ferramenta e técnica relacionada à possibilidade de ordenar atividades com base em características que lhes podem ser atribuídas - como por exemplo classificações da EAP, fases do projeto, tipo de atividades e etc.

Capítulo 72

Gerenciamento de Custo

Os processos englobados pela área de Gerenciamento de Custo são:

1. *Planejamento dos Recursos*: determinar quais recursos serão necessários e em que quantidade;
2. *Estimativa dos Custos*: associar custo aos recursos enumerados;
3. *Orçamentação dos Custos*: alocação das estimativas globais de custo são utilizadas para formalizar o orçamento;
4. *Controle dos Custos*: revisar o orçamento, identificar e administrar desvios.

72.1 Técnicas de Estimativas de Custos

As estimativas de custos emprega cinco ferramentas e técnicas para gerar estimativas:

- Estimativa Análoga
- Modelagem Paramétrica
- Estimativas bottom-up
- Ferramentas computadorizadas

72.1.1 Estimativas Análogas

As estimativas análogas, também chamadas de estimativas top-down, são uma forma de opinião especializada. Nessa técnica usa-se a duração real de uma atividade similar realizada num projeto anterior para projetar a duração da atividade atual. Essa técnica é muito útil quando as atividades anteriores de referência são realmente próximas com a atividade que estão sendo avaliada. É particularmente útil quando não há informações minuciosas sobre o projeto - como ocorre em suas fases iniciais, por exemplo.

72.1.2 Modelagem Paramétrica

É um modelo matemático que usa parâmetros - ou características do projeto - para prever custos. A idéia é descobrir um ou vários parâmetros que se modifiquem paralelamente aos custos do projeto, incorporando-os ao modelo para chegar a um custo total. Para utilizar essa técnica é preciso haver um padrão no trabalho, de modo que possamos usar uma estimativa desse elemento para gerar a estimativa total do projeto. A modelagem paramétrica é semelhante à estimativa análoga, na medida em que prevê o custo do projeto inteiro de cima para baixo.

72.1.3 Estimativa bottom-up

Esta técnica é oposta à top-down, pois estima-se cada atividade ou item de trabalho separadamente para depois reuni-las ou agregá-las numa estimativa total do projeto. Trata-se de um método de estimação muito preciso, entretanto é muito demorada, já que o pacote de trabalho deve ser avaliado e estimado com exatidão para ser incluído no cálculo. Essa não é uma alternativa adequada para fazer uma estimativa de custo na fase inicial do projeto, dada a inexistência de dados suficientes nessa etapa.

Capítulo 73

Gerenciamento de Riscos

A área de gerenciamento de riscos é responsável por minimizar a probabilidade de ocorrência de desvios ao longo do ciclo de vida do projeto. Os processos do Gerenciamento de Riscos são:

1. *Planejamento*: definir qual será a abordagem aos riscos;
2. *Identificação*: identificar, descrever e classificar os riscos;
3. *Análise Qualitativa*: associar probabilidades e graus de impacto aos riscos através de técnicas como *Matriz de Graduação*, *Matriz Probabilidade x Impacto*;
4. *Análise Quantitativa*: associar probabilidades e graus de impacto de forma numérica utilizando técnicas como *Análise de Sensibilidade*, *Árvore de Decisão* e *Simulações*;
5. *Planejamento de Respostas*: definir qual ação será tomada mediante os riscos identificados (ex: evitar, transferir, mitigar, aceitar);
6. *Controle e Monitoração*: garantir a execução do plano, identificar novos riscos, monitorar riscos residuais e realizar ajustes necessários.

73.1 Análise Qualitativa

A Análise Qualitativa de Riscos visa à detecção do impacto dos riscos identificados sobre os objetivos do projeto e sua probabilidade de ocorrência. Também classifica os riscos por prioridade, de acordo com os efeitos sobre os objetivos do projeto. Nesse processo, basta atribuir ao risco um valor alto, médio ou baixo (ou alguma combinação similar).

As ferramentas desse processo, de uma forma geral, visam a descobrir a probabilidade de um evento de risco e a determinar seu impacto caso venha a ocorrer. Nesse sentido, a ferramenta e técnica Probabilidade e Impacto do Risco atribui probabilidades aos eventos de risco identificados e calcula seu efeito sobre os objetivos do projeto. Esses métodos permitem determinar que riscos demandam o gerenciamento mais agressivo.

Outra ferramenta e técnica é a Matriz de classificação de riscos por probabilidade e impacto (PI) cujo objetivo é atribuir uma classificação genérica a cada um dos riscos identificados do projeto, geralmente expressa como alta, média ou baixa. Segundo o guide to PMBOK, os riscos elevados correspondem a uma situação vermelha, os médios a uma situação amarela e os baixos, verde. Esse tipo de classificação é denominado de escala ordinal, porque os valores são ordenados de forma descendente. Ao fim desse processo, portanto, os riscos podem ser enumerados por ordem de prioridade. A prioridade é estabelecida por meio da matriz PI.

A matriz de PI é determinada a partir da multiplicação de dois valores: o valor da probabilidade do risco vezes o valor do seu impacto (cada qual gerado na escala correspondente). A avaliação da probabilidade e impacto dos riscos costuma se dar por meio de técnicas de opinião especializada e entrevistas. Antes de se chegar a matriz PI, porém, são elaboradas a escala de probabilidades e a escala de impacto dos riscos. O objetivo dessas escalas é desenvolver medidas predefinidas que determinem que valor atribuir a determinado evento de risco.

73.2 Análise Quantitativa de Riscos

A Análise Quantitativa de Riscos avalia os impactos e quantifica a exposição do projeto aos riscos por meio da atribuição de probabilidades numéricas a cada um e aos seus impactos sobre os objetivos do projeto. Para tanto, são usadas técnicas como a simulação de Monte Carlo e a análise de decisões. Pode-se usar tanto a Análise Qualitativa quanto a Análise Quantitativa para avaliar os riscos, ou usar apenas uma das duas. Essa decisão será tomada com base na complexidade do projeto. As ferramentas e técnicas desse processo são:

- Entrevistas;
- Análise de Sensibilidade;
- Análise da árvore de decisões;
- Simulação.

Análise de Sensibilidade é um método de análise do possível impacto dos eventos de risco sobre o projeto e determinação de que evento (ou eventos) de risco tem maior impacto potencial. Também pode ser empregada para averiguar os níveis de tolerância aos riscos por parte dos stakeholders. Ela examina a extensão a qual a incerteza de cada elemento do projeto afeta o objetivo que está sendo examinado quando todos os outros elementos incertos são mantidos em seus valores iniciais.

A análise da árvore de decisões são diagramas que mostram a seqüência de decisões inter-relacionadas e os resultados esperados de acordo com a alternativa escolhida. Em geral existe mais de uma escolha ou opção disponível quando nos deparamos com uma decisão - ou, nesse caso, possíveis consequências de um evento de risco. As escolhas disponíveis são expostas em forma de árvore, começando à esquerda com a decisão relacionada ao risco, e ramificando para a

direita, com as possíveis consequências. Essas árvores costumam ser usada para os eventos de riscos associados a tempo e custo.

A simulação tem como principal representante a técnica de Monte Carlo. Ela ajuda a quantificar os riscos associados ao projeto com um todo. Os riscos identificados e seus possíveis impactos sobre os objetivos do projeto são examinados sob o prisma do projeto global. A técnica é usada para determinar os possíveis resultados pr meio da simulação do projeto diversas vezes seguidas.

Capítulo 74

Gerenciamento de Qualidade

Os processos para realização do Gerenciamento de Qualidade como definidos no PMBOK são:

1. *Planejamento*: definição dos padrões de qualidade relevantes (*Benchmarking*, *Diagrama Espinha de Peixe* (causa/efeito) e *Fluxograma Processo*);
2. *Garantia*: verificar relevância dos processos definidos;
3. *Controle*: analisar se os padrões estão sendo seguidos, identificar causas de desvios. Ferramentas muito utilizadas são a *Carta Controle*, *Diagrama de Pareto* (direcionar ações corretivas).

74.1 Técnicas de Planejamento da Qualidade

O processo de Planejamento da Qualidade visa ao cumprimento de padrões de qualidade para o projeto em questão, elaborando, para tanto, um plano. Segundo o guide to PMBOK a qualidade é planejada, não inspecionada. Portanto, uma saída desse processo é o plano de gerenciamento da qualidade que especifica como a política de qualidade será implementada pela equipe de gerência de projeto no decorrer das atividades. Para isso, várias ferramentas são empregadas. Os principais exemplos são: análise de custo/benefício, benchmarking, fluxograma, elaboração de experimentos e custo da qualidade.

74.1.1 Análise Custo/Benefício

Análise de custo/benefício aborda as compensações no custo da qualidade. É mais barato e eficiente prevenir problemas do que ter de perder tempo e dinheiro para corrigi-los mais tarde. O guide to PMBOK salienta que o custo básico do cumprimento dos requisitos de qualidade num projeto equivale às despesas implicadas na execução das atividades de gerenciamento da qualidade.

74.1.2 Benchmarking

É um processo de comparação de atividades anteriores similares às do projeto atual para obter um parâmetro de referência para a avaliação do desempenho. A comparação acaba sugerindo ideias para melhorar a qualidade no projeto atual. Por exemplo, se a impressora imprime 8 páginas por minuto e o projeto está considerando comprar uma que imprime 14 páginas por minuto, a referência é a impressora usada no projetos até então.

74.1.3 Fluxograma

São diagramas que mostram as etapas lógicas a serem executadas para atingir um objetivo. Também podem discriminar como se dão as relações entre os diversos elementos de um sistema. Os fluxogramas podem ajudar a identificar possibilidades de ocorrência de problemas no projeto. Há dois tipos mais comuns de fluxogramas: os diagramas de processo ou sistema e os diagramas de causa e efeito.

Os diagramas de causa e feito mostram a relação entre os efeitos dos problemas e suas causas, apresentando todas as possíveis causas primárias e secundárias dos problemas, bem como os efeitos de cada solução sugerida. Esse diagrama também é chamado de espinha de peixe ou de Ishikawa.

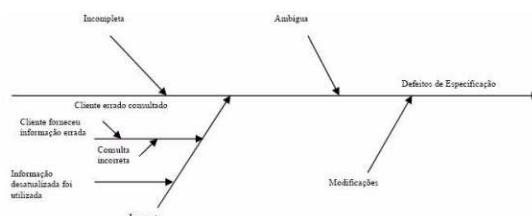


Figura 74.1: Diagrama Causa/Efeito

O fluxograma de sistema ou processos exibe as etapas lógicas necessárias para atingir determinado objetivo, bem como a relação entre diferentes elementos de um sistema. Esse é o mais conhecido; geralmente é construído com retângulos e paralelogramos que, ordenados numa seqüência lógica, permitem ramificações baseadas em sim e não

74.1.4 Elaboração de Experimentos

É uma técnica estatística que identifica os elementos ou variáveis que surtirão os efeitos mais profundos sobre os resultados globais do projeto. Ela é aplicada com mais freqüência ao produto do projeto, mas também pode ser utilizada nos processos de gerenciamento que examinam relações de compensação. O processo cria e realiza experimentos para chegar à solução ideal de um problema, a partir de um número limitado de exemplos.

74.1.5 Custo da Qualidade

O custo da qualidade diz respeito ao custo total da geração do produto ou serviço do projeto de acordo com os padrões de qualidade, e engloba todo o trabalho necessário para atender os requisitos do produto - quer tenha sido planejado ou não. Existem três custos associados ao custo da qualidade:

- Custo de Prevenção;
- Custo de Avaliação;
- Custo de falhas, divididos em custos internos e custos externos.

Os custos de prevenção são aqueles associados ao cumprimento dos requisitos dos clientes por meio da geração de produtos livres de defeitos; vêm a tona logo no começo do processo e compreendem fatores como planejamento da qualidade, treinamento, análise de projeto.

Os custos de avaliação são aqueles em que se incorre para examinar o produto ou processo e averiguar se determinados requisitos estão sendo atendidos. Podem incluir custos associados a elementos como inspeções e testes.

Custos de falhas são aqueles que ocorrem quando as coisas não se passam conforme o planejado. São dois tipos: internos (também chamado de custo de erro) e externos (também chamado de custo de defeito). O primeiro se impõe quando os requisitos do cliente deixam de ser satisfeitos com o produto ainda sob controle da organização. Os custos de falhas externas se dão quando o produto já chegou ao cliente e este entende que suas determinações ainda não foram atendidas.

Três teóricos em particular são responsáveis pela expansão do movimento pelo gerenciamento da qualidade e das teorias a respeito do custo da qualidade: Crosby, Juran e Deming. Crosby criou a prática do zero defeito - que significa, basicamente, acertar de primeira. A prevenção é a peça central da teoria de Crosby. Juran é conhecido pelo seu princípio da adequação ao uso - que significa, em termos simples, a satisfação ou superação das expectativas dos stakeholders clientes. A conformidade é peça central na teoria de Juran. Deming afirma que até 85% do custo da qualidade são problemas gerenciais. Quando a questão da qualidade chega à base, ao patamar dos operários, estes têm pouco controle a exercer. Ao lado dos três teóricos, pode-se citar como de importante relevância a Abordagem Kaizen que significa melhoria contínua.

74.2 Técnicas de Controle da Qualidade

O controle de qualidade preocupa-se particularmente com o monitoramento dos resultados do trabalho, a fim de verificar se estão cumprindo os padrões estabelecidos no plano de gerenciamento da qualidade. Existem várias ferramentas e técnicas nesse processo:

- Inspeção;
- Gráficos de Controle (Carta de Controle);
- Diagramas de Pareto;
- Diagramas de Dispersão;
- Amostragem Estatística;
- Elaboração de Fluxogramas (Diagramas de Causa e Efeito, Fluxograma de Sistemas);
- Análise das Tendências.

74.2.1 Gráficos de Controle

Medem os resultados dos processos ao longo do tempo e exibem-nos em formato gráfico; constituem uma maneira de mensurar variações a fim de averiguar se as variações dos processos estão ou não sob controle. Os gráficos de controle se baseiam na mediação de variações de amostras escolhidas e mensuradas das quais se determinam a média e o desvio-padrão. O controle da qualidade costuma ser mantido - ou considerado sob controle - dentro de mais ou menos três desvios-padrão. Em outras palavras, o Controle da Qualidade considera que o processo encontra-se sob controle quando se sabe que 99,73% das peças a ele submetidas vão recuar num intervalo aceitável da média. Caso de descubra uma peça fora desse intervalo, deve-se averiguar a necessidade de eventuais medidas corretivas.

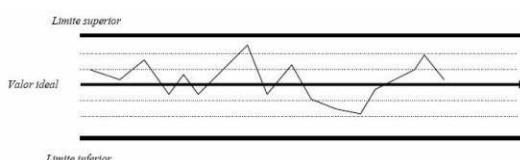


Figura 74.2: Gráfico de Controle

Existem dois tipos de gráficos de controle que são (i) Controle por variáveis e (ii) Controle por atributos.

74.2.2 Diagramas de Pareto

Baseia-se na regra 80/20 onde é afirmado que 80% dos problemas são causados por um pequeno número de causas, 20%. Os diagramas de Pareto assumem a forma de histogramas que classificam os fatores mais importantes (como atrasos, custos, defeitos ou outros) conforme a freqüência ao longo do tempo. A teoria prega que se obtém o máximo benefício se for dedicado a maior parte do tempo à correção dos problemas mais relevantes. Os problemas são classificados de acordo com a freqüência e a porcentagem de defeitos.

74.2.3 Diagramas de Dispersão

Os diagramas de dispersão usam duas variáveis, uma denominada variável independente, que é entrada, e outra chamada variável dependente, que é uma saída, e mostram a relação entre ambas, sob a forma de pontos no gráfico.

Capítulo 75

Gerenciamento da Comunicação

Segundo o PMBOK, o Gerenciamento da Comunicação é dividido nos seguintes processos:

1. *Planejamento*: definição das informações necessárias, objetos de comunicação e layouts de documentos;
2. *Distribuição das Informações*: disponibilizar as informações necessárias às partes envolvidas;
3. *Relato de Desempenho*: relatórios de situação, de progresso e previsões;
4. *Encerramento Administrativo*: encerramento formal do projeto. Divulgação dos resultados do projeto para patrocinadores e cliente, considerações finais, análise dos pontos fracos e fortes.

75.1 Um mais sobre Planejamento da Comunicação

Compreende a determinação das necessidades de comunicação dos stakeholders por meio da definição dos tipos de dados necessários e do formato em que serão transmitidos. Dois fatores influenciam esse processo: os requisitos de comunicação e a tecnologia envolvida. A saída desse processo é o plano de gerenciamento das comunicações. Ele estabelece como coletar, armazenar, arquivar e corrigir ou atualizar materiais já divulgados. Contém também uma lista das informações a serem distribuídas, o momento de sua comunicação e a estrutura de distribuição, além de descrever como os stakeholders podem ter acesso a informações do projeto antes das datas de divulgação. As informações a serem compartilhadas e os métodos de distribuição dependem das necessidades dos stakeholders, da complexidade do projeto e das políticas organizacionais.

Capítulo 76

Gerenciamento das Aquisições

A área de gerenciamento de comunicações é composta pelos seguintes processos:

1. *Planejamento*: identificação das necessidades que serão melhor atendidas por terceiros. Elaboração da *SOW* (*Specification of Work*);
2. *Preparação*: elaboração da documentação para licitações ou concorrência;
3. *Obtenção de Propostas*: seleção de fornecedores utilizando critérios definidos;
4. *Administração de Contratos*: acompanhar desempenho de fornecedores;
5. *Encerramento Contrato*: formalizar fim de contrato e arquivamento de informações.

76.1 Um pouco mais sobre Planejamento de Aquisições

O Planejamento de Aquisições é um processo que identifica os bens ou serviços a serem comprados fora da organização - o que inclui confirmar se devem adquirir os produtos ou serviços e, em caso afirmativo, quanto e quando. Duas ferramentas são usadas com freqüência nesse processo: análise ?make or buy? e seleção do tipo de contrato. A principal decisão a ser tomada por médio dessa análise é se vai ser melhor, em termos de custos para a organização, comprar ou fabricar os bens ou serviços necessários para ao projeto. Já a outra técnica trata da seleção do tipo de contrato usado para aquisição de produto ou serviço. Os tipos de contrato são:

- Contratos de preço fixo ou preço global: O contratado arca com a maior parte dos riscos;
- Contrato de custos reembolsáveis: O contratante arca com a maior parte dos riscos;
 - + Custo mais remuneração fixa

+ Custo mais remuneração de incentivo

- Contrato por tempo e material: Tipo híbrido. As duas partes dividem os riscos.

A principal saída desse processo é a Declaração de Trabalho(DT) ou Especificação de Trabalho (SOW - Specification of Work). A DT descreve o item a ser contratado com suficiente detalhamento para permitir que os potenciais fornecedores possam avaliar se são capazes de atender o edital. Deve conter no mínimo:

- Os objetivos do projeto;
- Uma descrição do trabalho do projeto;
- Especificações concisas do bem ou serviço necessários;
- Um cronograma do projeto.

Capítulo 77

Gerenciamento da Integração

Para assegurar que áreas estão bem coordenadas o PMBOK define a área de Gerenciamento da Integração, que é composta pelos seguintes processos:

1. *Desenvolvimento do Plano do Projeto*: reune todas as áreas em um documento chamado *Plano de Projeto*);
2. *Execução do Plano*: onde é gasto maior parte Orçamento;
3. *Controle Integrado de Mudanças*: analisar as possíveis mudanças, motivos e respectivos impactos ao longo do projeto. Os ajustes devem ser feitos de forma integrada.

77.1 Ferramentas de Apoio à Integração

Duas ferramentas são usadas nessa área: Gerenciamento de Valor Agregado (GVA ou EVM - Earned Value Management) e os softwares de gerenciamento de projetos. O GVA é uma metodologia de integração de projetos empregada para integrar processos e mensurar o desempenho do projeto ao longo do ciclo de vida do projeto. O MS Project é um exemplo de software de apoio à gerencia de projetos. Outro exemplo é o Primavera TeamPlayer.

O GVA pode ser empregado tanto no processo de controle de custos quanto no processo de planejamento. No último caso, o GVA, que visa à mensuração e ao relato do andamento do projeto, incorpora o cronograma do projeto, seu escopo e recursos como um todo para determinar se existem discrepâncias no projeto. No primeiro caso, OGVA apóia o cálculo das variações de custo.

Capítulo 78

Sobre os Ciclos do Projeto e Processos de Gerenciamento

Todos os projetos são divididos em fases e, sejam grandes ou pequenos, têm um ciclo de vida parecido. O conjunto de fases coletivas atravessadas pelo projeto é denominado **ciclo de vida do projeto**. O término de cada fase pode ser reconhecido pela apresentação de uma entrega específica (ou várias), marcando o final daquela etapa. **Entrega** é tudo o que deve ser produzido para que a fase ou o projeto sejam encerrados; são elementos tangíveis, que podem ser avaliados e comprovados com facilidade. As avaliações empreendidas ao final de cada estágio recebem vários nomes tais como: **saídas de fase, passagem de estágio ou pontos de encerramento**.

Como mencionado anteriormente, os projetos possuem um ciclo de vida parecido. Esse ciclo de vida pode ser organizado nos seguintes grupos processos: iniciação, planejamento, execução, controle e encerramento. Os cinco grupos são iterativos. A figura 78.1 apresenta os grupos de processos em um ciclo de vida de projeto típico.

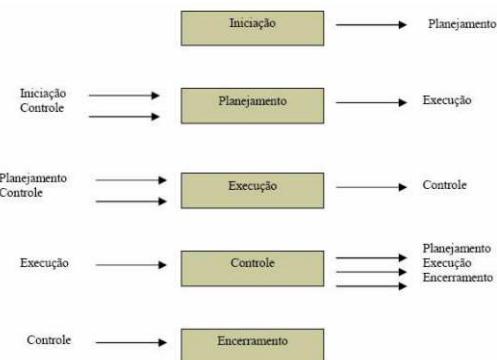


Figura 78.1: Ciclo de Vida do Projeto