Serviço de Pós-Graduação EESC/USP
EXEMPLAR REVISADO
Data de entrada no Serviço: 22 / 03 / 02
Ass.: <i>Guilherme</i>

Proposta de um Modelo para Redes de Dispositivos Telemáticos

Francisco José Monaco

Tese apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica

ORIENTADOR: Prof. Dr. Adilson Gonzaga

DEDALUS - Acervo - EESC

31100036911

São Carlos
2002

FOLHA DE JULGAMENTO

Candidato: Engenheiro **FRANCISCO JOSÉ MONACO**

Tese defendida e julgada em 20-03-2002 perante a Comissão Julgadora:

Prof. Dr. **ADILSON GONZAGA (Orientador)**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Dr. **EVANDRO LUIS LINHARI RODRIGUES**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Profa. Assoc. **LIRIA MATSUMOTO SATO**
(Escola Politécnica/USP)

APROVADO

Profa. Dra. **ROSELI DE DEUS LOPES**
(Escola Politécnica/USP)

APROVADO

Prof. Assoc. **LUIS CARLOS TREVELIN**
(Universidade Federal de São Carlos/UFSCar)

Aprovado

Prof. Doutor **MURILO ARAUJO ROMERO**
Coordenador do Programa de Pós-Graduação
em Engenharia Elétrica

JOSÉ CARLOS A. CINTRA
Presidente da Comissão de Pós-Graduação

Dedicatória

Ao Geraldo, meu pai e melhor amigo, a quem devo o que de mais importante aprendi e todas as boas coisas que conquistarei em toda minha vida.

À Heloísa, minha mãe, sábia conselheira, que abençoa nosso lar com o amor que nos conduz adiante a cada dia, nos incentivando e nos fazendo felizes.

À Fabíola, minha irmã, cujo carinho e amizade verdadeiros me ensinam a acreditar que toda luta vale a pena se os amigos estão por perto.

...palavras jamais serão suficientes para agradecer o carinho e a dedicação que de vocês tenho recebido ao longo de toda minha vida.

Dedico este trabalho também, com especial carinho, à Patrícia, minha companheira, com quem desejo compartilhar a alegria desta singela conquista e de tantas mais que alcançaremos juntos.

Ofereço-o, ainda, aos meus amigos de todos os dias, companheiros neste caminho de dificuldades e alegrias que é a vida, a Deus e seus mensageiros.

Ao Professor Adilson Gonzaga pela confiança em mim depositada para o desenvolvimento deste trabalho, e pela sua sábia orientação que me possibilitou realizá-lo.

Aos funcionários do Departamento de Engenharia Elétrica e da Escola de Engenharia de São Carlos, cuja solicitude a todos nós ofertada tem sempre superado suas atribuições profissionais.

Aos colegas e professores pela colaboração e amizade.

À Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP - pela concessão da bolsa de estudos que permitiu minha dedicação integral a este projeto.

Sumário

Lista de Figuras	vi
Lista de Símbolos	xiv
Resumo	xv
Abstract	xvi
1 Introdução	1
2 Revisão	4
2.1 Experimentos Precursores	4
2.2 Experimentos Científicos Pioneiros	6
2.3 O Estado da Arte	13
3 Investigações	16
3.1 Abordagem Convencional	16
3.1.1 Ensaios	19
3.2 Princípio de Funcionamento	20
3.3 Conclusões Preliminares	22
4 Um Modelo de Rede de Dispositivos Telemáticos	28
4.1 Contextualização	28
4.2 Elementos Básicos	31
4.3 Estrutura da Rede	36
4.4 Funcionamento Interno	44
4.5 Encaminhamento	47

4.6	Termos Utilizados na Descrição do Modelo	51
4.7	Comunicação	54
4.8	O Protocolo da RDI	60
5	Concepção de um Sistema-Piloto	64
5.1	Alguns Esboços	64
5.1.1	Mecanismos do Gerenciador de Interfaces	65
5.1.2	Mecanismos da Interface de Dispositivo	69
5.2	Ensaaios com Algumas Configurações	71
5.3	Algumas Conclusões	80
5.3.1	Limitações Anotadas	80
5.3.2	Um Protótipo Aprimorado	81
6	Implementação	87
6.1	O Gerenciador de Interfaces	89
6.1.1	Parâmetros do Servidor	90
6.1.2	Inicialização do Serviço TCP	92
6.1.3	O Monitor de Eventos	94
6.2	Estruturas de Dados	100
6.2.1	Procedimento de Conexão	102
6.2.2	Manipulação de Mensagens	110
6.2.3	Processamento de Mensagens	118
6.3	Alocação de Cadeias de Instâncias	122
6.3.1	Processamento de Entradas	130
6.3.2	Processamento de referências	133
6.3.3	Processamento de Saídas	135
6.3.4	Processamento de Reportes	141
6.3.5	Desestabelecimento de Cadeias	143
6.3.6	Envio de mensagens para a rede	145
6.3.7	Procedimento de Desconexão	146
6.3.8	O Processador	149
6.3.9	O Dispositivo de Sistema	151
6.3.10	O IDN Shell	160
6.3.11	O GI: Unindo os Componentes	162

6.4	A Interface de Dispositivo	168
6.4.1	Representação da ID	169
6.4.2	Inicialização da ID	172
6.4.3	A Linha da Rede	175
6.4.4	A Linha do Operador	177
7	Ensaio e Resultados	183
7.1	Instruções de Configuração e Inicialização do GI	183
7.2	Operação Elementar do Gerenciador de Interfaces	191
7.3	Utilização da API	204
7.4	Execução de Programas	213
7.5	Ciclos de Operação Concorrentes	226
7.6	Replicação de Recursos	232
7.7	Avaliação de Desempenho	237
8	Conclusões	243
8.1	Avaliação do Sistema	245
8.2	Síntese das Contribuições	248
	Referências Bibliográficas	250

Lista de Figuras

FIGURA 2.1	Consulta realizada ao servidor coke do RIT em 1998.	5
FIGURA 2.2	Aparato mecânico e interface de operação do Projeto Mercury . .	7
FIGURA 2.3	Interface do telescópio remoto da Universidade de Bradford . . .	9
FIGURA 2.4	Imagens do tele-robô australiano em operação.	10
FIGURA 3.1	Interação entre elementos no arranjo cliente-servidor	20
FIGURA 3.2	Esquema de telecomando utilizando o mecanismo Web-CGI . . .	22
FIGURA 3.3	Possibilidades habilitadas pela rede telemática	24
FIGURA 4.1	Um ambiente de dispositivos cooperativos em rede	30
FIGURA 4.2	Sistema interativo e sistema responsivo	31
FIGURA 4.3	Arranjo operador-mecanismo	31
FIGURA 4.4	O comportamento simétrico das interfaces de dispositivos	36
FIGURA 4.5	Algumas topologias de canais ponto-a-ponto	38
FIGURA 4.6	A rede real e a rede virtual no modelo RDI	39
FIGURA 4.7	Diagrama de estados da interface de dispositivo	44
FIGURA 4.8	Interfaces de dispositivos em uma composição em camadas	45
FIGURA 4.9	Fluxograma simplificado do algoritmo de encaminhamento	47
FIGURA 4.10	Programação dual implícito para o exemplo da Figura 4.8	49

FIGURA 4.11	Uma transação completa envolvendo três IDs	49
FIGURA 4.12	Arquitetura da Internet	56
FIGURA 4.13	Formato das mensagens do protocolo de comunicação da RDI . . .	60
FIGURA 5.1	Estados das representativas na versão preliminar	66
FIGURA 5.2	Potenciais situações de conflito na rede	67
FIGURA 5.3	Algoritmo de encaminhamento (versão 1)	70
FIGURA 5.4	Saída inicial do módulo RDCS	71
FIGURA 5.5	Processo de conexão e desconexão no módulo RDCS	72
FIGURA 5.6	Uma ID como interface de usuário gráfica	74
FIGURA 5.7	Uma aplicação simples em RDI	74
FIGURA 5.8	Uma mini-aplicação ID embutida em uma página Web	75
FIGURA 5.9	Um dispositivo complexo via interface Web	76
FIGURA 5.10	Um exemplo de programação na RDI	77
FIGURA 5.11	Interface gráfica para configuração da porta RS-232	78
FIGURA 5.12	Instruções enviadas para o módulo ADAM	78
FIGURA 5.13	Tabela de tradução para o experimento com LEDs	79
FIGURA 5.14	Espaço dual de representação interna no GI	86
FIGURA 6.1	Parâmetros operacionais do servidor	90
FIGURA 6.2	Possíveis eventos na rede	91
FIGURA 6.3	Configuração do servidor	92
FIGURA 6.4	Ativação do servidor	93
FIGURA 6.5	À espera de eventos na rede	97
FIGURA 6.6	Detectando requisições de conexão	98

FIGURA 6.7	Atendendo pedidos dos clientes	99
FIGURA 6.8	Atendimento de mensagens e desconexão	99
FIGURA 6.9	O manipulador de interfaces	101
FIGURA 6.10	Lista duplamente encadeada utilizada no programa	101
FIGURA 6.11	Macros para manipulação da lista de dispositivos	102
FIGURA 6.12	Criação e liberação de uma lista	103
FIGURA 6.13	Adição de elementos na lista	104
FIGURA 6.14	Remoção e localização de elementos na lista	105
FIGURA 6.15	Inicialização do manipulador	106
FIGURA 6.16	O dispositivo	107
FIGURA 6.17	O conector	107
FIGURA 6.18	Procedimento de conexão (parte 1)	109
FIGURA 6.19	Procedimento de conexão (parte 2)	110
FIGURA 6.20	Código de status de retorno IDNP/1.0	113
FIGURA 6.21	CMI estabelecido pelo IDNP/1.0	113
FIGURA 6.22	CIEA do dispositivo de sistema estabelecido pelo IDNP/1.0 . . .	113
FIGURA 6.23	Declarações do IDNP	115
FIGURA 6.24	A mensagem IDNP como uma estrutura	115
FIGURA 6.25	Verificação de mensagens	116
FIGURA 6.26	Tratamento de entradas (parte 1)	118
FIGURA 6.27	Recupera o dispositivo dado o descritor	119
FIGURA 6.28	Tratamento de entradas (parte 2)	120
FIGURA 6.29	Recupera o conector dado o nome	120
FIGURA 6.30	Dispositivo com interrupção pendente	121

FIGURA 6.31 Tratamento de entradas (parte 3)	122
FIGURA 6.32 A instância	123
FIGURA 6.33 Alocação de instâncias (parte 1)	124
FIGURA 6.34 Formação de uma cadeia (intermediária)	124
FIGURA 6.35 Formação de uma cadeia (completa)	125
FIGURA 6.36 Alocação de instâncias (parte 2)	128
FIGURA 6.37 Alocação de instâncias (parte 3)	129
FIGURA 6.38 Processamento de entradas (parte 1)	130
FIGURA 6.39 Processamento de entradas (parte 2)	131
FIGURA 6.40 Processamento de entradas (parte 3)	132
FIGURA 6.41 Processamento de referências (parte 1)	134
FIGURA 6.42 Processamento de referências (parte 2)	135
FIGURA 6.43 Processamento de saídas (parte 1)	136
FIGURA 6.44 Processamento de saídas (parte 2)	139
FIGURA 6.45 Transferência da saída (parte 1)	139
FIGURA 6.46 Transferência da saída (parte 2)	140
FIGURA 6.47 Processamento de reportes (parte 1)	142
FIGURA 6.48 Localização de cadeias obsoletas	144
FIGURA 6.49 Desestabelecimento de cadeias	144
FIGURA 6.50 Transmissão de mensagens para a ID	145
FIGURA 6.51 Procedimento de desconexão (parte 1)	147
FIGURA 6.52 Procedimento de desconexão (parte 2)	148
FIGURA 6.53 Chamada do despachador	149
FIGURA 6.54 O despachador A	150

FIGURA 6.55 Máscaras de bits utilizadas no despachador A	151
FIGURA 6.56 O dispositivo de sistema (parte 1)	153
FIGURA 6.57 O processo do GI: linhas do servidor e do sistema	153
FIGURA 6.58 O dispositivo de sistema (parte 2)	154
FIGURA 6.59 A linha de execução do sistema (parte 1)	155
FIGURA 6.60 A linha de execução do sistema (parte 2)	156
FIGURA 6.61 Sincronização entre a linha do servidor e do sistema	157
FIGURA 6.62 Processamento de comandos do sistema (parte 1)	158
FIGURA 6.63 Processamento de comandos do sistema (parte 2)	158
FIGURA 6.64 Vetor de comandos do dispositivo de sistema	159
FIGURA 6.65 Processamento de comandos do sistema (parte 3)	159
FIGURA 6.66 Implementação de um comando de sistema (implicit)	160
FIGURA 6.67 Inicialização do IDN shell	162
FIGURA 6.68 IDN : programa principal (parte 1)	163
FIGURA 6.69 IDN : programa principal (parte 2)	164
FIGURA 6.70 IDN : programa principal (parte 3)	165
FIGURA 6.71 IDN : programa principal (parte 4)	166
FIGURA 6.72 IDN : programa principal (parte 5)	167
FIGURA 6.73 Representação esquemática do sistema IDN	167
FIGURA 6.74 Dois canais de entradas em uma ID	168
FIGURA 6.75 Estrutura da API para interface de dispositivos em RDI	170
FIGURA 6.76 Estrutura da ID	171
FIGURA 6.77 Configuração inicial da ID	173
FIGURA 6.78 Ativação da ID	174

FIGURA 6.79	O laço da linha principal (parte 1)	175
FIGURA 6.80	O laço da linha principal (parte 2)	176
FIGURA 6.81	O laço do operador (parte 1)	178
FIGURA 6.82	Vetor de requisições	178
FIGURA 6.83	O laço do operador (parte 2)	179
FIGURA 6.84	“ <i>Dummy Functions</i> ” fornecidas pela API	180
FIGURA 6.85	Funções suplementares para reconfiguração da ID	180
FIGURA 6.86	Funções suplementares para composição de mensagens	181
FIGURA 6.87	Macros suplementares para comparação de mensagens	182
FIGURA 7.1	Makefile para compilação automática do software	184
FIGURA 7.2	Invocação do módulo <i>idn</i> no terminal	185
FIGURA 7.3	Iniciando o módulo <i>idn</i>	186
FIGURA 7.4	Conectando-se ao IDN shell	187
FIGURA 7.5	Formato de saída do comando <i>lc</i>	187
FIGURA 7.6	Finalizando o GI	188
FIGURA 7.7	Conexões ativas (<i>netstat</i>)	190
FIGURA 7.8	Redução do intervalo de <i>TIME_WAIT</i> (linux 2.2.18)	191
FIGURA 7.9	Simulação de uma ID via aplicação <i>telnet</i>	192
FIGURA 7.10	Exemplo A (<i>idn</i> , parte 1)	192
FIGURA 7.11	Exemplo A (<i>IDN shell</i> , parte 1, condição inicial)	193
FIGURA 7.12	Exemplo A (<i>IDN shell</i> , parte 1, cadeia estabelecida)	194
FIGURA 7.13	Exemplo A (<i>IDN shell</i> , parte 1, manipulador completo)	194
FIGURA 7.14	Exemplo A (<i>idn</i> , parte 2)	196
FIGURA 7.15	Exemplo A (<i>idn</i> , parte 3)	197

FIGURA 7.16 Exemplo A (<i>idn</i> , parte 4)	199
FIGURA 7.17 Exemplo A (<i>idn</i> , parte 5)	199
FIGURA 7.18 Exemplo A (<i>idn</i> , parte 6)	201
FIGURA 7.19 Exemplo A (<i>idn</i> , parte 7)	202
FIGURA 7.20 Exemplo A (<i>idn</i> , parte 8)	203
FIGURA 7.21 Implementação mínima de uma ID utilizando a API	205
FIGURA 7.22 Inicialização de <i>example_1</i>	206
FIGURA 7.23 Registros de <i>idn</i> para a invocação de <i>example_1</i>	207
FIGURA 7.24 Um ciclo de operação (<i>example_1</i>)	208
FIGURA 7.25 Utilizando o vetor de configurações	209
FIGURA 7.26 Implementação da funcionalidade da ID	210
FIGURA 7.27 Saída de <i>example_3</i>	211
FIGURA 7.28 Implementando a aplicação	212
FIGURA 7.29 Uma RDI com duas IDs	213
FIGURA 7.30 Listagem de <i>op.c</i> (parte 1)	214
FIGURA 7.31 Listagem de <i>op.c</i> (parte 2)	215
FIGURA 7.32 Listagem de <i>dev.c</i> (parte 1)	216
FIGURA 7.33 Listagem de <i>dev.c</i> (parte 2)	217
FIGURA 7.34 Comunicação entre <i>op</i> e <i>dev</i>	218
FIGURA 7.35 Listagem de <i>proxy.c</i> (parte 1)	219
FIGURA 7.36 Listagem de <i>proxy.c</i> (parte 2)	220
FIGURA 7.37 Invocação de <i>proxy</i>	221
FIGURA 7.38 Estabelecimento da cadeia $A - B - C$	222
FIGURA 7.39 Símbolos que representam eventos do despachador	222

FIGURA 7.40 Exemplo B	223
FIGURA 7.41 Execução da cadeia A-B-C	224
FIGURA 7.42 Quebra abrupta de uma cadeia.	225
FIGURA 7.43 Algoritmo do despachador	227
FIGURA 7.44 Possíveis caminhos para dois ciclos concorrentes	228
FIGURA 7.45 Manipulação de requisições pendentes (parte 1)	229
FIGURA 7.46 Manipulação de requisições pendentes (parte 2)	230
FIGURA 7.47 Possíveis caminhos para três ciclos concorrentes	231
FIGURA 7.48 Operações reentrantes (parte 1)	233
FIGURA 7.49 Operações reentrantes (parte 2)	234
FIGURA 7.50 Configuração de réplicas	235
FIGURA 7.51 Transferência de dispositivos	236
FIGURA 7.52 Acessos simultâneos a um recurso replicado	237
FIGURA 7.53 Utilização de recursos da plataforma	238
FIGURA 7.54 Estações que tomaram parte nos ensaios	241
FIGURA 7.55 Valores do RTT para diversos caminhos na Internet	241
FIGURA 7.56 Valores de RTT entre duas estações durante 30 dias	242
FIGURA 7.57 Desempenho da comunicação através da Internet	242

Lista de Símbolos

RDI	Rede de dispositivos interativos
ID	Interface de dispositivo
GI	Gerenciador de interfaces
CID	Conjunto de instruções do dispositivo (CMI+CIEA+CIED)
CMI	Conjunto mínimo de instruções obrigatórias em qualquer DI
CIEA	Conjunto de instruções específicas obrigatórias para determinada aplicação
CIED	Conjunto de instruções específicas do dispositivo
conector	representação de uma ou mais DIs no interior do GI
instância	avatar de um conector no mecanismo de comunicação interno do GI
ciclo de operação	sucessão de eventos relativos a uma interação entre duas DIs
interrupção	mensagem que inicia um ciclo de operação
retorno	mensagem resultante de um processamento
programa	configuração do GI que organiza um conjunto de conectores em seqüência
cadeia	conjunto de instâncias seqüencialmente conectadas conforme um programa
requisição	mensagem trocada entre duas DIs na fase ascendente do ciclo de operação
resposta	mensagem trocada entre duas DIs na fase descendente do ciclo de operação
referência	emissão de uma requisição
reporte	emissão de uma resposta
A:B:C	programa envolvendo três DIs nos conectores <i>A</i> , <i>B</i> e <i>C</i>
A-B-C	cadeia envolvendo três DIs nos conectores <i>A</i> , <i>B</i> , <i>C</i>

Resumo

Avanços técnicos na integração dos sistemas de comunicação e computação têm revelado ao interesse científico as potencialidades das redes de dispositivos telemáticos — nas quais interagem equipamentos dotados de capacidade de transmissão e processamento de informação — em áreas que cobrem desde o ensino à distância, até telemedicina, automação remota e tele-instrumentação científica. Especialmente em virtude do reconhecimento das destacadas facilidades da Internet, dentre as aplicações mais visadas encontra-se o compartilhamento de recursos (dados e instrumentos) em escala global.

Em um terreno ainda predominantemente marcado pelo empirismo característico das frentes de exploração recentes, este estudo investiga os fundamentos da operação à distância em redes telemáticas, através da identificação das atuais tendências no emergente campo do comando remoto via Internet, examinando os aspectos tecnológicos no que concerne ao seu desenvolvimento e aplicações. A partir de uma extensa revisão do assunto, abarcando tanto o cenário acadêmico quanto a iniciativa industrial, o trabalho apresenta uma abordagem alternativa em relação às técnicas atualmente empregadas.

Mediante a construção de um protótipo, o estudo introduz uma arquitetura experimental distribuída, baseada no conceito de ambiente virtual cooperativo, através da qual dispositivos tele-interativos exportam sua funcionalidade para os demais elementos na rede. Então, através de uma série de ensaios e avaliações, discutem-se as aplicações e limitações do mecanismo, bem como dos recursos tecnológicos atualmente disponíveis para a construção de efetivos sistemas de compartilhamento de dispositivos telemáticos através da Internet.

Abstract

Technical advances in the integration of computer and communication systems have drawn to the attention of scientific interest the potentialities of the telematic-device networks — where pieces of equipment endowed with information processing and transmission capabilities interact — in areas ranging from distance learning to telemedicine, remote automation and scientific tele-instrumentation. Particularly with the perception of the benefits brought about by the Internet, one of the most sought after applications is the global-wide resource (data and instruments) sharing systems.

In a field still distinguished by the empirical practices that are typical in recent exploration fronts, this study investigates the foundations of distance operation in telematic networks by identifying the current trends in the emerging area of Internet-based remote command, examining the technological aspects that concerns to its development and application. Starting with an extensive review on the subject, covering both academic and industrial initiatives, this work introduces an alternative approach to the techniques currently employed.

By means of a prototype, the study presents an experimental distributed architecture based on the concept of virtual cooperative environment, through which tele-interactive devices export their functionality to the other elements on the network. Then, through a set of tests and evaluations, applications and limitations of the mechanism are discussed, as well as the technological resources currently available for the development of effective telematic-device sharing systems over the Internet.

Capítulo 1

Introdução

Durante os últimos 50 anos, exaustivos esforços têm sido dedicados ao desenvolvimento das tecnologias que habilitam o acesso efetivo a recursos distantes nos diversos domínios de aplicação onde têm sido pesquisadas. De fato, decorre da inerente multidisciplinaridade desse campo não ser tarefa fácil discorrer a seu respeito no âmbito técnico em meio às incongruentes definições e terminologias que o permeiam na vasta extensão que o terreno abrange. Não obstante, é possível ainda assim distinguir alguns passos importantes na sua evolução ao longo desse período e as diversas facilidades por ela habilitadas.

Intensivamente empregados desde as primeiras décadas do século XX em ambientes científicos e industriais, especialmente em instalações nucleares e militares, para evitar a perigosa exposição de pessoas a substâncias químicas ou radioativas, os sistemas *mecânicos* são descendentes distantes das pinças manuais utilizadas na antiguidade pelos ferreiros para manusear peças aquecidas [1]. A principal aplicação desses equipamentos, puramente mecânicos ou dotados de componentes hidráulicos ou pneumáticos, tem sido a manipulação de objetos à distância, particularmente quando a intervenção direta não é desejável ou possível em função de restrições técnicas ou relativas à segurança — de onde a designação “telem manipuladores”, como são referenciados na vasta literatura dedicada a área, ou “telechirics” [2] *apud* [1], uma alternativa àquela etimologicamente imperfeita palavra greco-latina.

Uma outra classe de aplicações é trazida pelos sistemas *eletrônicos*, que estendem o

conceito para além da manipulação mecânica até o genérico acionamento de dispositivos. A tecnologia dissemina-se sob um amplo leque de utilizações, incluindo o maquinário industrial e também os equipamentos domésticos orientados ao conforto. Iniciando com os atuadores elétricos motorizados, beneficia-se então dos avanços seguintes que permitem a comunicação a longas distâncias. A expressão “tele”, que a partir do evento tem sido algumas vezes estritamente associada com as telecomunicações, produz o assim prefixado termo “telecomando”, que em geral refere-se à transmissão de sinais de controle através daqueles sistemas [3].

Apenas nos anos 70 os sistemas *computadorizados* de operação à distância tornaram-se efetivos, introduzindo a área na era da automação. Dispositivos mecânicos programáveis conduziram então à tele-robótica, aos dispositivos móveis remotamente operados, também denominados “teleoperadores” em alguns contextos [4] *apud* [1] e ainda ao controle remoto — aqui, no significado específico dentro da disciplina da automática, posto que este é também o nome comercial pelo qual fabricantes promovem a facilidade encontrada em aplicações domésticas modernas.

Nos dias atuais, outra vez uma emergente tecnologia surge com possibilidades revolucionárias para o campo do acesso à distância. Valendo-se do consórcio de sistemas de computação e comunicação, as aplicações telemáticas entram em evidência no início deste século estabelecendo-se dentre as mais promissoras conquistas no campo da chamada área IC (Informação e Comunicação).

As redes de computadores, contudo, sejam locais ou de longa distância, não obstante prestem-se ao exemplo mais evidente da atualidade, não constituem a única realização dentro desse universo de possibilidades — daí a referência, neste estudo, às *redes telemáticas* como uma designação mais geral e abrangente que inclui todos os sistemas onde elementos interagem através das capacidades integradas de processamento e transmissão de informação. Cabem nessa categoria, portanto, uma crescente gama de conceitos emergentes, tais como as redes de dispositivos inteligentes [5], como em sistemas de automação residencial e mecanismos distribuídos de sensores e atuadores autônomos em plantas de controle industrial. Tais avanços orientam-se com os rumos da tecnologia dos sistemas de processamento embutidos em direção à ubiqüidade computacional de equipamentos e sistemas de

serviços, aparecendo, por exemplo, na forma dos sistemas permeáveis¹, assim nominados a fim de salientar a desintegração dos dispositivos auto-suficientes em favor da crescente descentralização da funcionalidade entre unidades cooperantes que se alastram pelo ambiente à maneira de um fluido permeando o meio.

Assim, seguindo-se aos antecessores mecânicos, eletrônicos e computadorizados, os sistemas telemáticos que habilitam o acesso a recursos distantes — seja essa distância espacial, temporal, cultural ou de qualquer outra natureza — suscitam a atenção para novas possibilidades que se agregam às já conquistadas através das técnicas anteriores. Desde os terminais de informação interligados e os caixas eletrônicos à vasta miríade de serviços interativos disponíveis na Internet, as potencialidades da tecnologia telemática têm sido prontamente percebidas em praticamente todas as esferas de aplicação.

Com efeito, o advento das redes globais, com a Internet mundial em destaque, tem estimulado o recente interesse no aproveitamento das potencialidades das redes telemáticas em promissoras possibilidades que incluem o *compartilhamento de recursos em âmbito global*, não apenas na sua aplicação no âmbito da comunicação e computação de dados, mas na integração com sistemas convencionais existentes e a serem concebidos, provendo a interação com dispositivos genéricos — de equipamentos domésticos a instrumentos científicos, de serviços de utilidade pública à tele-robótica industrial.

Inserido nesse contexto, e motivado pelas potencialidades que se abrem nesse emergente fronte tecnológico, este trabalho propõe-se a explorar as possibilidades da tecnologia em relevo, suas características e limitações. Principiando por uma revisão dos avanços em uma área ainda caracterizada pelo empirismo próprio das vertentes recentes, o estudo tem como objetivo ensaiar, sob a perspectiva da engenharia, um passo adiante na efetiva exploração da tecnologia e aplicações das *redes de dispositivos telemáticos*.

¹Tradução literal para o termo *pervasive systems*.

Capítulo 2

Revisão

2.1 Experimentos Precursores

Assim como tantas outras interessantes criações da tecnologia Internet, os primeiros exemplos de operação à distância através de redes telemáticas são oriundos de iniciativas despretensiosas conduzidas por investigadores voluntários na busca de soluções para problemas triviais ou talvez na invenção de novas fórmulas de entretenimento. Dentre os primeiros experimentos de que se têm notícia encontram-se os lendários servidores coke (*coke servers*), que surgiram inicialmente nos anos 70, quando estudantes da Universidade de Carnegie Mellon ligaram a máquina de refrigerantes da sala de computadores à Internet [6], de tal sorte que qualquer usuário pudesse ler os valores registrados pelos sensores eletrônicos a ela acoplados¹, a fim de verificar remotamente a disponibilidade e a temperatura da bebida no interior do equipamento — evitando, assim, caminhadas frustradas através das escadas do edifício.

O servidor coke desenvolvido no Instituto de Tecnologia de Rochester [8] é ainda disponível ao acesso público através da Internet, e o resultado de uma consulta² a ele realizada durante esta pesquisa pode ser visto na Figura 2.1. Já o projeto do servidor do Departamento de Computação da Universidade de Copenhague, incluindo descrição do aparato

¹A leitura podia ser conhecida através da Internet por meio de um programa *finger* [7] adaptado.

²Consulta realizada em julho de 1998.

```

$ finger drink@csh.rit.edu
[csh.rit.edu]
Login name: admin                      In real life: drink admin stuff
Directory: /usr/local/libdata/drink    Shell: /bin/sh
Never logged in.
No Plan.

Login name: drink                      In real life: Drink Account
Directory: /u2/u0/drink                Shell: /usr/local/bin/drink
Last login Thu Aug 14, 1997 on ttyp6 from d-ma-kingston-13
Plan:

                                Balance: $ 0.00

1) JOLT!!           $ 0.50  Empty  (0/44)
2) Mountain Dew!!  $ 0.50  Empty  (0/44)
3) Mystery Slot?!  $ 0.50  Empty  (0/44)
4) Diet Stuff      $ 0.50  Empty  (0/44)
5) Coke Classic    $ 0.50  Empty  (0/44)

```

	Drink	3+ hrs	1-3 hrs	0-
1 hrs	Total			
	:::::	:::::	:::::	:::::
	JOLT!!	0	0	0
	Mountain Dew!!	0	0	0
	Mystery Slot?!	0	0	0
	Diet Stuff	0	0	0
	Coke Classic	0	0	0

```

+-----+
| Computer Science House |
|      Drink Machine     |
+-----+-----+-----+-----+
| JO | Mo | My | Di | Co |
| 0 | 0 | 0 | 0 | 0 |
|$.50|$.50|$.50|$.50|$.50|
+-----+-----+-----+-----+
|               +-----+ + |
|               | +---+ ' | | | | | |
|               ||__| |__| | |
|               |  ||  |  | |
|               | +---+ . | |
|               +-----+ + |
+-----+-----+-----+-----+

```

Figura 2.1: Consulta realizada ao servidor coke do RIT em 1998.

O sistema, bastante simples, consiste em um programa *finger* adaptado. Ao envio de uma requisição de consulta, o servidor responde com os dados coletados do dispositivo sensor e informa o número de frascos de cada tipo de refrigerante, a quanto tempo estes estão sendo resfriados no interior do equipamento e o preço da bebida. O serviço continua disponível por motivações históricas, mas presumivelmente não é mais operacional, visto que consultas repetidas ao servidor têm reportado sempre a indisponibilidade de refrigerantes.

eletrônico, rotinas de software, notas de teoria e implementação, pode ser conhecido em detalhes a partir da sua página Web [9]. Outros exemplos de servidores coke incluem os protótipos da Universidade de Missouri-Rolla [10], da Universidade do Oeste Australiano [11], do Instituto de Tecnologia da Califórnia [12], do Instituto de Tecnologia de Massachusetts [13], da Universidade de Colúmbia [14], dentre outros.

Esse inusitado experimento adquiriu notoriedade e logo passou a ser reproduzido em versões alternativas utilizando outros dispositivos “não convencionais”, tais como a folclórica cafeteira elétrica da Universidade de Cambridge [15], dezenas de sensores de umidade e temperatura ambiente e até mesmo refrigeradores e torradeiras que, a exemplo da robótica recreativa, terminaram também por alcançar o status de experimento clássico para estudantes de engenharia e computação [16].

Um exemplo particularmente curioso é conhecido como *Random Hall Laundry* [17], um serviço criado pelos alunos do Instituto de Tecnologia de Massachusetts, que ligaram as máquinas da lavanderia do alojamento estudantil à Internet para facilitar o cotidiano de seus usuários, conforme relata o coordenador do projeto em resposta a um e-mail enviado-lhe pelo autor, em 1997, durante o desenvolvimento desta pesquisa:

```
''Quando precisamos limpar as roupas podemos visitar a página Web  
para verificar se há alguma lavadora disponível, antes de descer as  
escadas para a lavanderia. Podemos também conferir quando a máquina  
terminou o serviço.'' (Schuartz, quark@mit.edu)
```

Não obstante a natureza pitoresca e despretensiosa, tais iniciativas têm o mérito do reconhecimento das possibilidades da aplicação, evidenciando suas potencialidades e, assim, de fato inspiraram o desenvolvimento subsequente, introduzindo o conceito ao interesse científico.

2.2 Experimentos Científicos Pioneiros

Desde os primeiros ensaios até os protótipos operacionais atuais, a idéia de compartilhar a utilização de equipamentos através da Internet tem ganhado consistência e motivado trabalhos pioneiros de pesquisa, tais como o Projeto Mercury [18] na Universidade da Califórnia do Sul, cujo objetivo propunha demonstrar a viabilidade da Internet como canal

Figura 2.2: Aparato mecânico e interface de operação do Projeto Mercury

O diagrama à esquerda (a) mostra esquematicamente o braço-robô IBM SR5427 montado sobre um tanque de areia semi-circular do qual objetos podiam ser desenterrados mediante um jato de ar soprado através do bocal acoplado ao dispositivo. A interface de operação via Web à direita (b) fornece imagens capturadas com uma câmera CCD ligada ao dispositivo.

simples e econômico para oferecer acesso a equipamentos comandados remotamente.

Freqüentemente lembrado como referência clássica na recente literatura que tange à jovem vertente de investigação, o então inovador experimento apresentava um robô escavador (Figura 2.2) controlado através da Web, cujo acesso era disponibilizado ao público. A média de acessos durante os seis primeiros meses de operação calculada em 430 usuários por dia (o que constitui uma soma expressiva em vista da dimensão da Web na época) e a estimativa de que o Projeto Mercury tenha sido responsável por cerca de metade de todos os acessos aos servidores da universidade, foram considerados indicativos do sucesso do ensaio [19].

Motivados pelo destaque conferido pelo público à realização através da mídia científica e em geral, seus responsáveis prosseguiram com novas versões do sistema, apresentando a visita remota ao herbário robótico do premiado projeto Telegarden [20], hoje em exposição permanente no Centro de Artes Eletrônicas na Áustria, e também o inusitado experimento com “tele-epistemologia” através do ShadowServer [21].

Reconhecendo a influência daqueles experimentos, os autores do projeto DIGIMUSE convidam os usuários da Web a uma visita virtual à Galeria de Artes da Universida-

de da Califórnia do Sul através das lentes de um conjunto de câmeras estereoscópicas tele-robóticas que podem ser interativamente posicionadas em torno de esculturas reais, oferecendo a oportunidade de visualização tridimensional para estudantes de artes em todo o mundo [22].

A possibilidade da utilização compartilhada de recursos únicos ou raros, seja em virtude do seu elevado custo, especialização ou outras peculiaridades que dificultem sua replicação em larga escala³, tem sido de fato um notado estímulo para essa segunda fase da exploração da tecnologia telemática em sistemas de operação à distância.

A motivação resulta da observação de que, freqüentemente, equipamentos científicos e industriais de aplicação muito específica tendem a alcançar custos extremamente elevados. Somem-se a esse obstáculo as dificuldades inerentes à instalação e, principalmente, à manutenção desses recursos, e ainda a necessidade da disponibilidade de operadores capacitados à sua utilização adequada, treinamento e suporte, e revelam-se as razões pelas quais a adoção generalizada de tais instrumentos não é uma realidade atual. Em verdade, não obstante as vantagens permitidas pelo estágio de desenvolvimento em muitos setores, constata-se o panorama onde o acesso aos avanços tecnológicos permanece ainda restrito a uma parcela reduzida de usuários potenciais que dele poderiam fazer uso conveniente, especialmente na área pública e em utilização no âmbito social. Juntamente com políticas governamentais que busquem mitigar tais discrepâncias, as redes telemáticas têm sido consideradas como um importante instrumento de ação prática nessa esfera.

Por meio do compartilhamento do acesso em escala global, um conceito que evolui velozmente da condição de utopia tecnológica a uma possibilidade em vias de realização, redes de dispositivos podem fornecer a infra-estrutura para sistemas que permitam o emprego de recursos raros a instituições que não possuam meios de adquirir ou manter similares próprios, servindo por exemplo para fins de treinamento, estudo e pesquisa.

O telescópio robótico da Universidade de Bradford [23], que atende a essa proposta, foi ligado a um sistema de comando remoto via Internet para o acesso de cientistas, estudantes e amadores em todo o mundo, oferecendo-lhes ainda a integração com outros recursos, tais

³Por exemplo, laboratórios de medidas ambientais, geológicas ou astronômicas, únicos em função da sua localização geográfica.

Figura 2.3: Interface do telescópio remoto da Universidade de Bradford

Através da interface na Web, cientistas, estudantes e amadores têm acesso ao telescópio robótico, um refletor newtoniano de 46cm localizado em West Workshire. Com o auxílio da base de dados astronômicos e dispositivos de sensoriamento ambiental interligados, o sistema autônomo gerencia as prioridades das observações, adquire e preprocessa as imagens, que são disponibilizadas no servidor.

Figura 2.4: Imagens do tele-robô australiano em operação.

O braço-robô industrial ASEA de 6 eixos pode ser remotamente operado através da Web para empilhar blocos de madeira sobre uma plataforma horizontal, com o auxílio das imagens geradas à cada movimento por câmeras posicionadas em direções ortogonais. Estas cenas foram obtidas mediante uma tentativa de operação realizada pelo autor em 1998, após haverem todos os blocos sido inabilmente derrubados fora da mesa.

como detalhadas instruções de operação, lições de astronomia e boletins técnicos, todos acessíveis através da mesma interface na Web⁴ (Figura 2.3).

O conceito encontra também aplicação na questão da aquisição de equipamentos de alto custo que, em função da tecnologia envolvida, tendem à obsolescência em pouco espaço de tempo, ou igualmente quando seu emprego é infreqüente, tornando assim o investimento desencorajador. Nesses casos, a possibilidade de as várias partes interessadas se consorciarem para a aquisição e atualização periódica de sistemas de última geração propicia grande economia, evitando a duplicação desnecessária de recursos e favorecendo a eficiência da sua utilização.

Com efeito, demonstrar tais possibilidades fora a motivação do projeto *Austrália Tele-Robot* [24] reconhecido pela ainda restrita bibliografia da área como uns dos primeiros trabalhos a explorar a tele-robótica industrial via Internet. No estudo, os autores observam que a maioria das aplicações em teleoperação de robôs tem sido empregada como um recurso extremo, freqüentemente em missões críticas em ambiente espacial, submarino ou radioativo, onde não são praticáveis as alternativas convencionais. Tais sistemas se caracterizam pela elevada sofisticação e tecnologia dedicada de alto desempenho, de onde decorre que considerações econômicas não são em geral prioritárias.

⁴O endereço do telescópio é <http://www.telescope.org>.

O projeto do tele-robô australiano, que admitindo “o importante estímulo da cafeteria de Cambridge” [25], oferece ao público, através da Internet, os controles de um robô industrial (Figura 2.4), e procura investigar a viabilidade de um sistema de baixo custo e simples operação como forma de reverter o reduzido índice da utilização desse tipo de equipamento, o qual se apresenta na razão 1/10 em relação as previsões estatísticas de então.

A complexidade de operação de instrumentos de elevada sofisticação é igualmente um dos constrangimentos que, em muitos casos, se convertem em fatores impeditivos à adoção de novas tecnologias em ambientes não especializados ou, não menos inconvenientemente, resulta na sua utilização inadequada. Este o cenário explorado, por exemplo, em uma aplicação na área de telemedicina, em cujo projeto fora facultada a participação durante esta pesquisa, que sugere um sistema de telediagnóstico auxiliado por computador [26]. A incidência da patologia endereçada pelo estudo de caso é associada à condição sócio-econômica desfavorecida da população atingida, freqüentemente resultando em dificuldades de locomoção e acesso ao serviço de saúde. O sistema de processamento e inferência computacional de imagens médicas acessível através da Internet pode ser compartilhado por diversos centros clínicos por meio de uma unidade de aquisição de imagens portátil de baixo custo, remotamente operada pela central de diagnósticos.

Em outra área correlata, a qual é merecedora do interesse especial dentro do grupo científico no qual se insere este estudo, concebem-se sistemas de monitoramento remoto de dispositivos eletrônicos e sua aplicação em telemanutenção. Para instrumentos de medida e outros equipamentos científicos ou industriais de precisão, a tele-aferição contínua através da Internet tem grande utilidade, por exemplo, no monitoramento periódico de equipamentos hospitalares. Além da aferição, na telemanutenção, como caso geral, pretende-se que equipamentos inteligentes identifiquem suas condições de operação e, à detecção de mal funcionamento ou de sua iminência, informe o evento através da rede; mediante o alarme, a intervenção de mecanismos de manutenção remota podem proceder testes ou mesmo providenciar a reconfiguração sanadora em tempo efetivo.

O teletrabalho e telemanufatura através do compartilhamento cooperativo de unidades fabris tele-robóticas interligadas através de redes de longa distância é também um conceito

explorado pelo grupo de trabalho [27].

Ensaaios com sistemas telemáticos de operação à distância também têm sido realizados com vistas às aplicações em ensino à distância, tais como o projeto *ChickScope* [28] da Universidade de Illinois, no qual, através de um equipamento de aquisição de imagens por meio de ressonância magnética nuclear, acessível através da Web, estudantes — de crianças a universitários — acompanham o desenvolvimento embrionário de uma ave. Destacam-se também nessa área os tele-laboratórios didáticos de sistemas de controle, para alunos de engenharia elétrica na Universidade do Estado do Óregon [29], e o projeto similar desenvolvido em conjunto pela Universidade de Jaén Avda e Universidade Nacional de Educação à Distância da Espanha [30], assim como o microscópio eletrônico do Laboratório Nacional de Oak Ridge, que leva a estudantes de ensino fundamental e médio, o acesso, através da Internet, a um instrumento científico de um milhão e meio de dólares [31]. Na mesma linha, o programa CORE [32] do Laboratório de Ciências Moleculares Ambientais conjuntamente com o DoE/NSF⁵ e outras instituições nos Estados Unidos, leva aos estudantes das universidades conveniadas, para fins de ensino e pesquisa, o acesso remoto a instrumentos científicos como um espectômetro de massa utilizado no estudo dos mecanismos cinéticos de degradação de CCl_4 em água subterrânea.

Dezenas de outros exemplos explorando a utilização compartilhada de equipamentos por meio de sistemas de operação à distância em redes telemáticas têm sido propostos em diversos campos — do entretenimento à ciência, da arte aos sistemas industriais — podem ser conhecidos através da Internet.

Após os precursores experimentos rudimentares com telecomando de eletrodomésticos, a segunda geração da pesquisa, a ainda recente área ensaia os primeiros passos no âmbito científico. Nessa fase demonstra-se a viabilidade da aplicação e examinam-se as possibilidades e limitações da tecnologia disponível à sua construção.

⁵(US) Department of Energy / National Science Foundation.

2.3 O Estado da Arte

A atenção recentemente devotada às redes de dispositivos telemáticos, como se propõe chamá-las neste estudo, dá-se em virtude da evidência da rápida evolução tecnológica dos seus meios de implementação e, em particular, da acelerada expansão da Internet, tanto através das redes locais cada vez mais comuns em ambientes acadêmicos, industriais e domésticos, quanto das redes de longa distância que já atingem cobertura global. A sua tecnologia em progressivo alcance — não apenas em função dos custos e disponibilidade de meios de acesso, mas também da facilidade de utilização que o seu amadurecimento proporciona nos últimos anos — tem se disseminado entre um número (vertiginosamente) crescente de usuários, os quais passam a reconhecer prontamente as incontáveis multi-utilidades que a telemática tem a lhes oferecer. A percepção praticamente instintiva frente aos poderosos recursos disponíveis tem resultado em contínuo surgimento de novas invenções para a utilização da Internet, dentre elas os populares *web devices*, uma tecnomania nascida nos anos 90: “web cams”⁶, sensores de temperatura e uma infinidade de outras curiosidades as quais, mais do que reedições avançadas dos ancestrais servidores coke, adentram a década seguinte com uma forte tendência no desenvolvimento de sistemas de operação à distância via Internet.

Aplicações baseadas na Web orientadas ao telecomando de dispositivos já principiam a surgir nos ensaios dos primeiros sistemas comerciais, assimilando a tecnologia desde nos controles de instrumentos eletrônicos interfaceados com microcomputadores, monitoramento de segurança em plantas residenciais, e até em sistemas embutidos (*embedded systems*). Exemplos da influência dessa tendência no âmbito industrial podem ser encontrados na implementação de servidores Web em circuitos integrados, como o IPic desenvolvido por pesquisadores da Universidade de Massachussets [33], que implementa uma pilha TCP/IP compatível com a RFC-1122 [34] em um microprocessador PIC 12c50x com as dimensões aproximadas de uma cabeça de fósforos⁷ e que, assim como outros projetos na mesma linha [35], [36], orientam-se a aplicações de telecomando de equipamentos via Web.

⁶Câmeras que capturam ao vivo imagens estáticas ou vídeo, que podem ser visualizadas através da Web.

⁷Acesso *on-line* ao IPic Web Server: <http://128.119.241.131:9080/index0.html>.

Todavia, é oportuno observar que, em meio a todo um contexto de conquistas tecnológicas no campo dos sistemas de processamento e transmissão de informação — e inclusive em vista do decorrente apelo que se lhes tem conferido na esfera comercial — a inadequada composição do entusiasmo acentuado, combinado à insuficiência de domínio efetivo das questões de engenharia referentes à emergente aplicação, suscita a atenção ponderada no exame de seus caminhos de desenvolvimento.

Cabe salientar, desse modo, que em verdade, os sistemas em comentário constituem ainda uma tecnologia pouco amadurecida e, não obstante seu notado potencial progressivamente desvendado pelos esforços concentrados na sua exploração, o campo carece ainda de importantes aquisições no que concerne ao tratamento sistemático dos aspectos técnicos e metodológicos pertinentes.

Com efeito, o estudo mais atencioso permite notar que as aplicações de telecomando baseadas na Web existentes, desde as mais rudimentares até as mais elaboradas, por mais populares que tenham se tornado, raramente podem ser citados como exemplos de sistemas telemáticos de operação à distância propriamente desenvolvidos com critério de engenharia, na medida em que, em larga extensão, trata-se basicamente de improvisações sobre sistemas legados e freqüentemente recorrem a recursos arbitrários para se esquivar da inadequação das técnicas disponíveis. De fato, em virtude da insuficiência de subsídios metodológicos, mesmo trabalhos representativos de reconhecível contribuição ao estado-da-arte muitas vezes correspondem a artifícios *ad hoc* em lugar de soluções sistematicamente desenvolvidas.

O panorama pode ser em parte creditado ao fato de o terreno em análise ser freqüentemente relegado a experimentação empírica ou, quando não, explorado com pouca sistematização, em geral com o auxílio de técnicas e ferramentas emprestadas de outras áreas, projetadas para servir às peculiaridades daquelas, e não da aplicação em relevo. Na prática, tais condições refletem-se no desempenho das implementações atualmente disponíveis, em muitas delas aquém das possibilidades habilitadas pela tecnologia atual, resultando também na inflexibilidade que os sistemas assim concebidos apresentam frente às configurações genéricas das necessidades reais.

No estágio atual, superado o furor acerca da novidade e da descoberta das potencialida-

des de um recurso recém conquistado — inevitável sempre que a célere marcha do progresso científico-tecnológico surpreende a ficção a passos largos — germina a nova geração exploratória, onde se insere este estudo. Depois de apresentada às possibilidades dos sistemas telemáticos de operação à distância e de haver sido convencida da sua viabilidade através das iniciativas antecessoras, a investigação científica agora lança-se ao reconhecimento mais detalhado do terreno, procurando pelos fundamentos técnicos e conceituais que envolvem a aplicação. Então, a partir deles torna-se possível empreender avanços decisivos na efetiva exploração dessa tecnologia, e já com olhos voltados à criação de aplicações operacionais e à sistematização do desenvolvimento sob a perspectiva da engenharia.

Capítulo 3

Investigações

3.1 Abordagem Convencional

Assim como em outras áreas onde exploram-se as aplicações dos sistemas de computação e comunicação, o desenvolvimento da tecnologia de acesso a dispositivos telemáticos é atualmente focalizado nas facilidades trazidas pela Internet.

Naturalmente, os primeiros experimentos com a conexão de dispositivos à rede de computadores¹ constituíam-se em mecanismos bastante rudimentares de telecomando, basicamente acionamento remoto simples e leitura de sensores. Na falta de princípios metodológicos específicos e uma vez que almejavam tão somente apresentar uma idéia ou, no mais ambicioso dos objetivos, demonstrar sua aplicabilidade, os autores desses ensaios preliminares não têm exitado em utilizar as ferramentas mais simples ao seu alcance, e que em geral representavam os recursos então mais óbvios.

Assim, os nostálgicos servidores *coke*, que precedem à Internet gráfica, eram baseados em utilitários *finger* (RFC 1288) [7] e *telnet* (RFC 854) [37] modificados. Contudo, tão logo difundiu-se, a criação de Tim Berners-Lee [38] batizada originalmente de *World Wide Web*² no início dos anos 90, esta passou a assumir o lugar de destaque quase absoluto desde

¹Excetuam-se nesta discussão as aplicações tradicionais do compartilhamento de periféricos convencionais na rede local como impressoras e dispositivos de armazenamento.

²“A grande teia global”, abreviada WWW ou simplesmente Web.

então — na verdade, em tal extensão que a associação entre Internet e Web tornou-se tão forte a ponto de causar confusão entre não especialistas, que não raro ainda empregam equivocadamente ambos os termos indiscriminadamente, tomando um pelo outro.

Existem sólidas justificativas para a evolução da Web desde o sistema original de recuperação de informação à interface universal para aplicações em Internet — e também, portanto, para os sistemas em foco. Com efeito, na maioria dos exemplos comentados anteriormente, a disponibilidade de implementações para diversas plataformas de computador e a familiaridade que conquistou entre os usuários da Internet propicia que novas aplicações Web se lhes tornem prontamente acessíveis [39] [23]. Adicionam-se a essas vantagens a integração com outras aplicações disponíveis unindo acesso a bases de dados, sistemas de comunicação interativos e recursos multimídia [23]. A seleção da Web deve-se também a sua adequação como “interface de baixo custo e ampla disponibilidade” para “teleoperação via *desktop*” [19], ou para “classes de aplicação para tele-robótica onde a motivação é econômica” [25].

Compartilhamento em larga escala, da mesma forma, sugere a necessidade de uma interface de operação simples, o que também conduz a Web [40] [21]. Finalmente, é notório ainda que sua arquitetura, no que se refere ao conjunto de recursos essenciais, é uma tecnologia simples, sendo admitidamente uma ferramenta de desenvolvimento de rápido aprendizado e relativamente descomplicada mesmo para programadores não especialistas.

Em síntese, a Web tem sido preferida por proporcionar um recurso de custo reduzido, bastante difundido, simples de usar, fácil de construir e, sobretudo porque é possivelmente a tecnologia percebida como a mais conveniente dentre as alternativas disponíveis para a pretendida utilização; mas não porque se tenha necessariamente considerado-a como mecanismo ideal para o acesso a dispositivos através de redes telemáticas. De fato, na sua maioria, os sistemas descritos na Seção 2.2 não apenas utilizam a Web como interface — conforme a tendência em todas as áreas de aplicações para a Internet — mas também são modelados como aplicações Web. Cabe então observar que um modelo de interface interativo, como o que se tem na Web, não constitui garantidamente uma arquitetura adequada para implementar o mecanismo essencial de aplicações genéricas³ [41].

³Uma discussão sobre a diferença entre utilizar-se a Web como interface para sistemas legados e a

Assim, seja utilizando os implementos mais simples ou incrementada com novos artifícios como Java e objetos multimídia, na migração dos antigos servidores coques baseados em improvisações várias para os sucessores *web devices*, a essência do procedimento não se altera; constitui-se na adaptação de ferramentas disponíveis para uma nova aplicação, mais em função da disponibilidade do recurso do que como consequência do reconhecimento ponderado de sua adequação.

Em meio a esse cenário, distinguem-se duas linhas principais de abordagem, ambas parcialmente orientadas pelo empirismo. Uma delas tem origem no âmbito da robótica e é possivelmente a primeira a se desenvolver, posto que é nesse meio que se originam os primeiros trabalhos científicos de repercussão. A partir dela difundiu-se o termo “Web robot⁴”, como são também muitas vezes conhecidos os dispositivos Web de modo geral.

Nessa área, o significado da palavra robô, antes fortemente associado com os dispositivos mecânicos (tradicionalmente braços articulados e manipuladores) dilata sua abrangência para englobar sistemas genéricos “cujo mecanismo é controlado por um computador” [21]. Nesse contexto, a Web tem sido vista como uma tecnologia estabelecida e solução natural dada sua hegemonia em outras áreas, e que pode ser adaptada para aplicações em tele-robótica (*web tele-robotics*) [39], com destaque em sistemas móveis (*web mobile tele-robotics*) [42] [43]. A principal motivação nesse campo é a utilização compartilhada de equipamentos e o telecomando de baixo custo para aplicações não críticas, inclusive domésticas [44].

A outra linha de abordagem de certo modo evolui no sentido inverso, partindo da experiência nos sistemas de informação baseados na Web e seguindo em direção ao desejo de ir além da virtualidade original do espaço da Internet para tanger o mundo físico [21] [45] [22]. Uma sugestão para classificar o nível de interação entre as duas “realidades” chegou a ser proposta [23], a qual distingue quatro categorias indo da simulação computacional (puramente virtual), sensoramento remoto (percepção do ambiente), comando remoto simples

reengenharia desses sistemas para convertê-los em aplicações Web, particularmente em sistemas de comando remoto, é oferecida pelo autor em *I Workshop on Web Engineering, International Conference on Software Engineering, Los Angeles, 1999*.

⁴Vale a ressalva de que certos mecanismos de busca de informações na Web também são algumas vezes referenciados por esse termo.

(atuação no ambiente) até controle completo em “tempo real” (interação total).

Em ambas as linhas, que se encontram e se confundem em muitos aspectos, em essência, a abordagem parte do mesmo princípio empírico de adaptação de uma tecnologia disponível para uma nova aplicação; o que não traduz necessariamente uma medida inapropriada, posto que é coerente com o procedimento natural de investigação explorar as metodologias existentes e avaliar sua adequação e limitações antes de partir na elaboração de novos recursos. Com efeito, tal é admitidamente o propósito da maioria dos trabalhos científicos revisados neste estudo.

3.1.1 Ensaios

A área em exame constitui definitivamente uma vertente interdisciplinar bastante recente e de evolução muito acelerada, o que é consistente com a importância estratégica das tecnologias envolvidas. Ao longo do desenvolvimento desta pesquisa tem-se observado a crescente convergência de esforços no seu desenvolvimento, verdadeiramente em um ritmo tão veloz que acompanhá-la tem se revelado uma tarefa árdua.

Quando do início dos trabalhos, estes inseriam-se em um cenário diverso do que existe atualmente, onde a quantidade de referências científicas era ainda mais escassa do que a que se conhece hoje. Assim, inicialmente este estudo recorrera principalmente ao labor experimental, onde buscou-se, a partir dos subsídios então disponíveis na bibliografia elementar, prosseguir na investigação daquele ponto em diante.

Com esse propósito, foram reproduzidos os mecanismos básicos introduzidos pelos trabalhos anteriores a fim de identificar-lhes as características e aspectos técnicos passíveis de um exame mais completo. A experiência desse modo angariada proporciona a constatação prática de limitações encontradas na abordagem convencional, algumas das quais já reportadas por autores antecessores e outras que se revelaram mediante investigação adicional. A seguir são sistematizadas tais observações, as quais constituem o ponto de partida para o desenvolvimento subsequente.

Figura 3.1: Interação entre elementos no arranjo cliente-servidor

No modelo cliente-servidor, as duas entidades comunicantes interagem através do envio de mensagens. O cliente inicialmente envia uma requisição ao servidor (Req); este, ao seu recebimento e após haver processado a solicitação, envia uma resposta (Resp) ao cliente para completar a operação.

3.2 Princípio de Funcionamento

O modelo básico largamente empregado para construção de aplicações interativas na Internet é conhecido como *cliente-servidor*. Nele o sistema compõe-se de dois elementos complementares que interagem de modo particular na realização do serviço⁵, conforme esquematizado na Figura 3.1.

O elemento da esquerda, o cliente, implementa a interface de operação através da qual o usuário tem acesso ao sistema mediante o envio de requisições através da rede. Estas são recebidas pelo servidor, à direita, que então processa a solicitação e devolve ao cliente uma resposta.

No serviço *finger*, por exemplo, a resposta é um bloco de texto informando quais usuários possuem uma ou mais seções em andamento no momento da requisição na estação onde está sendo executado o servidor; no serviço POP⁶ (RFC 1939) [46] a resposta pode ser o conteúdo da caixa postal eletrônica de mensagens do usuário e no serviço FTP⁷ (RFC 959) [47], um arquivo a ser recuperado do servidor.

No serviço Web, o cliente é freqüentemente nomeado *navegador* (“browser”), e o servidor, *servidor Web*. Assim como o serviço FTP, a Web implementa um sistema de recuperação de arquivos; todavia, diferente daquele, baseia-se no conceito de hipermídia.

Hiperdocumentos (ou *páginas* na terminologia da Web⁸) são documentos que podem

⁵Serviço é o nome genérico para referir-se às facilidades oferecidas pelas aplicações na rede.

⁶POP: *Post Office Protocol*.

⁷FTP: *File Transfer Protocol*.

⁸O *World Wide Web Consortium*, W3C, fornece uma descrição dessa terminologia [48].

conter uma composição de texto e outros recursos como imagens e objetos multimídia⁹. O que os caracteriza dessa forma é que referências (*links*) a outros hiperdocumentos podem ser associadas a porções do texto e outros objetos. Essas indicações, por sua vez, expressas no formato URI¹⁰ (RFC 2396) [49], indicam a localização do referido documento na Internet, fornecendo dados como o endereço do servidor onde pode ser encontrado e sua identificação dentro do sistema de arquivos daquele *host*¹¹ e outros dados que servem à sua recuperação.

A utilização básica da Web pode ser resumidamente descrita como se segue. Requisições explícitas são disparadas pelo cliente (navegador) mediante a entrada do usuário através do fornecimento da URI a ser consultada ou mediante a seleção de um *link* em uma página Web. Uma conexão é então estabelecida entre cliente e servidor para a transferência do recurso solicitado (via protocolo HTTP¹² (RFC 2616) [50]), quando, então, a mesma conexão é eventualmente encerrada pelo servidor. O arquivo ou conjunto de arquivos recuperados são, então, processados e exibidos pelo navegador.

O mecanismo conhecido como CGI (*Common Gateway Interface*), criado para atender a demanda pela utilização de documentos interativos na Web, funciona por meio da seguinte operação. Em lugar de um arquivo contendo texto (HTML/XML)¹³, um link pode indicar um programa executável como alvo. O servidor Web, nesse caso, em vez de retornar o conteúdo do referido arquivo, comanda sua execução ao sistema operacional do *host* onde se encontra e responde ao cliente com a saída gerada pelo programa. Por meio desse procedimento, o usuário é capaz de acionar aplicações no equipamento remoto e interagir com elas através da interface da Web, inclusive mediante a passagem de parâmetros juntamente com as requisições.

A estratégia básica utilizada na implementação dos *web devices* origina-se na observação de que o mecanismo CGI pode ser utilizado não somente para gerar documentos interativos, mas também para executar qualquer operação comum às aplicações ordinárias. A facilidade de aproveitar esse esquema descomplicado para telecomandar equipamentos é, assim, uma

⁹Documentos na Web são escritos na linguagem HTML/XML, conforme recomendações do W3C.

¹⁰*Universal Resource Identifier*.

¹¹Host é o nome genérico para computadores na rede.

¹²*Hypertext Transfer Protocol*.

¹³*Hypertext Markup Language / Extensible Markup Language*, www.w3.org.

Figura 3.2: Esquema de telecomando utilizando o mecanismo Web-CGI

O operador à esquerda utiliza o navegador para enviar requisições através da rede, via protocolo HTTP, ao servidor Web. Este, comanda a execução do programa CGI referenciado, o qual transmite e recebe informações do equipamento por meio do sistema de I/O do computador remoto.

consequência direta da observação, bastando para tal utilizar programas CGI para executar o acesso aos dispositivos de I/O (entrada/saída) do *host* onde é executado o servidor Web e, através deles, enviar instruções de controle para o equipamento interfaceado com o computador (Figura 3.2).

Esse artifício simples e derivações similares têm sido amplamente explorados e respondem propriamente não somente pela implementação da maioria dos exemplos estudados, mas também pelo modo usual de construção de aplicações em telecomando e teleoperação de equipamentos na Internet que, motivados pelo destaque da Web como tendência comercial, anunciam-se sob títulos diversos, tais como “*web-enabled*”, “*web-accessible*” etc.

3.3 Conclusões Preliminares

As primeiras constatações mediante os ensaios reproduzindo o funcionamento básico dos sistemas de telecomando baseados no mecanismo Web-CGI dizem respeito ao desempenho e flexibilidade das aplicações nele baseadas. Algumas limitações anotadas decorrem do esquema de interações elementares no modelo da Web e da especialização do protocolo HTTP para necessidades distintas das vinculadas à aplicação pretendida, dentre outros aspectos técnicos que serão oportunamente tratados no desenvolvimento seguinte.

A conclusão mais relevante, todavia, concerne à extensão conceitual à qual se devem tais limitações práticas e conduzem à reflexão sobre a adequação da abordagem. Por exemplo, a análise permite observar que a submissão aos princípios elementares do mecanismo utilizado resulta em um modelo no qual, para a aplicação em destaque, a Internet passa a se resumir em um canal para a transmissão de telecomandos, não muito especial em re-

lação ao fornecido por outros sistemas de comunicação, com talvez as vantagens de utilizar uma tecnologia bastante difundida e amplamente disponível; e as desvantagens de oferecer ainda hoje, em geral, limitações de velocidade de transmissão, imprevisibilidade de atraso na comunicação e não negligenciáveis restrições de segurança.

Especificamente, a análise permite conjecturar que, de uma maneira geral, o desenvolvimento até este ponto, representado pelos exemplos examinados, tem se focalizado em apenas uma fração das propriedades das redes telemáticas — provavelmente mediante a percepção intuitiva que seu uso convencional suscita, mas possivelmente não sob a compreensão ampla de como os recursos almejados e as dificuldades encontradas na realização da aplicação pretendida se relacionam com um conjunto maior de paradigmas que compreende aqueles sistemas. Com efeito, diversas considerações reportadas no desenvolvimento dos sistemas estudados, mesmo que não explicitamente reconhecidas como tal, constituem questões pertinentes à tal perspectiva — um aspecto que motiva a abordagem alternativa deste trabalho.

A observação que fundamenta o enfoque desta pesquisa é a de que a integração das capacidades de transmissão e processamento de informação habilitadas pela rede telemática não proporciona tão somente um meio de comunicação, mas um ambiente computacional dotado daquela facilidade. Isso lhe atribui potencialidades próprias daqueles sistemas, dentre as quais as exploradas pelas técnicas convencionais constituem apenas uma parte menor.

Sintetizando as conclusões dessa análise, tem-se a Figura 3.3. Nela, descrevem-se as diversas possibilidades habilitadas pelas redes de dispositivos telemáticos, ilustradas pelo exemplo de interesse especial dado pelo compartilhamento de acesso em nível global através da Internet.

Em seu arranjo mais simples, experimentos com teleoperação e telecomando através da Internet, com menção especial aos *Web devices*, têm buscado ensaiar mecanismos que permitam ao usuário o acesso remoto a recursos distantes através da infraestrutura de comunicação proporcionada pela Internet; trata-se, portanto, de telecomando através de uma nova mídia, cuja vantagem precípua constitui-se na cobertura de vasta extensão e

Figura 3.3: Possibilidades habilitadas pela rede telemática

Um usuário, um recurso: a infraestrutura de comunicação possibilita o acesso remoto a recursos distantes. Um usuário, vários recursos: a abrangência mundial da rede permite o compartilhamento de recursos em escala global. Vários usuários, um recurso: além da utilização compartilhada, o controle multi-usuário propicia a utilização colaborativa. Vários usuários, vários recursos: no caso mais geral, a natureza distribuída da rede telemática conduz aos ambientes virtuais cooperativos.

capilaridade de um sistema já instalado, assim como na disseminação de seu acesso, cada vez mais simplificado. A cultura dos tele-serviços e dos sistemas de informação on-line que se popularizou na década de 1990 naturalmente sugere a extensão dessas facilidades para dispositivos telemáticos em geral, incluindo equipamentos de toda sorte.

Todavia, mais do que um canal, a Internet é em si uma rede. Logo, diferentemente dos sistemas convencionais de operação à distância, como o controle remoto tradicional, onde um operador tem acesso a um dispositivo mediante um par de interfaces que lhes permite a comunicação, a rede telemática torna possível que o usuário tenha acesso a todos os dispositivos a ela conectados. Cabem, portanto, os mesmos benefícios que se têm com relação às facilidades convencionais disponibilizadas pelas redes de computadores aos seus usuários, tais como servidores de arquivo, terminais de impressão, base de dados dentre outros itens — em uma rede de dispositivos telemáticos (RDT) de âmbito global, a consideração da diversidade de recursos disponíveis ao usuário é uma concepção positivamente promissora.

Enquanto ambas as possibilidades tenham constituído o objeto principal de investigação nas primeiras fases da exploração na área até o momento, é bastante reconhecido que a rede telemática provê não somente um ambiente multi-recursos, como anotado, mas

também multi-usuário, i.e no qual diversos operadores podem compartilhar o acesso a um mesmo dispositivo. A razão de esta configuração encontrar-se ainda consideravelmente menos explorada em relação às anteriores encontra justificativa no fato de as questões técnicas que se impõe no projeto de sistemas de acesso compartilhado constituírem um conjunto notoriamente mais complexo, necessitando mecanismos essenciais de controle a fim de endereçar as complicações intrínsecas que se destacam ao estender-se dos processos seqüenciais simples para os processos concorrentes. Em um passo seguinte na sua sofisticação, para além da utilização compartilhada, é possível conceber a utilização colaborativa, na qual usuários interagem coerentemente na realização de tarefas em conjunto.

Assim, da generalização das condições anteriores, unindo-se as propriedades multi-recurso e multi-usuário, evolui-se naturalmente para o caso mais geral, onde usuários e recursos interagem entre si em um ambiente distribuído. No contexto das redes de dispositivos telemáticos, isto conduz ao conceito que este estudo se refere por *ambiente virtual cooperativo*, onde uma coleção de elementos unidos por uma rede de comunicação compartilham o acesso a facilidades comuns e colaboram na execução de tarefas. Eis o princípio que se aplica em instâncias outras onde analogias são pertinentes, a exemplo dos sistemas operacionais onde múltiplos processos se utilizam de serviços do *hardware*, dos servidores e clientes em uma instalação computacional distribuída ou dos sensores e atuadores em uma rede de automação fabril.

Orientando-se nessa direção, a abordagem desta investigação sugere destacar algumas possibilidades assaz interessantes que se tornam possíveis em uma rede de dispositivos telemáticos concebida de tal forma. Dentre elas, o *acesso multipontual equivalente* diz respeito à conveniência de que usuários em distintos pontos de acesso na rede desfrutem de equivalentes condições de utilização dos recursos disponíveis no sistema, independente de sua localização, a fim de que a tecnologia em destaque cumpra seu papel elementar de mitigar os obstáculos a tal homogeneidade.

Um tal sistema seria, contudo, grosseiramente ineficiente se apenas um usuário pudesse ter acesso a todos os recursos da rede a cada instante, especialmente na sua aplicação no compartilhamento em âmbito global. Logo, é extremamente desejável que se possa contar com o *acesso multi-usuário simultâneo* e, mais do isso, também com o *acesso a recur-*

ses compartilhados, que adiciona a condição de a utilização concomitante de um mesmo conjunto de recursos não resulte em conflitos ou degradação excessiva de desempenho.

Ainda em se tratando de compartilhamento em escala global, é bastante possível que na prática, em determinadas situações, subsistemas devam ser duplicados dentro da rede, a fim de acomodar a carga de utilização através do *acesso a recursos redundantes*, o qual presta-se igualmente a garantir a confiabilidade global, aprimorando a tolerância à faltas isoladas.

Em tempo, para que essas intenções se tornem viáveis em um sistema de configuração dinâmica, a exemplo da Internet, é preciso que constantes alterações na topologia da rede ou na localização dos recursos dentro da mesma não afetem o modo como os usuários têm acesso a eles; analogamente, atendendo às tendências rumo às tecnologias da *mobile computig*¹⁴ é desejável que também os usuários possam mover-se através da rede sem que isso comprometa a utilização do sistema¹⁵.

Com relação às propriedades enumeradas, é oportuno notar que, enquanto tais se apresentam de forma revolucionária para a aplicação em foco — nos sistemas de acesso à dispositivos genéricos, como na tele-instrumentação, equipamentos robóticos e facilidades remotas de um modo geral — seu conjunto representa um tema bastante estudado em contextos específicos, a exemplo da área dos sistemas distribuídos, na qual é referenciado na literatura pelo termo *transparência*. De fato, a cada uma das características de acesso comentadas corresponde uma modalidade específica dessa propriedade, respectivamente a transparência de localização, de paralelismo, de concorrência, replicação e migração [52].

Constata-se, mediante essa perspectiva, que as motivações para o desenvolvimento de sistemas de compartilhamento de dispositivos através de redes telemáticas norteiam-se pela habilitação de benefícios que, de fato, convergem para os atributos de um modelo distribuído; as técnicas rudimentares experimentadas pelas abordagens convencionais, como as revisadas anteriormente, não contemplam essa perspectiva e tampouco capturam em

¹⁴Computação móvel, em uma tradução literal, encontra grande atenção na área dos *hand helds* (portáteis), *wearable devices* (que se carregam à maneira de peças de vestuário) etc.

¹⁵A necessidade de independência do ponto de acesso levou engenheiros da NASA à consideração de sistemas de teleoperação via Internet para a missão Rover em Marte, a fim de permitir-lhes a mobilidade entre as diversas instalações de controle terrestres [51].

plenitude as possibilidades da tecnologia já efetivamente disponíveis.

Em conclusão, mediante à análise preliminar, as considerações assinaladas permitem formular o objetivo básico deste trabalho, o qual pretende investigar um modelo para redes de dispositivos telemáticos baseado em uma organização distribuída. Com efeito, a transposição das conquistas angariadas na teoria da computação distribuída para os domínio do controle de sistemas é um argumento não apenas plausível, mas praticamente mandatórias para o projeto de grandes instalações confiáveis, como em aquisição de dados e automação de processos [53] — ressaltam-se as redes de dispositivos telemáticos globais como aplicação direta.

Capítulo 4

Um Modelo de Rede de Dispositivos Telemáticos

4.1 Contextualização

É digno de menção o quão rapidamente este tema de investigação tem atraído a atenção da comunidade acadêmica no período mais recente. A crescente injeção de esforços tem acontecido tão aceleradamente que, no curto espaço de tempo do desenvolvimento deste estudo, desde o início das pesquisas até o momento, já é possível notar apreciável mudança no panorama tecnológico, caracterizado principalmente pela organização das primeiras conferências de vulto dedicadas ao terreno e pela chegada de seu assunto ao temário das revistas científicas de grande relevância — dentre eles o *Workshop “Robots on the Web”* realizado durante a IROS’98 (*IEEE/RSJ International Conference on Intelligent Robots and Systems*), e a edição especial de mesmo título, primeira dedicada ao tema, na revista *IEEE Robotics and Automation*, (março de 2000).

Tal reconhecimento tem assim encorajado e contribuído para a realização deste projeto, sobretudo trazendo à tona o interesse de outras iniciativas científicas no mesmo campo. De fato, durante o andamento das atividades deste trabalho, formuladas as conclusões preliminares à respeito da abordagem convencional e já em fase adiantada na obtenção dos progressos assinalados com relação à elaboração do modelo proposto, deram-se a conhecer

alguns trabalhos sendo realizados em outros centros de pesquisa, os quais corroboram as conclusões deste estudo e indicam que sua orientação encontra suporte em iniciativas científicas recentes.

Cita-se, a título de exemplo, um projeto sendo concomitantemente desenvolvido em uma universidade australiana, em continuidade ao experimento com comando remoto via Web [24], comentado anteriormente (página 10), onde algumas considerações reportadas pelos seus autores [54] vieram mais tarde confirmar as conclusões previamente delineadas na fase inicial desta pesquisa. O referido estudo, focalizando a aplicação em tele-robótica, concebe similarmente um ambiente virtual cooperativo baseado em uma organização distribuída como forma de superar limitações de desempenho e flexibilidade dos sistemas de telecomando baseados no mecanismo Web-CGI. Entretanto, diversamente da proposta trazida por este trabalho, aquele constrói-se através de uma base *middleware* análoga a CORBA [55] e JRMI¹ [56], exceto pelo suporte para comunicação assíncrona através de mensagens [57].

O objeto de interesse desta pesquisa, por outro lado, define-se de maneira distinta. Na investigação de uma arquitetura distribuída para redes de dispositivos telemáticos, como em pauta, esta proposta desenvolve-se em um nível subjacente às abstrações do *middleware*, da orientação a objetos e de outras tecnologias específicas para integração de equipamentos interativos tais como Jini² [58]. Trata-se de um enfoque *bottom-up* onde se deseja, outrossim, um modelo simples de interação entre dispositivos, no qual estes exportem suas funcionalidades para os outros elementos do sistema mediante interfaces de telecomando. Orientada às aplicações em operação remota de equipamentos com capacidade telemática, a arquitetura a ser apresentada não supõe a coadjuvação de mecanismos complementares, quer por primitivas de serviços implementadas em camadas funcionais inferiores (tais como sistema operacional), quer por interfaces de aplicação proporcionadas por camadas superiores (como RPCs³). Assim, além da independência de padrões de *software* e *hardware*, importante para atender à diversidade presente em uma rede global, o modelo pode ser

¹JRMI: *Java Remote Method Invocation*.

²Jini resgata o destino originalmente reservado para Java, quando era esta ainda chamada Oak, projetada para a criação de aplicações de TV interativa e eletrodomésticos inteligentes.

³*Remote procedure call*: chamada de procedimento remoto.

Figura 4.1: Um ambiente de dispositivos cooperativos em rede

Em uma rede cooperativa, pessoas operam equipamentos (a), interagem entre si (b) e dispositivos se comunicam compartilhando recursos (c).

implementado desde em plataformas orgânicas sofisticadas como PCs, até em sistemas embutidos microcontrolados, diretamente em linguagem de programação de sistemas (como C) ou mesmo em linguagens montadoras (*assembly*).

O sistema em desenvolvimento representa um conceito ao qual referimos por rede cooperativa de dispositivos, um ambiente virtual de trabalho onde equipamentos interconectados interagem colaborativamente requisitando, provendo serviços e compartilhando recursos com os outros elementos na rede (aspectos dessa idéia têm sido explorados por tecnologias recentes em outras áreas, como engenharia de controle industrial e automação). Nesse modelo, a noção de compartilhamento não inclui apenas a interação de pessoas e equipamentos, mas também a destes entre si, como dispositivos autônomos que se comunicam e utilizam cooperativamente os recursos de toda a rede — em escala global, possivelmente de toda a Internet (Figura 4.1). Como ilustração, equipamentos dedicados podem utilizar-se da capacidade de processamento, bases de informação e da funcionalidade proporcionadas pelos outros elementos na rede, solicitando-lhes tarefas especializadas (sem a dispendiosa e freqüentemente impraticável necessidade de embutir todos esses recursos dentro de cada dispositivo).

A síntese dos princípios fundamentais do sistema proposto é tema de um artigo publicado pelo autor em edição especial da revista *IEEE Transactions on Computer* [59].

Figura 4.2: Sistema interativo e sistema responsivo

Figura 4.3: Arranjo operador-mecanismo

4.2 Elementos Básicos

Uma maneira bastante genérica de representar um sistema qualquer, natural ou artificial, susceptível de receber ou exercer influências no ambiente externo é através do modelo conhecido como *sistema interativo*, esquematicamente representado da Figura 4.2. Por definição, as *interações* se processam com relação ao *ambiente* e, dependendo de sua direção, correspondem às *entradas* e *saídas* do sistema, respectivamente. Quando considera-se uma relação causal entre as entradas e as saídas, i.e se estas são produzidas em resultado de uma ação aplicada por aquelas, pode-se especializar esse modelo para representar um *sistema responsivo*. Nele, os estímulos do meio sobre o sistema correspondem às entradas, e as suas reações mediante àquela influência correspondem às saídas.

Sistemas responsivos proporcionam uma base conceitual para projeto de dispositivos artificiais cujo objetivo é oferecer um conjunto de funções ao usuário, e seu modelo é usualmente adotado para representar a interação entre o *operador* e um *mecanismo* qualquer (Figura 4.3), como um equipamento de aquisição de dados, um atuador mecânico, um recurso computacional etc. Assim, no contexto das engenharias de controle de sistemas, o termo “sistema interativo” usualmente refere-se a sistemas interativos responsivos — convenção a ser seguida neste texto.

Quando existe interesse no *tempo de resposta*, i.e. na demora da reação mediante o estímulo, sistemas responsivos são formalmente chamados *sistemas de tempo real* — tempo real flexível, se a não observância de um dado limite de tempo implica na degradação, mas

não necessariamente na inviabilização do sistema; tempo real rívido, do contrário⁴ [60].

A arquitetura concebida neste estudo é a que resulta da interconexão de vários sistemas interativos em uma rede, de onde haver sido denominada *Rede de Dispositivos Interativos* (RDI). Dentre as características que a distinguem de outras organizações possíveis em RTDs, ressalta-se a de que todos os elementos da rede são modelados como sistemas interativos, indistintamente — um aspecto que contrasta com a prática usual em se tratando de sistemas de serviço ao usuário, na qual o arranjo operador-dispositivo é tomado como base.

Tal esquema, assaz conveniente para representar sistemas segundo a abordagem na qual os papéis de ambos elementos naquele par são implicitamente determinados de forma única, tem sido utilizado, dentre outros casos, para modelar serviços interativos implementados através das redes de computadores. Em geral, a propriedade identifica-se com o modo de operação desejado em sistemas dirigidos ao usuário (*user systems*), como são os mecanismos tradicionais de recuperação de informações na Internet. Esses sistemas caracterizam-se pela inerente assimetria com relação ao modo de interação entre as duas entidades elementares que compõem o modelo, na qual o operador é um componente ativo, ao passo que o dispositivo é sempre passivo, no sentido de que é sempre o operador que inicia qualquer transação. A especificação da Web, por exemplo, identifica de forma clara o usuário e o provedor do serviço, explicitamente estabelecendo que é o primeiro quem deve solicitar a informação; o servidor, ao menos com relação à intenção original, não deve molestar o usuário enviando ao navegador documentos não requisitados; filosofia análoga se aplica a outros serviços como FTP e POP.

O referido modelo, que influencia os experimentos com telecomando via Internet originados nesse contexto, é conhecido no domínio das redes de computadores como *cliente-servidor* e pode ser entendido como um sistema responsivo no qual, não obstante a comunicação se dê por meio de mensagens, a semântica da operação é a mesma da chamada de procedimentos, i.e. mediante uma requisição de serviço emitida pelo cliente, é estabelecida

⁴Do inglês “*soft*” e “*hard*” *real-time systems*. Recentemente, o termo *firm real-time systems* tem sido empregado para designar sistemas com rígidos tempos de resposta nos quais eventuais atrasos podem ser tolerados [60].

uma reação do servidor na forma da correspondente resposta.

No âmbito de controle de sistemas, tal arranjo é conhecido como *mestre-escravo*, em referência ao fato de haver um elemento comandando toda a operação, e um outro elemento subordinado, apenas reagindo ao estímulo daquele. Claramente, esta é uma condição que, embora viável em muitas circunstâncias, impõe-se bastante restritiva frente a generalidade demandada pela aplicação em vista. Presta-se bem aos mecanismos de telecomando simples, como em teleacionamento de chaves eletrônicas e sistemas de controle centralizado, onde uma unidade ativa é responsável pela operação dos subsistemas passivos. Não é adequada, todavia, para comando assíncrono, como no caso em que se deseja que o dispositivo controlado possa sinalizar a ocorrência de eventos não esperados — alarmes de prioridade, por exemplo) [41]. Eis, pois, uma das limitações dos sistemas de telecomando baseados no mecanismo Web/CGI, onde a assimetria intrínseca das interações elementares do modelo Web constrange sua aplicabilidade aos casos onde a condição mestre-escravo é aceitável.

Cabe notar que a limitação tem motivado a criação de artifícios vários, tais como fazer com que o operador “monitore” o dispositivo buscando pela ocorrência de eventos mediante o envio de requisições sucessivas a intervalos regulares, a exemplo da facilidade conhecida como “refresh”, disponível em alguns navegadores padrão, utilizada na maioria das *webcams* e *websensors* espalhados pela Internet. A técnica, conhecida como *polling*, em alguns contextos, é usada, por exemplo, em mecanismos de tempo-real síncronos, mais freqüentemente em sistemas embutidos. Por outro lado, no caso de comunicação através da rede, especialmente em canais de longa distância sujeitos a sensíveis latências, o processo se torna demasiado custoso, acentuando-se sua ineficiência quando os eventos observados são caracteristicamente de baixa freqüência ou esporádicos — e mais ainda quando o dispositivo é consultado por muitos operadores concomitantemente, gerando cargas de comunicação desnecessárias no canal compartilhado.

Adaptações auxiliares das mais diversas têm sido experimentadas em combinação com a Web, incluindo o envio contínuo de dados do servidor para o cliente através de mecanismos não padrões, como o sistema *push*, disponível em alguns navegadores comerciais [42] e, especialmente, os *applets*⁵ Java. Em tempo, a criação da linguagem Java acontecia con-

⁵Mini-aplicações destinadas à serem recuperadas do servidor Web pelo navegador, e serem executadas

comitantemente com o início das investigações preliminares desta pesquisa, e demandou algum tempo até que as notícias mais precisas sobre seu lançamento viessem à tona. Na oportunidade, dedicou-se atenção à averiguação da nova tecnologia, tendo-se inclusive o cuidado de recuperar através da Internet, diretamente do servidor oficial [56], uma versão de pré-lançamento do livro de referência [61] da linguagem, ao qual se seguiram outros, também consultados [62], [63], [64].

A estratégia básica, nesse caso, tem sido utilizar *applets* embutidos na página Web para transmitir informação de orientação, geralmente imagem, na forma de seqüência temporizada de quadros estáticos, ou conteúdo multimídia com vídeo compactado [40], [45]. Ainda que estabelecendo um canal auxiliar de monitoramento, o sistema é, ainda assim, essencialmente construído como uma aplicação Web: as transações correspondentes às requisições e respostas da operação em si se processam através dos mecanismos padrões do conjunto navegador-servidor, subordinado, portanto, à inerente assimetria das interações elementares daquele modelo — razão pela qual, a solução não sugeriu ser a direção para endereçar a questão em pauta.

Especialmente no caso de comando à distância, é um parâmetro a ser considerado a efetividade de vias sensoriais entre o operador e o dispositivo. Quando essas não são adequadas, como no comando a longas distâncias (*comando remoto*, para distinguir esse caso especial), o operador deve contar apenas com a interatividade proporcionada pelo sistema em si; é freqüentemente desejável minimizar tal insuficiência por meio da implementação de níveis de autonomia na funcionalidade do dispositivo, para que este possa cooperar com o operador, executando procedimentos autônomos e minimizando, assim, a necessidade de intervenção — da mesma forma, em canais de comunicação de baixa velocidade e sujeitos a atraso e latência variável.

Tal modo de funcionamento desvia-se da orientação unidirecional da operação mestre-escravo, e sugere um menor desbalanceamento da relação operador-dispositivo. O mesmo argumento aplica-se à habilitação da interação cibernética, como destacadamente se conhece no campo da teleoperação robótica, por exemplo, onde a evolução tecnológica permitiu

em ambiente seguro dentro destes, embutidas no layout da página.

substituir os atuadores irreversíveis (mestre-escravo) pelos dotados de servo-mecanismos com controle bilateral reversível [1], implementando a retroalimentação sensorial.

Mais genérica para o caso de operação de dispositivos, portanto, torna-se a organização *par-a-par* (*peer-to-peer*), na qual as entidades se comunicam de forma assíncrona. Não somente para permitir que o dispositivo controlado possa notificar eventos esporádicos ao operador, mas igualmente a fim de acomodar uma gama maior situações, nas quais a distinção dos papéis de operador e dispositivo em si não seja útil. Esta tem sido importante, tradicionalmente, no modelamento de sistemas de serviços interativos tendo em vista um operador humano e um dispositivo dado por um mecanismo eletrônico, mecânico ou componente de software. Contudo, em se tratando da aplicação em relevo, o par operador e dispositivo pode representar composições variadas, seja entre dois equipamentos que se operam mutuamente mediante sinais assíncronos, seja entre um software em um microcontrolador e um equipamento a tele-interfaceado, ou ainda entre dois sistemas de software entre si — enfim, em diversas configurações entre dois elementos onde as reações de cada um são disparadas pela sinalização da contraparte.

Assim, contextualizadas no domínio da aplicação de interesse, as considerações assinaladas sugerem um ambiente cooperativo onde os elementos guardem propriedades simétricas com respeito ao modelo interacional. A arquitetura experimental RDI atende a esse princípio constituindo uma rede de dispositivos cujas interfaces exibem comportamento tanto ativo quanto passivo, i.e. todos os elementos da rede são entidades de mesma natureza interagindo em arranjos par-a-par. Dessa maneira, diversamente dos esquemas cliente-servidor convencionais, onde tem sido conveniente definir a interface homem-máquina e interface de instrumentação do sistema interativo [53], a RDI assume apenas a *interface de dispositivo* (ID) para a qual a condição usuário-serviço corresponde a uma configuração momentânea entre duas unidades, passível de inversão na transação seguinte.

A Figura 4.4(a) ilustra esse comportamento dual, inicialmente indicando a situação onde um usuário humano envia comandos para o equipamento, ambos respectivamente representados no diagrama. A ID da esquerda, implementando a interface do operador, e sua complementar à esquerda, na função de interface de dispositivo, formam, nesta ordem, um par cliente-servidor: mediante o recebimento da requisição enviada pela primeira

Figura 4.4: O comportamento simétrico das interfaces de dispositivos

Em (a) uma ID na função de interface de operação envia uma requisição à sua parceira, que atua como controladora de dispositivo junto do equipamento. A configuração inversa se dá quando a segunda ID envia uma requisição à primeira para notificar um evento assíncrono.

interface, a segunda retorna-lhe uma resposta, possivelmente uma indicação de sucesso ou falha na correspondente ação transmitida ao equipamento conectado. O oposto se tem na 4.4(b): o equipamento envia um sinal não requisitado ao operador, por exemplo um alarme, uma notificação de evento ou um pedido de dados para executar determinada tarefa autônoma. Desta vez, invertem-se os papéis, passando o equipamento a operador e o usuário a se lhe configurar como serviço interativo; uma vez na posição de agente reativo, a ID da direita deve responder à da esquerda como na interação servidor-cliente, cumprindo o compromisso da simetria.

4.3 Estrutura da Rede

À noção de RDI não se associa uma rede física; define-se apenas no nível lógico. A título de referência, somente para apontar um paralelo, poder-se-ia situá-la acima da quarta camada no modelo OSI [65], não especificando propriedades das camadas física, de enlace de dados ou de rede. A única suposição é a de um canal confiável, livre de erros e que entregue as mensagens em ordem, proporcionado pelo serviço da camada de transporte subjacente. Assim, pode-se imaginar uma RDI implementada tanto em uma rede fabril em barramento serial de alta velocidade, como através da Internet sobre um serviço confiável

orientado a conexão, por exemplo.

Dentro da rede, um número indeterminados de ID interage trocando mensagens, solicitando e fornecendo serviços umas às outras em um ambiente cooperativo distribuído. No que se refere à topologia lógica, e assumindo cada elemento comunicante como um nó na rede, a RDI é uma rede ponto-a-ponto de conexão completa, i.e. cada ID comunica-se diretamente com todas as outras IDs.

Há mais de uma forma de implementar a comunicação entre os elementos em um conjunto como esse e diversos esquemas encontram aplicação prática. Uma possibilidade é conceber um sistema puramente distribuído no que concerne à comunicação, i.e no qual esta se processa exclusivamente por meio de mensagens diretamente enviadas de unidade para unidade na rede.

Cabem algumas considerações quanto a essa configuração na visada aplicação. Em primeiro lugar, há a questão do endereçamento das mensagens. Em sistemas operacionais distribuídos, por exemplo, a solução para o problema costuma requerer uma coordenação global para manter o conhecimento do espaço de nomes onipresentes entre todos os elementos. Uma alternativa é a utilização de *broadcast*⁶ para comunicação direta, ou para mapear endereços lógicos em físicos, como o protocolo ARP⁷ utilizado para converter endereços de rede em endereços físicos na rede local. Uma terceira possibilidade é o método de *lookup* (consulta) via um serviço de nomes proporcionado por um ou mais elementos especializados, como no sistema DNS⁸ utilizado na Internet para converter nomes simbólicos em endereços de rede.

Todas as estratégias encontram aplicação na prática, cada uma delas sujeita a complicações próprias. A técnica de *broadcast* é geralmente bem empregada em redes com canal de difusão, em barramento compartilhado, onde apenas uma transmissão pode ser recebida diretamente pela totalidade dos elementos, tal como entre estações na IEEE 802.3, ou entre CPUs⁹ através do barramento de dados em um multiprocessador; sua utilização em topo-

⁶ *Broadcast*: difusão; técnica de comunicação em grupo onde uma mesma mensagem é recebida por todos os elementos na rede.

⁷ ARP: *Address Resolution Protocol* (Protocolo de Resolução de Endereços).

⁸ DNS: *Domain Name System* (Sistema de Nomes de Domínio).

⁹ CPU: *Central Processing Unit* (Unidade Central de Processamento).

Figura 4.5: Algumas topologias de canais ponto-a-ponto

logias físicas ponto-a-ponto (Figura 4.5), ou mistas como no caso mais geral, implica em mecanismos de replicação de mensagens entre segmentos, o que se torna consideravelmente mais complexo, por exemplo, em inter-redes como a Internet. Além disso, aumenta a carga de comunicação na rede, causando a diminuição da eficiência, assim como o faz os sistemas de consulta via serviço de nomes, o qual requer uma transação de consulta adicional feita ao servidor para cada requisição normal endereçada a outros elementos; alguma melhora pode ser obtida mediante o uso de *cache*¹⁰ local, com o custo adicional de necessitar de uma coordenação global para manter a informação atualizada e coerente entre os pontos da rede.

De uma maneira geral, a consistência de estados globais em sistemas distribuídos é reconhecidamente sempre um problema complexo. Se não há um espaço de memória compartilhado, qualquer elemento que necessite conhecer o estado global do sistema (união dos estados de todos os elementos) deve inferi-lo apenas mediante a comunicação por mensagens. Especialmente em sistemas assíncronos, incertezas no atraso da comunicação e nas velocidades relativas dos procedimentos em cada elemento aumentam a susceptibilidade de estimativas obsoletas, incompletas ou inconsistentes [66]; se a acurácia da previsão pode ser elevada com a frequência da comunicação, esta não elimina o problema, tampouco é

¹⁰Repositório ou memória de dados de uso freqüente, utilizado para evitar a necessidade de repetidas consultas a outras fontes de informação.

Figura 4.6: A rede real e a rede virtual no modelo RDI

As IDs conectam-se ao GI (a), o qual provê serviços básicos de um ambiente distribuído. Sua ação é transparente para o conjunto, de tal sorte que as IDs interagem em uma rede virtual (b).

factível em qualquer escala. Novamente citando a atual Internet como exemplo, onde se tem o estudo de caso da pesquisa, essa taxa tem admitidos limitantes técnicos.

A alternativa adotada no modelo RDI utiliza de uma abordagem em camadas, dividindo conceitualmente o processo de comunicação entre as IDs em dois planos lógicos justapostos. O nível inferior refere-se ao modo como os elementos efetivamente trocam mensagens, e foi por isso designado o modo da *rede real*. Neste, a rede se constitui mediante a conexão das IDs a uma unidade de coordenação central, e através da qual o fluxo de dados é conduzido como em uma configuração ponto-a-ponto em topologia tipo estrela, como na Figura 4.6(a).

O componente central, denominado *gerenciador de interfaces* (GI) funciona à maneira de um roteador, ao qual todas as IDs se ligam ao se juntarem à rede, sendo portanto visível apenas no procedimento de conexão; desse ponto em diante, torna-se transparente, servindo apenas como um canal de comunicação, agregado de algumas facilidades. Dentre elas, GI provê um espaço de nomes global com resolução automática de endereços.

Por meio desse recurso, as IDs podem se comunicar enviando mensagens explicitamente umas às outras no modo complementar ao descrito, que foi chamado *rede virtual*. Conforme representado na Figura 4.6(b), do ponto de vista de cada ID, o arranjo funciona como um sistema distribuído, onde os elementos interagem endereçando-se diretamente. É tarefa do roteador traduzir referências simbólicas embutidas na mensagem, correspondente aos nomes com os quais as IDs se registram na RDI para o endereço respectivo, e cuidar para que cada mensagem chegue ao seu destinado receptor.

Não obstante seja motivada pelo compartilhamento de recursos em âmbito global, em lugar de um único sistema mundial esparso, análogo à teia mundial resultante de todos os servidores Web espalhados pelo globo, as RDIs compõe-se a maneira de ilhas, formadas pela interação de diversas interfaces de dispositivos em torno de um gerenciador de interfaces. Não há nessa decisão uma limitação, posto que na concepção do modelo não é interessante, do ponto de vista prático, que um sistema como o estudado sirva simultaneamente a um número exageradamente grande de usuários (a natureza da aplicação sugere condicionais técnicos inerentes ao acesso concorrente, operação em tempo real etc.). É provável, por outro lado, que instituições mantenedoras de tais recursos constituam redes específicas para servirem a grupos restritos de usuários, bem como que sistemas de acesso público adotem diferentes políticas de utilização. Dessa forma, o conceito de redes independentes é assaz adequado para a RDI.

Além de proporcionar uma solução efetiva para o endereçamento, a decisão pela topologia da rede real atende também a outras conveniências relativas às características da aplicação pretendida.

Substantial parte do desenvolvimento na área de sistemas distribuídos tem se dado no domínio das máquinas paralelas e dos sistemas operacionais [52]. Naturalmente, nessas classes de aplicações, o desempenho no que concerne à eficiência da comunicação entre as partes constituintes é prioritária para a viabilidade da solução. No primeiro caso, o requisito vem sendo progressivamente contemplado dentro dos barramentos que ligam os multi-processadores, de modo até a aproximar-se de majorantes teóricos da física. No segundo, convenientemente, a tecnologia atualmente disponível já se presta à construção de redes locais, ou LANs ¹¹, como consensualmente denominadas, com confiabilidade e taxas de transmissão satisfatórios para motivar a pesquisa nesse campo — razão pela qual conhecem-se muitos sistemas experimentais, usualmente projetados para instalações restritas, servindo a aplicações específicas em ambiente acadêmico, científico ou industrial.

A motivação pelo modelo RDI, todavia, não se restringe ao âmbito das redes locais. Ao contrário, abarca, mesmo com destaque, as aplicações em redes de longa distância. Tornam-

¹¹LAN: *Local-area network* (Rede de área local).

se importantes, portanto, as diferenças que existem entre ambos os casos: a distinção entre LAN e WAN não se faz meramente em função da extensão das redes (uma WAN não é uma grande LAN) mas, principalmente, das tecnologias que lhes são associadas. O aspecto relevante nessa classificação é que redes locais são em geral ambientes controlados, i.e. sob responsabilidade de um única entidade mantenedora — usualmente a própria usuária — e circunscritos a uma instalação física de pequeno a médio porte, onde são viáveis altas velocidades de comunicação.

As WANs, por outro lado, são resultantes da interconexão entre LANs distintas, abrangendo canais de longa distância que, por sua vez, podem compor-se de vários circuitos construídos com tecnologia diversa, de linhas telefônicas a enlaces de microondas, troncos de fibra ótica, canais via satélite, dentre outros sistemas. Resulta assim uma infraestrutura de comunicação bastante heterogênea e cuja qualidade não é tão previsível em comparação com o caso das LANs. Da mesma forma, diferentemente do que ocorre naquelas, as WANs não são geridas pelos usuários diretos, mas pela operadora de telecomunicações e possivelmente por mais de uma delas ao longo dos diversos segmentos que compõem a rede; decorre daí que não é possível especificar as características do serviço prestado. Na prática, não obstante os troncos de alta capacidade, estes são compartilhados entre um grande número de usuários, de tal sorte que as redes de longa distância tendem a oferecer menores taxas de transmissão; além do fato de que a manutenção e atualização de enlaces de comunicação em distância são extremamente mais caros do que em LANs. Além disso não é possível garantir localmente a integridade do sistema que extrapola a instalação física do usuário: linhas de comunicação podem ser congestionadas ou mesmo segmentadas em pontos externos, alterando a capacidade de interconexão com outros canais. Ainda, uma vez que são acessíveis de outras localidades, não há como impedir tentativas de acesso não autorizado; o máximo que se pode pretender é implementar mecanismos para detectar intrusões e evitar que estas tenham sucesso.

Em resumo, um sistema distribuído estendendo-se através de WANs deve assumir um ambiente heterogêneo, não controlado e potencialmente hostil. Seu projeto deve, portanto, buscar não somente a interoperabilidade para suportar a diversidade de tecnologias de comunicação e computação, mas, na mesma medida, a segurança do sistema. Trata-se,

na verdade, de uma hipótese válida em geral para qualquer sistema multi-usuário e é adotada, por exemplo, por projetistas de sistemas operacionais [67], nos quais as aplicações são executadas em espaços de memória separados, não tendo acesso direto ao *hardware* senão através do *kernel* (cerne) do sistema, o qual atua como um *proxy* (procurador) responsável por atender pedidos das aplicações de forma segura e segundo regras bem estabelecidas.

Eis, portanto, a analogia que descreve o funcionamento da RDI, onde espera-se que IDs sejam construídas por usuários, e, portanto, passíveis de comportamento imprevisível em função de projetos equivocados ou maliciosos. Embora as IDs se enderecem diretamente na rede virtual, a transmissão de mensagens se dá através dos serviços do GI, de modo comparável à comunicação entre processos no SO. Esse esquema permite que, além de prover mecanismos de utilidade comum a todas as IDs, o GI imponha também uma política central de segurança objetivando a integridade do sistema. Dentre os serviços disponibilizados pelo GI, além do espaço de nomes com resolução de endereço, incluem-se autenticação (que de outra forma deveria ser, necessariamente, realizada por cada ID), gerenciamento de acesso, arbitragem de recursos, controle de concorrência e outras funções importantes em sistemas multi-usuário.

O conhecimento necessário para que o GI possa executar tais serviços depende da informação sobre o estado global do sistema a cada instante, o qual, por sua vez é o resultado das transações efetuadas pelas IDs. O modo como essas transações se relacionam determina a complexidade do controle. Em particular, dois aspectos dessa propriedade são importantes: a correlação seqüencial, que expressa o modo como eventos consecutivos gerados pelo mesmo elemento se inter-relacionam, e a correlação combinacional, que descreve as interdependências entre seqüências de comandos sendo executados em paralelo.

Em sistemas onde ambas as dependências são de baixa relevância, como por exemplo na operação normal da Web (na recuperação de documentos de uma base de informação, requisições de consecutivas não guardam uma relação expressa; requisições emitidas por clientes distintos não se interferem), pouco ou nenhum controle por parte do sistema é requerido. Na prática, existe um grande número de aplicações simples nessa categoria, e as

necessidades de cada uma podem ser tão diferentes que a ISO¹² não se animou a introduzir mais do que uma camada “magra” [65] para esse serviço no modelo de referência OSI¹³ para arquitetura de redes.

Na aplicação em questão, essa noção é fundamental para que se possa fazer uma distinção entre *comando* e *operação*. A partir da sua acepção literal, assume-se nesta abordagem, que o primeiro termo refere-se às transações elementares por meio das quais o operador tem acesso à funcionalidade de determinado dispositivo; o segundo, por outro lado, traduz a seqüência de comandos correspondentes à execução de uma tarefa completa. Assim, a operação é efetiva se produz resultado útil, i.e. se o operador tem sucesso ao utilizar-se do dispositivo — submetendo comandos — de acordo com a ação planejada.

Nesse sentido, pode-se considerar que os mecanismos de acionamento remoto via Web anteriormente comentados apresentam sistemas de telecomando; teleoperação, no significado literal de operação à distância, não tem sido desenvolvida senão em esboços rudimentares. Tendo em consideração, que esta define uma relação intrínseca entre comandos consecutivos (correlação seqüencial), torna-se claro que a memória de eventos é importante; o mesmo vale com respeito à concorrência desejável em sistemas multi-usuários com compartilhamento de recursos (correlação combinacional). Logo, o controle é necessário.

O conhecimento sobre o estado do sistema e sua evolução pode ser implementado através do conceito de *sessão*. A cada sessão é associado um conjunto de parâmetros que estabelecem as condições negociadas para ela (que podem incluir regras de acesso, níveis de prioridade etc.) e informações de estado. Na RDI, o GI, além de roteador, executa o gerenciamento de sessão para coordenar a atividade cooperativa das IDs, implementando um *ambiente virtual de trabalho* onde são disponibilizadas as informações de estado atualizadas e consistentes e são coibidas ações de IDs desobedientes. Dado que o GI age como um observador onisciente capaz de intervir de modo transparente quando necessário, uma vez admitidas nesse ambiente, as IDs na rede virtual podem assumir como garantidas algumas premissas convenientes com relação ao comportamento das outras IDs, como em um ambiente confiável.

¹²ISO: *International Standards Organization*, (Organização Internacional de Padronização).

¹³OSI: *Open System Interconnection* (Interconexão de Sistemas Abertos).

Figura 4.7: Diagrama de estados da interface de dispositivo

Ao receber uma mensagem a ID comporta-se como um servidor convencional de dois estados; o envio da requisição, por outro lado, é um evento assíncrono que não causa alteração de estados.

4.4 Funcionamento Interno

Uma forma amplamente utilizada para representar sistemas interativos é o modelo de máquina de estados, em virtude da clareza com que especifica o modo como o estado de um sistema determinístico se altera face à ocorrência de um conjunto de eventos. Uma máquina de estados constitui-se de variáveis de estado e comandos que os transformam mediante o recebimento de requisições [68].

Quando descrita dessa forma, uma ID, do ponto de vista do GI, associa-se a uma máquina de dois estados designados, respectivamente, por PRONTO e ATENDE. No estado PRONTO, dois eventos são possíveis. Um deles ocorre quando uma mensagem é recebida (r); nesse caso, uma ID assemelha-se a uma entidade reativa em um arranjo cliente-servidor convencional: a mensagem é interpretada como uma requisição de comando, fazendo com que o estado da ID mude de PRONTO para ATENDE e esta se ponha a processar o pedido. Findo o trabalho, a ID eventualmente emite (e) uma mensagem como resposta, cujo evento acompanha o retorno do estado ATENDE para PRONTO. A mesma identidade não vale, por outro lado, com relação ao papel do cliente. Naquela configuração a entidade no estado PRONTO, ao emitir uma requisição, passa para o estado de espera, onde permanece até que uma resposta lhe seja entregue. Em vez disso, nas mesmas circunstâncias, um evento e não causa alteração de estado para uma ID que, estando no estado PRONTO, envia uma mensagem.

Esse modo de funcionamento pode ser comparado ao de um sistema operacional multitarefas, onde as IDs, por analogia, comparam-se aos acionadores de dispositivos (*device drivers*) e o GI, ao núcleo do sistema. Cabe a este último entregar requisições de serviços geradas pelo processos do usuário aos primeiros. Ao recebimento de tal pedido, os aciona-

Figura 4.8: Interfaces de dispositivos em uma composição em camadas

O modelo de interação das interfaces de dispositivos permite a sua composição em camadas, e assim, a organização de níveis funcionais abstratos. O gerenciador central controla o fluxo de mensagens através do arranjo e implementa a política de acesso especificada pelo usuário.

dores, como as IDs, dedicam-se ao seu atendimento até que, eventualmente, uma resposta é gerada e devolvida ao núcleo, ou ao GI, para que possa ser encaminhada ao cliente. A outra circunstância em que um dispositivo se comunica com o sistema operacional é na indicação de eventos assíncronos, como o término de uma operação de entrada-saída ou a chegada de dados em uma porta de comunicação; assim também acontece quando uma ID envia uma mensagem sem que esteja em atendimento. Em resumo, no estado PRONTO, o recebimento de uma mensagem tem o significado de uma chamada de procedimento, e o envio de uma mensagem, o significado de uma interrupção.

Note-se, portanto, que o envio de uma mensagem acontece como uma requisição assíncrona (se parte do estado PRONTO) ou como uma resposta à uma requisição anteriormente recebida (se é emitida a partir do estado ATENDE). O recebimento de uma requisição, por outro lado, é sempre interpretada pela ID como uma requisição de serviço. Esse modo original de funcionamento é a base tanto para a simetria da dualidade ativo-passiva desejada pela RDI, como também para a segunda característica essencial do modelo: a construção modular do sistema através da estruturação interna em camadas. A Figura 4.8 exemplifica como esse aspecto é explorado.

No diagrama, a ID *A* fornece uma interface de usuário através da qual o operador (à esquerda) pode enviar comandos à ID *C*, que atua como interface de instrumentação do equipamento (à direita). Em lugar de fazê-lo diretamente, contudo, ambas as interfaces de dispositivo utilizam-se de *B*, que lhes serve como intermediária na comunicação. O conceito fundamental dessa configuração é o mesmo válido onde quer que se utilize o princípio da decomposição de sistemas complexos em níveis funcionais abstratos. A partir dessa divisão é possível, então, construir o sistema como uma coleção de módulos independen-

tes hierarquicamente conexos, onde cada camada intermediária apresenta à sua superior, determinadas primitivas de serviço.

O procedimento confere ao projeto qualidades de flexibilidade e escalabilidade que são convenientes na aplicação a que se destina. Uma situação típica onde o esquema da Figura 4.8 torna-se útil, encontra-se na implementação do mecanismo de teleoperação do dispositivo, onde pode-se acrescentar camadas especializadas entre o módulo de comando e de instrumentação. Isso permite desvincular o projeto da interface de usuário da interface do controlador do equipamento, possibilitando, por exemplo, que um tradutor apresente à primeira, um conjunto de funções mais abstratas que as disponíveis diretamente na segunda (conversão de comandos de alto nível em instruções diretas de acionamento fornece uma aplicação do conceito). Adicionar mecanismos de segurança para que requisições mal especificadas, não autorizadas ou potencialmente perigosas alcancem dispositivos críticos, constitui outro exemplo de interesse, assim como a execução condicional de comandos, em função de parâmetros variáveis calculados pelo nível intermediário, proporcionando graus de autonomia.

O objetivo é a fragmentação de dispositivos monolíticos em subsistemas especializados, distribuindo a funcionalidade, tanto quanto conveniente, entre módulos distintos na rede. Com isso, determinado dispositivo real pode exportar sua funcionalidade através de uma única ID, ou através de um conjunto delas, cuidando cada qual de um subconjunto de funções — facilitando a concorrência na sua operação compartilhada ou colaborativa, ou oferecendo interfaces alternativas para usuários com necessidades diversas.

Desse modo, torna-se possível compor sistemas sofisticados virtualmente constituídos da interação cooperativa de unidades menores, as quais podem ser eventualmente disponibilizadas por outros sistemas (um sistema, nesse contexto, poderia ser informalmente descrito como uma porção “conexa” da funcionalidade disponibilizada por uma coleção distribuída de IDs). Por exemplo, equipamentos sensores e de aquisição de dados podem ser compartilhados entre aplicações essencialmente distintas, bem como unidades de pré e pós-processamento, bases de dados e mesmo atuadores. Ainda, definindo-se especificações para interfaces de dispositivos padrões (como portas de comunicação serial, paralela, autenticadores etc.) é possível a criação de “*toolkits*” prontos para serem utilizados como

Figura 4.9: Fluxograma simplificado do algoritmo de encaminhamento

componentes na construção de sistemas maiores.

A configuração de tais *dispositivos complexos* é feita através de programações fornecidas ao gerenciador central (GI), o qual manipula internamente estruturas lógicas conectando os dispositivos singulares (IDs) envolvidos na operação e cuidando da alocação dinâmica dos recursos necessários. A base para a execução desses programas é o *algoritmo de encaminhamento* utilizado pelo GI para rotear as mensagens.

4.5 Encaminhamento

A descrição do princípio básico do algoritmo de encaminhamento, abordado em mais detalhes nas seções subseqüentes, complementa a explicação a respeito do modo como o sistema de fato funciona, e que pode ser explicado com o auxílio da Figura 4.9, onde o trecho de fluxograma mostra simplificada como o GI processa as mensagens recebidas.

Quando uma ID, inicialmente em PRONTO envia espontaneamente uma mensagem, ela o faz endereçando-a diretamente à outra ID. Estando esta em condições de atender à solicitação, o GI cumpre a ordem emitida pela entidade cliente e cuida para que a mensagem chegue corretamente à destinatária especificada, caso em que se dá o *encaminhamento explícito* (ramo I no diagrama). A ID receptora, em seu turno, após concluir o processamento da solicitação, emite uma resposta. Desta vez, porém, a mensagem parte de um dispositivo

no estado ATENDE, situação em que duas decisões alternativas podem ser tomadas pelo GI.

Em ambas, a informação de endereçamento se torna irrelevante e tem lugar o *encaminhamento implícito*. Se não existe qualquer programação específica associada à ID emissora, então o GI necessariamente deve rotear a mensagem de volta à ID credora, i.e. aquela que enviou a requisição anterior que levou a ID emissora para o estado ATENDE (ramo II no diagrama). Por outro lado, a programação, se houver, oferece uma segunda possibilidade desse modo de encaminhamento e se dá na forma da indicação de um *alvo implícito* para a referida ID. Este alvo é o endereço de uma outra ID na rede, para qual mensagens implicitamente encaminhadas devem, em determinadas condições, ser entregues (ramo III no diagrama). O que determina qual das duas opções terá efeito é o conhecimento do GI acerca do histórico dos eventos que lhe permite distinguir, em uma operação combinada entre uma sequência de IDs, se a emissora está reencaminhando uma requisição para um nível mais interno em uma estrutura de camadas ou se está retornando a resposta para o plano mais externo.

Retomando o exemplo da Figura 4.8, a seguinte ilustração oferece-se útil a fim de esclarecer como funciona o algoritmo de encaminhamento. A programação do GI para o arranjo formado pelas três IDs poderia ser dada instruindo o gerenciador para que associe a B o alvo implícito C , como na Figura 4.10. A Figura 4.11 mostra, de forma esquemática a sequência de passos desenvolvida na comunicação entre os dispositivos. A primeira coluna dá os estados das três IDs, conforme estas estejam em PRONTO ou ATENDE. A segunda coluna representa a memória de eventos do GI. Para cada ID, um par na forma (Rp, Rf) indica, nessa ordem, o endereço do dispositivo ao qual uma mensagem é devida, e do qual uma mensagem é esperada. O primeiro campo, inicialmente vazio, é atualizado ao recebimento da próxima mensagem, passando conter o seu remetente. O segundo campo é atualizado quando uma mensagem é emitida para um outro dispositivo que não o indicado no primeiro campo, quando este não está vazio, e passa a conter o destinatário da mensagem. Ambos os campos retornam ao valor original quando uma mensagem é enviada à ID indicada no primeiro campo. A informação armazenada nesse par completa os dados de que o GI necessita para conduzir encaminhamento.

Estando todos os elementos no estado PRONTO (i), a transação inicia-se quando A gera

ID	alvo implícito
<i>A</i>	-
<i>B</i>	<i>C</i>
<i>C</i>	-

Figura 4.10: Programação dual implícito para o exemplo da Figura 4.8

i)	PRONTO PRONTO PRONTO	(- -)	(- -)	(- -)
ii)	PRONTO ATENDE PRONTO	(- B)	(A -)	(- -)
iii)	PRONTO PRONTO ATENDE	(- B)	(A C)	(B -)
iv)	PRONTO ATENDE PRONTO	(- B)	(A C)	(- -)
v)	ATENDE PRONTO PRONTO	(- B)	(- -)	(- -)
vi)	PRONTO PRONTO PRONTO	(- -)	(- -)	(- -)

Figura 4.11: Uma transação completa envolvendo três IDs

uma mensagem endereçada à *B*; o encaminhamento explícito tem lugar e o GI entrega a mensagem a essa ID, que passa ao estado ATENDE. Note-se que, não obstante do ponto de vista de *A*, tenha havido a emissão de uma interrupção assíncrona, o modelo da RDI requer que o receptor da mensagem, que sempre a interpreta como uma solicitação de serviço, termine por devolver-lhe uma resposta para completar a interação cliente-servidor (Figura 4.4, pág. 36). Com efeito, a aplicação que da primeira ID faz uso, e que é a responsável em última instância pela emissão da requisição, espera, no caso geral, uma resposta — que virá de qualquer modo como dita o mecanismo descrito. O GI toma nota desse compromisso registrando que *A* se torna credora de *B*, e que *B* se torna devedora de *A* (ii).

Transcorrido determinado tempo, *B* deve concluir o processamento da requisição recebida e, com isso, produzir uma mensagem de resposta que, sendo referente a uma transição ATENDE para PRONTO, é tratada por encaminhamento implícito. Uma vez que a programação de *B* tem como seu alvo implícito a terceira ID, *C*, é a esta última que a mensagem é entregue em virtude o estado em que *B* se encontra e independentemente da informação que esta tenha eventualmente indicado como endereçamento da mensagem. Nesse ponto *B* retorna para PRONTO e *C*, para ATENDE, como requer o diagrama de estados da Figura 4.7 (pág. 44). O débito de *B* para com *A*, portanto, não fora saldado, mas aquele dispositivo torna-se credor de *C* (iii), e esta devedor de *B*.

Na seqüência, há o processamento da requisição em C e o envio, por esta, de uma mensagem, a qual também é tratada, segundo o algoritmo da Figura 4.9, por endereçamento implícito. Desta vez, no entanto, como não há programação associada a C , a mensagem é entregue a B , que pela ordem dos eventos é a quem a resposta é devida. Tendo C saldado seu débito, sua condição no registro de estados do GI retorna à condição inicial, ao passo em que B é levado mais uma vez para o estado ATENDE, (iv).

Novamente, B deve terminar o processamento e retornar para PRONTO mediante a emissão de uma mensagem. Como da outra vez, o algoritmo estabelece o encaminhamento implícito mas, agora, encontrando B com obrigações para com uma ID credora (o que caracteriza essa condição é, como ficará claro em seguida, o fato do campo Rf de B não estar vazio). Neste caso, não o alvo implícito, mas o débito tem prioridade e, assim, para A a mensagem é conduzida pelo GI, retornando a emissora ao estado inicial (v).

Nesse ponto, o dispositivo A , responsável pelo início da transação, já tem a resposta referente à requisição original que disparou todo o processo. De acordo com o mecanismo descrito, essa ID deve encontrar-se agora no estado ATENDE. A fim de retornar também à condição inicial, A deve retornar para o estado PRONTO mediante a emissão de uma mensagem, que é igualmente tratada por encaminhamento implícito. O detalhe importante para a determinação do receptor da mensagem é que, tendo A ainda a informação da sua ex-devedora B — (Rp, Rf) só é zerado quando uma mensagem é implicitamente encaminhada à ID credora — a observação que o GI tem de A nesse ponto é a mesma que tinha de B no passo anterior. Logo, a mesma decisão se aplica, e a resposta de A deve seguir para sua credora, que no caso é um campo vazio. Encaminhar, pois, a resposta para “ninguém” é a forma conceitualmente consistente utilizada para detectar o final da transação; na prática, a mensagem é simplesmente descartada.

É interessante observar, ainda com relação ao exemplo, que após A haver iniciado a transação, cada dispositivo se comporta de fato como sistema interativo exatamente da forma como este foi apresentado, i.e. como um elemento meramente responsivo. A participação de B como intermediária entre A e C se dá de forma transparente, uma vez que a ID não tem “consciência” de tomar parte em uma seqüência de interações casadas; do seu ponto de vista, B foi consultada duas vezes (ii e iv) e em ambas apenas emitiu uma

resposta (*iii* e *v*) sem preocupar-se com o endereçamento. É, portanto, um caso completamente diferente daquele onde um *proxy* mantém um registro de requisições pendentes vinda de clientes, enquanto espera pelo atendimento das correspondentes requisições por ele repassadas a outro servidor. Mesmo *A*, ao receber a resposta à qual tem direito (*vi*), também comporta-se reativamente emitindo a mensagem que encerra a transação. Ainda, o caso especial da mensagem inicial gerada por *A* (*i*), para tornar o modelo consistente, pode ser visto como uma reação da ID à uma hipotética requisição anterior que esta houvesse recebido de algum outro dispositivo.

Essa descrição do algoritmo de encaminhamento é bastante simplificada. À sua extensão para atender às diversas condições não consideradas, mas que se impõe nas aplicações reais — tais como requisições não autorizadas, indisponibilidade do recurso solicitado, disputa pelo acesso a dispositivo compartilhado etc. — cabem diversas possibilidades no que concerne a estratégias de abordagem e técnicas de construção. Pelo menos duas variações destas foram experimentadas a título de ilustração, conforme comentado adiante.

4.6 Termos Utilizados na Descrição do Modelo

A fim de simplificar as discussões que se seguem, é conveniente sintetizar algumas definições introduzidas até este ponto, bem como atribuir significado a mais alguns termos úteis na descrição do modelo RDI. Conforme apresentado, a rede compõe-se de interfaces de dispositivo (IDs, ou simplesmente *interfaces*) que trocam mensagens entre si por intermédio de um gerenciador de interfaces (GI, ou simplesmente *gerenciador*). Este exerce o controle da comunicação mediante o conhecimento do estado global do sistema, o qual mantém através de uma representação abstrata do conjunto de IDs que é constantemente atualizado. Distingue-se, desse modo, o estado da *interface* do estado da sua *representativa* no GI: o primeiro refere-se à condição PRONTO ou ATENDE, na qual a interface de dispositivo pode encontrar-se, conforme esteja no estado inicial ou em processamento; o segundo, por outro lado, diz respeito à informação correspondente a cada ID que o GI guarda internamente. Inferido a partir do registro de eventos, o estado do dispositivo (*representativa*) é dado pelo par (Rp, Rf) que contém, nesta ordem, os endereços da credora e da devedora da interface

(ID) representada, os quais são calculados conforme as regras descritas anteriormente.

O conjunto dos estados de cada representativa forma o estado completo do sistema, utilizado pelo GI para executar o roteamento das mensagens enviadas pelas IDs, segundo um algoritmo de encaminhamento. O encaminhamento explícito, no qual a mensagem é entregue ao destinatário especificado pelo emissor, dá-se quando este causa uma *interrupção*: mensagem espontaneamente gerada a partir do estado PRONTO. A receptora, então, mediante a chegada da mensagem, passa ao seu processamento, findo o qual, emite um *retorno*: mensagem liberada a partir da transição do estado ATENDE para PRONTO.

Ao contrário das interrupções, um retorno é sempre encaminhado implicitamente. Logo, continuando a seqüência referida, a mensagem recém retornada ou é devolvida para a primeira ID (a interruptora) ou, se a emissora tem especificado um alvo implícito, é repassada para um terceiro dispositivo. A partir daí, se a segunda receptora também possuir uma configuração desse tipo, o processo se repete recursivamente e a mensagem é repassada de ID em ID até que, eventualmente, uma receptora final não indique um alvo implícito.

Uma seqüência completa de IDs, na qual cada um aponta sua sucessora como alvo implícito, constitui um *programa*, com início no único dispositivo não apontado por qualquer outro, e término naquele que não tem um sucessor — para efeito de notação, o programa $A : B : C$ tem início em A e término em C . O retorno produzido por esta última ID é sempre encaminhado para aquela outra que, pela ordem dos eventos qualifica-se como credora da mensagem, especificamente a remetente do pedido recebido pela ID terminal, i.e. a registrada na primeira coordenada do estado da representativa dado por (Rp, Rf) . A partir desse ponto, o retorno de cada interface vai sempre sendo encaminhado para sua respectiva credora até que, finalmente, um deles alcança a interruptora — esta, por sua vez, também recebe a mensagem e gera um derradeiro retorno que é consumido pelo sistema sem ser roteado adiante.

O processo completo, desde a interrupção pela primeira ID até seu retorno final, é chamado *ciclo de operação* e tem duas fases: primeiro uma ascendente, na qual a mensagem é recursivamente encaminhada através das IDs do programa, e depois uma descendente, que começa com o retorno da ID terminal e vai até o retorno da interruptora. Durante a

primeira fase, diz-se que uma ID *refere-se* à sua sucessora na sequência, ao passo em que na fase seguinte, cada ID *reporta-se* a que a antecede. A terminologia expressa a idéia de que um dispositivo intermediário, mediante o recebimento uma ordem, recorre ao concurso de outro dispositivo e, após receber o pretendido auxílio, presta contas ao seu cliente — daí os símbolos utilizados para o estado da representativa (Rp, Rf).

A fim de completar o arcabouço de palavras-chaves, é ainda oportuno definir que doravante os termos *requisição* e *resposta* aplicam-se às mensagens trocadas entre um determinado par de IDs e indicam, respectivamente, a referência que a primeira faz à segunda na fase ascendente do ciclo de operação, e o correspondente reporte, na fase descendente. Seja em uma ou outra fase, cada mensagem transmitida de uma ID para outra representa o resultado do processamento executado naquele *passo* do ciclo de operação; são, portanto, mensagens distintas. O termo *tarefa* é a forma utilizada para identificá-las indiferenciadamente, i.e. uma tarefa atravessa todo o ciclo de operação de ID em ID. Note-se que este caminho percorrido pela tarefa não corresponde necessariamente a um programa completo, uma vez que um dispositivo pode interromper outro em qualquer parte daquela sequência. O que se estabelece é uma *cadeia* que parte da ID interruptora (que não pertence, necessariamente, ao programa), atinge em seguida a primeira ID no programa (não obrigatoriamente no seu início), e engloba as restantes até o término do programa. Assim, se D interrompe B no programa $A : B : C$, forma-se a cadeia denotada por $D - B - C$.

O modo como chegou-se a esse modelo reflete um processo essencialmente indutivo, no qual partiu-se do tradicional mecanismo cliente-servidor empregado nos sistemas experimentais de telecomando ensaiados na fase preliminar da pesquisa e, gradualmente foram-se concebendo alterações fundamentais necessárias a fim de adequá-lo às propriedades desejadas. Iniciando com o conceito de sistema responsivo (tempo-real) como representação de dispositivos genéricos, a utilização de uma interface comum garante a simetria da relação par-a-par e a operação assíncrona, enquanto a dualidade da forma de interação com interrupções e retornos no comportamento ativo e reativo, respectivamente, permite requisições não bloqueantes. O controlador central provê uma solução efetiva para o espaço de nomes e roteamento, além de proporcionar o gerenciamento de sessões e administração do ambiente de trabalho distribuído. Baseado no algoritmo de encaminhamento, a construção

de dispositivos virtuais em camadas pode ser realizada através de programação interna do GI, o qual também tem condições de cuidar da alocação de recursos, manipulação de filas de ações pendentes e demais tarefas necessárias à operação concorrente.

4.7 Comunicação

As velocidades e confiabilidades relativamente elevadas alcançadas pelas LANs atuais, associadas à homogeneidade de *hardware* e *software* que usualmente caracterizam sistemas computacionais distribuídos projetados para instalações desse porte, permite diversas simplificações nos mecanismos de comunicação utilizados. Em geral, apenas um subconjunto mínimo das funções disponíveis no modelo de referência OSI é utilizado, o suficiente para implementar a interação básica entre clientes e servidores. Na prática, três níveis são necessários [52], sendo que os dois inferiores (camada física e de enlace de dados) são fornecidos pelo *hardware* de comunicação, restando ao software apenas as funções relativas à quinta camada, onde apenas um protocolo simples de requisição-resposta é empregado. As complicações relativas aos serviços de roteamento e conexão são propositalmente evitados em prol da eficiência, enquanto os níveis superiores, quando necessários, ficam a cargo da aplicação. Os esforços para otimização do sistema concentram-se, portanto, na arquitetura de software, onde deseja-se tirar o máximo proveito possível das capacidades da rede.

A situação é sensivelmente diferente em redes de longa distância. A baixa velocidade das WANs, em comparação com as obtidas nas redes locais, faz com que o limitante seja o desempenho da comunicação. Em lugar de dispensar serviços da arquitetura de rede, torna-se mais útil valer-se deles a fim de lidar com as imperfeições do canal, como a baixa confiabilidade e as imprevisibilidade dos atrasos que são característicos. Portanto, não se incorre em perda de desempenho apreciável pela utilização de mais níveis funcionais no modelo da rede; ao contrário, é freqüentemente necessário a fim de suprir as notadas deficiências.

Conforme comentado na seção 4.3, as noções exploradas na RDI são relativas às funções superiores do modelo de referência; as camadas abaixo dependem da rede sobre a qual aquela deve ser implementada. Se esta for simplesmente um barramento de alta velocidade

de (como em uma rede de instrumentação industrial), um protocolo simples com primitivas para transmitir e receber dados é apenas o que as IDs necessitam. Por outro lado, retornando ao eleito estudo de caso, se a RDI deve servir a uma rede de dispositivos telemáticos para compartilhamento de recursos através da Internet, as camadas inferiores devem implementar serviços adicionais para procurar oferecer um canal confiável para o tráfego de mensagens.

Nesse caso em particular, vale lembrar que a arquitetura da Internet não segue o modelo de referência OSI; na verdade ela o antecede em mais de uma década e difere daquele tanto no número e nas funções das camadas, como na própria terminologia. Sua estrutura é oriunda do projeto ARPANET, uma das mais importantes pesquisas em redes de computadores baseadas em chaveamento de pacotes, conduzida pela então ARPA¹⁴, e que entrou em operação em dezembro de 1969 [69]. Inicialmente conectando quatro computadores através de linhas telefônicas, originou, na década seguinte a rede militar MILNET, a qual se ligaria em seguida a duas redes via satélite (SATNET e WIDENET) [65] e, mais tarde, a várias universidades nos EUA através dos esforços da NSF. Forma-se assim a ARPA *internet*¹⁵, e posteriormente a Internet, cujo acesso em massa começou a ser disseminado na década de 80, alcançado a conectividade global em poucos anos.

Conforme explícito em seu nome, a Internet, mais que propriamente uma rede, pode ser identificada como o resultante da inter-conexão de múltiplas redes formando uma infra-estrutura que se estende por praticamente todo o planeta. Contudo, não obstante o conceito de Internet exista de fato a partir do que no padrão OSI corresponderia à terceira camada, sua arquitetura é normalmente considerada como sendo composta de quatro níveis distintos [70], os quais cobrem aproximadamente o mesmo conjunto de funções previstas no modelo de referência da OSI. A Figura 4.12 fornece uma representação desse arranjo.

A camada de enlace, ou de interface de rede, é de algum modo comparável às duas camadas inferiores do modelo OSI (física e enlace de dados) e significa basicamente toda a tecnologia subjacente à própria Internet: desde as linhas telefônicas da ARPANET, LANs,

¹⁴Hoje DARPA: *Defence Advanced Research Projects Agency*, EUA.

¹⁵É convenção corrente grafar Internet com inicial maiúscula para diferenciar o nome próprio do termo em inglês *internet* (inter-redes).

Figura 4.12: Arquitetura da Internet

canais de ATM, microondas, fibras óticas etc. A camada de rede, ou de inter-rede, é similar à sua correspondente de mesmo nome no padrão OSI, e tem como principal função o roteamento de pacotes através da rede. Acima desta, a camada de transporte, também análoga à presente no modelo da ISO, é responsável por fornecer um fluxo de dados para a camada de aplicação, que a seu turno, equivale à sua contraparte no topo do modelo da ISO.

Com referência à sua implementação, a camada de enlace é em geral fornecida pelo *hardware* de comunicação, enquanto ambas as camadas intermediárias — a de rede e de transporte — são disponíveis como uma pilha fornecida pelo sistema operacional, acessível através de APIs da biblioteca de programação. A última camada representa, na prática, as próprias aplicações do usuário, as quais fornecem elas mesmas os protocolos de comunicação nesse âmbito — e na ausência das camadas de sessão e apresentação, ficam todos esses serviços também a cargo da aplicação, se necessários.

Trazendo, então, o exemplo-objeto para esse contexto, uma RDI construída sobre a Internet corresponde a um sistema na camada de aplicação. Faz-se importante, pois, verificar quais as características dos serviços que lhe são disponíveis, as quais dependem, por sua vez, das propriedades das camadas inferiores.

O protocolo atualmente utilizado para comunicação ponto a ponto na camada de rede da Internet é o IP¹⁶, introduzido no início dos anos 80. Basicamente, sua função é o roteamento de *datagramas*: pacotes individualmente endereçados que viajam independentemente através da rede, da origem ao destino, possivelmente por meio de rotas diferentes (Cada *host* na rede é identificado por um endereço que é formado de 32 bits na atual versão

¹⁶IP: *internet protocol* (protocolo de inter-redes).

IPv4 (RFC 791) [71], ou 128 bits, na versão IPv6 (RFC 1883) [72] em andamento). Como o serviço oferecido pelo IP é *não confiável*, i.e. não há garantia de que os pacotes alcancem adequadamente (íntegros e em ordem) o destino, a qualidade disponível à aplicação depende dos recursos agregados pela camada de transporte responsável pela comunicação fim a fim.

Uma das principais opções nesse nível é fornecida pelo protocolo UDP¹⁷ (RFC 768) [73] que, como a camada inferior, implementa um serviço não confiável baseado em datagramas, deixando à aplicação a tarefa de verificar a ordem e a correção dos pacotes transferidos. Em virtude do grande número de aplicações que necessitam desse controle, a camada de transporte oferece também um serviço confiável *baseado em conexão*, implementado pelo protocolo TCP¹⁸ (RFC 793) [74]. Uma conexão cria um circuito virtual ligando origem e destino; o TCP tem como objetivo que este apresente-se ao usuário do transporte como um canal livre de erros. Para tal, o protocolo implementa um *serviço confirmado*: ambas as entidades nas extremidades do circuito trocam informações de controle durante a comunicação para notificar o recebimento correto dos datagramas ou providenciar a sua retransmissão, caso necessário.

Um e outro tipo de serviço encontram aplicações práticas. O protocolo UDP é frequentemente utilizado em situações, nas quais tempo de transmissão é mais importante do que a correção dos dados, por exemplo em áudio digitalizado, onde a omissão eventual de pacotes ou o seu recebimento com alguns erros é preferível a prejudicar a constância da taxa de transferência pelo envio de confirmações e retransmissões (ruídos esporádicos podem ser mais facilmente tolerados do que distorções causadas pela alteração de frequência e longos hiatos sonoros).

O inverso se dá, por exemplo, com relação à transmissão de programas executáveis, onde o erro de apenas um bit pode inutilizar toda a comunicação — nesse caso, o TCP é em geral empregado. A expressiva fração de sistemas reais onde tal condição é verdadeira, faz com que este seja extensivamente utilizado, razão pela qual disseminou-se a designação genérica para os protocolos da Internet sob o nome de família TCP/IP. Este é o caso também da

¹⁷UDP: *User Datagram Protocol*, (Protocolo de Datagrama do Usuário).

¹⁸TCP: *Transmission Control Protocol* (Protocolo de Controle de Transmissão).

aplicação de interesse, a RDI, onde não obstante taxas constantes de transmissão sejam desejáveis no caso de operação em tempo real, a integridade das informações de comando transferidas através da rede não pode ser rescindida — logo, tem-se no TCP um candidato.

Diversamente dos serviços de rede e de transporte, que se ocupam da transferência de dados, a última camada concerne à aplicação em si, i.e. ao trabalho para o qual o sistema foi projetado. Logo, cada tipo de aplicação define seu próprio protocolo, convenientemente desenvolvido para atender às necessidades específicas de cada caso. Alguns serviços “clássicos” da Internet têm produzidos protocolos que se tornaram amplamente conhecidos tais como o FTP, para transferência de arquivos, e o SMTP¹⁹ (RFC 821) [75], para correio eletrônico.

Conforme comentado, grande número de experimentos com telecomando via Internet, seja a título de investigação preliminar, seja seguindo tendências tecnológicas, baseiam-se na utilização adaptada de protocolos de aplicação já existentes. O preço pago pela praticidade, nesse caso, manifesta-se no uso ineficiente dos recursos de comunicação e nas limitações práticas das facilidades implementadas pelo protocolo no que concerne à utilidade desejada.

Como ilustração, o comando remoto baseado em Web atualmente popular em protótipos científicos, ou mesmo oferecido como interface de tele-instrumentação em alguns produtos comerciais, encontra sensíveis constrangimentos de flexibilidade e desempenho oriundos da inadequação do protocolo HTTP para tais sistemas. É o caso, por exemplo, do esquema de transações com conexões distintas do modelo básico de interação da Web, no qual sucessivas recuperações de documentos constituem operações independentes. A especificação orienta que o cliente (navegador) solicite uma conexão de transporte com o servidor através da qual a requisição é enviada; o servidor, atendido o pedido, deve devolver ao cliente a respectiva resposta, findo o que, a conexão deve ser eventualmente encerrada.

É oportuno anotar que este é o comportamento padrão especificado no HTTP/1.0 [76], então em uso corrente quando das investigações iniciais desta pesquisa, e que viria a ser alterado na versão seguinte, o HTTP/1.1 [50] publicada em 1997, definindo a conexão per-

¹⁹SMTP: Simple mail Transfer Protocol.

sistente como opção preferencial. Todavia, a decisão visa solucionar problemas da versão anterior relativos a conexões via *proxy*, bem como para permitir que o cliente possa transmitir em lote, sobre uma mesma conexão de transporte, várias requisições correspondentes às referências implícitas (recursos embutidos) dentro de um mesmo documento. De qualquer modo, visto que para o propósito legítimo da Web, a recuperação de uma página e todos os seus objetos inclusos constitui uma operação completa, e também que a natureza hipermídia do espaço de Web não justifica supor que novas requisições devam ser endereçadas ao mesmo servidor, é sugerido que implementações — navegadores, servidores e *proxies* — definam números e tempos limites para tal método.

Essa decisão está de acordo com o modo de utilização esperado na Web, na qual documentos são recuperados do servidor, interpretados pelo software cliente e, provavelmente com frequência, lidos pelo usuário humano antes que uma nova requisição seja emitida. Assim, normalmente tem-se a situação em que o intervalo entre sucessivas requisições é sensivelmente maior que o tempo despendido na transferência dos dados — manter conexões ativas nesse entremcio não é, portanto, eficiente. Em tele-acionamento, por outro lado, que é o caso da RDI, o comportamento oposto pode ser esperado. Em geral, a teleoperação de um dispositivo se processa como uma sucessão de requisições de comandos, arbitrariamente longa, e que pode apresentar-se com frequências relativamente maiores, de tal sorte que o tempo de transferência de dados torne-se superior ao intervalo entre requisições. Especialmente em redes de longa distância como a Internet, onde o estabelecimento e encerramento de conexões de transporte costuma ser custoso, maior eficiência pode ser obtida mediante um modelo de interação utilizando uma única conexão persistente.

Além disso, a alternativa é conveniente para a implementação do gerenciamento de sessões, resolvendo de forma mais adequada, por exemplo, o problema da manutenção da informação de estado que, no caso do protocolo HTTP, normalmente tem de ser realizada através da passagem dos parâmetros da seção como argumentos a cada requisição [54] (essa limitação constitui uma das primeiras dificuldades observadas nesta pesquisa durante ensaios iniciais com o protótipo de telecomando baseado em Web/CGI). A tática alternativa permite que a camada de transporte detecte o eventual encerramento abrupto da conexão

PROTOCOLO	CONTROLE	ENDEREÇO	AÇÃO	COMANDO
-----------	----------	----------	------	---------

Figura 4.13: Formato das mensagens do protocolo de comunicação da RDI

na entidade parceira²⁰ e informe à camada superior para que medidas adequadas possam ser tomadas, tais como o restabelecimento da conexão, ou o encerramento da seção associado à liberação de recursos e restabelecimento do sistema a estados consistentes.

Resta ainda considerar, com base na natureza da aplicação, o significado e o formato das mensagens trocadas entre os elementos da rede. Diferentemente de servidores de arquivos, por exemplo, que transferem unidades de dados arbitrariamente grandes, a comunicação entre as IDs carrega exclusivamente instruções de acionamento de dispositivos, as quais, em geral, podem ser encapsuladas em pacotes de informação relativamente menores, bem estruturados e padronizados. Com relação à sua representação, e valendo-se de elementos da terminologia em uso [68], uma unidade cliente envia à servidora uma requisição contendo um *comando* que, sendo executado pela receptora (possivelmente através do concurso de outras IDs em um programa), tem como efeito a atuação sobre o dispositivo associado e a geração de uma resposta para a primeira.

4.8 O Protocolo da RDI

Lembrando que o retorno emitido por uma ID é recebido como uma ordem de processamento pelo próximo dispositivo na cadeia, faz sentido estabelecer um esquema único para o formato das mensagens, sejam requisições ou respostas, o qual deve incluir, além do comando, um cabeçalho contendo a informação de endereçamento para que o GI possa rotear o pacote até o destinatário e, possivelmente, também informações de controle adicionais. A Figura 4.13 mostra como esses dados são representados no protocolo experimental concebido para comunicação na RDI.

Cada mensagem constitui uma única linha de texto, codificada na forma usual para

²⁰O termo “entidade” é como são chamadas as implementações de cada serviço na pilha de protocolos; entidades parceiras, por conseguinte, refere-se às entidades equivalentes nos dois elementos comunicantes.

aplicações na Internet, como uma cadeia de caracteres no padrão ASCII²¹ terminada pelo par CRLF — caracteres *carriage return* (retorno de carro) e *line feed* (alimentação de linha) , códigos 13 e 10, respectivamente). A seqüência, de comprimento variável, é internamente estruturada em campos formados pelas cinco primeiras subsequências adjacentes, delimitadas por um ou mais caracteres de espaço (ASCII 32). Vale lembrar que a mensagem originária de uma ID, antes de alcançar seu destino, passa pelo gerenciador de interfaces. Na prática o que ocorre é que o GI recebe a mensagem da emissora, a processa e compõe uma nova mensagem, que então é enviada à receptora. Nesse procedimento, o valor de cada campo pode ou não ser alterado.

Em uma interrupção, o campo COMANDO transporta a instrução de acionamento e, como último na seqüência, pode conter uma ou mais subcadeias de caracteres também delimitadas por espaços. A primeira destas constitui a *função* do dispositivo à qual se refere o comando; a ela associa-se a lista de *argumentos* formada pelas cadeias seguintes. Quando se trata especificamente de um retorno, o campo traz o resultado do processamento recém concluído pela emissora; contudo é ainda assim adequado denominá-lo “comando”, uma vez que esse mesmo resultado será tomado pela receptora como um genuíno pedido de serviço a solicitar-lhe uma reação correspondente — seja a mensagem uma requisição ou uma resposta. Cada dispositivo é livre para especificar seu próprio conjunto de comandos.

A informação sobre o processamento do comando é indicado no campo AÇÃO. Para o emissor, o campo contém um código de *status* de resposta, um dentre um conjunto finito de valores pré-estabelecidos pelo protocolo que indica se o pedido foi ou não executado com sucesso; neste último caso, informa também razão associada à falha (se esta é transiente ou permanente; se é oriunda de um problema com o dispositivo cliente, com o dispositivo servidor ou com o sistema gerenciador etc.). Para o receptor, a seu turno, o campo indica o modo como o comando deve ser interpretado: se o código indica sucesso, o valor esperado no caso normal, este é equivalente a uma solicitação para que o comando seja executado; uma notificação de falha, por outro lado, pode determinar que o comando seja ignorado. Isto pode se dar, por exemplo, quando a emissora decide que não fora possível atender corretamente o pedido. Pode ocorrer também em virtude da intervenção do gerenciador

²¹ ASCII: *American Standard Code for Information Interchange*.

central, que pode alterar o valor no campo para desautorizar o subsequente processamento da mensagem. Sendo especificado pelo protocolo, este campo permite aos dispositivos produzir reações automáticas mediante o recebimento da mensagem, tais como o reenvio de pedidos não atendidos ou sinalização de falta para outros dispositivos na rede.

O campo ENDEREÇO é utilizado pela emissora de uma interrupção para especificar o destinatário da sua mensagem; já para a ID que produz de um retorno, a informação aí introduzida não é relevante, posto que o encaminhamento implícito terá lugar (versões aprimoradas do protocolo podem vir a utilizar o endereço em um retorno para outras finalidades). O campo, porém, é sempre alterado pelo GI quando do seu recebimento e preenchido com o endereço da ID emissora, de tal sorte que a receptora possa conhecer o remetente da sua mensagem. Evidentemente, a própria emissora poderia tê-lo feito, mas a chancela do GI proporciona um nível de autenticação conveniente para a receptora, que pode certificar-se, tanto quanto sua confiança no GI permite, da procedência do pedido a ser processado.

No modelo ensaiado nesta pesquisa, nenhuma informação foi atribuída ao campo CONTROLE, mas sua presença foi reservada para que versões posteriores do protocolo possam fazer dele uso na implementação de facilidades suplementares. A idéia é que sistemas aprimorados possam especificar recursos adicionais e ações complementares a fim de atender a necessidades específicas de determinada classe aplicações. Por exemplo, em determinadas situações onde segurança de dispositivos críticos seja primordial, pode ser desejável que os dados transmitidos como instrução de acionamento sejam encriptados a fim de evitar a interceptação da comunicação; nesse caso, o campo de controle pode transportar chaves de codificação ou informação de seqüenciamento de comandos. Outra possibilidade é utilizar o campo para registrar o tempo de viagem da tarefa no ciclo de operação (ou suas marcas parciais em cada ID), dado que pode ser útil em sistemas de tempo real. Essas e outras informações de controle podem ser codificadas dentro do campo de forma a estender as possibilidades contempladas pela descrição ora apresentada.

Finalmente, o campo PROTOCOLO contém a assinatura do protocolo, que constitui-se do seu nome diretamente seguido de uma indicação da versão em uso, a qual determina como os outros quatro campos devem ser interpretados. Uma versão experimental bas-

tante rudimentar (com o campo de controle sem efeito e um reduzido conjunto de ações padronizadas), mas contendo os recursos mínimos para a utilização como exemplo, foi implementada a título de demonstração.

Não obstante caiba a cada dispositivo em particular implementar uma coleção própria de comandos, o qual reflete a *funcionalidade* (conjunto de funções acessíveis através da interface de operação) que deseja exportar na rede, sugere-se assaz interessante conceber um *conjunto mínimo de instruções* (CMI) que proporcione à rede uma base para interação elementar. Nesse arcabouço, são de utilidade comandos para, por exemplo, verificar se o determinado dispositivo está operante, inquirir-lhe a cerca de sua funcionalidade para certificar-se de quais comandos implementa, consultar-lo sobre condições de acesso (permissões, prioridade etc.), dentre outras possibilidades. O mencionado protocolo experimental, a ser descrito adiante, especifica um pequeno CMI que ilustra como sistemas mais elaborados podem estender esse conceito.

Analogamente, há propósito também em considerar que classes de dispositivos possam definir cada qual um *conjunto de instruções específicas da aplicação* (CIEA), determinando, assim, condições para a compatibilidade com especificações particulares. A medida pode mostrar-se conveniente, por exemplo, para que dispositivos sensores estabeleçam convenções para interface de operação através de uma funcionalidade padronizada com comandos tais como “reativar”, “obter leitura”, “informar precisão” etc. A união do CMI com o CIEA, mais um *conjunto de instruções específicas do dispositivo* (CIED) compõe, nesse caso, o *conjunto de instruções do dispositivo* (CID), parte da especificação de cada dispositivo.

Capítulo 5

Concepção de um Sistema-Piloto

As discussões tecidas nos capítulos anteriores fornecem uma visão geral dos conceitos fundamentais utilizados na concepção do modelo RDI. Este capítulo apresenta alguns experimentos simples que foram elaborados durante o estudo, com a finalidade de verificar concretamente a aplicação de tais princípios e sua adequação do ponto de vista prático, tanto com relação à sua eficiência frente às configurações típicas dos sistemas reais, quanto no que concerne às limitações das estratégias e ferramentas de desenvolvimento permitidas pela tecnologia disponível.

5.1 Alguns Esboços

Diversos esboços foram produzidos a fim de investigar técnicas convencionais utilizadas em sistemas de telecomando via Internet, bem como ensaiar alguns mecanismos projetados para implementação do modelo alternativo. Uma sortida coleção de pequenos módulos de software, perfazendo no total aproximadamente 7 mil linhas de código, foi escrita em linguagens diversas tendo C como opção preferencial, mas incluindo também alguns programas em C++, segundo sugestão de colaboradores interessados na possível integração da RDI em aplicações em desenvolvimento. Ainda, atendendo à expectativa conhecida em diversas ocasiões quando do intercâmbio científico com núcleos envolvidos em pesquisa na

área de OOP¹, realizou-se o porte de alguns trechos para a linguagem Java. Além dessas, outras linguagens tais como Perl, HTML e *shell scripting* foram eventualmente empregadas. A menção a alguns excertos desse software e a referência à sua implementação aparece no texto como registro apenas, uma vez que o protótipo final os substitui completamente.

O desenvolvimento do sistema-piloto foi conduzido segundo um processo recursivo de implementação-testes-avaliação, partindo dos princípios gerais do modelo de interação proposto e revisando passo a passo o projeto conforme necessidades práticas fossem anotadas. Isso conferiu à abordagem uma perspectiva essencialmente experimental na qual cada recurso elaborado corresponde a necessidades detectadas no âmbito da observação prática. Esta seção descreve alguns ensaios realizados durante essas etapas, discutindo as principais conclusões angariadas a partir de sua avaliação.

5.1.1 Mecanismos do Gerenciador de Interfaces

O primeiro protótipo reproduzindo os recursos elementares da RDI, ainda bastante rudimentar, foi construído implementando os mecanismos básicos descrito na seção 47. No GI, um vetor de estruturas estaticamente alocado foi utilizado para armazenar objetos contendo as representativas das IDs conectadas à rede, encapsulando dados como o nome e endereço do dispositivo, seu alvo implícito e a informação de estado. Após um *handshake*², os dispositivos conectados ao sistemas eram associados à suas representativas, as quais eram então atualizadas à cada mensagem recebida pelo gerenciador.

Basicamente, o mecanismo interno constituía-se de uma tabela onde cada ID é representada pelo par (Rp, Rf) . Na Figura 5.1(a), três dispositivos A , B e C estão inicialmente desocupados: nada a ser reportado à qualquer cliente e nenhuma referência feita à qualquer servidor. Um artifício utilizado para representar esse estado particular foi a definição de um hipotético “dispositivo nulo” (em analogia ao dispositivo */dev/null* do SO tipo unix, ou ao ponteiro NULL da linguagem C) cujo endereço é denotado pelo valor zero, de maneira que o estado $(0, 0)$ da representativa indica uma transação com o dispositivo nulo.

¹ *Object-oriented programming* (programação orientada a objetos).

² *Handshake* (aperto de mãos): negociação inicial dos parâmetros de uma sessão.

	Rp	Rf		Rp	Rf		Rp	Rf
A	0	0	A	0	B	A	0	B
B	0	0	B	A	0	B	A	C
C	0	0	C	0	0	C	B	0
(a)			(b)			(c)		

	Rp	Rf		Rp	Rf		Rp	Rf
A	0	B	A	0	B	A	0	0
B	A	C	B	0	0	B	0	0
C	0	0	C	0	0	C	0	0
(d)			(e)			(f)		

Figura 5.1: Estados das representativas na versão preliminar

Se a representativa está nesse estado, o GI infere que a ID correspondente deve encontrar-se no estado PRONTO e livre para ser requisitada por outro dispositivo. Seja então o caso em que existe apenas um programa registrado no GI dado por $B : C$. Se então A interrompe B , o estado do sistema evolui primeiro para o mostrado na Figura 5.1(b), e em seguida, após o retorno de B seguir como referência até C , chega ao estado representado na Figura 5.1(c). A este passo, a mensagem atinge o término da cadeia $A - B - C$, de modo que o retorno de C é reportado a B , que por sua vez reporta a A , fazendo com que o estado evolua no sentido inverso passando pelo estado da Figura 5.1(d) até o da Figura 5.1(e). Nesse ponto, a ID A , que acaba de receber a resposta de B , está no estado ATENDE e sua representativa em $(0, B)$. Para terminar o ciclo de operação, o algoritmo interpreta consistentemente que A , que esperava uma mensagem de B , deve uma resposta ao dispositivo nulo. Para este é então entregue o retorno de A ; na prática, a mensagem é simplesmente descartada e o estado atinge a situação final da Figura 5.1(f).

Se mais IDs estiverem presentes na rede, pode acontecer de haver mais de um ciclo de operação ativo a qualquer momento e, dado que nada impede que haja interseção de dois ou mais programas, tem-se então um terreno propício para conflitos.

O caso mais simples de ser analisado é aquele no qual duas cadeias nas fases ascendentes de seus respectivos ciclos se interceptam em um ponto no qual o dispositivo encontra-se processando a primeira tarefa a alcançá-lo. O cenário é ilustrado na Figura 5.2(a). Logo após A haver interrompido B no programa $B : C$, o próximo evento é uma tentativa de D

	Rp	Rf		Rp	Rf
A	0	B	A	0	B
B	A	0	B	A	C
C	0	0	C	B	0
D	0	0	D	0	0

(a) (b)

Figura 5.2: Potenciais situações de conflito na rede

de fazer o mesmo. Ocorre que essa mensagem encontraria sua receptora no estado ATENDE e, lembrando que o comportamento reativo da ID é idêntico ao de uma entidade servidora simples, não há por parte desta qualquer provisão para registro de pedidos pendentes — logo, B não tem condições de receber a interrupção.

Uma das estratégias para lidar com uma situação como essa, é fazer com que D espere para liberar sua mensagem até que B esteja livre para assimilá-la, utilizando o conceito que em outros domínios é conhecido como tentativa bloqueante: a entidade a executá-la permanece bloqueada até que seja possível executar a ordem expedida. Antes de aprofundar-se nessa possível solução, é conveniente notar que não é preciso tal coincidência para gerar um conflito. Este pode se dar ainda que na seqüência descrita, B já tenha feito a referência a C , como na Figura 5.2(b). Nesse caso, a julgar pelo estado de B , que deve estar em PRONTO, o dispositivo teria condições de receber o comando de D .

Todavia, se assim fosse feito, ambas as IDs B e C estariam em atendimento. Se C fosse então a primeira a terminar o processamento, não poderia se reportar a B e, logo, bloquear-se-ia até que esta fosse liberada — o que não tem chances de acontecer, desde que B também não pode referir-se a C pelo mesmo motivo e seria, portanto, igualmente bloqueada. O problema não melhora se B tentasse retornar primeiro: encontrando C indisponível, seria bloqueada, estado do qual jamais sairia posto que C faria o mesmo mais tarde. O nome consagrado pela literatura para essa trágico cenário é o sugestivo veredicto *deadlock* (travamento morto), cujos métodos de prevenção e recuperação estão entre os mais desafiantes temas de estudo nas áreas onde se manifesta.

A solução para esquivar-se dessa complicação adotada na primeira versão do sistema foi a mais simples, e também mais rude: abortar. O algoritmo de encaminhamento determina

que, se a receptora de uma mensagem encontrar-se indisponível no momento da entrega, a mensagem deve ser recusada com sinalização de erro para a emissora, informando-a de que o recurso solicitado está ocupado (tentativa não bloqueante).

A forma de recusar uma mensagem, seja porque o destino esteja ocupado ou porque o destinatário da interrupção é inexistente, inválido (o algoritmo previne um dispositivo de enviar mensagens para si mesmo, por exemplo), não autorizado para o remetente, ou qualquer outro motivo, é novamente através de uma ID virtual, o “dispositivo de sistema”. Quando uma requisição não pode chegar ao seu destino, ela é virtualmente redirecionada pelo GI a esse dispositivo (que é um avatar do próprio GI), o qual responde, então, ao emissor em nome do pretendido destinatário, informando a razão da interceptação.

O dispositivo de sistema na verdade representa a funcionalidade do GI disponível diretamente para as IDs. A idéia é de que estas possam endereçar-lhes comandos assim como o fazem para outros dispositivos, requisitando-lhe serviços tais como informar ou alterar o nome do dispositivo e alvo implícito ou solicitar desconexão, dentre outras ações.

Nesta versão do protótipo foram ainda ensaiadas algumas estratégias de controle de acesso, especificando quatro diferentes níveis de permissões para as IDs. Ao conectar-se à rede, o dispositivo entra necessariamente no nível de autenticação, associado ao valor 0. Nesse nível, a ID pode apenas se comunicar com o dispositivo servidor para solicitar-lhe a promoção para o patamar seguinte, mediante o fornecimento de uma senha. O segundo nível é o modo normal, de valor 1, no qual as IDs gozam de plena liberdade, exceto a de se comunicar com os dispositivos no próximo nível, o especial, cujo valor é 2.

Isso permite proteger partes críticas da rede contra o acesso potencialmente danoso (intencional ou oriundo de imperícia) por parte das IDs normais, de modo análogo ao recurso disponível em microprocessadores modernos que limitam as capacidades dos processos de usuário em relação aos do sistema. Quanto às permissões de acesso, o grupo especial é idêntico ao normal; apenas, para que um dispositivo do grupo 1 possa acessar o grupo superior, ele deve fazê-lo através de uma ID desta classe que tenha seu parâmetro *trust mode* (modo de confiança) habilitado. Esta está autorizada a receber mensagens do nível inferior e transmiti-las a qualquer dispositivo.

Finalmente o nível 3, ou nível mestre, permite a ID alterar também os parâmetros dos outros dispositivos, e serve para a intervenção do administrador no sistema ou para agentes de controle automatizados. Tais mecanismos, também conhecidos na Internet como *bots* (de *robots*, populares em redes de IRC³ e servidores de jogos interativos), são entidades de software que funcionam como agentes autônomos utilizados para monitoramento e regulação de ambientes multi-usuário. Na RDI, *bots* são úteis para implementar serviços de informação às outras IDs e exercer políticas de segurança.

O algoritmo de encaminhamento que realiza as regras descritas pode ser resumidamente esquematizado no fluxograma da Figura 5.3, onde a notação $x.y$ indica a propriedade y do objeto x , ao passo que $x.y.z$, recursivamente, refere-se à propriedade z do objeto $x.y$ e assim por diante. No diagrama, s (*sender*) é o endereço do emissor da mensagem; Rp e Rf dão informação de estado da representativa (*report*, *refer*); a (*addressee*) é o destinatário especificado e r (*receiver*), o receptor calculado pelo GI; S e N são os endereços dos dispositivos de sistema e nulo, respectivamente (*system*, *null*); d (*default*) é o alvo implícito; in é o comando na mensagem recebida (*input*) e out , o campo depois de modificado pelo GI (*output*).

Uma variação mais rudimentar do protocolo descrito na seção 4.8 foi utilizado, a qual terminou por mostrar-se deficiente, sendo, portanto, substituída na próxima versão.

5.1.2 Mecanismos da Interface de Dispositivo

Para construir a parte complementar que compõe o sistema, foi criada uma classe de objeto representando um modo básico de funcionamento de uma ID. Instâncias derivadas dessa classe foram, então, utilizadas na implementação de algumas interfaces especializadas, com as quais diversos testes foram realizados.

A ID constitui um dispositivo de tempo-real com dois canais de entrada. Um deles corresponde ao GI, do qual mensagens são recebidas e para o qual são retornadas após o devido processamento. O outro é o que interage com a aplicação em si — seja um equipamento eletrônico, quando a ID é uma interface de instrumentação, seja o operador

³Internet Relay Chat, redes de servidores de conversação escrita *on line* na Internet.

Figura 5.3: Algoritmo de encaminhamento (versão 1)

humano, quando é uma interface de usuário, ou qualquer outro sistema que se sirva do modelo para se comunicar com a RDI.

Uma questão prática que se impõe imediatamente é como atender a ambos os canais e tratá-los de maneira adequada. Na sua solução deve-se levar em conta, por exemplo, que ambas as fontes de eventos são assíncronas com respeito ao mecanismo da ID e podem gerar ocorrências independentemente. Assim, se a aplicação produz espontaneamente dados que devem ser convertidos em uma interrupção, a ID deve dedicar-se a recebê-los, pré-processá-los e encapsulá-los em uma mensagem apropriada a ser subsequente enviada à rede. Todavia, uma vez que a duração desse processo não é determinada (a leitura de um longo arquivo, ou a aquisição de uma sequência de amostragens a partir de um equipamento sensor) espera-se que nesse ínterim a ID permaneça ainda à espera de mensagens do GI e pronta a processá-los.

Se a plataforma de desenvolvimento o permitir, uma das estratégias para resolver o problema é realizar o atendimento de cada uma das fontes de entradas em linhas de execução paralelas. Nesta versão inicial do protótipo, as duas linhas de execução foram implementadas como dois processos distintos que se sincronizam mutuamente por meio de um dos

Saída de RDCS

```
[megryan]$ RDCS  
  
Remote Device Command Server - RDCS/1.0  
Starting....  
Adding virtual devices  
>Client 0 from has just connected  
>Client 1 from has just connected  
>Client 2 from has just connected  
>Client 3 from has just connected  
Server up and ready !
```

Figura 5.4: Saída inicial do módulo RDCS

mecanismos de IPC⁴ nativos do sistema — neste caso um *pipe*. A *linha da rede* bloqueia-se na espera de mensagens vindas do GI, ao passo que a *linha do operador* bloqueia-se enquanto a aplicação não gera dados a serem convertidos em comandos para a rede. A qualquer momento, cada uma das duas pode ser ativada pelo correspondente gerador de eventos, e deve então tratar ela mesma de receber os dados e processá-los.

Os detalhes de como esse esquema é realizado na linguagem de programação ficam adiados para quando da descrição da segunda versão do sistema, na qual as observações concernentes tornam-se mais relevantes.

5.2 Ensaios com Algumas Configurações

Uma série de ensaios realizados com o conjunto de programas proporcionaram subsídios para uma avaliação preliminar do modelo, seu desempenho, flexibilidade e principais limitações. Alguns desses experimentos acompanhados das conclusões que propiciaram são reportados nesta seção.

A Figura 5.4 mostra as primeiras saídas geradas pelo GI quando o programa é invocado. O módulo, então nomeado RDCS, inicializa o serviço de rede e ativa os quatro dispositivos que representam IDs virtuais (o dispositivo nulo, de sistema, despachador e replicador), das quais faz uso interno⁵.

O processo de conexão de uma ID ao módulo pode ser acompanhado passo a passo

⁴IPC: *inter-process communication*, mecanismo de comunicação inter-processos.

⁵Detalhes do funcionamento das IDs virtuais não são apresentados no texto, uma vez que a versão final substitui esse expediente por um mecanismo mais simples e eficiente.

Saída de RDCS	
<pre> -----BOC (1)... Processor got: RDCP ANYONE SERVER ACTION CONNECT Comming from : DELIVER (1) AL:3 Adressed to : SERVER (3) AL:3 Sender status: ff=1 ft=3 fW=0 fR=0 fdt=1 ftW=0 ftR=0 fdtR=0 Delivering... This goes to : SERVER (3) >Server processing : RDCP DELIVER SERVER ACTION CONNECT command : ACTION argument 1 : CONNECT >Client 4 from 143.107.235.62 has just connected >Server response : RDCP SERVER DELIVER 200 Client added Processor got: RDCP SERVER DELIVER 200 Client added Comming from : SERVER (3) AL:3 Adressed to : DELIVER (1) AL:3 Sender status: ff=3 ft=1 fW=0 fR=1 fdt=1 ftW=3 ftR=0 fdtR=0 Answering... This goes to : DELIVER (1) Processor got: RDCP ANYONE SERVER 200 Thank you Comming from : DELIVER (1) AL:3 Adressed to : SERVER (3) AL:3 Sender status: ff=1 ft=3 fW=3 fR=0 fdt=1 ftW=0 ftR=0 fdtR=0 Answering... This goes to : NULL (0) ----- ...EOC (1) </pre>	<pre> >Client 4 has just disconnected -----BOC (4)... Processor got: RDCP ANYONE SERVER ACTION DISCONNECT ME Comming from : CLIENT4 (4) AL:0 Adressed to : SERVER (3) AL:3 Sender status: ff=4 ft=3 fW=0 fR=0 fdt=1 ftW=0 ftR=0 fdtR=0 The interface is locked Delivering... This goes to : SERVER (3) >Server processing : RDCP CLIENT4 SERVER ACTION DISCON- NECT ME command : ACTION argument 1 : DISCONNECT argument 2 : ME ...EOC (4) >Server response : RDCP SERVER NULL 200 OK -----BOC (3)... Processor got: RDCP SERVER NULL 200 OK Comming from : SERVER (3) AL:3 Adressed to : (not found) (0) AL:3 Sender status: ff=3 ft=0 fW=0 fR=0 fdt=1 ftW=0 ftR=0 fdtR=0 Delivering... This goes to : NULL (0) ----- ...EOC (3) </pre>

Figura 5.5: Processo de conexão e desconexão no módulo RDCS

na Figura 5.5, que mostra a saída do GI gerada por esse evento. No esquema, que foi posteriormente modificado na versão seguinte, uma ID virtual despachadora (*deliver*) age como anfitriã da recém chegada, solicitando ao dispositivo de sistema a sua conexão. A desconexão prematura do cliente é mascarada como um pedido normal de desligamento do dispositivo, enviado por este ao sistema; a resposta do sistema é entregue ao dispositivo nulo para cumprir o protocolo. A decomposição das mensagens trocadas internamente pelos dispositivos e as ações internas do GI são mostradas no exemplo.

A fim de compor uma pequena RDI, uma coleção de programas na forma de IDs foi escrito. Fornecendo interfaces de instrumentação para alguns equipamentos disponíveis no laboratório de pesquisa, o conjunto inclui um software que retorna a leitura do relógio interno do computador, um dispositivo que exporta a funcionalidade da porta de comunicação serial RS-232 e um mecanismo que acessa diretamente as instruções de acionamento do dispositivo de CD-ROM — as referências para programação com acesso ao hardware em sistema GNU/Linux podem ser consultadas em [77], [78], [79] e [80], assim como diretamente no arquivo de cabeçalho da biblioteca de sistema *sys/ioctl.h*. Em complemento, algumas interfaces gráficas de usuário foram construídas para operar esses equipamentos.

A Figura 5.6(a) mostra uma interface genérica escrita em Java — rodando no ambiente gráfico X sobre a máquina virtual *kaffe* no sistema GNU/Linux — através da qual se pode interagir com dispositivos na RDI. O botão *Configure* dá acesso a uma janela de diálogo que permite configurar o endereço do GI (IP ou nome do servidor) e a senha para o nível de acesso normal. A mesma aplicação pode ser vista na Figura 5.6(b) capturada do monitor de uma estação Sparc (SO Solaris), de onde um dos membros da equipe acadêmica telecomanda o “jukebox” — este, implementado pela ID controladora do equipamento de CD-ROM instalado em um PC, que funciona como servidor de som ambiente para o recinto. Os comandos são digitados na caixa de texto *Input*, e disparados através do botão *Send*. O interesse na programação musical do laboratório popularizou o aplicativo, o qual passou a ser utilizado também em plataforma Windows, como ilustra a Figura 5.6(c). O diagrama da Figura 5.7 descreve a organização da aplicação.

Não obstante tenham sido feitas comparações entre sistemas de teleoperação em redes telemáticas baseados no mecanismo Web/CGI e no modelo ora em exame, vale notar que a tecnologia da RDI não exclui a da Web. A fim de provar o alcance da assertiva, e atendendo a indagações que se tem feito a esse respeito quando das discussões sobre este estudo, demonstra-se que uma ID pode ser criada como uma mini-aplicação e embutida dentro da interface Web, como o arranjo mostrado na Figura 5.8. O pequeno relógio marcando a hora corrente é um *applet* cuja cor de fundo se confunde com a da página Web onde foi carregado pelo trecho mostrado abaixo, contido no código HTML do documento. A ID cliente interrompe a servidora (dispositivo relógio), localizada em outro computador na LAN — note-se, portanto, a diferença entre implementar o telecomando como uma aplicação Web, por um lado, e pelo outro, utilizar a Web como interface.

O entusiástico interesse despertado entre usuários que experimentam o acesso à sistemas de teleoperação via Internet é uma constatação anotada por autores de diversos trabalhos nessa área, levando-os a concluir que há um consenso bem assimilado de que essa tecnologia constitui não o anúncio de uma possibilidade futura, mas uma conquista em vias de consumação e até mesmo como uma meta urgente. A experiência inicial com o módulo RDCS corrobora a observação e, afim de propiciar sua exploração, criou-se uma segunda versão do cliente jukebox utilizando a Web.

(a)

(b)

(c)

Figura 5.6: Uma ID como interface de usuário gráfica

Figura 5.7: Uma aplicação simples em RDI

Figura 5.8: Uma mini-aplicação ID embutida em uma página Web

Figura 5.9: Um dispositivo complexo via interface Web

A Figura 5.9 mostra o painel de comando embutido em uma página HTML disponibilizada pelo servidor Web do laboratório, através da qual usuários na rede têm acesso aos controles da interface de instrumentação do CD-ROM (mais tarde, colaboradores da equipe decidiram instalar um servidor de áudio junto àquele equipamento para que os operadores externos pudessem também apreciar a seleção musical via Internet, o que na época ainda se apresentava como uma curiosa novidade).

Este experimento é na verdade um exemplo de dispositivo complexo (seção 4.4, página 47). Cada um dos controles — os três botões e os dois campos de texto — é na verdade uma ID distinta, e todas elas são conectadas na RDI. São, portanto, 6 *applets* carregados independentemente dentro da página Web. Os campos de texto interrompem a ID do equipamento a cada segundo, requisitando seu estado atual (se está reproduzindo alguma trilha do CD e a quanto tempo o faz). Os botões permitem enviar instruções de acionamento específicas para iniciar, suspender, continuar e interromper a reprodução. Durante algum tempo os estudantes e pesquisadores no laboratório puderam apreciar a estação musical operada por colegas e anônimos na Internet, os quais retribuíram com comentários descrevendo suas impressões.

Um exemplo mais complexo, utilizando a programação do GI foi montado a partir de uma pequena coleção de IDs especializadas distribuídas entre alguns computadores dentro da LAN do laboratório (Figura 5.10). Ao GI conectam-se as quatro IDs indicadas no diagrama. O equipamento comandado é uma pequena rede de sensoramento e atuação constituída de alguns módulos passivos interconectados por meio de um barramento de comunicação serial. O conjunto, adquirido de um fornecedor de equipamentos de automação industrial [81], inclui um conversor RS-232/RS-485 (ADAM-4520), conectado à porta de

Figura 5.10: Um exemplo de programação na RDI

comunicação do PC, como mostra o diagrama, e um módulo de entrada-saída conversor serial/paralelo (ADAM-4050), na outra extremidade do barramento.

Ao último módulo foi, então, conectado um circuito eletrônico contendo três LEDs⁶, cada um ligado a uma das três primeiras dentre as oito saídas da porta paralela do dispositivo. Mediante cadeias de caracteres ASCII⁷ enviadas pelo computador através do conversor serial, segundo o protocolo específico do equipamento, é possível ativar independentemente cada um dos diodos, obtendo-se, portanto, oito combinações diferentes.

Para interfacear com equipamento foi desenvolvida uma ID capaz de acessar as funções de controle da UART⁷ que comanda a porta RS-232 do computador, ao qual o módulo é conectado. O programa aceita comandos para informar e alterar as configurações básicas da porta, tais como taxa de transmissão, controle de fluxo, tamanho da palavra, paridade e número de bits de parada — instruções que podem ser enviadas através de uma segunda ID para esse fim produzida, a qual oferece os controles em um painel gráfico (Figura 5.11).

Além das instruções de configuração, a ID controladora da porta serial recebe ordens para o envio e recepção de dados, os quais lhe podem ser enviados pelo operador através de qualquer uma das interfaces descritas. O comando para escrita é dado pela função SEND.BYTES, que aceita uma lista de argumentos de comprimento variável contendo o valor decimal de cada byte a ser enviado, e produz o comando OK como saída, se bem sucedido. A leitura é solicitada pela função READ.CHARS que toma como argumento o número de bytes a serem recebidos da linha serial. No barramento, o protocolo utiliza

⁶LED: *light emission diod* (diodo emissor de luz).

⁷UART: *universal asynchronous receiver-transmitter*, (transmissor-receptor universal assíncrono).

Figura 5.11: Interface gráfica para configuração da porta RS-232

Entrada	Saída	Padrão
#010000 (cr)	00	000
#010000 (cr)	01	001
#010000 (cr)	02	010
#010000 (cr)	03	011
#010000 (cr)	04	100
#010000 (cr)	05	101
#010000 (cr)	06	110
#010000 (cr)	07	111

Figura 5.12: Instruções enviadas para o módulo ADAM

mensagem de 8 bytes, iniciando com uma assinatura “#” (ASCII 35) seguido da representação em dois dígitos do endereço do destinatário, máscara de inibição e valor de saída, e terminando com um caracter CR (ASCII 13); o módulo responde com a sequência “>CR” (ASCII 62 13). Sendo 01 o endereço do módulo serial/paralelo, e representando um LED aceso por 1 e apagado por 0, a Figura 5.12 relaciona a cadeia de caracteres a ser enviada para obter a respectiva combinação de saída.

Mediante a adição de um dispositivo intermediário entre a interface de operação e de instrumentação, contudo, é possível oferecer ao usuário um conjunto de funções mais conveniente do que o fornecido pelo equipamento. Um tradutor de comandos foi criado para essa finalidade; sua missão é converter determinados comandos recebidos como entrada para correspondentes comandos de saída. A tabela de tradução, transcrita na Figura 5.13, é fornecida na forma de um arquivo de texto que o programa lê quando da sua invocação.

Ao conectar-se à rede, a ID tradutora inicialmente solicita ao dispositivo de sistema a

Excerto de *translation.txt*

```
# Translation Table
# ADAM 400 -> leds

# Patterns

TURN 0 SEND.BYTES 23 30 31 30 30 30 30 13
TURN 1 SEND.BYTES 23 30 31 30 30 30 31 13
TURN 2 SEND.BYTES 23 30 31 30 30 30 32 13
TURN 3 SEND.BYTES 23 30 31 30 30 30 33 13
TURN 4 SEND.BYTES 23 30 31 30 30 30 34 13
TURN 5 SEND.BYTES 23 30 31 30 30 30 35 13
TURN 6 SEND.BYTES 23 30 31 30 30 30 36 13
TURN 7 SEND.BYTES 23 30 31 30 30 30 37 13

# Some useful comandas

TURN ALL SEND.BYTES 23 30 31 30 30 30 37 13
TURN NONE SEND.BYTES 23 30 31 30 30 30 13

# Run

SEQ ZERO SEND.BYTES 23 30 31 30 30 30 30 13
SEQ ONE SEND.BYTES 23 30 31 30 30 30 31 13
SEQ TWO SEND.BYTES 23 30 31 30 30 30 32 13
SEQ FOUR SEND.BYTES 23 30 31 30 30 30 34 13

# Read

GET REPLAY READ.CHARS 1
```

Figura 5.13: Tabela de tradução para o experimento com LEDs

sua promoção para o nível normal e depois para o nível especial. Em seguida, registra-se sob o nome de “LEDS” e requisita a configuração do seu alvo implícito apontando-o para a controladora da porta de comunicação (de nome SERIAL), também no nível de acesso especial, o que resulta no programa LEDS:SERIAL. O dispositivo LEDS ordena a ativação do seu modo de confiança (*trust mode*) e torna-se, assim, acessível ao usuário no nível normal. A partir desse ponto, o operador pode, no lugar de submeter diretamente ao dispositivo SERIAL as cadeias de caracteres correspondentes a cada padrão de ativação dos diodos, enviar instruções mais abstratas ao dispositivo LEDS solicitando ações como “acender todos os LEDS” ou “acender o terceiro LED”. A conversão para a sequência apropriada a ser enviada para SERIAL é realizada e o comando é executado pelo equipamento.

5.3 Algumas Conclusões

5.3.1 Limitações Anotadas

Essas e algumas outras montagens experimentais foram realizadas no intuito de ilustrar algumas configurações de interesse em aplicações reais do modelo RDI. Por meio delas foi possível conhecer algumas limitações importantes que os mecanismos implementados apresentam.

A mais grave é seguramente a precária estratégia de resolução de conflitos que, dependendo das características do sistema onde é empregada, revela-se absolutamente impraticável. Com efeito, abortar uma tarefa nem sempre pode ser considerado um procedimento viável em qualquer parte do ciclo de operação e, em muitos casos, pode ter conseqüências tão catastróficas quanto o que seria causado pelo conflito que a técnica objetiva evitar. Isso porque o ciclo de operação segue acionando cada dispositivo ao longo da cadeia que percorre, e, assim, potencialmente causando modificações no ambiente a cada passo da fase ascendente e descendente. O efeito líquido de todas as ações executadas constitui o resultado da tarefa e espera-se que este seja consistente. Se o ciclo for interrompido prematuramente antes que a tarefa planejada chegue a termo, contudo, é possível que a resultante das ações efetivadas até aquele ponto deixe o meio em um estado inconsistente, correspondente à tarefa inacabada.

Na implementação descrita, um ciclo de operação pode ser abortado apenas na sua fase ascendente e, portanto, o que ocorre devido a uma tarefa abortada depende dos dispositivos acionados. A leitura de um sensor, por exemplo, é normalmente uma sub-operação que, se frustrada, não resulta em grande transtorno a não ser, possivelmente, a necessidade de ser repetida posteriormente. Já a atuação sobre mecanismos externos, como o acionamento de um equipamento, se abortada é susceptível de conseqüências mais graves. Uma situação onde a hipótese se realiza pode ser imaginada em um programa onde um ou mais dispositivos intermediários ativem funções de determinado mecanismo (abertura de uma válvula, p.ex.), as quais devem ser desativadas pelos dispositivos que os sucedem na cadeia. Se o programa não for completado com sucesso, aquelas funções permanecerão ativadas — a dimensão do problema que isso pode causar depende evidentemente da aplicação, mas em

geral são indesejáveis.

Com relação a esse detalhe, uma virtude do esquema utilizado é que as cadeias são progressivamente desfeitas no transcorrer da fase descendente do ciclo, à medida em que cada ID reporta-se à sua predecessora. Enquanto isso minimiza o tempo em que uma cadeia permanece estabelecida e diminui, por conseguinte, a probabilidade de colisão entre tarefas de programas que se interceptam, a medida não elimina a chance de que ainda assim o conflito aconteça, a qual tende a aumentar em função da extensão dos programas e do número de ciclos concomitantemente ativos.

É pertinente também observar que tal colisão não implica necessariamente em desastre, posto que em alguns casos é perfeitamente aceitável que duas cadeias se interceptem, desde que o ponto da cadeia onde o encontro ocorra o permita. Por exemplo, um dispositivo sensor pelo qual uma tarefa já passou, pode ser consultado novamente pela tarefa seguinte sem que isso cause qualquer problema. Logo, além de inadequada para um sistema de propósito geral, a técnica de abortar tarefas, mesmo quando aceitável, mostra-se altamente ineficiente, bem como a alternativa baseada em tentativas bloqueantes.

Essas razões, somadas ao fato de que o mecanismo não oferece facilidades para explorar técnicas de replicação úteis na elevação da confiabilidade e disponibilidade de recursos, levaram a uma série de reformulações, as quais resultaram no desenvolvimento de um protótipo mais elaborado, cuja descrição é o tema da seção seguinte.

5.3.2 Um Protótipo Aprimorado

Em ambientes multi-usuário de maneira geral, sabe-se que, dependendo de características inerentes à sua natureza, alguns dentre os recursos disponíveis prestam-se adequadamente ao acesso compartilhado, i.e. à utilização concomitante por dois (ou mais) processos distintos. Por “concomitante” entende-se que, não obstante apenas cada um dos usuário tenha efetivamente acesso ao recurso a cada instante, este serve-lhes a todos alternadamente na execução de seus respectivos afazeres — operação de leitura, no âmbito de sistemas de software, ou sensoramento e aquisição de dados, no campo da tele-instrumentação, pertencem a essa categoria.

Como contra-exemplo, quando executando funções de escrita de dados ou tele-acionamento, respectivamente em seus domínios, recursos de uso comum precisam ser alocados exclusivamente para um único usuário a fim de evitar conflitos. O cenário que resulta da não observância dessa necessidade é freqüentemente denominado condição de disputa (*race condition*), a qual tem lugar quando seus participantes entram simultaneamente em uma região crítica; complementando o jargão técnico, a palavra-chave na sua prevenção é a exclusão mútua. No contexto do presente estudo e da terminologia empregada, isso significa que duas tarefas que se interceptam e que podem acionar desordenadamente um mesmo dispositivo de maneira incompatível devem ser proibidas de fazê-lo.

A tática adotada para esquivar-se desse perigo fora simplesmente abortar a tarefa mediante a iminência de um potencial conflito — uma medida inaceitável no caso geral, conforme argumentado. A fim de evitá-la, a alternativa inicialmente cogitada é a baseada na tentativa bloqueante, que suspende o emissor da mensagem até que o receptor esteja livre. Essa estratégia, no entanto, torna-se ineficiente no caso anterior se duas tarefas forem impedidas de se interceptarem em uma região não-crítica (um dispositivo compartilhável), o que sugere a possibilidade de permitir que um mesmo dispositivo possa receber mensagens consecutivas de clientes diferentes, mesmo estando envolvido em ciclos de operação ainda incompletos.

É preciso notar, todavia, que se isso for permitido, o mecanismo de comunicação, da forma como implementado, faz com que em determinado momento haja dois dispositivos necessitando trocar mensagens entre si em uma condição de espera circular. Esta, juntamente com a exclusão mútua (somente uma das IDs pode liberar a sua mensagem no passo seguinte) e a condição de posse-e-espera (*hold and wait*) em que se encontra os emissores⁸, necessitam apenas de mais um item a fim de completar a quadra de requisitos [82] *apud* [52] para a ocorrência de *deadlocks*. Infelizmente, este também se faz presente: o de não preempção, o que significa que o algoritmo utilizado não permite que um dispositivo tome à força os recursos sob posse de outro dispositivo — embora o *deadlock* que tem lugar nesse caso não é especificamente de recursos, mas de comunicação, este pode ser sempre

⁸A tarefa toma posse do dispositivo obtendo o *lock* (trava), e põe-se a esperar pela liberação do próximo dispositivo na cadeia.

modelado como tal [52].

É um consenso entre autores na área que o tratamento de *deadlocks* costuma ser complicado e custoso, tão custoso que até, em algumas circunstâncias, pode não ser sequer compensador, como propõe a solução do “avestruz” [52]. Em outros casos são aplicáveis algoritmos de detecção e recuperação que, face à confirmação de um travamento, buscam restaurar o sistema a um estado anterior, por exemplo, fornecendo provisões para que um dos processos possa ser abortado e reiniciado a partir de determinado ponto. Quando certas condições são satisfeitas pode-se mesmo tentar evitar que *deadlocks* aconteçam, com base em uma estimativa da utilização de recursos por parte de cada processo, embora freqüentemente essa previsão seja precária. O procedimento usual é portanto a prevenção, que se faz eliminando uma das quatro condições mencionadas.

A condição de posse-e-espera circular é intrínseca do mecanismo construído, ao passo que a não-preempção não pode ser violada sob pena de restringir a generalidade da aplicação. Resta, pois, evitar a exclusão mútua, se possível. No sistema em questão, o impasse ocorre entre duas entidades tentando enviar uma mensagem uma à outra, onde nenhuma delas tem condições de receber a entrega da parceira enquanto não realizar a sua própria. Isso deve ocorrer invariavelmente se for permitido que duas cadeias se interceptem, posto que a partir desse ponto ambos os programas são necessariamente coincidentes (lembrando, no modelo, cada representativa tem um alvo implícito pré-estabelecido e é este que aponta para o próximo dispositivo do programa).

Há, porém, uma saída para evitar a exclusão mútua utilizando-se primitivas de comunicação “bufferizadas”⁹. Se o receptor da mensagem dispuser de um *buffer* (espaço temporário de armazenamento) para receber mensagens mesmo quando de posse de uma entrega pendente, então é possível habilitar a comunicação assíncrona, i.e. não bloqueante, entre os dispositivos — em verdade, não apenas nesse caso, mas inclusive naquele onde o destinatário final encontra-se ocupado (ID no estado ATENDE). Mantendo o modelo original das IDs como um servidor simples, o *buffer* para cada uma delas pode ser proporcionado internamente pelo GI através do artifício conhecido como caixa postal (*mailbox*). Se estas

⁹Utilizando *buffers*; anglicismo informal bastante difundido, sem tradução usual que lhe preserve a imediata clareza de seu significado técnico neste contexto restrito.

tiverem capacidade suficiente para acomodar o fluxo de comunicação entre os dispositivos, então a possibilidade do *deadlocks* é afastada.

Basicamente este é o princípio adotado na segunda versão do sistema-piloto. Cabe notar, no entanto, que esse novo mecanismo torna-se sensivelmente mais sofisticado que o anterior, não apenas em função da necessidade de manter estruturas de dados internas, mas também em virtude do modo consideravelmente complexo com que estas devem ser manipuladas. Em algumas aplicações onde o método de armazenamento temporário de processamentos pendentes é utilizada, tal como o subsistema de impressão disponível na maioria dos sistemas operacionais modernos, a técnica conhecida como *spooler*¹⁰ prova-se razoavelmente simples e eficaz: as ordens são armazenadas sequencialmente em uma fila, de onde são posteriormente retiradas pelo executor do serviço, uma a uma, conforme o trabalho vai sendo completado. No caso particular em questão, infelizmente, a solução não é tão direta, uma vez que a ordem na qual as mensagens devem ser retiradas do *buffer* não é necessariamente como em um FIFO (*first in, first out*; primeiro a entrar, primeiro a sair).

Se o dispositivo estiver ocupado, não está apto a receber mensagens; caso contrário, uma mensagem da sua caixa postal deve ser selecionada para envio. Nesse caso, se o dispositivo não estiver envolvido em nenhuma cadeia, qualquer mensagem é elegível; senão, é preciso levar em conta o estado de sua representativa. Se este indicar que o dispositivo pertence a uma cadeia descendente (supondo que a cadeia não tenha sido desfeita), então sua contribuição no ciclo foi completada e o dispositivo pode ser requisitado para outra tarefa. Se estiver no ciclo ascendente, então é necessário determinar a possibilidade de acesso compartilhado, o que depende de como a mensagem a ser entregue pode interferir com a tarefa em execução. Se as condições não o permitirem, apenas respostas (mensagens vindas do seu sucessor na cadeia) lhe podem ser entregues; de outro modo, outras mensagens são elegíveis desde que a função que solicite não resulte em conflito com a tarefa corrente. Podem ainda sobrepor-se a essas determinações aquelas impostas pela política de prioridade, tais como a de dar segmento imediato a mensagens urgentes do dispositivo de sistema, bem como por mecanismos de reserva exclusiva de recursos para sessões duradouras. Ainda,

¹⁰ *Spool*: a tradução literal seria “carretel”, uma analogia que cabe bem à técnica a qual se refere.

em caso de desconexão prematura de um dado dispositivo ou na desautorização do envio de determinadas mensagens, é necessário providenciar-lhes um destino adequado.

O processo para se chegar a um algoritmo que traduza todas essas regras pode resultar admitidamente complicado, dependendo da perspectiva pela qual for conduzido. A abordagem adotada neste estudo, a qual se provou bastante conveniente, foi um caminho essencialmente diferente daquele que se acabou de descrever. No lugar de considerar que o GI mantém uma fila de requisições pendentes para cada representativa, concebeu-se o modelo original bem simples, sem provisões para *buffers*, e assumiu-se um cenário hipotético onde convenientemente tudo ocorresse espontaneamente da melhor maneira possível. Evidentemente, para efeito de aplicação prática, a suposição não é de modo algum realística; o artifício adotado está em fazer com que assim o seja, pelo menos virtualmente. Isso é conseguido concebendo um ambiente dual para a representação interna do conjunto de IDs conectadas à rede, o qual pode ser esquematicamente comparado com as duas esferas concêntricas da Figura 5.14. Na camada mais externa tem-se uma correspondência biunívoca entre IDs (clientes do GI) e representativas (contrapartes daquelas no modelo do GI). No nível interior, constrói-se um espaço virtual onde um conjunto variável de entidades efêmeras dinamicamente criadas, associadas e destruídas, interagem como avatares daquelas representativas de forma ideal. De posse do conhecimento de todos os programas e de um número suficiente de avatares, o GI pode simular, nesse âmbito, todas as tarefas em desenvolvimento e, a partir daí, transferir o resultado para as representativas. Sinteticamente, o truque é que, se estiver ocupada no instante em que deve receber uma mensagem, uma entidade no plano de simulação pode ser clonada quantas vezes seja necessário a fim de assimilar a demanda; por conseguinte, cadeias não se interceptam. Cada grupo de avatares serve coordenadamente a uma ou mais representativas, às quais está vinculado, podendo eventualmente migrar para outro grupo.

A questão fundamental nesse sistema é que um ciclo de operação é completado através do concurso de uma série de dispositivos, cada qual contribuindo com o serviço que lhe cabe. Deseja-se que uma tarefa iniciada seja efetivamente completada ou, se isso não for possível, que ela não seja absolutamente executada; tarefas prematuramente terminadas podem resultar em inconsistência e, logo, devem ser evitadas. Espera-se também que

Figura 5.14: Espaço dual de representação interna no GI

tarefas concorrentes não se interfiram de forma incompatível, o que equivale a dizer que o resultado de um conjunto de ciclos de operação concomitantes deve ser análogo ao efeito líquido produzido por cada um deles independentemente, de uma forma consistente. Em termos sucintos, trata-se de lidar com falhas e concorrência.

Assim declaradas, ambas as especificações identificam-se diretamente com o conceito bem conhecido de *transações*. Valendo-se da terminologia consagrada na literatura, a primeira corresponde à propriedade da *atomicidade*, a qual faz uma transação parecer indivisível com relação ao efeito provocado: apenas os estados iniciais e finais são possíveis; os intermediários não são visíveis. A segunda é usualmente referida por *isolamento* ou *serializabilidade* e é expressa exigindo que haja uma ordem na qual transações simultâneas possam ser executadas seqüencialmente com o mesmo resultado. Completa a trinca de propriedades a *permanência*, ou *durabilidade* [83], garante que, uma vez finalizada a transação, seu efeito é definitivo.

É preciso ressaltar que o protótipo construído não foi elaborado com vistas a garantir tais propriedades, e sequer foi submetido a qualquer prova formal de correção, tampouco à verificação experimental exaustiva; o que se procurou criar são mecanismos básicos que ofereçam a possibilidade da implementação do modelo de transações em investigações futuras. A explicação de como opera o arranjo descrito pode ser melhor contemplada se ilustrada no sistema implementado através da descrição de seu código fonte, onde o algoritmo e as estruturas lógicas podem ser visualizados de maneira concreta.

Capítulo 6

Implementação

A implementação definitiva do sistema-piloto foi completamente escrita a partir do início sem o aproveitamento de qualquer parte do código anterior, desta vez inteiramente em linguagem C. Favorecendo a legibilidade do programa, a medida logrou torná-lo mais adequado à finalidade didática a que se propõe o esforço, resultando em um código mais simples e claro, que evidencia mais de perto os detalhes da tecnologia (*software* e *hardware*) que se deseja explorar. A empreitada proporcionou também a oportunidade para aprimorar o sistema mediante o acréscimo de recursos mais elaborados, desenvolvidos a partir de contribuições angariadas na forma das valiosas sugestões recebidas, as quais possibilitam estender as capacidades do ambiente virtual de trabalho da RDI.

Como plataforma de desenvolvimento, o sistema operacional Debian GNU/Linux [84] foi considerado adequado para o propósito do projeto. Seu núcleo, o *kernel* Linux [85], reputado tanto no meio acadêmico quanto industrial em função de sua eficiência, estabilidade e flexibilidade, proporciona um ambiente tipo unix, multi-tarefa, multi-usuário cujas características são singularmente convenientes para aplicações em rede, em comparação com outros em sua categoria — razão pela qual tem sido amplamente adotado como recurso em pesquisas científicas em diversas áreas, dentre elas em vários dos experimentos com telecomando via Internet descritos. A coleção de software do projeto GNU [86] compõe, em conjunto com o núcleo, a base do sistema operacional, proporcionando a biblioteca de sistema *glibc* [87], o compilador *gcc* [88] e diversas ferramentas de desenvolvimento

avançadas, como o gerenciador de compilação automática *GNU make*, o “debugger” *gdb* — todos intensivamente utilizados na construção do protótipo da RDI, bem como alguns recursos auxiliares oriundos de outras fontes, tais como o sistema de controle de versões *CVS* [89], dentre outros.

Todos esses e demais programas utilizados no desenvolvimento do sistema experimental classificam-se como software livre, segundo definição da FSF¹ e, portanto, baseiam-se em código-fonte aberto e padrões não proprietários, o que lhes isenta de eventuais restrições técnicas ou legais quanto a sua livre utilização, subsequente aprimoramento e distribuição. Além de contar com simples instalação e configuração, o sistema operacional Debian encontra-se dentre as mais completas e consistentes distribuições GNU/Linux; pode ser obtido gratuitamente através da Internet e é amplamente documentado, o que lhe confere grande vantagem para utilização em pesquisa. O programa faz uso exclusivo de recursos normais da biblioteca compatíveis com o padrão ANSI/ISO C e POSIX² e eventuais extensões do sistema GNU, o que possibilita que seja facilmente portado para plataformas compatíveis [80] — de fato, as ferramentas selecionadas permitem o porte para variadas arquiteturas de *hardware*, inclusive para sistemas microcontroladores embutidos.

Para a finalidade do estudo, tão relevante quanto a apresentação da interface de usuário e de amostragem de dados é a explicação precisa acerca da interação entre as diversas partes que compõem o software, razão pela qual a atenção deste capítulo se concentra na descrição pormenorizada da sua implementação — proporcionando, dessa forma, a informação necessária não apenas para sua efetiva compreensão, mas também para candidatos eventualmente interessados em aprofundar a investigação, estendendo ou modificando o sistema.

Todavia, não obstante tenha-se buscado seguir as recomendações do *GNU Coding Standards*, que pretere o convidativo estilo conciso das construções da linguagem C em favor da clareza e da abundância de comentários, trata-se, ainda assim, de um programa consideravelmente complexo, com algumas estruturas de dados dinâmicas não muito simples e

¹Free Software Foundation, Fundação Software Livre <http://www.fsf.org>.

²Portable Operating System Interface (Interface de Sistema Operacional Portável), padrão ISO/IEC 4495, ANSI/IEEE 1003.

três linhas independentes de execução concorrentes, as quais interagem em tempo real com uma coleção de outros processos independentes através da rede, necessitando de alguns cuidados com sincronização que não são necessariamente intuitivos. Assim, para que o trabalho tenha serventia, nas seções seguintes, procede-se à descrição comentada do código fonte que constitui o software, aproveitando-se pontos oportunos para discutir a respeito da tecnologia atualmente ao alcance da aplicação em relevo.

O sistema, nomeado IDN (*Interactive-Device Network*), é disponível³ como software livre nos termos da licença GPL⁴ [90] e por essa razão, segundo a prática consagrada pela comunidade que suporta a filosofia de desenvolvimento aberto cooperativo, aderiu-se à recomendação de utilizar o idioma inglês para a confecção do código fonte e comentários (onde o tradicional estilo informal se faz respeitado) como forma de facilitar a interação entre possíveis futuros contribuidores. O programa foi desenvolvido sobre o sistema Debian GNU/Linux 2.2r3, *kernel* 2.2.18, *gcc* 2.95.2, *glibc* 2; é válido anotar que o sistema carece ainda de ser adequadamente testado e é, portanto, passível de conter ainda inconsistências (i.e. “bugs”) não detectadas.

Quando da sua referência no texto, todas as funções, variáveis e tipos de dados implementados pelo protótipo podem ser distinguidos pelo prefixo `idn_`; os demais símbolos são variáveis auxiliares ou funções da biblioteca do sistema.

6.1 O Gerenciador de Interfaces

O software que implementa o GI compõe-se de uma coleção de vários módulos distintos (cerca de 30 arquivos, perfazendo aproximadamente 4 mil linhas de código C), cada qual implementando uma função específica no programa. No texto que se segue, os números anotados na margem direita fazem referência às listagens fornecidas e dão a indicação das linhas a que se refere a respectiva passagem. O acompanhamento das discussões através desses excertos contribui para a compreensão das descrições pertinentes, além de

³A versão comentada neste volume encontra-se disponível no repositório CVS do projeto; informações complementares no endereço <http://winona.sel.eesc.sc.usp.br/idn>.

⁴GNU General Public Licence, (Licença Pública Geral GNU), <http://www.gnu.org/copyleft/gpl.html>.

Excerto de *idn/network.h*

```
1 /* network.h - network-related structures and functions
2 *
3 *      $RCSfile: network.h,v $ $Revision: 1.20 $
4 *      IDN Project, SEL/EESC-USP.
5 *      Refer to idn.txt for further information.
6 */
7
8 #ifndef _IDN_NETWORK
9 #define _IDN_NETWORK
10
11 #include <sys/socket.h>
12 #include <netinet/in.h>
13
14 /* A structure to represent the TCP server parameters. */
15
16 struct idn_server
17 {
18     struct sockaddr_in address; /* server socket's address */
19     int socket_fd; /* server socket's file descriptor */
20     fd_set active_fd_set; /* file descriptors being monitored */
21 };
22
23 typedef struct idn_server idn_server_t;
```

Figura 6.1: Parâmetros operacionais do servidor

proporcionar informações suplementares anotadas na forma de comentários no código-fonte.

6.1.1 Parâmetros do Servidor

16-21 No que concerne à rede real da arquitetura RDI (seção 4.3, página 39), o GI desempenha as funções elementares de um típico programa servidor, cujos parâmetros operacionais são reunidos na estrutura auxiliar *idn_server*, declarada no arquivo *network.h* (Figura 6.1); todos os procedimentos relativos à comunicação através da Internet são realizados com referência a essa estrutura.

No domínio das aplicações em redes TCP/IP, o recurso básico oferecido pela maioria das plataformas de desenvolvimento, desde em arquiteturas orgânicas até sistemas embutidos, é a API conhecida como *socket*, derivada do sistema 4.2 BSD (Berkely Unix, 1983) [91], hoje uma especificação POSIX. Por meio desse recurso a biblioteca de programação dá acesso ao PAS (ponto de acesso ao serviço [65]) da camada de transporte (PAST), possibilitando
18 ao processo⁵ registrar um endereço exclusivo junto ao sistema operacional: quando do estabelecimento de uma conexão TCP, dados recebidos neste endereço são transferidos para o processo associado; dados enviados através desse endereço são destinados à entidade parceira na outra extremidade do canal. A comunicação é realizada através das primitivas

⁵Processo é o nome genérico para designar um programa em execução no computador.

Excerto de *idn/network.h*

```
25 /* Macros to represent kinds of network events */
26
27 #define IDN_NEW_CONNECTION      0
28 #define IDN_NEW_MESSAGE        1
29 #define IDN_DISCONNECTION      2
30 #define IDN_NETWORK_ERROR      3
31 #define IDN_NETWORK_INTERRUPTED 4
32
33 /* A structure to represent a network event. */
34
35 struct idn_network_event
36 {
37     int type;          /* the kind of event */
38     int socket_fd;     /* associated file descriptor */
39     char *message;     /* the message, if any, as a null-terminated string */
40 };
41
42 typedef struct idn_network_event idn_network_event_t;
```

Figura 6.2: Possíveis eventos na rede

convencionais de entrada-saída (ler, escrever) que tomam como argumento um descritor de arquivo vinculado ao *socket*.

Um servidor TCP multi-cliente deve, portanto, manter um vetor de descritores de arquivo referentes aos *sockets* ativos, um para cada cliente conectado. Basicamente, o que tem a fazer é permanecer à espera de eventos em cada um dos *sockets* e, mediante a ocorrência, reagir apropriadamente.

O conjunto de macros⁶ declarado pelo programa para representar as possíveis ocorrências é listado na figura 6.2. O primeiro evento gerado por uma ID é o pedido de conexão, a qual, após efetivada, habilita-lhe o envio de mensagens para o GI. Um evento de desconexão é produzido por uma ID que se desliga prematuramente da rede e é distinto de um erro; este, causado por falha no recebimento de uma mensagem. O último tipo é gerado quando, estando no aguardo de novos eventos, o programa servidor é subitamente interrompido por um sinal assíncrono gerado pelo sistema operacional (evocado pelo usuário para finalizar o programa, por exemplo).

A estrutura *idn_network_event* contém, então, os dados pertinentes ao evento, incluindo o *socket* originário, a natureza da ocorrência e, se aplicável, um ponteiro para um *buffer* contendo a mensagem enviada pelo cliente.

⁶Macro: diretiva do compilador que associa nomes simbólicos a constantes definidas pelo programador diretamente no código fonte.

Excerto de *idn/network.c*

```

17 /*
18  Return a pointer to a newly created and initialized TCP server.
19  On success, the server is associated to the given well-known
20  port; the value zero means the default port. On failure, the
21  function returns NULL.
22 */
23
24 idn_server_t *
25 idn_new_server (int port)
26 {
27     idn_server_t *new_server;
28
29     /* Try to allocate memory for the new server */
30
31     new_server = (idn_server_t *) malloc (sizeof (idn_server_t));
32     if (new_server == NULL)
33     {
34         idn_system_failure ();
35         return NULL;
36     }
37
38     /* Internet namespace, well-known port, any network interface */
39
40     new_server->address.sin_family = AF_INET;
41     new_server->address.sin_port = htons (port ? port : IDN_DEFAULT_SERVER_PORT);
42     new_server->address.sin_addr.s_addr = htonl (INADDR_ANY);
43
44     return new_server;
45 }

```

Figura 6.3: Configuração do servidor

6.1.2 Inicialização do Serviço TCP

24-45 A função `idn_new_server` definida no arquivo *network.c*, cujo excerto é mostrado na Figura 6.3, retorna um ponteiro para uma estrutura `idn_server` dinamicamente alocada — se o procedimento é bem sucedido, valores apropriados são utilizados para inicializar o endereço do servidor que, por sua vez, é também uma estrutura. Na listagem, seu primeiro 31-36 membro determina o formado de endereço do *socket*, neste caso designando o espaço de 40 nomes da Internet (`AF_INET6` referir-se-ia alternativamente ao protocolo IPv6).

Na terminologia TCP/IP, cada ponto de acesso ao serviço de transporte é identificado por uma *porta*: um número exclusivo atribuído ao *socket*, por meio do qual, o sistema operacional faz a multiplexação das mensagens entre os diversos usuários dessa camada (os processos). Dessa forma, para que um cliente possa endereçar uma requisição de conexão ao servidor, é preciso que, além do endereço de rede deste (o número IP), aquele especifique também a porta à qual o respectivo processo destinatário atende (no serviço TCP). Embora o sistema operacional possa atribuir uma porta arbitrária para um *socket*, na prática é conveniente que os servidores utilizem portas conhecidas a fim de que possam ser mais 41 facilmente encontrados. A função descrita inicializa a porta do *socket* com o valor do

Excerto de *idn/network.c*

```

47 /* Prepare to start the network job. */
48
49 int
50 idn_start_server (idn_server_t *server)
51 {
52     /* Create a socket */
53
54     server->socket_fd = socket (AF_INET, SOCK_STREAM, 0);
55     if (server->socket_fd < 0)
56     {
57         idn_system_failure();
58         return EXIT_FAILURE;
59     }
60
61     /* Bind the socket to the selected port. */
62
63     if ( bind (server->socket_fd, &server->address,
64             sizeof (server->address) ) < 0 )
65     {
66         idn_system_failure();
67         return EXIT_FAILURE;
68     }
69
70     /* Start to listen to the socket. */
71
72     if ( listen (server->socket_fd, 1) < 0 )
73     {
74         idn_system_failure();
75         close (server->socket_fd);
76         return EXIT_FAILURE;
77     }
78
79     /* Prepare the vector of client sockets */
80
81     FD_ZERO (&server->active_fd_set);
82     FD_SET (server->socket_fd, &server->active_fd_set);
83
84     return EXIT_SUCCESS;
85 }
86

```

Figura 6.4: Ativação do servidor

argumento fornecido; se este for igual a zero, um valor *default*⁷ é utilizado. Em *hosts* com mais de um endereço de rede, o servidor pode opcionalmente especificar qual deles deve ser associado ao *socket*.

Após configurado, o servidor pode ser ativado (Figura 6.4). A função *socket* retorna um descritor de arquivo para um novo *socket*. Seus argumentos especificam que a comunicação dar-se-á na Internet, utilizando um canal de dados em um serviço com conexão (a alternativa seria *SOCK_DGRAM* para serviço de datagramas) e o protocolo padrão dessa configuração, no caso o TCP. A seguir, a função *bind* executa a *syscall*⁸ que efetivamente registra o *socket* recém-criado no endereço previamente especificado pelo servidor.

O próximo passo é a transformação do *socket* de ativo (condição original quando é

⁷*Default*: valor que por definição é atribuído a um parâmetro caso nenhum outro seja explicitamente especificado.

⁸*Syscall*: de *system call*, chamada ao sistema (operacional).

criado) em passivo através da função `listen`, a qual cria um *buffer* para o recebimento de mensagens e instrui o sistema operacional a aceitar conexões naquele endereço em favor do processo do servidor. Finalmente, o vetor de *sockets* é zerado e lhe é adicionado o descritor do servidor.

6.1.3 O Monitor de Eventos

O gerenciador de interfaces é, em essência, um sistema de tempo real, na medida em que ajusta-se à definição genérica dos “sistemas que mudam de estado em função do tempo (real)” [53], sob influência do ambiente externo. Dependendo do conjunto de hipóteses assumidas acerca do comportamento deste último, uma dentre duas abordagens alternativas pode revelar-se mais adequada ao projeto de sistemas dessa natureza, a saber: a dos sistemas acionados por tempo (AT) e a dos sistemas acionados por eventos (AE)⁹. Em ambos os casos, o sistema recebe informações do meio externo, as processa e reage correspondentemente. A diferença fundamental está na forma como as entradas são recebidas. Sistemas AT efetivamente monitoram o ambiente recolhendo amostragem periódicas de um conjunto de variáveis de interesse; sistemas AE permanecem inativos, apenas reagindo à ocorrência de eventos selecionados. São, portanto, duas perspectivas complementares do modo de interação com o ambiente: ativo na primeira, passivo na segunda.

A coleta de dados do ambiente em sistemas AT é regulada por um temporizador interno que determina o intervalo de observação. Se as características dinâmicas do meio forem bem conhecidas, é possível calcular o período de amostragem de modo a suportar mesmo as variações de mais alta frequência e, uma vez que todas as entradas são registradas no mesmo instante, aplicam-se regras de prioridade estáticas no seu processamento. Em função disso, o pior tempo de resposta é sempre conhecido, razão pela qual sistemas AT são, sempre que possível, a opção preferencial em controle de tempo real rígido, onde a estrita observância desse limite é imprescindível. São naturalmente adequados para controle de processos periódicos e sua verificação é relativamente simples.

Sistemas AE, por outro lado, são meramente reativos, sendo acionados pelas mudanças

⁹De *event-triggered systems* e *time-triggered systems*, respectivamente [53].

no ambiente e não por um relógio intrínseco. Diferentemente dos anteriores, estes sistemas recebem as entradas independentemente em instantes distintos e as processam na mesma ordem — são, portanto, mais adequados para o tratamento de eventos esporádicos. Conseqüentemente, se um evento reclamando atendimento urgente é detectado durante o processamento de outro em condições normais de operação, é necessário executar controle de prioridade dinâmica, possivelmente com preempção do processo em andamento, o que complica sensivelmente o projeto. Outra dificuldade em sistemas AE surge quando é preciso evitar que rajadas bruscas de eventos sobrecarreguem momentaneamente o sistema, o que não ocorre no tipo anterior. Ainda, o pior tempo de resposta não é determinístico, de onde resulta a aplicação preferencial de sistemas AE em controle de tempo real flexível.

Sistemas AE são admitidamente mais complexos com respeito a sua construção e verificação, além de mais susceptíveis a variações no tempo de resposta. Pesa a seu favor, contudo, o fato de que sistemas AT dependem da precisão no modelamento do ambiente (por isso são preferidos em ambientes fechados e controlados): seu pior tempo de resposta é ótimo em contraste com o obtido nos sistemas AE, mas pode falhar miseravelmente se aquelas hipóteses não se provarem corretas (p.ex., se o aumento da frequência de variação da entrada monitorada mascarar eventos importantes). Os sistemas AE, contrariamente, são capazes de suportar cargas mais altas exibindo degradação sem se tornarem absolutamente inutilizáveis. É relevante também notar que os sistemas AT trabalham sempre em plena carga, mesmo quando a configuração de entradas não determina qualquer atuação por parte do sistema. Pelo mesmo motivo, qualquer ocorrência no meio pode ter sua detecção adiada por um tempo equivalente a até um ciclo de amostragem, sendo que seu atendimento pode demorar ainda mais, dependendo da ordem de processamento das entradas — se a frequência de amostragem encontrar limitantes técnicos ou se o número de entradas for muito elevado, isso pode tornar-se um obstáculo.

No que concerne ao objeto em exame, a RDI, cabe identificar, com base nas propriedades da interação entre IDs e GI, qual a hipótese que melhor se aplica a esse caso em particular. Para tal, observa-se inicialmente que a comunicação no modelo é assíncrona e não subordinada a qualquer periodicidade imposta por um temporizador global, resultando assim em um cenário de eventos esporádicos (não completamente aleatório, mas permane-

cendo dentro de alguns limites quanto à frequência) — há suporte científico para que essa suposição possa ser considerada realística na maioria dos mecanismos do mundo real [92] *apud* [53]. É igualmente válido realçar que sistemas acionados por tempo dependem de um canal de comunicação capaz de entregar mensagens com certa regularidade, um requisito que a tecnologia Internet, conforme se apresenta hoje, não consegue cumprir a contento — essa é uma restrição válida, em geral, para redes de longa distância, onde atrasos e latências variáveis impedem o estabelecimento de um limite para o tempo de comunicação.

Ambas as observações conduzem aos sistemas AE, e assumindo válidas as suposições, conclui-se que são adequadas ao mecanismo originalmente concebido, posto que a interação ID-GI é inerentemente baseada em eventos. Acrescenta-se ainda que o tempo de comunicação (da ordem de dezenas, centenas ou mesmo milhares de milissegundos) através da Internet tende a ser sensivelmente maior do que o tempo de processamento dos eventos (da ordem de frações de milissegundos) no equipamento onde executa-se o GI. Assim, dado que o tempo de restabelecimento (intervalo entre o recebimento de um evento e o instante em que o sistema torna-se apto a receber o próximo evento) é suficientemente menor que o período de chegada de novas requisições (tudo que o GI faz é rotear mensagens), é razoável admitir que o sistema esteja sempre disponível. Exclui-se, portanto, a possibilidade da detecção de um evento enquanto outro esteja em processamento, o que simplifica bastante o funcionamento do sistema (na eventualidade de isso acontecer, como resultado de uma ocorrência atípica, projeta-se um *buffer* auxiliar para requisições pendentes, o qual deve ser rapidamente consumido).

Esse mecanismo é implementado no protótipo IDN pelo código de `idn_next_network_event`, mostrada na Figura 6.5, a qual responde pela primeira função chamada dentro do laço de eventos principais do programa. O esquema funciona como se segue.

93-94 Ponteiros para o servidor (a estrutura `idn_server` descrita) e para um evento de rede (uma estrutura `idn_network_event` previamente alocada) são passados para a função. Em 99 seguida o vetor de *sockets*¹⁰ do servidor é copiado em uma variável auxiliar.

¹⁰O texto tem feito referência ao tipo de dado `fd_set` como um “vetor”, embora seja na realidade um vetor de bits. A variável contém mais apropriadamente o conjunto de descritores de arquivos associados aos respectivos *sockets*; contudo, em alguns contextos é mais conveniente referir-se a eles por *sockets* em prol da concisão.

Excerto de *idn/network.c*

```

88 /*
89  Block until a network event occurs. This function returns successfully
90  unless a network error occurs (this does not include the raising of signals).
91  */
92
93 int
94 idn_next_network_event (idn_network_event_t *event, idn_server_t *server)
95 {
96     int ready;
97     fd_set read_fd_set;
98
99     read_fd_set = server->active_fd_set;
100     errno = 0;
101     ready = select (FD_SETSIZE, &read_fd_set, NULL, NULL, NULL);
102
103     /*
104      If the user issues an interrupt signal (C-c), select returns -1.
105      This is different from select returning -1 due to network errors,
106      so we check errno first.
107     */
108
109     if ( (ready < 0) && (errno == EINTR) )
110     {
111         event->type = IDN_NETWORK_INTERRUPTED;
112     }
113
114     /* That aside, if select returns -1, we do have a network error. */
115
116     else if ( ready < 0 )
117     {
118         event->type = IDN_NETWORK_ERROR;
119     }

```

Figura 6.5: À espera de eventos na rede

101 A função `select` faz então com que o programa seja bloqueado pelo sistema operacional até que a camada de transporte receba dados para qualquer um dos descritores em `read_fd_set`. A técnica produz de fato o comportamento de um sistema de tempo-real acionado por eventos, mantendo o programa inativo até que haja entradas para serem processadas. Na prática, a função pode retornar em duas outras circunstâncias além da atividade no serviço TCP, a saber: se o programa for interrompido por um sinal do sistema operacional ou na eventualidade de um erro de I/O. Em ambos os casos, a chamada produz

109-119 um valor negativo que é, então, testado e reportado no evento de rede (o valor da variável `errno` atualizada pelo sistema distingue as duas situações).

Se o valor retornado é positivo, isso é uma indicação de que há atividade no vetor de *sockets* (Figura 6.6). Na primeira vez que o laço é executado, o programa deve determinar

127 que se trata do descritor do próprio servidor — posto que é, até então, o único no vetor — o que é interpretado como um pedido de conexão. A ação pertinente, nesse caso, é

133 a criação de um novo socket para o cliente solicitante, por meio da função `accept` e a

140-141 inserção do descritor de arquivo associado no vetor de *sockets* do servidor, seguida pela

Excerto de *idn/network.c*

```

121  /*
122   * If select returns greater than zero, then one of the socket
123   * file descriptor is ready. If it's the server socket than we have
124   * a connection request.
125   */
126
127  else if (FD_ISSET (server->socket_fd, &read_fd_set) )
128  {
129      size_t size;
130      struct sockaddr address;
131
132      size = sizeof (address);
133      event->socket_fd = accept (server->socket_fd, &address, &size);
134      if (event->socket_fd < 0)
135      {
136          event->type = IDN_NETWORK_ERROR;
137          return EXIT_FAILURE;
138      }
139
140      FD_SET (event->socket_fd, &server->active_fd_set);
141      event->type = IDN_NEW_CONNECTION;
142  }

```

Figura 6.6: Detectando requisições de conexão

correspondente atualização do evento de rede.

158-168 Nas próximas iterações do laço principal, o retorno de `select` pode se dar em função de atividade em qualquer um dos *sockets* e, se não for no do servidor, uma busca (Figura 6.7) através do vetor encarrega-se de identificar qual dos descritores dos clientes está pronto para leitura. O programa faz uma tentativa de leitura no descritor¹¹ encontrado, armazenando a mensagem recebida no *buffer* do servidor.

173-184 O procedimento pode resultar em dois eventos distintos (Figura 6.8). Se no lugar de uma mensagem o cliente enviara um sinal de desconexão, então a tentativa é frustrada resultando em um erro que é identificado pelo retorno negativo da função que faz a leitura — o descritor de arquivo do cliente é descartado e retirado do vetor de *sockets*. Caso contrário, o *buffer* do servidor, que contém a mensagem recebida, tem seus últimos dois bytes removidos (caracteres CR-LF do protocolo de comunicação, seção 4.8, pág. 60) e substituídos por um caracter NULL (terminador de cadeias de caracteres na linguagem C).

¹¹Na prática, cria-se uma representação de arquivo a partir do descritor para que se possa utilizar funções de leitura orientada a linhas.

Excerto de *idn/network.c*

```

144  /*
145   * Otherwise, it's a message from an already connected client.
146   */
147
148  else
149  {
150      int i = 3;                /* Skip stdin, stdout and stderr */
151      int read_bytes;
152      ssize_t message_length;
153      int fd;
154      FILE *file;
155
156      /* Find which of the sockets in the vector is ready */
157
158      while ( (! FD_ISSET (i, &read_fd_set )) && (i++ < FD_SETSIZE) );
159      event->socket_fd = i;
160
161      /* Try to read the incoming message */
162
163      message_length = 0;
164      event->message = NULL;
165      fd = dup (event->socket_fd);
166      file = fdopen (fd, "r");
167      read_bytes = getline (&event->message, &message_length, file);
168      fclose (file);

```

Figura 6.7: Atendendo pedidos dos clientes

Excerto de *idn/network.c*

```

170  /* If there's no data, (getline returned a negative value)
171   * then we know it's a disconnection; otherwise, read the data */
172
173  if (read_bytes < 0)
174  {
175      event->type = IDN_DISCONNECTION;
176      close (event->socket_fd);
177      FD_CLR (event->socket_fd, &server->active_fd_set);
178  }
179  else
180  {
181      event->type = IDN_NEW_MESSAGE;
182      message_length = read_bytes - 2 ;
183      event->message[message_length] = 0;
184  }
185  }
186  return EXIT_SUCCESS;
187 }

```

Figura 6.8: Atendimento de mensagens e desconexão

6.2 Estruturas de Dados

A informação referente às IDs pertencentes à rede é mantida pelo GI na estrutura de dados fundamental do sistema, a qual se denominou *manipulador de interfaces* (ou simplesmente manipulador) e cuja descrição pode ser apresentada como se segue.

O manipulador de interfaces implementa o esquema comentado na seção 5.3.2 (pág. 85), acerca do espaço dual formado pelas representativas e suas contra-partes no plano virtual de simulação. No primeiro nível, para cada ID que se conecta à rede o GI aloca exatamente uma entidade no seu modelo interno, designada por *dispositivo* (representativa da interface); no segundo, cada dispositivo se faz representar por zero, uma ou mais instância de dispositivos, ou simplesmente *instâncias*, as quais são dinamicamente criadas e destruídas conforme as atividades no sistema se processam. Ambos os elementos associam-se por meio de um *conector*, que estabelece a correspondência entre o dispositivo e suas respectivas instâncias.

22-27 O manipulador de interfaces é declarado como a estrutura `idn.interface` no arquivo *interface.h*, listado na Figura 6.9. Cada um dos conjuntos mencionados — de dispositivo, conectores e instâncias — é implementado pelo programa na forma de uma lista duplamente encadeada, cujos cabeçalhos são os ponteiros que respondem pelos três primeiros elementos da estrutura, respectivamente. Três outros membros são utilizado como contadores seqüenciais incrementados a cada vez que uma entidade do tipo correspondente é acrescentada.

Para facilitar a manipulação dessas estruturas o programa declara um tipo abstrato de dado para a lista duplamente encadeada, o qual é utilizado por todas as funções que as acessam. É conveniente apresentá-lo, uma vez que referências a sua estrutura aparecem ao longo de todo código. A Figura 6.10 fornece a listagem correspondente.

13-18 Um elemento da lista, além dos ponteiros para os elementos antecessor e sucessor, respectivamente, contém um ponteiro genérico (*void*) que indica o dado associado. O cabeçalho da lista contém seu comprimento, ponteiros para o primeiro e o último elementos, e para as duas funções que devem ser utilizadas para, respectivamente, atribuir e descartar dados. A primeira destas deve alocar o bloco de memória para um dado do tipo que se

Excerto de *idn/interface.h*

```

1 /* interface.h - the main data structure
2  *
3  *          $RCSfile: interface.h,v $ $Revision: 1.17 $
4  *          IDN Project, SEL/EESC-USP.
5  *          Refer to idn.txt for further information.
6  */
7
8 #ifndef _IDN_INTERFACE
9 #define _IDN_INTERFACE
10
11 #include "device.h"
12 #include "connector.h"
13 #include "instance.h"
14
15 /*
16  * This is the program's main data structure. It represents
17  * the IDN by means of devices, connector and instances.
18  */
19
20 struct idn_interface
21 {
22     idn_device_set_t *device_set; /* The set of connected devices */
23     idn_connector_set_t *connector_set; /* their respective connectors */
24     idn_instance_set_t *instance_set; /* and instances within the IDN */
25     int last_device_id; /* Device counter */
26     int last_connector_id; /* Connector counter */
27     int last_instance_id; /* Instance counter */
28 };
29
30 typedef struct idn_interface idn_interface_t;

```

Figura 6.9: O manipulador de interfaces

Excerto de *idn/list.h*

```

11 /* An element of the doubly-linked list.*/
12
13 struct idn_list_element
14 {
15     struct idn_list_element *previous; /* the previous element (NULL if first) */
16     struct idn_list_element *next; /* the next element (NULL if last) */
17     void *data; /* the element's data (NULL if none) */
18 };
19
20 typedef struct idn_list_element idn_list_element_t;
21
22 /* The head of the doubly-linked list. */
23
24 struct idn_list
25 {
26     int size; /* the current number of elements */
27     idn_list_element_t *first; /* the first element (NULL if empty) */
28     idn_list_element_t *last; /* the last element (NULL if empty) */
29     void *(*assign_data)(); /* how to assign data to the element */
30     void (*release_data)(void *); /* how to release the elements' data */
31 };
32
33 typedef struct idn_list idn_list_t;

```

Figura 6.10: Lista duplamente encadeada utilizada no programa

Excerto de *idn/device.h*

```
/*  
 * A set of macros to manipulate the device set. These are actually  
 * casts for functions that handle a doubly-linked list (see list.h)  
 */  
  
#define idn_new_device_set() (idn_device_set_t *) \  
    idn_new_list (idn_assign_device_data, \  
        idn_free_device_data)  
  
#define idn_add_device(list) (idn_device_t *) \  
    idn_add_element ((idn_list_t *) list)  
  
#define idn_remove_device(list, element) \  
    idn_remove_element ((idn_list_t *) list, \  
        (idn_list_element_t *) element)  
  
#define idn_delete_device_set(list) \  
    idn_delete_list ((idn_list_t *) list)
```

Figura 6.11: Macros para manipulação da lista de dispositivos

deseja armazenar e retornar-lhe um ponteiro; a segunda deve receber um ponteiro para o elemento a ser removido e liberar o bloco de memória contendo o dado correspondente. Os ponteiros para ambas as funções são passados como argumentos para a função que cria o cabeçalho da lista e nele são armazenados para, posteriormente, serem invocados pelas funções que acrescentam e removem elementos.

Para cada uma das listas — *idn_device_set*, *idn_connector_set* e *idn_instance_set* — o programa declara um conjunto de macros auxiliares para adicionar, remover e localizar elementos (Figura 6.11), as quais são na verdade *casts*¹² para as funções genéricas de manipulação de listas definidas no arquivo *list.c* (Figuras 6.12, 6.13 e 6.14). As funções para adição e subtração de elementos nas listas são definidas, respectivamente, em *device.c*, *connector.c* e *instance.c* — cada uma delas aloca e libera memória para as estruturas de dados correspondentes, as quais são abordadas adiante.

6.2.1 Procedimento de Conexão

O manipulador de interfaces constitui uma estrutura de dados dinâmica razoavelmente complexa, formada por três listas duplamente encadeadas, cada qual contendo elementos cujos dados podem possuir ponteiros para os elementos das outras listas. O modo como é utilizada pelo GI pode ser melhor exposto através da descrição da sequência de passos

¹²Técnica de mascarar de um tipo de dado em outro tipo.

Excerto de *idn/list.c*

```
/*
 * Return a pointer to a newly allocated and initialized doubly-linked list, or
 * NULL on failure. The function (void *) (*assign_data) assigns data to the
 * element (must return non-NULL); (void) (*release_data) releases data.
 */

idn_list_t *
idn_new_list (void *assign_data, void *release_data)
{
    idn_list_t *new_list;

    new_list = (idn_list_t *) malloc (sizeof (idn_list_t));
    if (new_list == NULL)
    {
        idn_system_failure ();
        return NULL;
    }

    new_list->size = 0;
    new_list->assign_data = assign_data;
    new_list->release_data = release_data;
    new_list->first = NULL;
    new_list->last = NULL;

    return new_list;
}

/*
 * Delete a doubly-linked list and all its elements and release
 * the associated memory.
 */

void
idn_delete_list (idn_list_t *list)
{
    while (list->size > 0)
        idn_remove_element (list, list->last);

    free (list);
}
```

Figura 6.12: Criação e liberação de uma lista

 Excerto de *idn/list.c*

```

/*
 * Append a newly created and initialized element to the list. On success,
 * return a pointer to the element. If anything goes wrong, just return NULL.
 */

idn_list_element_t *
idn_add_element (idn_list_t *list)
{
    idn_list_element_t *new_element;

    /* All or nothing: try to allocate memory for both the new element and its
     * data. If anything goes wrong, just clean up the mess and return NULL.*/

    new_element = (idn_list_element_t *) malloc (sizeof (idn_list_element_t));
    if (new_element == NULL)
    {
        idn_system_failure ();
        return NULL;
    }

    /* Try assign_data; if it returns NULL, we consider this as an error */
    new_element->data = list->assign_data ();
    if (new_element == NULL)
    {
        free (new_element);
        return NULL;
    }

    /* Append the element to the doubly-linked list */

    new_element->next = NULL;
    if (list->last != NULL)
    {
        new_element->previous = list->last;
        new_element->previous->next = new_element;
    }
    else
    {
        new_element->previous = NULL;
        list->first = new_element;
    }
    list->last = new_element;

    /* Increment the list counter and return pointer to the element */

    list->size += 1;

    return new_element;
}

```

Figura 6.13: Adição de elementos na lista

Excerto de *idn/list.c*

```
/*
 * Remove an element from the list and release the associated memory.
 */

void
idn_remove_element (idn_list_t *list, idn_list_element_t *element)
{
    idn_list_element_t *is_an_element;

    /* Is there such element ? If not, we have a bug */

    is_an_element = idn_is_an_element (list, element);
    assert (is_an_element);

    /* Remove the element from the linked list */

    if (element->previous != NULL)
        element->previous->next = element->next;
    else
        list->first = element->next;

    if (element->next != NULL)
        element->next->previous = element->previous;
    else
        list->last = element->previous;

    /* Decrement list size */

    list->size -= 1;

    /* Release element's data with the appropriate function */

    list->release_data(element->data);

    /* And release element's memory */

    free (element);
}

/*
 * Return a pointer to the element or NULL if it's not in list
 */

idn_list_element_t *
idn_is_an_element (idn_list_t *list, idn_list_element_t *element)
{
    idn_list_element_t *tmp;
    tmp = list->first;
    while ( (tmp != element) && (tmp != NULL))
        tmp = tmp->next;
    return tmp;
}
```

Figura 6.14: Remoção e localização de elementos na lista

 Excerto de *idn/interface.c*

```

/*
 * Return a pointer to a newly created and initialized interface.
 * On failure, return NULL (try not to leave memory leaks behind).
 */

idn_interface_t *
idn_new_interface (void)
{
    idn_interface_t *new_interface;

    new_interface = (idn_interface_t *) malloc (sizeof (idn_interface_t));
    if (new_interface == NULL)
    {
        idn_system_failure();
        return NULL;
    }

    new_interface->device_set = idn_new_device_set();
    if (new_interface->device_set == NULL)
    {
        free(new_interface);
        return NULL;
    }

    new_interface->connector_set = idn_new_connector_set();
    if (new_interface->connector_set == NULL)
    {
        free(new_interface->device_set);
        free(new_interface);
        return NULL;
    }

    new_interface->instance_set = idn_new_instance_set();
    if (new_interface->instance_set == NULL)
    {
        free(new_interface->device_set);
        free(new_interface->connector_set);
        free(new_interface);
        return NULL;
    }

    new_interface->last_device_id = 0;
    new_interface->last_connector_id = 0;
    new_interface->last_instance_id = 0;

    return new_interface;
}

```

Figura 6.15: Inicialização do manipulador

executada durante a operação normal do sistema.

Durante a inicialização, um manipulador é alocado pela função `idn_new_interface` definida no arquivo *interface.c* (Figura 6.15), a qual retorna um ponteiro para uma estrutura `idn_interface` em uma variável global. Inicialmente, na estrutura retornada, `idn_device_set`, `idn_connector_set` e `idn_instance_set` são todas listas vazias alocadas através das macros correspondentes, e todos os contadores estão zerados.

Quando o GI recebe o pedido de conexão da primeira ID (e também de todas as seguintes), a primeira providência a ser tomada é atribuir-lhe um dispositivo no manipulador. Um dispositivo é um elemento de lista cujo dado associado é representado pela estrutura mostrada na Figura 6.16. Seu primeiro membro é um número de identificação interno que

Excerto de *idn/device.h*

```

15 /* Device data types. See below. */
16
17 typedef struct idn_device idn_device_t;
18 typedef struct idn_device_data idn_device_data_t;
19 typedef struct idn_device_set idn_device_set_t;
20
21 /*
22  The device data
23 */
24
25 struct idn_device_data
26 {
27     int id; /* an identifier (internal use only) */
28     int socket_fd; /* the socket file descriptor */
29     char *message; /* the message in the buffer (or NULL if none) */
30     int buffer_state; /* the buffer has an input/output or is empty */
31     idn_instance_t *servant; /* which instance serves this device */
32     idn_instance_t *interrupted; /* the interrupted instance (if any) */
33     idn_connector_t *connector; /* the connector to which the device is bound */
34 };

```

Figura 6.16: O dispositivo

Excerto de *idn/connector.h*

```

13 /* Connector data types. See below */
14
15 typedef struct idn_connector idn_connector_t;
16 typedef struct idn_connector_data idn_connector_data_t;
17 typedef struct idn_connector_set idn_connector_set_t;
18
19 /*
20  The connector data
21 */
22
23 struct idn_connector_data
24 {
25     char *name; /* the name by which the devices are known */
26     int device_count; /* the current number of devices */
27     int instance_count; /* the current number of instances */
28     int program_mark; /* zero if the connector starts a program */
29     idn_connector_t *implicit_target; /* the implicit target (or NULL if none) */
30     idn_connector_t *explicit_target; /* the explicit target (or NULL if none) */
31     int condition; /* current working condition */
32 };

```

Figura 6.17: O conector

é utilizado apenas como rótulo de referência em funções que geram saídas para o usuário, para distinguir um dentre os diversos dispositivos na rede. A estrutura contém também o descritor de arquivos atribuído ao *socket* da ID pelo módulo TCP e um *buffer* onde é armazenada a última mensagem associada ao dispositivo. Uma variável auxiliar informa se esta trata de uma mensagem recebida ou a ser transmitida pela rede, ou se o buffer está vazio. Se o dispositivo possui instâncias, um ponteiro indica qual dentre aquelas o está servindo em determinado instante e, se a ID é uma interruptora (iniciou um ciclo de operação), outro ponteiro informa qual instância constitui o início da cadeia.

Enquanto o dispositivo contém os dados relativos ao processo de comunicação com as

IDs, os parâmetros operacionais da representativa no modelo interno do GI são dados pelo
33 conector ao qual aquele é vinculado, fornecido pelo terceiro ponteiro da estrutura .

A segunda providência tomada pelo GI, mediante um pedido de conexão bem sucedido, portanto, é alocar para o dispositivo recém criado um novo conector que, por sua vez, é também representado por uma estrutura especial (Figura 6.17). Esta é que armazena o
25 nome simbólico do dispositivo, i.e. o valor do campo ENDEREÇO (seção 4.8, pág. 60) nas mensagens enviadas para (e recebidas de) a respectiva ID. Dois contadores guardam
26-27 o número de dispositivos e o número de instâncias vinculados ao conector, enquanto uma
28 variável especial informa se este pertence a um programa: uma associação de conectores
29 em uma seqüência, onde cada elemento intermediário possui um ponteiro para seu sucessor, ou seja, seu alvo implícito. O alvo explícito do conector, por outro lado, é um segundo
30 ponteiro que é atualizado pelo GI com o endereço do conector onde está a ID destinatária de uma interrupção recebida. Finalmente, um último campo é utilizado para indicar a condição do conector, conforme será explicado oportunamente.

81-87 A descrição do procedimento de conexão de uma ID pode ser, então, acompanhada com o auxílio da Figura 6.18, que fornece um excerto da função que o executa. O primeiro passo é alocar um novo conector no manipulador e atribuir-lhe a condição regular. Os números de dispositivos e instâncias associados são ambos corretamente inicializados em zero, e ao indicador de programa é atribuído o valor um, informando que o conector não é o início de um programa, conforme tornar-se-á claro adiante.

88 É preciso atribuir também um nome ao novo conector. Escolheu-se compô-lo utilizando o contador de conectores do manipulador de interfaces, o qual é continuamente incrementado. Assim, o primeiro conector chama-se “1”, o segundo “2”, depois “3” e assim por diante. Mais tarde, as IDs podem requisitar ao sistema a alteração de seus nomes para outros mais adequados conforme sua conveniência. A fim de que não haja conflito, contudo, estabelece-se que os nomes iniciados por dígitos numéricos ficam reservados para atribuição automática pelo sistema.

89-90 Finalmente, ajustam-se os alvos implícito e explícito. Nos experimentos anteriores, utilizara-se o artifício do “dispositivo nulo” como alvo implícito para simbolizar a ausência

 Excerto de *idn/interface.c*

```

68 /*
69  Add a new device to the interface with the given file
70  descriptor. On failure, return NULL.
71 */
72
73 idn_device_t *
74 idn_connect_device (int socket_fd)
75 {
76     idn_device_t *device;
77     idn_connector_t *connector;
78
79     /* First we add a new connector to the connector set. */
80
81     connector = idn_add_connector (interface->connector_set);
82     if (connector == NULL)
83         return NULL;
84     connector->data->condition = IDN_CONNECTOR_REGULAR; /* All right. */
85     connector->data->device_count = 0;
86     connector->data->instance_count = 0;
87     connector->data->program_mark = 1;
88     asprintf (&connector->data->name, "%d", ++interface->last_connector_id);
89     connector->data->explicit_target = connector;
90     connector->data->implicit_target = connector;
  
```

Figura 6.18: Procedimento de conexão (parte 1)

de programação. No protótipo IDN essa estratégia foi alterada: a mesma informação é representada fazendo-se que o alvo implícito aponte para o próprio conector; o alvo explícito, por sua vez, é relevante apenas na ocorrência de encaminhamento explícito e pode ser, portanto, arbitrariamente inicializado da mesma forma.

94-99 Após configurar o conector, o passo seguinte é alocar um novo dispositivo para a ID conectada (Figura 6.19). O identificador interno é atualizado incrementando o contador de dispositivos do manipulador, o descritor de arquivo recebe o *socket* fornecido como argumento à função (o qual fora criado para a ID pelo módulo TCP) e o *buffer* de mensagens
 109-110 do dispositivo e seu indicador de estado são inicializados apropriadamente. Na ausência de instâncias, os ponteiros para essas entidades são anulados e o dispositivo é finalmente atrelado ao conector recém criado, cujo contador de dispositivos é correspondentemente incrementado.

Nesse ponto está finalizada a operação de conexão. Se o procedimento for repetido várias vezes, o manipulador terá em suas listas de dispositivos e conectores tantos elementos quantos forem as IDs conectadas ao GI.

Excerto de *idn/interface.c*

```

92  /* Then we add a new device to the device set. */
93
94  device = idn_add_device (interface->device_set);
95  if (device == NULL)
96  {
97      idn_remove_connector (interface, connector);
98      return NULL;
99  }
100  device->data->id = ++interface->last_device_id;
101  device->data->socket_fd = socket_fd;
102  device->data->message = NULL;
103  device->data->buffer_state = IDN_DEVICE_BUFFER_EMPTY;
104  device->data->servant = NULL;
105  device->data->interrupted = NULL;
106
107  /* Finally, we bind the device to the connector */
108
109  device->data->connector = connector;
110  connector->data->device_count = 1;
111
112  return device;
113 }

```

Figura 6.19: Procedimento de conexão (parte 2)

6.2.2 Manipulação de Mensagens

Uma vez conectadas à rede, as IDs podem enviar mensagens umas às outras. Para tal, o protótipo IDN implementa um protocolo de aplicação orientado a mensagens nomeado IDNP (de *Interactive-Device Network Protocol*)¹³, cuja forma geral é aquela introduzida na seção 4.8 (pág. 60). Cada um dos cinco campos do protocolo pode ser descrito como se segue:

PROTOCOLO A assinatura do protocolo é a cadeia “IDNP”, seguida de uma barra “/” (ASCII 47), seguida pela versão do protocolo — esta, formatada da maneira usual na Internet, com um número principal de versão e um número complementar indicando o número da revisão, ambos separados por um ponto. A implementação fornece a versão “0.1” (experimental ilustrativa), cuja especificação informal define que os próximos campos sejam interpretados como se segue.

CONTROLE O campo de controle tem a forma $:d_1:d_2 \dots :d_n$, onde $d_1, d_2 \dots d_n$ são dados referentes às *opções de controle*. A forma como essa informação é processada deve ser negociada entre a ID e o GI durante a operação. O protótipo IDN não oferece suporte para qualquer opção de controle, mas eventuais implementações futuras podem

¹³Protocolo para Rede de Dispositivos Interativos.

se valer desse campo para transportar dados tais como códigos de verificação de erro, números de seqüência, autenticação digital, registros de tempos de execução, dentre outras anotações, as quais podem vir a ser de utilidade em condições específicas.

ENDEREÇO O campo de endereçamento contém o endereço simbólico do destinatário ou do remetente da mensagem e deve ser interpretado do seguinte modo:

- a) Entrada (recebimento da mensagem): para o GI o campo só tem relevância no caso de encaminhamento explícito, quando deve ser interpretado como o destinatário da mensagem (caso contrário, deve ser ignorado); a ID deve interpretar o campo como o remetente da mensagem.
- b) Saída (envio da mensagem): o GI deve anotar nesse campo o endereço do emissor da mensagem; a ID deve indicar o destinatário quando a mensagem for uma interrupção (caso contrário deve apenas preencher o campo com um símbolo de livre escolha).

A versão 0.1 do protocolo reserva o nome “SYSTEM” para o dispositivo de sistema e todos os nomes iniciados por caracteres numéricos (0, 1, 2, ... 9) para atribuição automática de nomes pelo GI.

AÇÃO O campo ação dá instruções adicionais para o processamento do comando. Baseado na prática convencional em alguns protocolos de aplicação na Internet (p.ex. HTTP, FTP), utilizou-se nesse campo um código de *status* de retorno (*return status code*) de três dígitos para representar as diversas ações previstas no protocolo conforme as regras abaixo:

- Primeiro dígito :
 - 1 : Mensagem nula
 - 2 : Mensagem positiva
 - 3 : Mensagem negativa, falha do emissor (ID)
 - 4 : Mensagem negativa, falha do receptor (ID)
 - 5 : Mensagem negativa, falha do sistema (GI)

- Segundo dígito:
 - 0 : Condição permanente
 - 1 : Condição transiente
- Terceiro dígito
 - 0 : ação padrão

As combinações formadas pelos três dígitos, seus significados e as ações solicitadas por elas são resumidas na Figura 6.20.

COMANDO O campo de comando contém a função e a lista de argumentos da mensagem. Em um retorno, o comando é o resultado do processamento do emissor e, ao mesmo tempo, a instrução de acionamento para o receptor. Não foram estabelecidas normas para o formato deste campo — basicamente, qualquer conjunto de uma ou mais cadeias de caracteres é válido. Registra-se como sugestão, todavia, que pode ser interessante definir algumas regras práticas para a sintaxe das funções, tais como reservar alguns nomes (iniciando com letra maiúscula, por exemplo) para uso exclusivo do dispositivo de sistema (evitando a poluição do espaço de nomes), criar uma função específica para retorno de resultados etc.

O protocolo estabelece o CMI¹⁴ descrito na Figura 6.21, o qual deve ser implementado pelas IDs como parte dos requisitos para a compatibilidade com a versão IDNP/1.0. Faz parte ainda da especificação definir um CIEA¹⁵ para o dispositivo de sistema, a fim de que as IDs possam adequadamente acessar os serviços do GI, os quais são listados na Figura 6.22.

Como é possível constatar, a implementação exclusiva das facilidades especificadas pelo IDNP/0.1 proporciona um sistema bastante rudimentar. Cabe ressaltar assim, conforme sugere o número de versão, que se trata evidentemente de uma definição estritamente experimental e foi elaborada com a finalidade meramente ilustrativa, fornecendo os recursos mínimos para alguns ensaios práticos que se deseja realizar. Ainda que precário, contudo, o

¹⁴CMI: Conjunto mínimo de instruções, seção 4.8, pág. 63.

¹⁵CIEA: Conjunto de instruções específicas da aplicação, seção 4.8, pág. 63.

Código	Significado	Ação correspondente
100	Mensagem nula	O receptor não deve processar a mensagem; deve passá-la adiante da forma como recebida
200	Sucesso/Executar	O emissor informa que o comando foi executado com sucesso; o receptor deve interpretar como uma ordem para que o comando seja executado
300	Falha permanente do cliente	O emissor gerou uma mensagem inválida; não há chances de que uma tentativa subsequente tenha sucesso a não ser que a mensagem seja reformulada. (ex. erro de sintaxe)
310	Falha transiente do cliente	Falha no processamento da mensagem causada pelo emissor; tentativas subsequentes podem ter sucesso (ex. falha de comunicação)
400	Falha permanente do servidor	O comando solicitado não pode ser executado pelo receptor; a mensagem deve ser reformulada (ex. função não implementada)
410	Falha transiente do servidor	Receptor temporariamente incapaz de processar o comando; tentativas subsequentes podem ter sucesso (ex. dispositivo ocupado)
500	Falha permanente do sistema	O sistema foi incapaz de processar a mensagem; novas tentativas não terão sucesso, intervenção no sistema necessária (ex. sistema inconsistente)
510	Falha transiente do sistema	Sistema temporariamente incapaz de processar a mensagem; tentativas subsequentes podem ter sucesso

Figura 6.20: Código de status de retorno IDNP/1.0

Função	Arg.	Resposta	Descrição
hello		(texto)	Resposta de eco, resposta de teste
version		(texto)	Versão da ID
functions		(número)	Número de funções implementadas pela ID
function	f	(texto)	Descrição do propósito da função f
arguments	f	(número)	Número de argumentos da função f
argument	$n f$	(texto)	Descrição do n -ésimo argumento da função f .

Figura 6.21: CMI estabelecido pelo IDNP/1.0

Função	Arg.	Resposta	Descrição
name		(text)	Informa nome do emissor
name	n	(text)	Configura o nome do emissor para n e retorna o novo nome
implicit		(text)	Informa o alvo implícito do emissor
implicit	n	(text)	Configura o alvo implícito do emissor para n e retorna o novo alvo
join		(text)	Informa o conector do emissor
join	n	(text)	Configura o conector do emissor para n e retorna o novo conector

Figura 6.22: CIEA do dispositivo de sistema estabelecido pelo IDNP/1.0

exemplo pode servir de subsídio para trabalhos futuros, onde este pode ser adequadamente estendido para acrescentar funcionalidades complementares, tais como controle de acesso, autenticação, reserva de recursos e, possivelmente, encriptação, dentre outras.

As ferramentas do protótipo para trabalhar com o protocolo IDNP são declaradas em *idnp.h* e *protocol.h*. O primeiro arquivo (Figura 6.23) oferece uma coleção de macros que representam parâmetros como a assinatura do protocolo, campo de controle, código de *status* de retorno etc., bem como as funções do seu CMI e CIEA.

O outro arquivo (Figura 6.24) declara uma estrutura para armazenamento de mensagens IDNP/1.0. As funções que manipulam mensagens, cuidam da sua recepção e transmissão são definidas em *protocol.c*. Neste arquivo, a função `idn_get_idnp_message` toma uma cadeia de caracteres como argumento e retorna um ponteiro para uma nova estrutura `idn_idnp_message` contendo a cadeia convertida para uma mensagem IDNP. Trata-se meramente de manipulações de cadeias de caracteres e, logo, se faz prescindível listar seu código. Como informação de ordem prática para o estudo complementar, todavia, dado que a maneira como o procedimento realiza o trabalho é um tanto artificiosa, cumpre anotar que este vale-se de um “efeito colateral” da função `strtok_r` (POSIX-1) para separar a cadeia de caracteres em subcadeias correspondentes aos campos da mensagem IDNP. Como resultado final, o *buffer* `message_vector` da estrutura `idn_idnp_message` contém uma cópia da mensagem original onde os delimitadores do protocolo são substituídos por caracteres NULL (final de cadeia, em C). Os cinco outros ponteiros indicam, então, o início das subcadeias assim formadas, fornecendo os campos de protocolo, controle, endereço, ação e comando, respectivamente.

O arquivo proporciona também a função `idn_new_message` (não listada) para criação de mensagens IDNP/0.1 textuais a partir dos campos ADDRESS, ACTION e COMMAND, e a função `idn_verify_message` para verificar se a mensagem é compatível com o protocolo. Esta última é listada na Figura 6.25, pois além de fornecer um exemplo da utilização das duas funções anteriores, introduz também a estratégia de interceptação de mensagens do GI.

Excerto de *idn/idnp.h*

```

/* idnp.h - IDNP/0.1 (Interactive-Device Network Protocol)
 *
 *      $RCSfile: idnp.h,v $ $Revision: 1.4 $
 *      IDN Project, SEL/EESC-USP.
 *      Refer to idn.txt for further information.
 */

#ifndef _IDN_IDNP
#define _IDN_IDNP

#define IDN_IDNP_PROTOCOL "IDNP/0.1" /* Protocol signature */
#define IDN_IDNP_DELIMITERS " " /* Field delimiter */
#define IDN_IDNP_CONTROL ":" /* Control field */
#define IDN_SYSTEM_NAME "SYSTEM" /* The system name. */

/* Response status codes */

#define IDN_STATUS_VOID 100
#define IDN_STATUS_SUCCESS 200
#define IDN_STATUS_PERMANENT_CLIENT_ERROR 300
#define IDN_STATUS_TRANSIENT_CLIENT_ERROR 310
#define IDN_STATUS_PERMANENT_SERVER_ERROR 400
#define IDN_STATUS_TRANSIENT_SERVER_ERROR 410
#define IDN_STATUS_PERMANENT_SYSTEM_ERROR 500
#define IDN_STATUS_TRANSIENT_SYSTEM_ERROR 510

/* Minimum instruction set */

#define IDN_MIS_HELLO "hello" /* The echo reply (talk to me...) */
#define IDN_MIS_VERSION "version" /* What is your version ? */
#define IDN_MIS_FUNCTIONS "functions" /* How many functions ? */
#define IDN_MIS_FUNCTION "function" /* What is the function foo for ? */
#define IDN_MIS_ARGUMENTS "arguments" /* How many args. has function foo ? */
#define IDN_MIS_ARGUMENT "argument" /* What about arg n or function foo ? */

/* Application-Specific Instruction set (System Device) */

#define IDN_DSIS_NAME "name" /* Set and get the device name */
#define IDN_DSIS_IMPLICIT "implicit" /* Set and get the implicit target */
#define IDN_DSIS_JOIN "join" /* Set and get the device connector */

#endif /* not _IDN_IDNP */

```

Figura 6.23: Declarações do IDNP

Excerto de *idn/protocol.h*

```

13 /*
14  * A structure to represent an IDNP message.
15  */
16
17 struct idn_idnp_message
18 {
19     char *message_vector; /* The message is stored in this buffer */
20     int vector_length; /* The buffer length */
21     char *protocol; /* A pointer to the protocol field */
22     char *control; /* A pointer to the control field */
23     char *address; /* A pointer to the address field */
24     char *action; /* A pointer to the action field */
25     char *command; /* A pointer to the command field */
26 };
27
28 typedef struct idn_idnp_message idn_idnp_message_t;

```

Figura 6.24: A mensagem IDNP como uma estrutura

 Excerto de *idn/protocol.c*

```

88 /*
89  Verify whether the given string is a good IDNP/0.1 message or not. The return
90  value is a pointer to a new buffer with either a copy of the string, if its ok,
91  or a special IDNP/0.1 addressed to the system, if it is not.
92
93  By now, a 'good' message is a message with five fields. No matter if the
94  protocol or control fields are wrong: we just overwrite them. In the future
95  the function should also check these values and reject bad messages.
96  */
97
98  char *
99  idn_verify_message (char *message)
100 {
101     idn_idnp_message_t *idnp_message;
102     char *new_message;
103
104     /* Get a new IDNP message from the given string message */
105
106     new_message = NULL;
107     idnp_message = idn_get_idnp_message (message);
108     free (message);
109
110     /* If any field is missing, force the sender to admit it to the system */
111
112     if ( (idnp_message->protocol == NULL)
113         || (idnp_message->control == NULL)
114         || (idnp_message->address == NULL)
115         || (idnp_message->action == NULL)
116         || (idnp_message->command == NULL))
117     {
118         idn_new_message (&new_message, IDN_SYSTEM_NAME,
119                         IDN_STATUS_PERMANENT_CLIENT_ERROR, "Syntax error");
120         idn_log ("syntax error; rewritten: %s", new_message);
121     }
122     else
123     {
124         /* Otherwise, overwrite the message with correct protocol and control */
125
126         idn_new_message (&new_message, idnp_message->address,
127                         atoi(idnp_message->action), idnp_message->command);
128     }
129     free (idnp_message->message_vector);
130     free (idnp_message);
131
132     return new_message;
133 }

```

Figura 6.25: Verificação de mensagens

106-108
112-118

O propósito da chamada é verificar se uma dada cadeia de caracteres é uma mensagem IDNP/0.1. Em caso positivo, seu retorno é um ponteiro para um novo *buffer*, contendo uma cópia da mensagem recebida como argumento; em caso negativo, o *buffer* é preenchido com uma nova mensagem construída pelo GI. Para tal, primeiro obtém-se uma estrutura de mensagem inicializada a partir da cadeia original. Em seguida, verifica-se se todos os cinco campos do protocolo estão presentes e, caso contrário, cria-se uma nova mensagem textual, mas sobrescrevendo os campos finais. Isso faz com que o comando original da ID cliente seja compulsoriamente reendereçado ao dispositivo de sistema, admitindo haver cometido uma falha permanente relacionada a um erro de sintaxe. A tática de forçar as IDs a confessarem suas próprias faltas forjando declarações pode soar um tanto próximo aos métodos da Inquisição, mas o objetivo é fazer com que a comunicação se processe de maneira realmente transparente, de tal sorte que o GI não se faça visível a não ser na forma do dispositivo de sistema (assumidamente, uma justificativa maquiavélica). O resultado, conforme comentado mais adiante, é que este último pode então tomar as providências adequadas, dentre elas, responder à ID equivocada instruindo-a acerca de seu engano. Assim, o conjunto funciona de modo consistente, sempre por meio da interação entre IDs e através dos mesmos mecanismos do ambiente virtual do GI.

126

A implementação em exame, contudo, é extremamente simplificada. A única verificação realizada é, de fato, a presença de cinco campos na mensagem. Quaisquer outros erros, tais como assinatura inválida ou opções de controle não suportadas, são simplesmente mascarados: uma vez que a mensagem tenha passado pelo teste anterior, é utilizada para compor uma nova, na qual os campos PROTOCOLO e CONTROLE são sobrescritos com os valores apropriados e, os três seguintes, com os dados aproveitados da mensagem original. Uma verificação mais completa pode ser elaborada estendendo-se a função para analisar um número maior de critérios.

A transmissão das mensagens através da rede é realizada pelas funções `idn_read_input` e `idn_write_output` (não listadas). A primeira efetua a leitura orientada a linhas através de uma função de biblioteca apropriada, armazenando o valor lido no endereço recebido como argumento. A segunda envia a cadeia de caracteres indicada através do descritor de arquivo fornecido via primitiva de I/O do sistema `write`.

 Excerto de *idn/interface.c*

```

147 /*
148  Hold your breath, we're moving inwards the core of the system. This function
149  takes the string 'message' that has just arrived from the network through the
150  file descriptor 'fd' and assigns it to the corresponding device. It then
151  drives the event into the virtual world of the instances. Read on...
152 */
153
154 int
155 idn_receive_message (int fd, char *message)
156 {
157     idn_instance_t *instance;
158     idn_device_t *device;
159
160     /* First, we assign the data to the device whose file descriptor is fd. */
161
162     device = idn_get_device_by_fd (fd); /* Find the device. */
163     assert (device); /* It'd be a bug if it wasn't found. */
164
165     idn_step ("message from device %d: %s", device->data->id, message);
166
167     message = idn_verify_message (message); /* check the message */
168
169     free (device->data->message); /* assign it to the device */
170     device->data->message = strdup (message);
171     if (device->data->message == NULL)
172     {
173         idn_system_failure ();
174         return EXIL_FAILURE;
175     }
176
177     device->data->buffer_state = IDN_DEVICE_BUFFER_INPUT; /* new buffer state */

```

Figura 6.26: Tratamento de entradas (parte 1)

6.2.3 Processamento de Mensagens

Quando o monitor de eventos detecta o recebimento de uma mensagem, seu tratamento fica a cargo da função `idn_receive_message`, definida em *interface.c*. Tem-se aí a fronteira entre a interação com a rede TCP e o ambiente virtual de trabalho proporcionado pelo GI — logo, trata-se de um ponto-chave na compreensão dos mecanismos essenciais do sistema e merece, portanto, uma descrição pormenorizada.

O procedimento passo a passo pode ser acompanhado no excerto de código da Figura 6.26. Dados o descritor de arquivo, através do qual a mensagem fora recebida, e o endereço do *buffer*, onde for armazenada — ambos registrados na estrutura que representa o evento de rede (pág. 91) — a primeira providência é localizar, no manipulador de interfaces, o dispositivo associado ao *socket* correspondente.

164-165 Para tal, a função `idn_get_device_by_fd` (Figura 6.27) vasculha a lista de dispositivos na busca daquele cujo descritor coincide com o associado ao evento (é consistente assumir que exatamente um dispositivo será localizado). Em seguida, a cadeia de caracteres 169 recebida é filtrada, utilizando-se as facilidades já descritas, para assegurar-se de que se

Excerto de *idn/interface.c*

```
/*
 * Return a pointer to the device in the interface whose
 * file descriptor 'fd'. If there is no such device, return NULL.
 */

idn_device_t *
idn_get_device_by_fd (int fd)
{
    idn_device_t *device;
    device = interface->device_set->first;
    while (device->data->socket_fd != fd)
        device = device->next;
    return device;
}
```

Figura 6.27: Recupera o dispositivo dado o descritor

trata de uma mensagem IDNP/0.1 válida. Então, a mensagem é transferida para o *buffer* do dispositivo encontrado e o indicador de estado daquele é alterado para informar que o dispositivo tem uma entrada pendente (neste contexto, entradas e saídas são consideradas do ponto de vista do interior do GI).

Nesse ponto, chega-se ao cerne do gerenciador de interfaces, onde se encontram o mecanismo de encaminhamento e a implementação da estratégia essencial de funcionamento do sistema. A decisão-chave tomada pelo algoritmo de roteamento nessa etapa é a identificação da natureza da mensagem recebida quanto ao fato de ser uma interrupção ou um retorno¹⁶.

182 O teste utilizado é o número de instâncias no conector do dispositivo: se não hou-
ver instâncias, então, certamente trata-se de uma interrupção (Figura 6.28). Este é o
caso, portanto, de encaminhamento explícito e, logo, é preciso recorrer ao campo de en-
193-194 dereço da mensagem recebida. Isso é feito obtendo-se uma estrutura de mensagem
IDNP através dos procedimentos já mencionados e convertendo o nome simbólico, assim
encontrado, em um ponteiro para o respectivo conector. Para tal, vale-se do auxílio da
função `idn_get_connector_by_name` (Figura 6.29), a qual, analogamente à chamada a
pouco comentada, vasculha a lista de conectores e localiza aquele cujo nome é dado como
argumento.

199-206 A busca, desta vez, pode não ter sucesso. Nesse caso, novamente adota-se a tática
de interceptação de comunicação, substituindo o conteúdo do *buffer* do dispositivo inter-

¹⁶Conforme seção 4.6, pág. 51.

Excerto de *idn/interface.c*

```

179  /* If there is no instances in the device's connector, than we know that we
180     have an interruption. It's key point. That's what we're going to do ...*/
181
182  if (device->data->connector->data->instance_count == 0 )
183  {
184      idn_idnp_message_t *idnp_message;
185      idn_connector_t *connector;
186      int found_it;
187
188      idn_log ("message arriving in an empty connector (interruption)");
189
190      /* We're dealing with explicit delivering, so we need to get the
191         addresse's connector from the corresponding message field */
192
193      idnp_message = idn_get_idnp_message (message);
194      connector = idn_get_connector_by_name (idnp_message->address);
195
196      /* Where is that connector after all ? If it can't be found, intercept the
197         message and use it to inform the system that connector is missing*/
198
199      found_it = 1;
200      if (connector==NULL)
201      {
202          idn_log ("** addressee not found, so forwarding to the system");
203          connector = IDN_SYSTEM_CONNECTOR;
204          found_it = 0;
205      }
206      device->data->connector->data->explicit_target = connector;

```

Figura 6.28: Tratamento de entradas (parte 2)

Excerto de *idn/interface.c*

```

/*
Return a pointer to the device in the interface whose
file descriptor 'fd'. If there is no such device, return NULL.
*/

idn_device_t *
idn_get_device_by_fd (int fd)
{
    idn_device_t *device;
    device = interface->device_set->first;
    while (device->data->socket_fd != fd)
        device = device->next;
    return device;
}

```

Figura 6.29: Recupera o conector dado o nome

Figura 6.30: Dispositivo com interrupção pendente

ruptor por uma mensagem endereçada ao sistema, notificando-o de uma falha transiente da ID parceira (considera-se que a falta é do destinatário em não estar disponível quando requisitado; este pode haver desconectado-se inesperadamente). De um modo ou de outro, obtém-se um conector destinatário para a mensagem, o qual é atribuído ao ponteiro de alvo explícito do dispositivo portador da mensagem.

Existe nesse instante um dispositivo com uma interrupção pendente a ser entregue a um destino certo. O manipulador de interfaces tem, então, a configuração ilustrada no diagrama da Figura 6.30. Na representação esquemática, o anel mais espesso denota que o dispositivo (círculos) possui uma entrada, e o segmento destacado unindo os conectores (retângulos) simboliza o ponteiro de alvo explícito do portador da mensagem (à esquerda) para receptor (à direita).

Resta agora transferir a mensagem, e neste processo é que se encontram todas as dificuldades discutidas na seção 5.3.2 acerca de operações concorrentes: interação ineficiente, condições de corrida, *deadlocks* etc. Figurativamente, o que a abordagem pretende é desmembrá-lo em duas camadas operacionais. À superior, cabe a interação do GI com a rede por intermédio dos dispositivos, e através dos mecanismos descritos. A comunicação entre dispositivos, a seu turno, é executada na camada inferior com o concurso de avatares para tal convocados, aos quais é delegada a função de mensageiros e que, em nome daqueles outros, assistem nesse âmbito.

O mesmo esquema pode ser descrito como um algoritmo de comunicação assíncrona baseado em primitivas de comunicação *bufferizadas* e um conjunto de regras de escalonamento para o processamento de tarefas pendentes e controle de concorrência. Contudo, este resulta consideravelmente complexo e a analogia adotada permite entender o mesmo sistema visto em uma perspectiva alternativa, a qual se provou mais intuitiva durante o

Excerto de *idn/interface.c*

```

208     /* And the witchcraft: we evoke a legion of avatars to serve us. */
209
210     instance = idn_evoke (device->data->connector, NULL);
211     if (! found_it)
212         instance->data->condition = IDN_INSTANCE_BAD_ADDRESS;
213
214     device->data->servant = instance;      /* Our main servant is the */
215     device->data->interrupted = instance;  /* one we've directly evoked. */
216     free (idnp_message->message_vector);
217     idn_step ("chain establishment completed");
218 }
219 else
220 {
221     idn_log ("message arriving in a non-empty connector (return)");
222
223     /* Other possibilities might be processed here (e.g. when after having
224        waited for too long, the client issues a second message to throw in the
225        towel... this is left as a suggestion for future implemenatations */
226 }
227 return EXIT_SUCCESS;
228 }

```

Figura 6.31: Tratamento de entradas (parte 3)

trabalho de desenvolvimento e responde pelo modo como de fato a solução foi concebida.

210 Tecnicamente a estratégia é realizada pelas providências finais tomadas pela função em exame, cujo excerto de código pode ser visto na Figura 6.31. A operação que produz um avatar no plano inferior é implementada pela função sugestivamente denominada *idn_evoke*. O retorno dessa chamada é um ponteiro para um bloco de memória contendo uma instância de dispositivo para o conector fornecido como argumento, a qual lhe servirá (ao dispositivo) como mensageira.

214-217 Em seus últimos passos, após evocar a instância que será incumbida de transportar a interrupção recebida, a função registra, na estrutura do conector, que este é servido pela instância recém-criada e que a mesma é o alvo da sua interrupção.

6.3 Alocação de Cadeias de Instâncias

O arquivo *instance.h* (Figura 6.32) contém as declarações dos tipos de dados referentes à entidade nomeada instância. A estrutura que a representa possui uma dezena de membros, os quais são utilizados como variáveis de um algoritmo um tanto intrincado — razão esta que sugere apresentá-los no contexto de sua utilização, conforme descrevem-se os procedimentos que deles fazem uso.

Excerto de *idn/instance.h*

```

/*
 * The instance data
 */

struct idn_instance_data
{
    int id; /* an identifier (internal use only) */
    int state; /* what the instance is doing now */
    int action; /* field 'action' of the message being ported */
    char *command; /* field 'command' of the message being ported */
    int pending_io; /* whether there is I/O to be processed */
    int condition; /* current working condition */
    idn_connector_t *connector; /* the connector to which the instance is bound */
    struct idn_device *master; /* who the instance is currently serving */
    idn_instance_t *refer_to; /* who the instance should refer to next */
    idn_instance_t *report_to; /* who the instance should report to next */
};

```

Figura 6.32: A instância

25-26 Instâncias são criadas pela mencionada função `idn_evoke`, cujo excerto de listagem é transcrito na Figura 6.33 (arquivo *router.c*). A correspondente chamada aloca uma nova instância no conector fornecido como alvo, em nome da instância apontada como evocadora — o procedimento é, então, recursivamente repetido, evocando-se uma outra instância no alvo implícito daquela recém criada (i.e. no alvo implícito do seu conector), até o final do correspondente programa. Eis como isso é realizado passo-a-passo.

33-41 Uma instância atua sempre como uma mensageira que recebe ordens do dispositivo emissor em seu conector e a entrega para sua parceira no conector onde está o dispositivo receptor. Logo, a primeira providência é criar uma nova instância e ligá-la ao conector-alvo.

45-50 Assim que é evocada, seus parâmetros são inicializados de modo a indicar não haver entradas ou saídas pendentes, i.e. sem mensagens a receber ou a entregar para o dispositivo correspondente. Também não há mensagens a serem entregues ou recebidas para as parceiras e, portanto, a instância está pronta para atender a novos pedidos. Como a instância está livre, as variáveis que carregam os campos “ação” e “comando” das mensagens em transporte estão vazias. Assume-se, então, que a instância está em condições normais de operação e que não está a serviço especificamente de qualquer dispositivo. A inicialização dos últimos dois membros da estrutura merecem atenção especial.

Durante um ciclo de operação, as instâncias conduzem a tarefa “de mão em mão”, obtendo a mensagem cada qual da sua vizinha anterior, e passando-a aos cuidados da que a segue na cadeia. Assim, em determinado momento, uma instância qualquer, no

Excerto de *idn/router.c*

```

18 /*
19  Evoke a new instance at 'target' in behalf of 'evocator' and return a pointer
20  to it. Do this recursively evoking another instance at the implicit target of
21  the just obtained one, until the end of the program. On failure, return NULL.
22  Here, 'target' is a connector; 'evocator', an instance. That's how it works:
23  */
24
25 idn_instance_t *
26 idn_evoke (idn_connector_t *target, idn_instance_t *evocator)
27 {
28     idn_instance_t *new_instance; /* This is the new instace we'll evoke. */
29     idn_connector_t *successor;    /* This is its sucessor in the chain. */
30
31     /* First, we create a new instance in the given connector. */
32
33     new_instance = idn_add_instance (interface->instance_set);
34     if (new_instance == NULL)
35         return NULL; /* This would mean a failure. */
36
37     /* Then we bind the instance to the given target. */
38
39     new_instance->data->connector = target; /* Attach it */
40     target->data->instance_count += 1;      /* Count it */
41     new_instance->data->id = ++interface->last_instance_id; /* Identify it */
42
43     /* Now, let's set the instance's initial status; we'll change it soon. */
44
45     new_instance->data->pending_io = IDN_IO_STAND_BY; /* No pending I/O. */
46     new_instance->data->state = IDN_STATE_READY;     /* It's idle. */
47     new_instance->data->action = IDN_STATUS_VOID;    /* No action. */
48     new_instance->data->command = NULL;              /* No command. */
49     new_instance->data->condition = IDN_INSTANCE_NORMAL; /* It's alright. */
50     new_instance->data->master = NULL;                /* No master. */
51     new_instance->data->refer_to = new_instance;      /* It means that */
52     new_instance->data->report_to = new_instance;     /* there's no chain. */

```

Figura 6.33: Alocação de instâncias (parte 1)

Figura 6.34: Formação de uma cadeia (intermediária)

caminho intermediário entre a interruptora e a terminal, tem sempre a quem referir-se e a quem reportar-se, respectivamente, na fase ascendente e descendente do ciclo. No sistema em apresentação, a informação relativa a esse encadeamento de compromissos é representada por um par de ponteiros em cada instância apontando, respectivamente, para a sua antecessora e sucessora na seqüência formada. O arranjo é ilustrado na Figura 6.34.

O exemplo mostra o trecho de uma cadeia onde as instâncias *A*, *B* e *C* interagem na fase descendente do ciclo de operação. No instante determinado, *A* aguarda pela resposta à requisição que fizera a *B* que, por sua vez, detêm a tarefa na via de regresso. Esta lhe fora reportada por *C*, que já tendo cumprido sua missão de levar adiante e de volta a tarefa,

Figura 6.35: Formação de uma cadeia (completa)

permanece agora inativa até o desestabelecimento da cadeia.

A configuração descrita é a de um segmento intermediário na cadeia; resta definir como representar seu ponto inicial e terminal. No primeiro lado há uma instância que deve referir-se à sua sucessora, mas não tem a quem se reportar; no outro extremo há uma instância que deve reportar-se à sua predecessora, mas não tem a quem se referir — no caso mais particular de uma instância que não pertence a qualquer cadeia (ou que é toda cadeia ela mesma, o que é equivalente), não há nem um, nem outro.

No esboço experimental anterior, conforme comentado, valera-se da definição de um “dispositivo nulo” como artifício conceitual para representar a condição de uma entidade que, no processo de troca de mensagens, não possui devedoras ou credoras. O protótipo que se descreve, contudo, utiliza uma tática essencialmente diferente. Nesta, uma instância que não possui parceira, seja para reporte ou referência, tem o ponteiro correspondente apontando para si mesma, como exemplificado na Figura 6.35.

51-52

A maneira como isso proporciona uma representação consistente com a lógica do mecanismo de roteamento tornar-se-á clara logo adiante. Por hora, compete completar a explicação do trecho de código da Figura 6.33 anterior, adicionando que a inicialização da instância recém-invocada, sem pertencer à uma cadeia, atribui a ambos os ponteiros o endereço da própria.

Com vários ciclos concomitantes, o resultado é o estabelecimento dinâmico de seqüências que se constituem quais listas com encadeamento duplo a se justaporem àquelas do manipulador de interfaces. A diferença fundamental desse sistema, com relação ao ensaiado anteriormente, contudo, é como seu mecanismo permite lidar mais adequadamente com os problemas concernentes à confiabilidade e concorrência. Tal se deve ao fato de que, em oposição ao que fora experimentado — nas montagens preliminares onde cadeias eram pro-

gressivamente constituídas e depois dispensadas conforme percorridas pela tarefa nas fases ascendente e descendente do ciclo de operação — neste novo esquema as cadeias podem ser estabelecidas *a priori*. O processo se dá, então, em três etapas.

- a) Ao recebimento de uma interrupção, primeiro evocam-se todas as instâncias necessárias, uma no conector de cada dispositivo a ser envolvido no ciclo de operação. Por meio dos ponteiros de reporte e referência, estas se conectam na seqüência correspondente à trajetória a ser percorrida pela tarefa.
- b) Em seguida, a primeira instância recebe a interrupção do dispositivo do qual está a serviço e a encaminha para sua sucessora na cadeia, a qual, a seu turno, a entrega para o dispositivo em seu conector. Este, então, faz sua parte transmitindo a mensagem para a ID que representa, e aquela procede ao atendimento do comando recebido. Findo o trabalho, a ID devolve o resultado ao dispositivo, que o armazena até que sua instância o procure. O processo, então, se repete, de instância a instância, nas duas fases do ciclo. A fim de facilitar sua descrição, denominar-se-á *entrada* a mensagem entregue pelo dispositivo à instância, e *saída*, a mensagem enviada pela instância ao dispositivo. A mensagem trocada entre instâncias será designada por *ordem*.
- c) Quando a tarefa retorna ao dispositivo interruptor, o ciclo está completo e a cadeia pode ser desfeita. As instâncias são, então, dispensadas do serviço e a memória para elas alocada é finalmente liberada.

Em comparação com o precário módulo anterior, onde a tarefa prosseguia seu caminho ascendente, requisitando ela mesma os recursos de que necessita — na temerária esperança de que estes estejam lá, e disposta a abortar, caso contrário — o mecanismo do sistema IDN assemelha-se mais com a técnica *two-phase locking* [83] (trava em duas fases), alocando primeiro todos os recursos para só depois lançar-se à sua utilização e, finalmente, liberá-los.

Isso só é possível em virtude do fato de que a trajetória de uma tarefa depende exclusivamente do destinatário inicial especificado pela interrupção e do programa previamente configurado. Assim, o GI dispõe do conhecimento necessário para projetar o desenvolvimento do ciclo de operação e calcular antecipadamente o estabelecimento da cadeia. Trata-se

de uma propriedade da estratégia de encaminhamento implícito: exceto pela interruptora, as IDs intermediárias não podem especificar o receptor da mensagem e, logo, esta segue um caminho que pode ser previamente determinado. Garantir os recursos imprescindíveis ao completamento de uma tarefa não é condição suficiente para a implementação de transações efetivas, mas é necessária. A reserva prévia de tais recursos, nesse caso, não incorre no risco de *deadlocks*, pois em um conector podem ser evocadas tantas instâncias quanto necessárias — dentro dos limites operacionais do sistema.

Em síntese, o estabelecimento de uma cadeia dá-se quando o GI recebe uma interrupção. Evoca-se, assim, uma instância no conector do dispositivo interruptor (o início da cadeia). Esta, então, evoca sua sucessora no conector do dispositivo receptor e este faz o mesmo, recursivamente, até atingir o conector terminal do programa. A cada passo, a instância evocadora tem seu ponteiro de referência direcionado para a evocada e esta, por sua vez, tem seu ponteiro de reporte apontado para aquela.

O que acontece com as instâncias inicial e terminal da cadeia, portanto, é uma aplicação direta da mesma regra. A primeira, não sendo evocada por qualquer outra, é considerada uma auto-evocação; logo, devendo reportar-se a quem a evocou, deve reportar-se a si mesma. A última, não devendo referir-se a nenhuma instância, deve referir-se a si mesma. Fecha-se, assim, o ciclo e a trajetória da tarefa simplesmente segue os ponteiros da Figura 6.35, primeiro através das referências e depois no sentido dos reportes.

56 O processo pode ser acompanhado com o auxílio da Figura 6.36. Quando `idn_evoke` é originalmente chamada para atender à interrupção, os argumentos recebidos indicam o conector do dispositivo interruptor como alvo e não especificam o evocador — fornece-se um ponteiro para NULL em seu lugar. Isso é interpretado como uma auto-evocação.

66 A instância recém-criada nomeia o alvo explícito de seu conector como alvo para a evocação da sua sucessora (o endereço do destinatário da interrupção fora armazenado nessa variável na ocasião da chamada à `idn_receive_message`¹⁷). Seu ponteiro `report_to`
70-71 mantém o valor da inicialização (instâncias auto-evocadas reportam-se a si mesmas). A instância tem agora uma entrada pendente que deve receber do dispositivo interruptor, e

¹⁷Função descrita anteriormente na seção 6.2.3, pág. 118.

Excerto de *idn/router.c*

```

54  /* Let's see whether it's an outset or an intermediary node. */
55
56  if (evocator == NULL) /* It means we have a self-evocation (chain outset). */
57  {
58      idn_log ("self-evocation at %s %d, so chain outset",
59              target->data->name, new_instance->data->id);
60      idn_log ("%t %s %d requests explicit target",
61              target->data->name, new_instance->data->id);
62
63      /* The evoked avatar shall choose its successor (implicit target).
64         The evoked avatar shall report only to itself (as initialized). */
65
66      successor = new_instance->data->connector->data->explicit_target;
67
68      /* There is a message to be read and to be referred to the successor. */
69
70      new_instance->data->pending_io = IDN_IO_PENDING_INPUT;
71      new_instance->data->state = IDN_STATE_REFERRING;
72  }
73  else /* It means we have a normal evocation */
74  {
75      idn_log ("%t and evokes the new instance %s %d",
76              new_instance->data->connector->data->name,
77              new_instance->data->refer_to->data->id);
78      idn_log ("%t which resorts to implicit target");
79
80      /* The evoked avatar shall choose its successor (explicit target).
81         The evoked avatar shall report to its evocator. */
82
83      successor = new_instance->data->connector->data->implicit_target;
84      new_instance->data->report_to = evocator;
85  }

```

Figura 6.36: Alocação de instâncias (parte 2)

seu indicador de estado informa que está atuando na fase ascendente do ciclo, prestes a referir-se à próxima instância.

73 Quando essa mesma instância finalmente evocar sua parceira, ela o fará por intermédio novamente desta função (passando um ponteiro para si própria como evocadora). Nesse caso, a instância recém-criada é uma instância intermediária ou terminal e, logo, sua su-
83 cessora deverá ser a indicada pelo alvo implícito em seu conector. A nova instância deve ainda atualizar seu compromisso, declarando que deve reportar-se à sua evocadora. Os demais valores iniciais permanecem.

Resta, então, conectar a instância à sua parceira, a qual já fora determinada no passo anterior. O próximo passo depende, desse modo, de estar-se configurando uma instância intermediária ou de haver-se chegado ao final da cadeia (Figura 6.37).

89 Se a instância criada é fruto de uma interrupção, este último caso pode se dar no inusitado, mas possível, caso de a ID haver endereçado uma mensagem a si mesma. Caso contrário, o término do programa é encontrado quando o conector atingido não especifica um alvo implícito. A convenção, já introduzida, de representar essa ausência impondo que

 Excerto de *idn/router.c*

```

87  /* Then, we check if we have an endpoint. */
88
89  if (successor == new_instance->data->connector) /* It is an endpoint. */
90  {
91      /* The avatar shall refer to itself. The mission has been completed. */
92
93      new_instance->data->refer_to = new_instance;
94      idn_log ("referring to itself, so chain endpoint at %s id",
95              successor->data->name,
96              new_instance->data->id);
97  }
98  else /* It isn't an endpoint. */
99  {
100     /* The avatar shall refer to its successor in the chain */
101
102     new_instance->data->refer_to = idn_evoke (successor, new_instance);
103  }
104
105  /* The program may be broken, though. Just check it out. */
106
107  if (new_instance->data->connector->data->condition
108      == IDN_CONNECTOR_BROKEN_PROGRAM)
109  {
110      new_instance->data->condition = IDN_INSTANCE_BROKEN;
111      new_instance->data->report_to->data->condition = IDN_INSTANCE_BROKEN;
112  }
113  return new_instance;
114 }

```

Figura 6.37: Alocação de instâncias (parte 3)

o ponteiro correspondente aponte para o próprio conector faz, então, sentido. Resulta dela que através do algoritmo descrito, este conector naturalmente selecione a si mesmo como

93 sucessor. Em ambas as possibilidades, tudo o que é preciso é atualizar os parâmetros da instância, determinando que esta deve fazer uma auto-referência.

102 Finalmente, se este não é o término do programa, a instância recém-criada deve referir-se a uma outra instância, em seu nome evocada, no conector definido como sucessor. A chamada inicia, então, um processo recursivo de evocações até o final do programa¹⁸.

107-112 Antes de retornar, entretanto, verifica-se a condição do conector onde foi produzida a instância. Se o programa ao qual este pertence estiver quebrado naquele ponto (porque seu alvo implícito fora eliminado do manipulador por alguma razão), então, novamente, tem-se a intervenção do GI. A instância nesse conector e sua antecessora são marcadas como inutilizáveis — e como a função é recursiva, a alteração se propaga daquele ponto até o início da cadeia. Isso previne que programas quebrados, capazes de levar o sistema a um estado inconsistente, sejam inadvertidamente executados.

¹⁸“Programa”, aqui, não se referindo ao software, mas como programa fornecido ao GI.

Excerto de *idn/router.c*

```

116 /*
117  Look for the first instance ready to receive a message from its device and
118  take care of it. A pointer to the select instance is returned; on error or
119  if no instances are eligible, the function returns NULL.
120 */
121
122 idn_instance_t *
123 idn_handle_pending_input()
124 {
125     idn_instance_t *instance;
126     idn_device_t *device;
127     idn_idnp_message_t *idnp_message;
128
129     idn_log ("checking for pending inputs");
130
131     /* Seek for an instance which is waiting for an input. */
132
133     instance = interface->instance_set->first;
134     while ( instance != NULL )
135     {
136         if (instance->data->pending_io == IDN_IO_PENDING_INPUT )
137         {
138             /* If we find one, look for the device which is served by it. */
139
140             device = interface->device_set->first;
141             while ( (device != NULL))
142             {
143                 /* it's redundant, but we keep it. */
144                 if ((device->data->connector == instance->data->connector)
145                     && (device->data->buffer_state == IDN_DEVICE_BUFFER_INPUT)
146                     && (device->data->servant == instance))
147                 {

```

Figura 6.38: Processamento de entradas (parte 1)

6.3.1 Processamento de Entradas

Conforme visto, assim que uma nova cadeia é estabelecida, a primeira de suas instâncias possui uma entrada a ser recebida do dispositivo interruptor. Cumpre-lhe, então, obtê-la e dar prosseguimento ao seu transporte dentro do manipulador de interfaces. Este é o trabalho da função `idn_handle_pending_input`, cujo excerto inicial aparece na Figura 6.38.

Um laço iterativo procura, na lista de instâncias, aquelas com uma entrada pendente. Para cada uma encontrada, a lista de dispositivos é vasculhada buscando, dentre os dispositivos cujos *buffers* contém mensagens recebidas da rede, aquele que é servido pela instância encontrada. No caso de uma interrupção, os indicadores que representam tais condições já foram devidamente ativados durante os procedimentos descritos até esse ponto; no caso de um retorno, ainda a ser visitado, as variáveis serão da mesma forma configuradas pelas funções correspondentes.

Se para a instância selecionada nenhum dispositivo se apresentar nas condições mencionadas (a indicação de entrada pendente não implica que esta já esteja disponível no dispositivo), repete-se a busca para a próxima instância que satisfaz o requisito e assim

 Excerto de *idn/router.c*

```

147      /* Consistence check. If this is false, we'll have a bug.*/
148      assert ( (instance->data->state == IDN_STATE_REFERRING)
149              || (instance->data->state == IDN_STATE_REPORTING));
150
151      /* Ok, let's handle the input. */
152
153      idn_step (" handling a pending input at %s id",
154              instance->data->connector->data->name,
155              instance->data->id);
156
157      /* Transfer the command from the device to the instance. */
158
159      idnp_message = idn_get_idnp_message (device->data->message);
160      instance->data->command = strdup(idnp_message->command);
161      if (instance->data->command == NULL)
162      {
163          idn_system_failure ();
164          return NULL;
165      }
166      free (device->data->message);
167      device->data->message = NULL;
168
169      /* Update the status of both the device and the instance. */
170
171      device->data->buffer_state = IDN_DEVICE_BUFFER_EMPTY;
172      instance->data->pending_io = IDN_IO_STAND_BY;
  
```

Figura 6.39: Processamento de entradas (parte 2)

por diante. Apenas uma instância é atendida pela função — o laço é finalizado logo que a primeira instância elegível é localizada ou o final da lista é atingido, o que ocorrer primeiro. Neste último caso, a função deve retornar NULL, sinalizando a ausência de entradas pendentes a serem processadas.

159-167 Se, ao contrário (Figura 6.39), um dispositivo com uma entrada pendente e uma instância apta a recebê-la forem encontrados — necessariamente ambos no mesmo conector — então o *buffer* do dispositivo é esvaziado e seu conteúdo transformado em uma mensagem IDNP, cujo campo de comando é entregue para a instância dele encarregada. O indicador 171-172 de estado do dispositivo é alterado para informar que este está liberado para receber novos dados, e o da instância, anunciando que não mais espera por entradas.

176-181 Da mesma mensagem IDNP, o campo de ação é também armazenado na instância. Porém, não naquela que recebeu a entrada, mas na sua antecessora ou sucessora na cadeia, dependendo de qual dentre ambas lhe segue na condução da ordem. Durante a primeira fase do ciclo de operação, a instância refere-se à sua seguinte na sequência e, logo, a esta é feita a atribuição; no semi-ciclo descendente, a instância reporta-se a sua anterior e, por conseguinte, é esta que deve receber o valor da ação.

Não é difícil compreender o que ocorre. Isso é feito desse modo porque, quando a

Excerto de *idn/router.c*

```

174         /* Set the field 'action' of the next instance. */
175
176         if (instance->data->state == IDN_STATE_REFERRING)
177             instance->data->refer_to->data->action =
178                 atoi (idnp_message->action); /* The forward path.*/
179         else
180             instance->data->report_to->data->action =
181                 atoi (idnp_message->action); /* The backward path.*/
182
183         /* The master dispenses the servant; but this remains loyal.*/
184
185         device->data->servant = NULL; /*In principle, we can reuse.*/
186         instance->data->master= device; /*Use this when coming back.*/
187
188         return instance; /* We want to handle only the first input. */
189     }
190     device = device->next;
191 }
192 }
193     instance = instance->next;
194 }
195     return NULL;
196 }

```

Figura 6.40: Processamento de entradas (parte 3)

mensagem for transferida da instância para o dispositivo, na operação inversa da que se descreve, seu campo de ação é retirado do valor armazenado na instância que entrega a saída. Assim, é preciso que a receptora de uma entrada armazene o valor da ação — o código de *status* de retorno gerado pelo processamento na ID correspondente — na instância seguinte, a fim de que esse seja passado a diante. Dessa maneira, uma ID que não tenha sucesso no atendimento de um pedido, ou que deseje desautorizar o processamento de determinada tarefa daquele ponto em diante, pode notificar tal decisão gerando um retorno com o valor de ação correspondente. O mecanismo introduzido fará com que essa indicação seja incluída na saída gerada pela instância seguinte e, assim, enviada para a próxima ID.

Nesse ponto, é importante notar o que acontece com a relação de compromisso entre o dispositivo e a instância. Antes da transferência da entrada, o dispositivo tinha o segundo elemento como seu servo — aquele convocado para lhe servir de mensageiro; a instância, contudo, não devia ao dispositivo qualquer satisfação, uma vez que nenhuma incumbência lhe fora dada. Após a operação, a situação de certo modo se inverte. Havendo delegado o transporte da mensagem para a instância, o dispositivo a exime da servidão. Esta, por sua vez, ao lhe ser confiada a missão pelo dispositivo, passa a tê-lo como seu mestre.

O recurso intuitivo tem a seguinte utilidade prática. Depois que sua entrada é deixada

a cargo da instância que lhe atende, o dispositivo é liberado do vínculo que o associa àquela cadeia. Assim, um dispositivo sem servos pode, em princípio¹⁹, ser reutilizado por outras cadeias de instâncias em ciclos de operação concorrentes. Já um dispositivo que dispõe de um servo, certamente possui ainda em seu *buffer* uma mensagem para ser tratada, ou a espera receber da sua ID, que deve estar em atendimento. Por outro lado, embora liberta da condição de serva, a instância que recebe uma solicitação de transporte, firma o propósito para com o dispositivo que lhe entregou, de prestar contas do paradeiro da mensagem quando encontrá-la de novo no caminho descendente da tarefa. A mecânica desse algoritmo será melhor revisada nas seções seguintes.

6.3.2 Processamento de referências

Continuando a descrever o sistema, através da seqüência de operações realizadas a partir do recebimento de uma interrupção, o próximo passo, após o processamento da entrada pendente, é dar seguimento ao transporte da mensagem por intermédio da cadeia de instâncias para esse fim criada.

A Figura 6.41 traz a listagem da função que cuida de transferir a ordem de uma instância a outra na primeira fase do ciclo de operação. O trabalho que esta realiza é semelhante àquele executado pela função anteriormente descrita, na medida em que atende a primeira instância pronta para referir-se à sua sucessora, embora sua implementação seja distinta.

213-217 Primeiro, a lista de instâncias é percorrida e são selecionadas aquelas que, nas etapas anteriores, haviam sido declaradas em trabalho de referência e sem entradas pendentes (i.e. aquelas que já receberam suas entradas). Se nenhuma for encontrada, a função irá
226 simplesmente retornar um ponteiro nulo. Caso contrário, determina-se a quem a instância deve referir-se, bastando para isso consultar o ponteiro correspondente.

238-240 O que se faz em seguida é transferir a ordem de uma instância a outra, por meio de uma permutação dos conteúdos do *buffer* que carrega o comando em cada uma delas. A técnica usual para realizar uma troca desse tipo, em termos de código de programa, é

¹⁹Outros critérios irão definir essa possibilidade, conforme discussão posterior.

Excerto de *idn/router.c*

```

198 /*
199  Look for the first instance ready to refer to its successor and take care
200  of it. A pointer to the select instance is returned; on error or if no
201  instances are eligible, the function returns NULL.
202 */
203
204 idn_instance_t *
205 idn_handle_referring ()
206 {
207     idn_instance_t *instance;
208
209     idn_log ("cheeking for pending references");
210
211     /* Seek for the first instance with a pending reference. */
212
213     instance = interface->instance_set->first;
214     while ( (instance != NULL)
215            && ! ((instance->data->state == IDN_STATE_REFERRING)
216               && (instance->data->pending_io == IDN_IO_STAND_BY)))
217         instance = instance->next;
218
219     if (instance)
220     {
221         idn_instance_t *receiver_instance;
222         char *tmp_msg;
223
224         /* If we find one, select the receiver (we're referring now) */
225
226         receiver_instance = instance->data->refer_to;
227         assert (receiver_instance); /* Everyone has someone to refere to. */
228
229         idn_step (" handling pending reference of %s %d to %s %d",
230                 instance->data->connector->data->name, instance->data->id,
231                 receiver_instance->data->connector->data->name,
232                 receiver_instance->data->id);

```

Figura 6.41: Processamento de referências (parte 1)

efetuada em três etapas com o auxílio de uma variável temporária²⁰.

Um artifício sutil tem lugar nessa operação e diz respeito ao que acontece com a instância terminal. Nesse ponto da cadeia, a instância refere-se a si mesma e deve, portanto, efetuar uma operação de permutação de comandos “degenerada”, de si para consigo. Todavia, se respeitada a ordem dos eventos, como estes teoricamente devem ocorrer — a mensagem é primeiro suprimida da sua portadora para depois ser adicionada à entidade parceira — então, o efeito é o esperado: a instância terminal despacha sua mensagem normalmente para, em seguida, recebê-la adiante.

244-254 Completada a transferência (Figura 6.42), a ex-portadora espera agora pela resposta da sua parceira, e a nova depositária passa a possuir uma saída pendente a ser entregue para
253-256 um dispositivo no seu conector. Dependendo de esta operação se dar na fase ascendente ou descendente do ciclo, a receptora da mensagem se prepara para referir-se ou reportar-se à próxima instância na cadeia. Apenas uma referência é atendida a cada chamada — assim,

²⁰Para permutar i e j , faz-se (em C): $t=i$; $i=j$; $j=t$ ou, alternativamente $t=j$; $j=i$; $i=t$.

 Excerto de *idn/router.c*

```

234     /* Order is important here: conceptually (as with physical entities),
235        first the sender releases the message, and then the receiver gets it.
236        Even at the chain endpoint we need to stick to this principle. */
237
238     tmp_msg = instance->data->command;
239     instance->data->command = receiver_instance->data->command; /* NULL anyway */
240     receiver_instance->data->command = tmp_msg;
241
242     /* Now sender waits and receiver has a pending output. */
243
244     instance->data->state = IDN_STATE_WAITING;
245     receiver_instance->data->pending_io = IDN_IO_PENDING_OUTPUT;
246
247     /* Consistence check. If this is false, we're in trouble. */
248     assert ( (receiver_instance->data->state == IDN_STATE_READY)
249             || (receiver_instance->data->state == IDN_STATE_WAITING));
250
251     /* What receiver will do next: to refere or to report ? */
252
253     if (receiver_instance->data->state == IDN_STATE_READY)
254         receiver_instance->data->state = IDN_STATE_REFERRING;
255     else
256         receiver_instance->data->state = IDN_STATE_REPORTING;
257 }
258 return instance;
259 }

```

Figura 6.42: Processamento de referências (parte 2)

uma vez completado o procedimento, a função retorna.

6.3.3 Processamento de Saídas

Recebida a ordem de sua parceira, a nova instância deve agora entregá-la ao dispositivo em seu conector. A função `idn_handle_pending_output` é encarregada do trabalho (Figura 6.43).

277-181 Similar à sua contraparte que cuida das entradas, esta função inicia localizando, na lista de instâncias, aquelas que possuem saídas pendentes. Desta vez, no entanto, há uma diferença importante no modo como o dispositivo que deve participar da operação é selecionado, e isso se deve à técnica utilizada pelo modelo IDN para suportar replicação.

Essa é na verdade a razão pela qual não se fez as instâncias ligarem-se aos dispositivos diretamente, mas sim através de conectores: o mecanismo possibilita que dois ou mais dispositivos compartilhem o mesmo conector e, por conseguinte, também as instâncias neste evocadas. Isso permite que uma instância que possua uma saída pendente disponha de mais opções para encaminhá-la — uma vez que há sempre uma ID por dispositivo, tal proporciona maior disponibilidade de serviço naquele conector. Para se duplicar o número de dispositivos em um conector, basta que a nova ID conecte-se à rede regularmente e

 Excerto de *idn/router.c*

```

261 /*
262  Look for the first instance ready to send a message to its device and
263  take care of it. A pointer to the select instance is returned; on error
264  or if no instances are eligible, the function returns NULL.
265 */
266
267 idn_instance_t *
268 idn_handle_pending_output ()
269 {
270     idn_instance_t *instance;
271     idn_device_t *device;
272
273     idn_log ("checking for pending outputs");
274
275     /* Seek for an instance with a pending output. */
276
277     instance = interface->instance_set->first;
278     while ( instance != NULL)
279     {
280         if (instance->data->pending_io == IDN_IO_PENDING_OUTPUT)
281         {
282             /* If we find one, select a device to receive the message. */
283
284             if ( (instance->data->master!= NULL) && (instance->data->master->data->servant==NULL) )
285                 device = instance->data->master; /* The servant shall serve its master first. */
286             else
287             {
288                 /* Otherwise, any available device will do as long as ...*/
289
290                 device = interface->device_set->first;
291                 while ( device != NULL)
292                 {
293                     /* ... it has an empty buffer and no servant as yet. */
294
295                     if ( (device->data->connector == instance->data->connector)
296                         && (device->data->buffer_state == IDN_DEVICE_BUFFER_EMPTY)
297                         && (device->data->servant == NULL ) )
298                     {
299                         /* Also, it must not be interrupter of another instance!*/
300
301                         if ((device->data->interrupted == NULL)
302                             || (device->data->interrupted == instance))
303                             break;
304                     }
305                     device = device->next;
306                 }
307             }

```

Figura 6.43: Processamento de saídas (parte 1)

depois solicite ao sistema, através do comando apropriado, o traslado de seu dispositivo associado para um outro conector já existente. Pode haver lá tantos dispositivos quantos necessários.

284-285 Assim, se uma instância possui uma saída pendente, é preciso decidir qual dentre os dispositivos (se mais de um houver) em seu conector deve ser selecionado para receber a mensagem. Na solução da questão, pesa o acordo anteriormente estabelecido entre dispositivo e instância e esse busca conseguir que uma tarefa encontre, no semi-ciclo descendente, as mesmas IDs pelas quais foi processada na fase anterior. Isso é importante porque, cada ID consultada duas vezes, o primeiro comando pode tê-la levado a determinado estado (p.ex. a abertura de uma válvula) do qual deseja-se que retorne (fechamento da válvula) quando a tarefa tomar o caminho de volta²¹. Assim, havendo mais de um dispositivo em determinado conector, é interessante que, no semi-ciclo descendente, a instância lembre-se qual dentre aqueles foi o que lhe conferiu uma entrada. Convenientemente a providência foi tomada: a instância o tem como seu mestre. A regra, portanto, determina que a preferência é dada ao mestre, se a instância o reconhecer — isso é equivalente a ativar um *lock* (trava) no recurso a ser utilizado, nesse caso, o dispositivo.

290-298 Se, por outro lado, o cenário apresenta-se na fase ascendente do ciclo, a instância ainda não responde a nenhum mestre, posto que nenhuma entrada lhe fora, até então, confiada. Pode portanto escolher qualquer um dos dispositivos em seu conector desde que este esteja disponível. Para sabê-lo, é preciso observar primeiro se seu *buffer* está vazio, caso contrário corre-se o risco de sobrescrever uma entrada lá depositada por uma ID participante de outra cadeia. Em seguida, verifica-se se o dispositivo não possui um servo; porque se o possuir, então o *buffer* livre significa apenas que o dispositivo espera receber o retorno da sua ID.

301-303 A condição final para que um dispositivo seja elegível para receber a saída é que não seja interruptor. A explicação é simples. Observando o sistema pela perspectiva da aplicação, o que provoca uma interrupção é um evento assíncrono, gerado por uma aplicação através de uma ID. Pelo princípio elementar do modelo, a receptora da requisição representa para aquela um dispositivo interativo que, mediante o estímulo, retorna-lhe uma resposta.

²¹Uma outra razão relacionada para que assim seja feito será oportunamente apresentada quando comentar-se sobre a migração de dispositivos.

A interação se dá em uma configuração momentaneamente polarizada como um arranjo cliente-servidor. De um lado está a ID interruptora; do outro, todo o resto da cadeia, que se apresenta à primeira como uma única entidade parceira (em outros termos, o cliente não tem noção de que o servidor eventualmente recorra a outras IDs auxiliares).

Considera-se, então, que a ID geradora da interrupção — ao contrário das demais que atuam reativamente — agira espontaneamente e o fizera em prol da aplicação-cliente, que por definição espera receber de volta uma nota do servidor. Assim, com relação à primeira ID da cadeia, o modelo assume que o par requisição-resposta possui um significado de operação final, não meramente intermediário como entre as outras instâncias da cadeia e, portanto, determina que a interruptora não seja compartilhada entre tarefas concorrentes. Em termos práticos, isso equivale a impor que a saída de um dispositivo interruptor só deve alcançar-lhe por intermédio da instância inicial da cadeia.

Os outros dispositivos, por outro lado, podem ser compartilhados. Mesmo aquele participando de uma tarefa em andamento, desde que já tenha tido sua entrada recebida por outra instância, pode ser reutilizado em outra cadeia — posto que terá seu *buffer* vazio e seu servo dispensado e, portanto, pode ser selecionado pelo algoritmo.

322-329 Daí em diante, o procedimento é análogo ao seguido pela função que recebe as entradas, mas realizando o inverso (Figura 6.44). A mensagem é transferida da instância para o dispositivo: a instância tendo cumprido o papel que lhe cabe é, então, liberada de seu mestre (significando que sua próxima ordem pode ser entregue a qualquer dispositivo); e o dispositivo, havendo sido bem servido pela instância que o escolhera para cuidar da saída, decide que a requisitará como sua serva na próxima oportunidade (o ciclo descendente).

Nesse processo, a função `idn_get_instance_message` que realiza a transferência da mensagem, merece que lhe seja devotada atenção particular. Sua definição encontra-se no arquivo *interface.h* e pode ser consultada na Figura 6.45.

237-238 A chamada recebe o endereço do *buffer* onde deve armazenar a cadeia de caracteres formando uma mensagem IDNP/0.1, e um ponteiro para a instância onde estão os valores a serem utilizados na sua composição. Primeiro, o bloco de memória do destino é liberado
243-244 (portanto, deve haver sido anteriormente inicializado).

Excerto de *idn/router.c*

```

309     if (device)
310     {
311         /* Consistence check. If this is false, we'll have a bug. */
312         assert ( (instance->data->state == IDN_STATE_REFERRING)
313                 || (instance->data->state == IDN_STATE_REPORTING));
314
315         /*Ok, let's handle the output. */
316
317         idn_step (" handling pending output at %s %d",
318                 instance->data->connector->data->name, instance->data->id);
319
320         /* Transfer the message from the instance to the device. */
321
322         idn_get_instance_message (&device->data->message, instance);
323         device->data->buffer_state = IDN_DEVICE_BUFFER_OUTPUT;
324         instance->data->pending_io = IDN_IO_PENDING_INPUI;
325
326         /* The servant becomes free; the master acknowledge it. */
327
328         instance->data->master = NULL; /*The next order, select any. */
329         device->data->servant = instance; /*After attending, hire this. */
330
331         if (device->data->interrupted == instance) /* The end is near... */
332         {
333             idn_log ("interruption from device %d at %s %d completed",
334                     device->data->id, device->data->connector->data->name,
335                     instance->data->id);
336             device->data->interrupted = NULL;
337         }
338         return instance; /* We want to handle just one output. */
339     }
340     instance = instance->next;
341 }
342 }
343 return NULL;
344 }

```

Figura 6.44: Processamento de saídas (parte 2)

Excerto de *idn/interface.c*

```

230 /*
231  * Transfer the fields action and command of 'instance' to a newly allocated
232  * IDNP/0.1 string message and stores it in the address of '(*buffer)'. The
233  * instance's buffer is released and a pointer to the string is returned (the
234  * values in the instance may be suitably overwritten. On error, return NULL.
235  */
236
237 char *
238 idn_get_instance_message (char **buffer, idn_instance_t *instance)
239 {
240     char *address, *command;
241     int action;
242
243     free(*buffer); /* It should have been */
244     *buffer = NULL; /* previously initialized. */
245
246     /* The address depends on whether the instance is referring or reporting */
247
248     if (instance->data->state == IDN_STATE_REFERRING)
249         address = instance->data->report_to->data->connector->data->name;
250     else
251         address = instance->data->refer_to->data->connector->data->name;

```

Figura 6.45: Transferência da saída (parte 1)

Excerto de *idn/interface.c*

```

253  /* Normally, action and command are those specified by the instance. */
254
255  action = instance->data->action;
256  command = instance->data->command;
257
258  /* However, this will depend also on eventual interventions. */
259
260  switch (instance->data->condition)
261  {
262      case IDN_INSTANCE_BAD_ADDRESS: /* The addressee was not found. */
263          action = IDN_STATUS_TRANSIENT_SERVER_ERROR;
264          command = idn_msg_bad_addressee;
265          break;
266      case IDN_INSTANCE_ORPHAN: /* The chain was broken while active. */
267          action = IDN_STATUS_TRANSIENT_SERVER_ERROR;
268          command = idn_msg_broken_chain;
269          break;
270      case IDN_INSTANCE_BROKEN: /* The program is broken. */
271          if (instance->data->report_to != instance) /* But let the connector talk */
272              /* to the SYSTEM, at least. */
273              {
274                  action = IDN_STATUS_TRANSIENT_SERVER_ERROR;
275                  command = idn_msg_broken_program;
276              }
277          break;
278  }
279
280  /* OK, fill in the new buffer and release the instance's buffer. */
281
282  asprintf (buffer, "%s is %s %d is",
283            IDN_IDNP_PROTOCOL, IDN_IDNP_CONTROL, address, action, command);
284
285  free (instance->data->command);

```

Figura 6.46: Transferência da saída (parte 2)

248-251 Em seguida, o endereçamento é preenchido, e pela regra do protocolo, esse campo deve conter o nome da ID emissora: a antecessora, se o ciclo está na fase ascendente; a predecessora, do contrário. Um nome, na realidade, identifica um conector; todas as instâncias cujos dispositivos são a ele ligados compartilham o mesmo nome. É, pois, desse modo que o GI pode distribuir as requisições de acesso a um dado nome simbólico entre as diversas réplicas que seu conector pode abarcar, a fim de acomodar a carga de utilização.

255-256 Quanto ao campo de ação e comando, estes dependem da condição da instância e são calculados conforme mostra a Figura 6.46. Salvo determinações em contrário, o campo de ação da mensagem deve ser preenchido com aquele especificado pela instância. A ressalva deve ser feita porque, como estabelecido, o GI pode intervir na comunicação em diversas circunstâncias nas quais julgar necessário.

260-277 Se uma intervenção fora exercida durante o processamento anterior — algumas circunstâncias onde isso ocorre foram vistas — então, o valor do campo de ação é alterado para indicar a condição de falha e o campo de comando recebe uma descrição textual da sua causa.

281-284

Por fim, os dados são utilizados para criar uma mensagem, esta é armazenada no endereço fornecido como argumento, e o *buffer* da instância é liberado.

6.3.4 Processamento de Reportes

Na fase descendente do ciclo, o procedimento pelo qual as instâncias reportam-se às suas antecessoras é praticamente idêntico àquele que realiza o trabalho inverso, e a função `idn.handle-reporting` que o implementa é listada na Figura 6.47.

Seu código é auto-explicativo e muito parecido com o da função comentada na seção 6.3.2. O único destaque que cabe à sua apresentação concerne às mudanças de estado dos elementos envolvidos. Na ordem natural dos eventos, conforme a sequência que vem sendo descrita, uma instância passa por diversos estados diferentes ao longo de sua existência:

- i) Assim que é criada, a instância inicia sua atividade no estado representado na implementação pela macro `IDN_STATE_READY`²², que significa que a entidade está pronta para o trabalho.
- ii) Quando recebe sua primeira ordem, a instância evolui desse estado para o indicado por `IDN_STATE_REFERRING`, o que informa estar esta agora em trabalho de referência (fase ascendente do ciclo de operação).
- iii) Em seguida, após referir-se à sua sucessora emitindo-lhe uma requisição²³, a instância passa para o estado declarado como `IDN_STATE_WAITING`, isto é, põe-se à espera da resposta que lhe cabe.
- iv) Quando a tarefa toma seu caminho de volta, deve encontrar a instância novamente, e esta, depois de recebê-la, passa ao estado designado por `IDN_STATE_REPORTING`, ou seja, prestes a reportar-se à sua antecessora.
- v) Após entregá-la a seu dispositivo e recebê-la novamente, a instância passa a ordem adiante (no sentido do início da cadeia) e chega finalmente ao estado dado pela macro `IDN_STATE_DONE`, indicando que a sua missão está completa.

²²Nenhuma relação com os estados PRONTO e ATENDE no modelo da ID.

²³Definição de *requisição* e *resposta*, seção 4.6, pág. 51.

Excerto de *idn/router.c*

```

346 /*
347  Look for the first instance ready to report to its antecessor and take
348  care of it. A pointer to the select instance is returned; on error or if
349  no instances are eligible, the function returns NULL.
350 */
351
352 idn_instance_t *
353 idn_handle_reporting ()
354 {
355     idn_instance_t *instance;
356
357     idn_log ("checking for pending reports");
358
359     /* Seek for the first instance with pending reports. */
360
361     instance = interface->instance_set->first;
362     while ( (instance != NULL)
363           && ! ( (instance->data->state == IDN_STATE_REPORTING)
364              && (instance->data->pending_io == IDN_IO_STAND_BY)) )
365         instance = instance->next;
366
367     if (instance)
368     {
369         idn_instance_t *receiver_instance;
370         char *tmp_msg;
371
372         /* If we find one, select the receiver (we're reporting now) */
373
374         receiver_instance = instance->data->report_to;
375         assert (receiver_instance); /* Everyone has someone to report to. */
376
377         idn_step (" handling pending report of %s %d to %s %d",
378                 instance->data->connector->data->name, instance->data->id,
379                 receiver_instance->data->connector->data->name,
380                 receiver_instance->data->id);
381
382         /* Order is important here: conceptually (as with physical entities),
383            first the sender releases the message, and then the receiver gets it.
384            Even at the chain outset we need to stick to this principle. */
385
386         tmp_msg = instance->data->command;
387         instance->data->command = receiver_instance->data->command; /* NULL anyway */
388         receiver_instance->data->command = tmp_msg;
389
390         /* Now sender is all done and receiver has a pending output. */
391
392         instance->data->state = IDN_STATE_DONE;
393         receiver_instance->data->pending_io = IDN_IO_PENDING_OUTPUT;
394
395         /* Consistence check. If this is false, we're in trouble. */
396         assert ( (receiver_instance->data->state == IDN_STATE_WAITING)
397              || (receiver_instance->data->state == IDN_STATE_DONE));
398
399         /* What the receiver will do next: to refere or to report ? */
400
401         if (receiver_instance->data->state == IDN_STATE_WAITING)
402             receiver_instance->data->state = IDN_STATE_REPORTING;
403         else
404             receiver_instance->data->state = IDN_STATE_OVER;
405     }
406     return instance;
407 }

```

Figura 6.47: Processamento de reportes (parte 1)

392 É para este último estado que evolui a instância que efetua a entrega da ordem nessa etapa — bem como todas as suas sucessoras na cadeia, pelas quais a mensagem evidentemente já passara. Uma cadeia onde todas as instâncias já terminaram seu trabalho não é mais necessária e pode ser descartada para livrar recursos da plataforma operacional. Na implementação do sistema IDN, procurou-se realizar essa intervenção de maneira consistente com o mecanismo até então descrito, sem violar as regras de comunicação inter-instâncias. Eis como é feito.

386-388 Quando uma instância reporta-se à sua parceira, o que ocorre na prática é que ambas permutam seus *buffers* — através do mesmo procedimento utilizado no semi-ciclo ascendente. Assim como naquela fase, nesta também vale a regra no extremo da cadeia: ao reportar-se a si mesma, a instância primeiro emite a ordem e em seguida a recebe de si mesma adiante. Ocorre, então, o seguinte fenômeno. Segundo a seqüência de transições de estado, ao liberar a mensagem, a instância passa do estado `IDN_STATE_REFERING`, no qual estava, para o estado `IDN_STATE_DONE`. Para as outras instâncias da cadeia, isso significa que seu trabalho terminou. Para a primeira instância, contudo, acontece diferente, pois a receptora da mensagem que esta acaba de despachar não chega a nenhuma outra senão ela mesma.

404 Assim, uma vez que a instância já estava em `IDN_STATE_DONE`, é preciso adicionar mais um estado para a transição que decorre do recebimento de uma mensagem nessa condição. Este foi denotado `IDN_STATE_OVER`, significando que a instância fora “sobre-requisitada”. Uma instância que aí chega é certamente o início de uma cadeia obsoleta e, assim, pode ser utilizada como ponto de partida para o desestabelecimento desta.

6.3.5 Desestabelecimento de Cadeias

A função responsável pela desativação de cadeias obsoletas é listada na Figura 6.48.
425-429 A chamada simplesmente procura pela primeira instância sobre-requisitada e passa seu
436 endereço como argumento para a função `idn_dismiss`, que é quem realmente realiza o trabalho.

Esta outra, mostrada na Figura 6.49, é bastante simples. Dada uma instância como

Excerto de *idn/router.c*

```

409 /*
410  Look for the first dismissable chain and take care of it.
411  A pointer to the first instance in the dismissed chain is returned
412  (but this can be dereferenced anymore, since the memory was released.)
413  If no instances are eligible, the function returns NULL.
414 */
415
416 idn_instance_t *
417 idn_handle_pending_dismissable ()
418 {
419     idn_instance_t *instance;
420
421     idn_log ("checking for dismissable chains");
422
423     /* We seek for the first instance with state IDN_STATE_OVER. */
424
425     instance = interface->instance_set->first;
426     while ( (instance != NULL) && (instance->data->state != IDN_STATE_OVER))
427         instance = instance->next;
428     if (!instance)
429         return NULL;
430
431     idn_step (" completed chain starting at %s %d",
432             instance->data->connector->data->name, instance->data->id);
433
434     /* If we find it, dismiss the entire chain. */
435
436     idn_dismiss (instance);
437
438     return instance;
439 }

```

Figura 6.48: Localização de cadeias obsoletas

Excerto de *idn/router.c*

```

/*
  This function dismiss the given 'instance' and, recursively, all other
  instances in the chain to which it belongs, from that point on.
*/

void
idn_dismiss (idn_instance_t *instance)
{
    /* Do it recursively until we reach the end of the chain. */

    if (instance->data->refer_to != instance)
        idn_dismiss (instance->data->refer_to);

    idn_log (" dismissing %s %d",
            instance->data->connector->data->name, instance->data->id);

    /* Remove the instance.*/

    instance->data->connector->data->instance_count -= 1;
    idn_remove_instance (interface->instance_set, instance);
}

```

Figura 6.49: Desestabelecimento de cadeias

argumento, a função faz chamadas recursivas até que encontre o final da cadeia. Antes de retornar, remove a instância da lista. Isso tem o efeito de percorrer a cadeia do início até o final e, então, tomar o caminho inverso, dispensando instância por instância.

É interessante notar, pois, que além da construção *a priori* da cadeia, o mecanismo difere daquele módulo anteriormente experimentado, no fato de que esta permanece esta-

Excerto de *idn/interface.c*

```

290 /*
291  We're moving outward the system, from the core to the ICP network. This
292  function examine the device set looking for those with pending outputs
293  in their buffers and transmit them to the respective DIs.
294 */
295
296 void
297 idn_send_message ()
298 {
299     idn_device_t *device;
300
301     idn_log ("looking for messages to be released");
302
303     /* Look for devices with pending outputs. */
304
305     device = interface->device_set->first;
306     while (device != NULL)
307     {
308         if (device->data->buffer_state == IDN_DEVICE_BUFFER_OUIPUT)
309         {
310             /* Ok, let's transmit the message and release its buffer. */
311
312             idn_step ("transmitting message of %s %d at device %d",
313                     device->data->servant->data->connector->data->name,
314                     device->data->servant->data->id, device->data->id);
315
316             idn_write_output (device->data->message, device->data->socket_fd);
317
318             free (device->data->message);
319             device->data->message = NULL;
320             device->data->buffer_state = IDN_DEVICE_BUFFER_EMPTY;
321
322             /* If we happen to have written to the system device, we shall block
323              ourselves until the system has done its job; otherwise we might
324              run into race conditions. It will tell us when it's all done. */
325
326             if (device == IDN_SYSTEM_DEVICE)
327                 idn_system_synchronize ();
328
329             device = device->next;
330         }
331     }
332     idn_log ("all releasable messages already transmitted");
333 }

```

Figura 6.50: Transmissão de mensagens para a ID

belecida até o final do ciclo de operação (e não se desfaz concomitantemente com o regresso da tarefa). Duas possibilidades se abrem através desse comportamento, ambas relacionadas à implementação de transações confiáveis, e que serão oportunamente comentadas nas seções posteriores.

6.3.6 Envio de mensagens para a rede

Completando as operações de comunicação, resta apresentar a função que recolhe a mensagem armazenada no *buffer* de um dispositivo e a envia de volta para a ID através da rede. Seu código é listado na Figura 6.50.

301-316

Um laço iterativo localiza, no manipulador, todos os dispositivos cujo indicador de estado informe que seu *buffer* contém uma mensagem a ser transmitida e, para cada um

deles, a função envia o respectivo conteúdo através do descritor de arquivo correspondente.

326-327

Cuidado especial cabe, entretanto, ao caso em que o dispositivo atendido é o dispositivo de sistema. Uma vez que através deste é possível alterar os parâmetros de operação das representativas das IDs conectadas à RDI, é necessário um mecanismo de exclusão mútua para evitar condições de corrida. Tomando-se, pois, a providência de suspender o monitor de eventos de rede até que o dispositivo de sistema tenha terminado seu trabalho, garante-se que a ID eventualmente afetada não tenha seus parâmetros modificados durante o processamento de uma de suas mensagens — o que poderia levar o manipulador a um estado inconsistente. A função `idn_system_synchronize`, responsável por essa sincronização, será subseqüentemente detalhada nas próximas seções.

6.3.7 Procedimento de Desconexão

Além da conexão de novas IDs e do recebimento de mensagens daquelas já conectadas, já descritos — bem como indicações de erros oriundos do sistema operacional e interrupções do servido de rede solicitada pelo usuário, que não necessitam maiores detalhes — o último evento de rede a ser abordado é o de desconexão.

Um evento desse tipo ocorre quando uma ID envia um sinal de desconexão de transporte para o GI — nessas circunstâncias, então, a função `idn_disconnect_device` é chamada para executar os procedimentos necessários para remover a sua representativa do manipulador de interfaces (Figura 6.51).

345

O primeiro passo é decrementar o contador de dispositivo do conector para que o manipulador possa saber quantos daqueles ainda restam. Se esse número eventualmente atingir zero, a manutenção do conector não tem mais sentido e mensagens a ele endereçadas devem ser devolvidas a seus remetentes indicando endereço inexistente.

354-355

Antes de remover o conector, contudo, é conveniente tomar alguma medida corretiva com relação aos programas do GI nos quais esteja incluído. Para tal, localiza-se na lista do manipulador cada conector cujo alvo implícito seja o elemento que está por ser eliminado, e redireciona-se o ponteiro correspondente para o dispositivo do sistema, o qual passa a servir de “terminador” artificial. A condição do conector imediatamente anterior a este

360-366

 Excerto de *idn/interface.c*

```

334 /*
335  Remove the given 'device' from the interface and take further actions if
336  necessary, in order to keep the system consistent.
337  */
338
339 void
340 idn_disconnect_device (idn_device_t *device)
341 {
342     /* First we decrement the usage count of the device's connector.
343      If it reaches zero, we remove the connector and it's instances, if any. */
344
345     if (--device->data->connector->data->device_count == 0) /*There might be a */
346     {                                                         /*watermark instead.*/
347         idn_instance_t *instance;
348         idn_connector_t *connector;
349
350         idn_step ("%s will be removed", device->data->connector->data->name);
351
352         /* Manage broken programs. */
353
354         connector = interface->connector_set->first;
355         while (connector)
356         {
357             /* The system is appended at the point where the system was broken.
358              The preceding connector's condition is set accordingly. */
359
360             if (connector->data->implicit_target == device->data->connector)
361             {
362                 idn_log (" broken program at %s", connector->data->name);
363
364                 connector->data->implicit_target = IDN_SYSTEM_CONNECTOR;
365                 connector->data->condition = IDN_CONNECTOR_BROKEN_PROGRAM;
366             }
367
368             /* for consistence */
369             if (connector->data->explicit_target == device->data->connector)
370                 connector->data->explicit_target = IDN_SYSTEM_CONNECTOR;
371
372             connector = connector->next;
373         }
374     }
375 }

```

Figura 6.51: Procedimento de desconexão (parte 1)

Excerto de *idn/interface.c*

```

370
371     connector = connector->next;
372 }
373
374 /* Please, take care of those poor orphaned instances... */
375
376 instance = interface->instance_set->first;
377 while (instance)
378 {
379     /* Transfer orphaned instance to the system's connector */
380
381     if (instance->data->connector == device->data->connector)
382     {
383         idn_log (" instance id is orphan", instance->data->id);
384
385         instance->data->connector = IDN_SYSTEM_CONNECTOR;
386         IDN_SYSTEM_CONNECTOR->data->instance_count += 1;
387         instance->data->condition = IDN_INSTANCE_ORPHAN;
388         instance->data->master = NULL;
389
390         /* If the instance has pending inputs, it must have already sent
391            its message: revert to previous state with pending outputs */
392
393         if (instance->data->pending_io == IDN_IO_PENDING_INPUT)
394             instance->data->pending_io = IDN_IO_PENDING_OUTPUT;
395     }
396     instance = instance->next;
397 }
398 /* Remove the connector. */
399
400 idn_remove_connector (interface->connector_set, device->data->connector);
401 }

```

Figura 6.52: Procedimento de desconexão (parte 2)

é, então, alterada para registrar o encerramento inesperado do programa a partir daquele ponto.

Como visto, a condição do conector tem efeito na evocação de cadeias. Assim, essa sinalização alerta o GI para que intervenha nas interrupções que acessem o programa, fazendo com que a instância gerada nesse conector — e por propagação, toda a cadeia — seja marcada como inválida. Isso previne que seja inadvertidamente executado até que o programa seja recuperado.

Além de cuidar dos programas quebrados, é preciso ainda considerar o destino das instâncias que possam existir no conector prestes a desaparecer. Uma maneira de cuidar das instâncias órfãs é transferi-las aos cuidados de outro conector — e o dispositivo de sistema é mais uma vez escolhido (Figura 6.52).

376-388

Procuram-se, então, todas as instâncias ligadas ao conector a ser descartado. Cada uma delas é transferida para o conector do dispositivo de sistema (cujo contador é devidamente atualizado), tem sua condição de órfã anotada e seu compromisso com o antigo mestre anulado. Ainda, como informação complementar, vale citar que pode ser necessário um

Excerto de *idn/router.c*

```
/*  
    Finally, the core of the system. This function is responsible for  
    calling one of the available dispatchers which will manage all  
    those on-order deliveries.  
*/  
  
void  
idn_process_deliveries ()  
{  
    idn_dispatcher_A (); /* A very simple, yet suitable, dispatcher. */  
}
```

Figura 6.53: Chamada do despachador

artifício a mais para liberar a cadeia de um possível “congelamento” perpétuo.

393-394

Se acontecer de uma instância tornar-se órfã quando ainda tem em sua conta uma entrada pendente, então, simplesmente transportá-la para o outro conector não basta para solucionar o problema. Isso porque a instância espera que seu novo dispositivo lhe entregue uma mensagem, estado em que permanecerá para sempre, posto que a esperada entrega ficou para trás e seu novo dispositivo, o de sistema, não tem nenhuma outra a lhe confiar. O artifício é reverter a situação para um estágio anterior, artificialmente retrocedendo do histórico de eventos para o instante em que a instância ainda não havia feito sua entrega para o dispositivo. Quando chegar sua vez, ela o fará novamente, agora para o dispositivo de sistema, e o processo continuará normalmente.

6.3.8 O Processador

As funções que realizam o recebimento de entradas dos dispositivos para as instâncias, o transporte de referências e reportes de ordens entre estas, e a transferência de saídas das instâncias para os dispositivos, todas elas possuem em comum o fato de atender apenas uma ocorrência a cada chamada. Foram, pois, projetadas dessa maneira para que possam ser utilizadas pelo despachador central do GI, que cuida de invocá-las em determinada ordem e número de vezes necessários para diligenciar todas as entregas pendentes.

Assim, após detectado um evento trazendo uma nova mensagem, e esta haver sido transferida para o dispositivo correspondente, a função listada na Figura 6.53 é chamada para dar seqüência ao seu tratamento.

A fim de que diferentes algoritmos pudessem ser testados, o código que realiza de fato

Excerto de *idn/dispatcher.c*

```

1  /* dispatcher.c - a dispatcher to handle pending deliveries
2  *
3  *      $RCSfile: dispatcher.c,v $ $Revision: 1.4 $
4  *      IDN Project, SEL/EESC-USP.
5  *      Refer to idn.txt for further information.
6  */
7
8  #include "dispatcher.h"
9  #include "interface.h"
10 #include "router.h"
11 #include "log.h"
12
13 /*
14  Dispatch in this order: input, refere, dismiss, output, report.
15  Do it again and again until there's no more possible deliveries.
16 */
17
18 void
19 idn_dispatcher_A ()
20 {
21     idn_instance_t *instance;
22     int transitions;
23
24     do
25     {
26         transitions = IDN_NONE_FLAG;
27
28         instance = idn_handle_pending_input (interface);
29         if (instance)
30             transitions |= IDN_INPUT_FLAG;
31
32         instance = idn_handle_referring (interface);
33         if (instance)
34             transitions |= IDN_REFER_FLAG;
35
36         instance = idn_handle_pending_dismissable (interface);
37         if (instance)
38             transitions |= IDN_DISMISS_FLAG;
39
40         instance = idn_handle_pending_output (interface);
41         if (instance)
42             transitions |= IDN_OUTPUT_FLAG;
43
44         instance = idn_handle_reporting (interface);
45         if (instance)
46             transitions |= IDN_REPORT_FLAG;
47
48         idn_log ("%sdeliveries this time: %s",
49                 (transitions > 0) ? "some " : "no ",
50                 (transitions > 0) ? "doing all over again." : "we're all done.");
51         if ( (transitions & IDN_DISMISS_FLAG) )
52             idn_log ("----- END OF CICLE -----");
53     }
54     while (transitions > 0);

```

Figura 6.54: O despachador A

o trabalho foi transferido para um arquivo separado nomeado *dispatcher.c*. Neste, diversas funções alternativas, visando o mesmo propósito, podem ser definidas e, então, uma dentre elas é escolhida para a chamada.

Dentre as configurações experimentadas, uma é mostrada na Figura 6.54. O despachador A, como foi chamado, executa as operações em uma ordem diferente daquela descrita neste capítulo e que, empiricamente, ajustou-se para atender aos ciclos de operação concorrentes.

Excerto de *idn/dispatcher.h*

```
#define IDN_NONE_FLAG      0x00 /* Handled none.          */
#define IDN_INPUT_FLAG     0x01 /* Handled one input.     */
#define IDN_REFER_FLAG     0x02 /* Handled one reference.  */
#define IDN_DISMISS_FLAG   0x04 /* Handled one dismission.*/
#define IDN_OUTPUT_FLAG    0x08 /* Handled one output.    */
#define IDN_REPORT_FLAG    0x10 /* Handlde one report.    */
```

Figura 6.55: Máscaras de bits utilizadas no despachador A

zada como máscara de bits que é inicializada com zeros. Dentro do laço, invocam-se, uma a uma, todas as funções que realizam entregas, na seguinte seqüência: *i*) receber entrada, *ii*) referir requisição, *iii*) dispensar cadeia obsoleta, *iv*) liberar saída, *v*) reportar resposta.

28-46 Para cada função chamada, verifica-se seu retorno. Se uma entrega foi atendida, então ativa-se o bit correspondente na máscara de bits, utilizando-se as macros definidas em *dispatcher.h* (Figura 6.55).

54 No final, a máscara é testada. Se qualquer um dos bits estiver ativado, i.e. houvera pelo menos uma entrega realizada, então todo o laço é executado de novo e, assim por diante, até que não haja mais nenhuma entrega. O objetivo é esgotar todas as possibilidades de atendimento mediante o recebimento de uma mensagem. O funcionamento do algoritmo pode ser melhor visualizado através de um exemplo prático, mas para isso é preciso antes completar a descrição do restante do sistema, introduzindo brevemente apenas mais alguns módulos.

6.3.9 O Dispositivo de Sistema

Completado o processo inicial de conexão, a próxima ação que uma ID possivelmente deve executar é requisitar ao GI a reconfiguração de seus parâmetros operacionais para, por exemplo, substituir o nome que lhe fora provisoriamente atribuído, por outro mais adequado. Conforme explicado, isso se faz endereçando uma mensagem ao dispositivo de sistema, que é conhecido através do protocolo pelo nome reservado “SYSTEM”.

No módulo descrito no capítulo anterior, esse dispositivo era implementado de maneira a simular uma ID: mensagens destinadas ao sistema eram interceptadas e convertidas em chamadas a uma determinada função que fazia as vezes deste. No sistema IDN, por outro

lado, o dispositivo de sistema é implementado como uma ID verdadeira, por meio do suporte à programação multitarefa do sistema operacional.

Através dela foi possível construir o dispositivo de sistema como uma *thread* distinta do programa principal. Uma *thread* é uma linha de execução sequencial dentro de um processo e, se o cerne do sistema operacional assim o permitir, pode haver diversas delas. De um modo geral, dividir um processo em duas ou mais linhas de execução é equivalente a utilizar diversos processos de usuário, com a diferença essencial de que aquelas compartilham a mesma memória, além de dados como descritores de arquivos, manipuladores de sinais, dentre outros (processos necessitam de mecanismos de IPC para trocar tais informações entre si).

62-63 O dispositivo de sistema é inicializado pela função `idn_system`, cujo trecho inicial do código-fonte é transcrito na Figura 6.56. Uma vez que esse dispositivo e os demais mecanismos do GI são executados em linhas distintas — e conforme já comentado, são susceptíveis de incorrerem em condições de disputa — um mecanismo de sincronização 73-77 se faz necessário. Dentre as técnicas disponíveis, a implementação faz uso de semáforos (POSIX 1003.1b)²⁴.

81-86 Além disso, para conectar-se à linha do servidor, a ID do sistema necessita também de um *socket* TCP, o qual é devidamente configurado para comunicar-se através da Internet, 87-89 para conectar-se à porta padrão do GI (através da interface de rede *loopback* ²⁵). Em seguida, requisita-se a conexão por meio da função `connect`. Note-se, portanto, que a linha do dispositivo de sistema comunica-se com a linha do servidor como qualquer outra ID na rede, i.e. através da camada de transporte, enviando mensagens IDNP/0.1 que são processadas da maneira como foi descrita (Figura 6.57).

Até esse ponto, todas as operações se deram na linha do servidor, que é a linha principal do processo²⁶ — é pois preciso criar uma nova linha de execução para o sistema. Na plata-

²⁴Não obstante o mesmo nome, semáforos POSIX não devem ser confundido com semáforos System V.

²⁵Posto que ambas as linhas residem necessariamente na mesma estação, essa interface otimiza sua comunicação através do serviço de transporte.

²⁶Vale lembrar, portanto, que o pedido de conexão recém-registrado por `connect` não foi ainda processado; isso só se dará quando o laço principal da linha atual chamar novamente `idn_next_network_event` para receber os eventos da rede.

Excerto de *idn/system.c*

```

58 /*
59  Initialize and launch the system thread.
60 */
61
62 int
63 idn_system ()
64 {
65     pthread_t system_thread;
66     int err;
67     struct sockaddr_in servername;
68
69     idn_log ("starting system device");
70
71     /* We'll need a POSIX semaphore to synchronize both threads. */
72
73     if (sem_init (&system_semaphore, 0, 0))
74     {
75         idn_system_failure ();
76         return EXIT_FAILURE;
77     }
78
79     /* We'll need also a socket to connect them. */
80
81     system_socket = socket (PF_INET, SOCK_STREAM, 0);
82     if (system_socket < 0)
83     {
84         idn_system_failure ();
85         return EXIT_FAILURE;
86     }
87     servername.sin_family = AF_INET;
88     servername.sin_port = htons (IDN_DEFAULT_SERVER_PORT);
89     servername.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
90
91     if (connect (system_socket, &servername, sizeof (servername)))
92     {
93         idn_system_failure ();
94         return EXIT_FAILURE;
95     }

```

Figura 6.56: O dispositivo de sistema (parte 1)

Figura 6.57: O processo do GI: linhas do servidor e do sistema

Excerto de *idn/system.c*

```

97  /* We've already issued the connection request, but we must bear in mind that
98     it hasn't been processed yet (this is still the main thread). */
99
100 /* Now start the system's thread. */
101
102 err = pthread_create (&system_thread, NULL, idn_system_job, NULL);
103 if (err)
104 {
105     idn_log ("system thread failed");
106     return EXIT_FAILURE;
107 }
108 return EXIT_SUCCESS;
109 }

```

Figura 6.58: O dispositivo de sistema (parte 2)

forma GNU/Linux, a biblioteca *GNU libc* implementa as facilidades para a programação de *threads* conforme o padrão POSIX, também conhecidas como *pthread*; o correspondente suporte no núcleo do sistema é oferecido pelo recurso denominado *LinuxThreads*.

102-107 A função `pthread_create` (Figura 6.58) solicita ao sistema operacional a criação de uma nova linha dentro do processo chamador, cujos parâmetros são armazenados no endereço fornecido como primeiro argumento. O segundo argumento é um ponteiro para a função que deve ser executada pela nova linha e o terceiro, um ponteiro para os argumentos a lhe serem passados (no caso, nenhum).

Após essa chamada, o processo do GI agora possui duas linhas. A linha do servidor, que é a linha atual, retorna da função que se acabou de descrever, e deve deter-se novamente no monitor de eventos, onde encontra a já conhecida `idn_next_network_event`. É provável, contudo, que naquele ponto `select` sequer a bloqueie, dado que o pedido de conexão recém realizado em nome do dispositivo servidor lá deve estar a seu aguardo, indicando atividade no vetor de *sockets*. Nesse caso, todo o procedimento de conexão de uma nova ID é efetuado normalmente, incluindo a alocação de um dispositivo e um conector para o sistema no manipulador de interfaces, após o que, a linha do servidor retorna mais uma vez ao monitor de eventos e, agora sim, bloqueia-se.

É a vez, então, da linha do sistema, recém criada, entrar em execução. A função invocada é `idn_system_job`, que responde pelo trabalho do dispositivo de sistema. Seu código é mostrado na Figura 6.59.

O processo de conexão da ID do sistema, da maneira como descrita a pouco, assume

Excerto de *idn/system.c*

```

111 /*
112  The system thread itself. We configure the system and enter
113  a loop where we receive and process command requests.
114  */
115
116 void *
117 idn_system_job (void *args)
118 {
119     int read_bytes;
120     FILE *file;
121
122     /* Hehe... we'll not be tricked by the scheduler :) Let's wait until the
123        main thread has allocated the system connector, just for the case... */
124
125     while (!interface->connector_set->first)
126         sleep(1);
127
128     /* We have to change the name by ourselves (who would do it for us ?) */
129
130     free(interface->connector_set->first->data->name);
131     interface->connector_set->first->data->name = system_name;
132
133     /* Let's get a file from the descriptor, so that we can read lines. */
134
135     file = fdopen (system_socket, "r");
136     if (file == NULL)
137     {
138         idn_system_failure ();
139         return NULL;
140     }

```

Figura 6.59: A linha de execução do sistema (parte 1)

que a linha principal que requisitou a criação da nova *thread* tenha tempo de retornar ao monitor de eventos para receber a solicitação de conexão TCP emitida por ela mesma e, então, criar o dispositivo e o conector do sistema, tudo antes que o escalonador do sistema operacional alterne a execução para a segunda linha.

125-126 Nas condições práticas durante todos os ensaios realizados, tal hipótese tem se verificado. Entretanto, bastaria um atraso da linha principal — que pode ser experimentado com uma chamada `sleep(1)` para detê-la por um segundo — para que a linha do sistema entre em execução antes dos respectivos dispositivo e conector serem criados. O resultado é observado na forma da falha total do programa e destruição do processo em decorrência de violação de segmentação de memória (conhecida como “*crash*”.) Por precaução contra o indesejável incidente, a linha do servidor verifica inicialmente se seu conector já fora alocado, permanecendo em um laço repetido a cada um segundo, enquanto isso não acontecer.

130-131 Em seguida, altera-se o nome do dispositivo do sistema provisoriamente atribuído pelo servidor (“1”) para “SYSTEM”, como determina o protocolo. No caso das outras IDs, isso é sempre feito mediante solicitação convencional ao sistema. Nesse caso, porém, evidentemente, isso é impossível, posto que é o próprio sistema que se prepara para entrar

Excerto de *idn/system.c*

```

142  /* Main loop within the system thread. */
143
144  for(;;)
145  {
146      int size;
147
148      /* Block until we can read data from system socket. */
149
150      system_buffer = NULL;
151      errno = 0;
152      read_bytes = getline (&system_buffer, &size, file);
153      if ( read_bytes < 0)
154      {
155          idn_system_failure ();
156          continue;
157      }
158      system_buffer[read_bytes - 2] = 0; /* strip trailing CRLF */
159      idn_system_log ("system got: %s", system_buffer);
160
161      /* Process the client's request and answer accordingly. */
162
163      idn_process_request ();
164
165      idn_write_output (system_buffer, system_socket);
166
167      idn_system_log ("system sent: %s", system_buffer);
168      free(system_buffer);
169
170      /* Finally, release the main thread, which has been blocked
171       since it sent the request to the system device. */
172
173      sem_post (&system_semaphore);
174  }
175  fclose (file);
176  return NULL;
177 }

```

Figura 6.60: A linha de execução do sistema (parte 2)

em operação; logo, uma intervenção mais direta é necessária.

135-140 Como último passo na inicialização do sistema, converte-se o descritor de arquivos do *socket* do sistema em um ponteiro auxiliar para arquivo (*stream*), a fim de que se possa efetuar leitura orientada a linhas através das funções da biblioteca.

144-158 Depois disso, a função entra no laço principal (Figura 6.60) da linha de execução do sistema e lá bloqueia-se até que mensagens sejam recebidas. Para cada uma, o sistema 163-165 processa o comando solicitado e envia a correspondente saída para o GI através da rede TCP.

Qualquer mensagem recebida pela ID do sistema lhe fora certamente enviada pela linha do servidor (que é o *proxy* de todas as IDs). Segundo previamente explicado, depois de havê-la submetido, àquela *thread* deliberadamente bloqueara-se, a fim de que não concorresse com o sistema nas alterações do manipulador de interface (seção 6.3.6, pág 146) — a 179-191 função invocada naquele instante pode ser vista na Figura 6.61. Assim, logo após despa-

Excerto de *idn/system.c*

```
/*
 * Suspend the thread until system semaphore is set. This function must be
 * called by the main thread just after having written to the system device
 * in 'interface.c', so that we avoid race conditions.
 */
void
idn_system_synchronize ()
{
    idn_log ("hold on: main thread suspended");
    sem_wait (&system_semaphore);
    idn_log ("go on: main thread released");
}
```

Figura 6.61: Sincronização entre a linha do servidor e do sistema

char o retorno, o sistema deve liberar a linha principal através da operação apropriada no semáforo para esse fim criado, antes de repetir o laço principal.

207-209 A função `idn_process_request` chamada pelo sistema para processar os comandos recebidos é mostrada na listagem da Figura 6.62. Esta primeiro obtém uma estrutura de mensagem IDNP/0.1 a partir do buffer do sistema e examina seus campos. Tendo em conta que no final de uma cadeia de instâncias há sempre uma auto-referência, a ID correspondente deve receber, nessa oportunidade, uma mensagem enviada por ela mesma.

213-214 O dispositivo do sistema lida com essa particularidade simplesmente ignorando comandos cujo remetente seja o próprio sistema, devolvendo-os ao GI tal qual os recebera.

218-219 Mensagens cujo código de *status* de retorno indiquem erro também são enviadas de volta intocadas. Dessa maneira, comandos que chegam ao sistema por meio de uma cadeia inválida — por exemplo, quando o dispositivo é empregado como terminador de programas quebrados — não são processados. O mesmo vale para instâncias órfãs e interrupções com destinatário inexistente, os quais são desviados para o sistema.

223-231 As outras mensagens são consideradas para processamento subsequente (Figura 6.63). Para isso, o comando é transformado em um vetor *argv*, que é uma representação de dados comum em linguagem C: a função relacionada e seus argumentos são separados em *buffers* distintos e obtém-se para eles um vetor de ponteiros.

235-238 Em seguida (Figura 6.65), faz-se uma busca dentre os nomes de funções que o sistema suporta, os quais constam de um vetor estaticamente alocado (Figura 6.64), para determinar se algum deles coincide com o nome solicitado. Em caso positivo, um pontei-

 Excerto de *idn/system.c*

```

193 /*
194  Process the client's request. The function is supposed to return sucessfully
195  even if the requested command is not kown. Failure means an OS failure.
196  */
197
198 int
199 idn_process_request ()
200 {
201     int i;
202     char *argz, **argv;
203     int length, argc;
204
205     /* Get an IDNP message from the system buffer. */
206
207     idn_idnp_system_message = idn_get_idnp_message (system_buffer);
208     if (idn_idnp_system_message == NULL)
209         return EXIT_FAILURE;
210
211     /* Skip messages from itself (self-reference at the chain endpoint). */
212
213     if ( !strcmp(idn_idnp_system_message->address, IDN_SYSTEM_NAME) )
214         return EXIT_SUCCESS;
215
216     /* If the action isn't and indication of success, do not touch the message. */
217
218     if ( atoi(idn_idnp_system_message->action) != IDN_STATUS_SUCCESS)
219         return EXIT_SUCCESS;

```

Figura 6.62: Processamento de comandos do sistema (parte 1)

 Excerto de *idn/system.c*

```

221 /* Get both an argz and an argv vectors from the idnp command */
222
223     argz_create_sep (idn_idnp_system_message->command, 32, &argz, &length);
224     argc = argz_count (argz, length);
225     argv = (char **) malloc ( (argc + 1) * sizeof (char *));
226     if (argv == NULL)
227     {
228         idn_system_failure (); /* we could alternatively answer something like: */
229         return EXIT_FAILURE; /* "I'm sorry, but I can't process your request" */
230     }
231     argz_extract (argz, length, argv);

```

Figura 6.63: Processamento de comandos do sistema (parte 2)

Excerto de *idn/system.h*

```

14 /* Auxiliary structure to manipulate system commands. */
15
16 struct idn_system_command
17 {
18     char *name;           /* Function name. */
19     void (*function)();   /* Function to be invoked. */
20 };

```

Figura 6.64: Vetor de comandos do dispositivo de sistema

Excerto de *idn/system.c*

```

233 /* See if the requested function matches one of the implemented commands. */
234
235 i=0;
236 while (idn_system_commands[i].name != NULL)
237 {
238     if (!strcmp (idn_system_commands[i].name, argv[0]))
239     {
240         idn_system_log ("executing: \"%s\"", idn_system_commands[i].name);
241         (idn_system_commands[i].function) (argc, argv);
242         break;
243     }
244     ++i;
245 }
246 /* No... no such command, sorry. */
247
248 if (!idn_system_commands[i].name)
249 {
250     free (system_buffer);
251     idn_new_system_message (IDN_STATUS_PERMANENT_CLIENT_ERROR,
252                             "What do you mean by \"%s\" ?", argv[0]);
253 }
254 free (argv);
255 free (argz);
256 return EXIT_SUCCESS;
257 }

```

Figura 6.65: Processamento de comandos do sistema (parte 3)

241 ro de função associado àquele nome é utilizado para invocar a chamada correspondente.
 249-254 Se a procura não tiver sucesso, por outro lado, o procedimento simplesmente gera uma mensagem de resposta padrão.

As funções que executam o comando do sistema são definidas no arquivo *syscmd.c*. São procedimentos que meramente alteram os parâmetros operacionais dos elementos conectados a rede, não se fazendo necessário listá-los todos. Apenas a título de ilustração, e porque oferece um exemplo interessante, a Figura 6.66 mostra a implementação do comando “implicit”, invocado pela ID que deseja trocar de alvo implícito.

77 Primeiro a função obtém um ponteiro para o conector da ID autora da mensagem, através de uma função auxiliar (não listada) que, com base no campo de endereço da mensagem correntemente armazenada no *buffer* do servidor, faz uma correspondente chamada à já introduzida *idn_get_connector_by_name*.

Excerto de *idn/syscmd.c*

```

68 /*
69  Get/set the connector's implicit target.
70 */
71
72 void
73 idn_system_command_implicit (int argc, char **argv)
74 {
75     idn_connector_t *sender_connector;
76
77     sender_connector = idn_sender_connector (); /* A pointer to the sender. */
78
79     /* If there are no arguments, just return the current value. */
80
81     if (argc == 2)
82     {
83         idn_connector_t *new_implicit_target;
84
85         new_implicit_target = idn_get_connector_by_name (argv[1]);
86         if (new_implicit_target == NULL)
87         {
88             idn_new_system_message (IDN_STATUS_TRANSIENT_SERVER_ERROR,
89                                     "No such device");
90             return;
91         }
92         /* Update program marks and restore the connector's condition. */
93
94         ++new_implicit_target->data->program_mark;
95         --sender_connector->data->implicit_target->data->program_mark;
96         sender_connector->data->condition = IDN_CONNECTOR_REGULAR;
97
98         sender_connector->data->implicit_target = new_implicit_target;
99     }
100     idn_new_system_message (IDN_STATUS_SUCCESS, "%s",
101                             sender_connector->data->implicit_target->data->name);
102 }

```

Figura 6.66: Implementação de um comando de sistema (implicit)

81-91 Se o comando solicitado pela mensagem IDNP/0.1 contiver um argumento, este é interpretado como o valor a ser atribuído ao alvo implícito para substituir a informação atual. Nesse caso, obtém-se também um ponteiro para este outro conector — a implementação não permite que seja especificado um alvo inexistente.

94-96 Atendendo, então, à convenção utilizada para o marcador de programas, incrementa-se o contador do novo alvo e, em contrapartida, decrementa-se o contador do alvo antigo (se nenhum argumento fora fornecido, então ambas as operações se efetuam sobre o mesmo contador e, logo, adequadamente se cancelam). Se a condição do conector indicava um programa quebrado, depois do processamento do comando esta é restituída ao valor normal.

98 Por fim, o novo (ou o mesmo) endereço é armazenado como alvo implícito do conector.

6.3.10 O IDN Shell

Visto do lado externo, conforme se apresenta para as IDs, o GI atua como um roteador de mensagens. Todo trabalho de manipulação de dados e processamento de comandos,

necessários para executar sua missão, são ocultos daquelas entidades, de tal sorte que se torna difícil de ser acompanhado, a menos que seja possível observá-lo internamente enquanto exerce suas funções. Isso não se sugere apenas conveniente no intuito de permitir a visualização do mecanismo em operação — a título de ilustração — mas revelou-se também necessário (imprescindível, de fato), durante seu desenvolvimento.

Assim, como ferramenta auxiliar, foi criado um módulo especial, o qual se denominou *IDN shell*. Trata-se de um interpretador de comandos simples que permite ao usuário inspecionar, em tempo de execução, cada um dos parâmetros operacionais dos dispositivos, conectores e instâncias alocados no manipulador de interfaces. O mecanismo permite também suspender o processamento do GI ou mantê-lo em *lock-step mode* (modo passo a passo), fornecendo a visualização dos programas registrados e da situação instantânea dos ciclos de operação ativos.

A implementação desse recurso é realizada por meio de uma terceira *thread* criada dentro do processo do GI, a qual possibilita o acesso assíncrono a todas as suas estruturas de dados e funções internas. A fim de que possa ser acessado remotamente, o shell apresenta-se na forma de um outro *socket*, ao qual um ou mais clientes podem se conectar ao mesmo tempo via Internet.

73-78 A função que inicializa o shell é mostrada na Figura 6.67. Um servidor TCP igual
aquele da linha principal é criado, configurado para uma porta pré-definida (distinta do
80-84 servidor principal) e inicializado. Um semáforo especial é utilizado para sincronizar as
86-90 outras duas linhas com a linha do shell, a qual é iniciada em seguida.

A função `_idn_shell`, invocada na linha do shell, realiza um trabalho similar ao executado pela linha principal no que concerne à interação com a rede TCP: atende pedidos de conexão e desconexão, recebe e envia mensagens, processa e executa comandos. Não há, pois, razão para se deter na sua descrição, ou na das demais funções do arquivo, as quais simplesmente produzem listagens dos parâmetros do manipulador e as enviam através do *socket* para os clientes.

O único destaque cabe à função `idn_step` que aparece ao longo de todo código examinado, a qual produz um texto formatado a partir dos argumentos fornecidos e o escreve na

Excerto de *idn/shell.c*

```

60 /*
61  Initialize and start shell thread.
62 */
63
64 int
65 idn_shell ()
66 {
67     pthread_t shell_thread;
68
69     idn_log ("starting IDN shell");
70     main_thread = pthread_self(); /* We may need to send it a signal. */
71     idn_step_mode = 0;           /* Lock-step mode off. */
72     idn_active_shells = 0;       /* Number of active shells. */
73     shell_server = idn_new_server (IDN_SHELL_PORT);
74
75     /* Initialize the shell server, semaphore and thread. */
76
77     if (idn_start_server (shell_server))
78         return EXIT_FAILURE;
79
80     if (sem_init (&idn_shell_semaphore, 0, 0))
81     {
82         idn_system_failure ();
83         return EXIT_FAILURE;
84     }
85
86     if (pthread_create (&shell_thread, NULL, _idn_shell, (void *) interface))
87     {
88         idn_log ("shell thread failed");
89         return EXIT_FAILURE;
90     }
91     return EXIT_SUCCESS;
92 }

```

Figura 6.67: Inicialização do IDN shell

saída padrão, assim como a outra função `idn_log`. Diferente desta, entretanto, `idn_step` também bloqueia a linha chamadora no modo passo-a-passo, através de um semáforo — sua inserção permite escolher pontos de parada dentro do programa onde se deseja inspecionar mais atentamente o estado interno do GI.

6.3.11 O GI: Unindo os Componentes

Todas as partes apresentadas, cumpre agora reuní-las na composição do programa final. As funções especiais definidas pelo protótipo que constam do excertos de código mostrados nas seções anteriores, também nas que se seguem, constituem basicamente ferramentas auxiliares, sem grande relevância para a discussão proposta. As que aparecem com maior frequência são utilizadas para a geração de *logs*²⁷, dentre elas as macros `idn_log` (anotações da linha do servidor), `idn_system_log` (anotações da linha do sistema) e `idn_system_failure` (anotações de falha do sistema operacional), todas declaradas no

²⁷*Log* (entrada), jargão que se refere aos registros geradas por um programa para reportar eventos de interesse.

Excerto de *idn/main.c*

```

1  /* main.c - The IDN (the body of the interface manager)
2  *
3  *      $RCSfile: main.c,v $ $Revision: 1.9 $
4  *      IDN Project, SEL/EESC-USP.
5  *      Refer to idn.txt for further information.
6  */
7
8  #include <stdlib.h>
9  #include <assert.h>
10 #include "log.h"
11 #include "interface.h"
12 #include "router.h"
13 #include "network.h"
14 #include "router.h"
15 #include "shell.h"
16 #include "system.h"
17 #include "options.h"
18
19 /* Shutdown flag: if non-zero terminates the main loop */
20
21 int idn_shutdown_flag;
22
23 /* Signal handler (SIGINT) */
24
25 void
26 idn_shutdown (int signum)
27 {
28     idn_log ("system shutdown requested by user");
29     idn_shutdown_flag = 1;
30     signal (signum, idn_shutdown );
31 }
32
33 /* The interface handler (it's a global variable). */
34
35 idn_interface_t *interface;

```

Figura 6.68: IDN : programa principal (parte 1)

arquivo *log.h*. Além dessas macros, citam-se também as funções *idn_get_device_address*, *idn_get_device_address_name* (retornam o IP e o nome do *host* de onde se conecta um dado dispositivo) e *idn_get_host_system_information* (colhe dados sobre a plataforma onde o GI é executado) — suas definições encontram-se em *log.c*.

1-6 O corpo do programa está no arquivo *main.c*, listado na Figura 6.68. O código-fonte corresponde à revisão assinalada nos comentários de cabeçalho, a qual é automaticamente atribuída pelo sistema de controle de versões concorrente — o mesmo vale para todos os excertos incluídos no volume, cada qual em seu número de revisão distinto (quando este não aparece nas figuras, conste que todos referem-se ao extrato instantâneo *thesis_release*)²⁸.

21 O preâmbulo do arquivo declara a variável que controla o laço principal do programa. Seu valor determina a execução ou não da próxima iteração e pode ser alterado através
25-31 da função *idn_shudwn*, a qual pode ser invocada a qualquer instante mediante a geração

²⁸Não obstante as futuras revisões possam ser substancialmente diferentes, é sempre possível obter um extrato anterior mediante a etiqueta de *release* no repositório CVS.

 Excerto de *idn/main.c*

```

37 /* Here we go. */
38
39 int
40 main (int argc, char **argv)
41 {
42
43     idn_server_t *server;
44     idn_network_event_t network_event;
45
46     /* Handle command-line arguments. */
47
48     idn_handle_arguments (argc, argv);
49
50     /* Setup the signal handlers. */
51
52     signal (SIGINT, idn_shutdown );
53
54     /* Print host system information. */
55
56     idn_get_host_system_information ();
57
58     /* Initialize the interface handler. */
59
60     interface = idn_new_interface ();
61     if (interface == NULL)
62         return EXIT_FAILURE;
63
64     /* Initialize the TCP server. */
65
66     server = idn_new_server (0);
67     if (server == NULL)
68         return EXIT_FAILURE;
69
70     if ( idn_start_server(server) )
71         return EXIT_FAILURE;
72
73     /* Initialize the IDN shell. */
74
75     if (idn_shell())
76         return EXIT_FAILURE;
77
78     /* Initialize the systemd device. */
79
80     if (idn_system ())
81         return EXIT_FAILURE;

```

Figura 6.69: IDN : programa principal (parte 2)

34 de um sinal assíncrono para o sistema operacional. Ainda nessa parte, o manipulador de interfaces é declarado como uma variável global, acessível a todos os módulos do sistema.

48-56 O processo tem início a partir da função mostrada na Figura 6.69. Como procedimentos preliminares, o programa processa os argumentos eventualmente fornecidos na linha de comando (apenas para fornecer informações relativas à versão e utilização), registra uma chamada para o sinal SIGINT e reporta na saída padrão algumas informações acerca do sistema onde está sendo executado.

60-52 Em seguida, é alocado o manipulador de interfaces e todas as suas sub-estruturas, contendo as listas (ainda vazias) de dispositivos, conectores e instâncias. O servidor TCP 66-71 da linha principal é criado na seqüência e inicializado para aceitar pedidos de conexão na porta padrão.

Excerto de *idn/main.c*

```

83  /* This is the main loop. It blocks waiting for the next network event.
84     When one takes place, it is handled accordingly. */
85
86  idn_shutdown_flag = 0;
87  network_event.message = NULL;
88  do
89  {
90      idn_log ("waiting for the next network event");
91      idn_next_network_event (&network_event, server);
92
93      switch (network_event.type)
94      {
95
96          case IDN_NEW_CONNECTION:
97          {
98              idn_device_t *new_device;
99
100             idn_step ("new connection request");
101             new_device = idn_connect_device (network_event.socket_fd);
102             if (new_device == NULL) /* non fatal */
103                 idn_log ("*** can't connect new client");
104             else
105             {
106                 char *ip, *host_name;
107                 idn_get_device_address (&ip, new_device);
108                 idn_get_device_address_name (&host_name, new_device);
109                 idn_log ("client %d connected from %s (%s)",
110                     new_device->data->id, ip, host_name);
111                 free (ip);
112                 free (host_name);
113             }
114             break;
115         }

```

Figura 6.70: IDN : programa principal (parte 3)

75-76 O próximo passo é a inicialização do shell: a chamada cria a nova linha de execução, onde um segundo servidor TCP põe-se a monitorar a sua respectiva porta. Finalmente, 80-81 a terceira linha é inicializada com o dispositivo de sistema, o qual deve conectar-se ao primeiro servidor TCP quanto entrar em execução.

86-91 Chega-se, assim, ao laço principal do programa (Figura 6.70), onde a linha principal bloqueia-se à espera de eventos na rede. À sua detecção, um dois cinco blocos condicionais é executado, dependendo da natureza da ocorrência.

96 A primeira possibilidade testada é a chegada de uma requisição de conexão. Em seu 101 atendimento, um novo dispositivo e seu conector correspondente são alocados no manipulador, sendo o evento reportado na saída de *log*.

120-125 Se, por outro lado, o evento for referente à chegada de dados proveniente de uma ID já conectada (Figura 6.71), então a mensagem é recebida e armazenada no dispositivo 127 associado ao respectivo *socket*. O despachador é então invocado : a mensagem é transferida do dispositivo para o nível das instâncias e transportada até seu destino. Então, se as condições internas assim o permitirem, a entrega é enviada para a ID receptora.

Excerto de *idn/main.c*

```

117     case IDN_NEW_MESSAGE:
118     {
119
120         if (idn_receive_message (network_event.socket_fd,
121                                 network_event.message))
122         {
123             idn_log ("can't handle incoming message");
124             break;
125         }
126
127         idn_process_deliveries ();
128         idn_send_message ();
129
130         break;
131     }
132
133     case IDN_DISCONNECTION:
134     {
135         idn_device_t *departing_device;
136
137         departing_device = idn_get_device_by_fd (network_event.socket_fd);
138         assert (departing_device != NULL); /* it would be a bug */
139
140         idn_log ("client %d leaving the network", departing_device->data->id);
141         idn_disconnect_device (departing_device);
142
143         idn_process_deliveries ();
144         idn_send_message ();
145
146         break;
147     }

```

Figura 6.71: IDN : programa principal (parte 4)

133-141 Um evento de desconexão faz com que o dispositivo associado ao *socket* originário seja removido. Com isso, o conector ao qual se ligava pode vir a se tornar obsoleto e, por essa razão, removido, fazendo com que as instâncias que por ventura lá estivessem sejam transferidas para outro conector. A decorrente reconfiguração da estrutura interna do manipulador pode fazer com que mensagens eventualmente armazenadas no nível das instâncias, à espera da liberação dos recursos necessários ao seu prosseguimento, possam finalmente emergir à esfera dos dispositivos e, de lá, serem transmitidas para o conjunto

143-144 de IDs. Logo, invoca-se também nessas circunstâncias o despachador e, em seguida, o procedimento de envio de mensagens pela rede.

149-151 Restam apenas dois casos. Um deles é a interrupção da rede, causada pela geração de um sinal de sistema enquanto a linha principal está bloqueada à espera de eventos no monitor da rede. Se o referido sinal for do tipo SIGINT — usualmente disparado pelo usuário via entrada padrão, através do caracter de controle ^C (Control-C) — então *idn_shutdown* será invocada assíncronamente, e a variável de controle do laço será alterada, de modo a

153-155 determinar seu encerramento à próxima iteração. O último evento é interpretado como um erro na interação com a rede. Nenhuma reação é prevista.

Excerto de *idn/main.c*

```

149     case IDN_NETWORK_INTERRUPTED:
150         idn_log ("network interrupted");
151         break;
152
153     default:
154         idn_log ("network error");
155         break;
156     }
157
158     idn_log ("so we're done with this event");
159     idn_log ("");
160 }
161
162 while ( ! idn_shutdown_flag );
163
164 /* When we reach this point, the IM has been shutdown. Sockets and file
165    descriptors will be closed, threads terminated and memory released. */
166
167 idn_log("see you latter");
168
169 return EXIT_SUCCESS;
170 }

```

Figura 6.72: IDN : programa principal (parte 5)

Figura 6.73: Representação esquemática do sistema IDN

162-170

Por fim, se o laço for finalizado, o processo termina, liberando todos os recursos alocados para o sistema operacional.

Sintetizando, o programa em execução tem a estrutura esquematizada na Figura 6.73, a qual ilustra o GI com suas três linhas de execução, suas formas de interação e pontos de sincronização. A fim de realizar alguns ensaios com o sistema-piloto, provou-se conveniente dispor também da parte complementar ao GI no modelo RDI: a ID. Para tal, um mecanismo-base que implementa seu funcionamento foi desenvolvido e, a partir deste, um reduzido conjunto de IDs foi criado. A próxima seção é dedicada à sua apresentação.

Figura 6.74: Dois canais de entradas em uma ID

6.4 A Interface de Dispositivo

Conforme comentado na seção 5.1.2, uma ID é um mecanismo interativo com duas fontes de entradas complementares (Figura 6.74). De um lado está o GI, do qual a ID recebe mensagens solicitando a execução de comandos, e para o qual os respectivos resultados são retornados (comportamento reativo); do outro, há a aplicação, que pode espontaneamente gerar mensagens para serem enviadas à RDI, na forma de interrupções (comportamento ativo).

Em plataformas onde não há suporte efetivo para multitarefa, o modo usual de monitorar ambas as entradas de dados é fazê-lo por meio de *polling* dentro de um laço iterativo que verifica se cada uma delas possui dados a serem lidos e, em caso positivo, chama a função de tratamento apropriada. A precária técnica, contudo, constitui um notório desperdício de recursos da plataforma, e responde por uma dentre as motivações da opção pelo sistema operacional GNU/Linux no desenvolvimento de aplicações em rede. Uma alternativa mais racional é utilizar-se da capacidade multitarefa do sistema e suspender o programa enquanto não houver dados a serem lidos. Quando uma entrada torna-se disponível, vinda da rede ou do usuário, o programa é notificado e liberado para execução, a fim de que possa receber e processar os dados recém chegados.

Dependendo da origem da ocorrência, esta é encaminhada para um dentre ambos os procedimentos correspondentes — respectivamente, para atender a RDI ou o operador. Relembrando o modelo da ID, o primeiro caso remete ao seu comportamento reativo: o elemento recebe a mensagem evoluindo do estado PRONTO para ATENDE, aí permanecendo até findo o processamento do comando solicitado, quando retorna para a condição inicial mediante a emissão de uma resposta. Logo, o referido procedimento, nesse caso, é o responsável pela funcionalidade da ID perante à RDI, i.e. pelo conjunto de funções que a entidade exporta para as outras IDs na rede. Durante sua execução, a ID permanece no

estado de atendimento e, portanto, incapaz de receber novas mensagens (o que sequer é um risco, visto que o GI cuida para que tal não aconteça). No que concerne à implementação, essa propriedade pode ser diretamente associada à uma chamada de função.

No segundo caso, o recebimento de uma entrada da aplicação dá-se diferentemente no modo de operação ativo. Ao ser recebida na correspondente entrada, a mensagem deve apenas ser encaminhada à RDI. Lá chegando, será recebida pelo GI, que lhe alocará uma cadeia de instâncias, através da qual desenvolverá seu ciclo de operação. Uma vez completado o circuito, do GI retornará uma mensagem de resposta à ID interruptora. Assim, ao recebê-la, novamente através da entrada da rede, a ID deve, desta vez, delegá-la não ao primeiro procedimento, mas aos cuidados daquele outro na aplicação que a aguarda. Sempre que uma mensagem é recebida, porém — seja uma consulta ou a resposta de uma interrupção anterior — a reação da ID é a mesma e, logo, também neste caso uma mensagem deve ser emitida (o derradeiro evento que finaliza o ciclo de operação).

A fim de ilustrar o esquema através de um exemplo concreto, elaborou-se uma API que implementa o funcionamento essencial do modelo descrito. O software, ainda que rudimentar, fornece uma biblioteca de programação que simplifica o desenvolvimento de aplicações para RDI e que pode ser utilizada para a construção de sistemas mais sofisticados.

6.4.1 Representação da ID

Sob determinado aspecto, o mecanismo proposto assemelha-se àquele do GI, descrito na seção anterior: o laço principal da ID detém-se em um monitor de eventos até que haja atividade em um de ambos os canais. O primeiro, o da rede, recebe as mensagens oriundas do GI. A aplicação, por sua vez, constitui uma segunda linha de execução, a qual responde pelo segundo canal. Ambas as linhas interagem e cooperam entre si de um modo peculiar e não trivial que pode ser esquematicamente ilustrado como na Figura 6.75.

No diagrama, o retângulo tracejado à direita representa a *linha principal*. Supondo-a inicialmente bloqueada à espera de eventos, seja o caso em que a ID recebe uma requisição da rede, o que a faz mudar do estado PRONTO para ATENDE. Internamente, o que ocorre é o seguinte: a entrada da rede é ativada e o modo de atuação reativo é selecionado. A

Figura 6.75: Estrutura da API para interface de dispositivos em RDI

mensagem recebida é encaminhada para o procedimento de atendimento externo, onde a aplicação define a sua funcionalidade (as funções a serem executadas para cada comando do seu CID²⁹). O resultado do processamento é então enviado de volta à RDI na forma de um retorno.

O outro canal de entrada é acessado pelo operador e esta pode ser, por exemplo, uma interface gráfica que apresenta ao usuário os controles de um dispositivo remoto, ou pode ser a interface de instrumentação de um equipamento qualquer. Em ambos os casos, a *linha do operador* (retângulo tracejado à esquerda) permanece bloqueada no aguardo de entradas (teclado, porta de comunicação, aquisição de dados etc.). À sua detecção, a linha retoma a execução e produz uma mensagem, que é transmitida à linha principal pelo canal do operador. Desbloqueando-se, esta diligencia a mensagem para a RDI na forma de uma interrupção.

Mais tarde, ao final do ciclo de operação gerado pela interrupção, uma resposta deve chegar à ID através do canal da rede. Desta vez, contudo, como a aplicação a espera, a ID seleciona o modo de operação ativo e delega a mensagem ao procedimento de atendimento

²⁹Conjunto de instruções do dispositivo, seção 4.8, pág 63.

Excerto de *idn/dis/di.h*

```

22 /*
23  Representation of a DI.
24 */
25
26 struct idn_di
27 {
28     struct sockaddr_in server;          /* The server where to connect. */
29     int socket_fd;                     /* The socket file descriptor. */
30     int threads_pipe[2];               /* A pipe (inter-thread communication) */
31     char *network_buffer;             /* A buffer for the network's thread. */
32     char *operator_buffer;            /* A buffer for the operator's thread. */
33     sem_t semaphore;                  /* Semaphore for thread synchronization. */
34     int current_job;                  /* What are we doing right now ? */
35     void (*attend) (idn_di_t *);       /* Function called to attend requests. */
36     void (*process) (idn_di_t *);      /* Function called to process replies. */
37     void (*issue) (idn_di_t *);        /* Function called to issue requests. */
38     char *name;                       /* Our name (to go in field 'address'). */
39     struct idn_request *setup_requests; /* The vector of setup requests. */
40     idn_idnp_message_t *last_message; /* The last message we've received. */
41 };

```

Figura 6.76: Estrutura da ID

interno (aos cuidados do operador) e não ao externo (para processamento automático).

A descrição da implementação do sistema deve auxiliar a compreensão de todo o processo. Sua apresentação pode principiar pela Figura 6.76, que mostra a estrutura básica utilizada para representar uma ID.

28-29 A linha principal conecta a ID ao GI através de um *socket* TCP — a entrada da rede é, portanto, o descritor de arquivo a este associado. A comunicação com a linha do operador, 30 por outro lado, é realizada através de um *pipe*, o qual fornece um canal bidirecional de dados onde cada extremidade dispõe de um par de descritores de arquivos: um para 31-32 escrita, outro para leitura. A cada uma das entradas é fornecido um *buffer* distinto para o armazenamento das mensagens.

33 Após haver entregue uma mensagem à linha principal, a linha do operador deve aguardar até receber sua resposta de direito antes de submeter a próxima³⁰ — essa sincronização 34 é realizada com o auxílio de um semáforo. Por outro lado, quando uma mensagem chega vinda da rede, a linha principal necessita saber se se trata de um pedido oriundo de outra ID, ou se constitui o resultado de uma interrupção anterior. Com base nessa informação, 35-36 a função que realiza o atendimento às consultas RDI ou a função que processa respostas é invocada.

³⁰A submissão de uma segunda requisição não é prevista no sistema atual, mas pode ser concebida em futuras extensões para, por exemplo, desistir de uma requisição ou enviar um sinal urgente.

37-38 Os pedidos da aplicação são realizados por uma terceira função, esta executada na linha do operador. Não obstante o remetente das interrupções não seja relevante, a ID preenche esse campo na mensagem com seu nome. Para configurá-lo, assim como outros parâmetros operacionais, a ID pode definir um conjunto de requisições de configuração
39 a serem enviadas ao GI assim que a ID entrar em operação. As respostas às requisições
40 podem ser consultadas pela aplicação em um *buffer* especial que contém a última mensagem recebida.

6.4.2 Inicialização da ID

31-38 Para criar uma aplicação na forma de uma ID, o programa deve primeiro obter uma estrutura do tipo descrito, por meio da função `idn_new_di`, listada na Figura 6.77, cuja primeira providência é alocar a memória necessária. Em seguida, o semáforo, os indicadores
42-45 e os *buffers* são inicializados com valores apropriados.

49-51 Provisoriamente até que seja reconfigurado, o *socket* da ID é preparado para conectar-se a um servidor IDN (via TCP) no mesmo *host*, através da porta padrão do sistema. Para facilitar ainda mais o uso da API, o arquivo define ainda um conjunto de funções auxiliares
55-58 que podem ser atribuídas aos procedimentos elementares da ID, proporcionando-lhe uma funcionalidade mínima — o que cada um desses procedimentos faz tornar-se-á claro adiante.

74-84 Obtida a estrutura, a ID pode ser ativada pela função `idn_di_start`, mostrada na Figura 6.78. Isso faz com que o programa obtenha os descritores de arquivo para *socket* (entrada da rede) e para o *pipe* (entrada do operador) e em seguida envie uma requisição
98-92 de conexão para o servidor IDN preconfigurado.

96-100 Se a tentativa tiver sucesso, a linha de execução do operador é criada. A partir desse ponto, a execução do programa se divide em duas seqüências paralelas. A linha atual
102-107 prossegue chamando `idn_main_loop`, onde entra no laço principal do programa. A linha do operador, a seu turno, invoca `idn_operator_loop`, onde também atinge um circuito iterativo sem fim. Cada uma das linhas comporta-se como se segue.

Excerto de *idn/dis/di.c*

```
21 /*
22  Return a newly created DI. On success, the DI is initialized with
23  default values; on failure, it returns NULL.
24  */
25
26 idn_di_t *
27 idn_new_di ()
28 {
29     idn_di_t *new_di;
30
31     /* Allocate memory for the new structure. */
32
33     new_di = (idn_di_t *) malloc (sizeof (idn_di_t));
34     if (new_di == NULL)
35     {
36         idn_system_failure ();
37         return NULL;
38     }
39
40     /* Initialize the flags, buffers and the semaphore. */
41
42     sem_init (&new_di->semaphore, 0, 0);
43     new_di->current_job = IDN_DI_ATTEND;
44     new_di->name = NULL;
45     new_di->last_message = idn_get_idnp_message (NULL);
46
47     /* By default, we specify the localhost as the target server. */
48
49     new_di->server.sin_family = AF_INET;
50     new_di->server.sin_port = htons (IDN_DEFAULT_SERVER_PORT);
51     new_di->server.sin_addr.s_addr = htonl (INADDR_ANY);
52
53     /* The default handlers and setup request vector. */
54
55     new_di->issue = _idn_default_issue;
56     new_di->attend = _idn_default_attend;
57     new_di->process = _idn_default_attend;
58     new_di->setup_requests = _idn_default_setup_requests;
59
60     return new_di;
61 }
```

Figura 6.77: Configuração inicial da ID

 Excerto de *idn/dis/di.c*

```

63 /*
64  * Starts the DI.
65  */
66
67 int
68 idn_di_start (idn_di_t *di)
69 {
70     pthread_t server_thread;
71
72     /* Create the DI's socket and pipe */
73
74     di->socket_fd = socket (AF_INET, SOCK_STREAM, 0);
75     if (di->socket_fd < 0)
76     {
77         idn_system_failure ();
78         return EXIT_FAILURE;
79     }
80     if (pipe (di->threads_pipe))
81     {
82         idn_system_failure ();
83         return EXIT_FAILURE;
84     }
85
86     /* Connect to the IDN */
87
88     if (connect (di->socket_fd, &di->server, sizeof (struct sockaddr_in)))
89     {
90         idn_system_failure ();
91         return EXIT_FAILURE;
92     }
93
94     /* Create the operator thread */
95
96     if (pthread_create (&server_thread, NULL, idn_operator_loop, di))
97     {
98         fprintf (stderr, "can't create server thread");
99         return EXIT_FAILURE;
100     }
101
102     /* Start the main loop. */
103
104     if (idn_main_loop (di))
105         return EXIT_FAILURE;
106     else
107         return EXIT_SUCCESS;
108 }

```

Figura 6.78: Ativação da ID

Excerto de *idn/dis/di.c*

```

110 /*
111  This is the main loop where we receive and handle the messages
112  from both the network and the operator thread.
113  */
114
115 int
116 idn_main_loop (idn_di_t *di)
117 {
118     FILE *socket_file, *pipe_file;
119     fd_set active_fd_set, read_fd_set;
120     int ready;
121
122     /* Prepare the set of descriptors we're going to monitor. */
123
124     FD_ZERO (&active_fd_set);
125     FD_SET (di->socket_fd, &active_fd_set);
126     FD_SET (di->threads_pipe[0], &active_fd_set);
127
128     socket_file = fdopen (di->socket_fd, "r");
129     if (socket_file == NULL)
130     {
131         idn_system_failure ();
132         return EXIT_FAILURE;
133     }
134     pipe_file = fdopen (di->threads_pipe[0], "r");
135     if (pipe_file == NULL)
136     {
137         idn_system_failure ();
138         return EXIT_FAILURE;
139     }
140
141     /* Then we enter the loop */
142
143     for (;;)
144     {
145         /* We stop and wait for either the network or the operator thread. */
146
147         read_fd_set = active_fd_set;
148         ready = select (FD_SETSIZE, &read_fd_set, NULL, NULL, NULL);
149         if (ready < 0)
150         {
151             idn_system_failure ();
152             return EXIT_FAILURE;
153         }

```

Figura 6.79: O laço da linha principal (parte 1)

6.4.3 A Linha da Rede

124-139 A linha da rede inicia preparando o vetor de entradas a serem monitorados (Figura 6.79), adicionando-lhe os descritores de arquivo do *socket* e do *pipe* (manipuladores de arquivos são também criados para as operações de leitura e escrita orientadas à linha).

143-153 A seguir a execução adentra o laço principal onde bloqueia-se em uma chamada de sistema *select*, estado em que permanecerá até que uma das duas fontes de entrada contenha dados a serem lidos. Quando isso acontecer, uma das ações a seguir especificadas é selecionada.

155 Se o canal ativo é o *socket*, então trata-se de um evento proveniente da RDI (Figura 6.80). Se os dados não puderem ser lidos do respectivo descritor, interpreta-se a falha

Excerto de *idn/dis/di.c*

```

155     if (FD_ISSET(di->socket_fd, &read_fd_set)) /* Operator or network ? */
156     {
157         /* It's the operator. Let's read (error means disconnection) */
158
159         if (idn_read_input (&di->network_buffer, socket_file))
160             return EXIT_SUCCESS;
161         verbose (1, "> %s\n", di->network_buffer);
162
163         free (di->last_message->message_vector);
164         free (di->last_message);
165         di->last_message = idn_get_idnp_message (di->network_buffer);
166
167         /* What are we going to do with the message ? */
168
169         if ((di->current_job == IDN_DI_ISSUE) && di->process)
170             (di->process) (di); /* If it's a reply, process it. */
171         else if (di->current_job == IDN_DI_ATTEND)
172             (di->attend) (di); /* If it's a request, attend to it. */
173
174         verbose (1, "< %s\n", di->network_buffer);
175
176         /* If the message we've just handled is a reply of an interruption
177            than the operator thread must be suspended by now. Let it go. */
178
179         if (di->current_job != IDN_DI_ATTEND)
180         {
181             di->current_job = IDN_DI_ATTEND;
182             sem_post (&di->semaphore);
183         }
184     }
185     else /* The operator or the network ? */
186     {
187         /* It's the operator. Let's get its message. */
188
189         idn_read_input (&di->network_buffer, pipe_file);
190         verbose (1, "<< %s\n", di->network_buffer);
191     }
192
193     /* Ok, now let's drive the resulting message to its destiny. */
194     idn_write_output (di->network_buffer, di->socket_fd);
195     free (di->network_buffer);
196 }
197 }
198 }

```

Figura 6.80: O laço da linha principal (parte 2)

como desconexão da entidade TCP parceira — isto é, o GI terminou prematuramente (provavelmente um *shutdown* sem aviso). Caso contrário, a mensagem é recebida e armazenada no *buffer* correspondente (“última mensagem lida”). O destino a ser-lhe dado depende da razão que a trouxe à entrada da rede: uma consulta vinda da RDI ou uma resposta desta mediante prévia interrupção.

169-172

Para que isso possa ser determinado, a linha do operador — conforme será mostrado na seqüência — ativa um indicador toda vez que gera uma mensagem. Assim, com base nessa variável invoca-se o procedimento adequado. Ao terminar seu trabalho, este deve alterar o *buffer* da linha da rede, armazenando lá o resultado na forma de uma mensagem a ser enviada para a RDI em retorno. A função que provisoriamente a inicialização da ID associa a ambos os ponteiros não executa qualquer ação; simplesmente produz uma

mensagem padrão como resposta.

179-183 Além de ativar o referido indicador, a linha do operador também emite um pedido ao sistema operacional para ser bloqueada até que a sua esperada resposta esteja disponível. Assim, nesse caso é ainda preciso reverter aquela variável e liberar a linha para execução.

185-191 Se, por outro lado, a mensagem for oriunda do operador, esta é somente lida e arma-
195-196 zenada no *buffer* da forma como recebida. Modificada por um dos procedimentos ou não, a mensagem é finalmente transmitida à RDI através do *socket* TCP.

6.4.4 A Linha do Operador

212-216 Imediatamente após algumas medidas preliminares, a função que responde pela linha do operador entra em uma laço infinito (Figura 6.81) onde deve permanecer indefinidamente. As primeiras iterações, entretanto, são tratadas diferenciadamente, da forma como se segue.

A ID possui um vetor de configuração cujos elementos são estruturas do tipo mostrado na Figura 6.82. Cada um corresponde a uma requisição a ser enviada ao GI, indicando como devem ser preenchidos o campo de endereçamento, ação e comando, bem como a função que deve ser chamada para processar a respectiva resposta, quando esta chegar.

221-226 Nas primeiras repetições do laço, as requisições desse vetor são uma a uma utilizadas para compor mensagens que, cada qual à sua vez, devem ser entregues à linha principal para
230-231 despacho na forma de interrupções. Antes, porém, é preciso ativar o indicador de requisição — para que a linha principal saiba como tratar a resposta vindoura — e sobrescrever seu procedimento de processamento com a chamada associada àquela instrução de configuração
240-243 em particular. O laço é, então, bloqueado até a chegada da resposta na linha principal, após o que, restaura o procedimento original definido pelo usuário.

247-262 Um elemento com o primeiro membro igual a NULL marca o final do vetor. Quando não há mais requisições, o laço passa a invocar a função definida pelo usuário. O restante do procedimento é análogo. Até que seja substituída, a função inicialmente atribuída pela API não faz nada além de suspender o laço para sempre — nenhuma interrupção é jamais gerada.

Excerto de *idn/dis/di.c*

```

200 /*
201  This is the operator loop. It generate messages which are then sent to
202  the main thread in order to be dispatched. The function never returns.
203  */
204
205 void *
206 idn_operator_loop (void *args)
207 {
208     idn_di_t *di;
209     int iterations;
210     void (*temp_process)();
211
212     di = (idn_di_t *) args;
213     temp_process = di->process;
214     iterations = 0;
215     di->operator_buffer = NULL;
216     for (;;)
217     {
218         /* The first times, we trigger one by one of the messages stored in the
219          setup request set, until there's no one left. */
220
221         if (di->setup_requests [iterations].device != NULL)
222         {
223             idn_di_new_message (&di->operator_buffer,
224                                di->setup_requests[iterations].device,
225                                di->setup_requests[iterations].action,
226                                di->setup_requests[iterations].command);
227
228             /* Process the reply with our own procedure.*/
229
230             di->process = di->setup_requests[iterations++].reaction;
231             di->current_job = IDN_DI_ISSUE; /* We have issues a request */
232
233             idn_write_output (di->operator_buffer, di->threads_pipe[1]);
234             free (di->operator_buffer);
235
236             /* Wait until the answer gets handled in the the main thread. */
237
238             verbose (2, "operator thread suspended\n");
239             sem_wait (&di->semaphore);
240             verbose (2, "operator thread released\n");
241
242             di->process = temp_process; /* Restore the user's choice. */
243         }

```

Figura 6.81: O laço do operador (parte 1)

Excerto de *idn/dis/di.h*

```

/* A structure to represent a request. */
struct idn_request
{
    char *device;           /* Who the request comes from. */
    int action;             /* The action requested. */
    char *command;         /* The command requested. */
    void (*reaction)(idn_di_t *); /* The function to be called. */
};

```

Figura 6.82: Vetor de requisições

Excerto de *idn/dis/di.c*

```
244     else /* No more setup requests. */
245     {
246         (di->issue) (di);          /* The user-defined procedure. */
247
248         di->current_job = IDN_DI_ISSUE; /* We have issued a request. */
249
250         idn_write_output (di->operator_buffer, di->threads_pipe[1]);
251         free (di->operator_buffer);
252
253         /* Wait until the answer gets handled in the the main thread. */
254
255         verbose (2, "operator thread suspended\n");
256         sem_wait (&di->semaphore);
257         verbose (2, "operator thread released\n");
258     }
259     return NULL;
260 }
261 }
```

Figura 6.83: O laço do operador (parte 2)

A Figura 6.84 mostra as funções provisórias com as quais a ID é inicializada. O arquivo *di.c* contém ainda outras funções auxiliares que, não obstante à parte do algoritmo da ID, são fornecidas como facilidades suplementares que simplificam o uso da API no desenvolvimento de aplicações. Ainda que sua descrição não seja imprescindível, por completeza seu código é listado no final deste capítulo. São estas as funções para reconfiguração da ID (Figura 6.85), para composição de mensagens e requisições (Figura 6.86) e para comparação de mensagens (Figura 6.87).

O próximo capítulo apresenta alguns ensaios utilizando ambos a API e o protótipo IDN, os quais são úteis para descrever o funcionamento do conjunto.

Excerto de *idn/dis/di.c*

```

/* Dummy procedure to process consults and replies (just say something). */
void
_idn_default_attend (idn_di_t *di)
{
    idn_di_reply (di, IDN_STATUS_SUCCESS, IDN_DI_STANDARD_REPLY);
}

/* Dummy procedure to generate requests (do nothing; block forever). */
void
_idn_default_issue (idn_di_t *di)    {
    for (;;)
        pause (1);
}

/* Dummy procedure to process setup replies. (just acknowledge). */
void
_idn_setup_action(idn_di_t *di)
{
    idn_di_reply (di, IDN_STATUS_SUCCESS, IDN_DI_STANDARD_THANKS);
}

/* Dummy vector of setup requests (ask your name). */
idn_request_t _idn_default_setup_requests[] =
{
    {IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "name", _idn_setup_action},
    {NULL, IDN_STATUS_VOID, NULL, NULL}
};

```

Figura 6.84: "Dummy Functions" fornecidas pela API

Excerto de *idn/dis/di.c*

```

/*
    Configure 'di' to connect to the server 'hostname' at port 'port'.
*/
int
idn_di_configure_server (idn_di_t *di, char *hostname, int port)
{
    struct hostent *host;
    host = gethostbyname (hostname);
    if (host == NULL)
    {
        fprintf (stderr, "(!s %d) %s\n", hostname, port, hstrerror (h_errno));
        return EXIT_FAILURE;
    }
    di->server.sin_port = htons (port);
    di->server.sin_addr = *(struct in_addr *) host->h_addr;

    return EXIT_SUCCESS;
}

/*
    Configure the DI 'di' to attend consults with the function 'attend' and issue
    requests with the function 'issue'. NULL means not to change the handlers.
*/
void
idn_di_configure_handlers (idn_di_t *di,
                          void (*attend)(idn_di_t *di),
                          void (*issue)(idn_di_t *di))
{
    if (attend)
        di->attend = attend;
    if (issue)
        di->issue = issue;
}

```

Figura 6.85: Funções suplementares para reconfiguração da ID

 Excerto de *idn/dis/di.c*

```

/*
 * Build a new IDNP/0.1 message and fill in the last three fields with 'address'
 * 'action' and 'command', and then stores it in 'buffer'.
 */

void
idn_di_new_message (char **buffer, char *address, int action, char *command)
{
    *buffer = NULL;
    asprintf (buffer, "%s %s %s %d %s", IDN_IDNP_PROTOCOL,
              IDN_IDNP_CONTROL, address, action, command);
}

/*
 * Shortcut for building a new request and storing it in the appropriate buffer.
 */

void
idn_di_request(idn_di_t *di, char *addressee, int action, char *command)
{
    idn_di_new_message (&di->operator_buffer, addressee, action, command);
}

/*
 * Shortcut for building a new reply and storing it in the appropriate buffer.
 */

void
idn_di_reply (idn_di_t *di, int action, char *command)
{
    idn_di_new_message (&di->network_buffer,
                        di->name ? di->name : IDN_DI_CLIENT_NAME, action, command);
}

/*
 * Fill in an idn_request_t structure.
 */

void
idn_di_build_request (idn_request_t *request, char *device, int action,
                     char *command, void (*reaction)() )
{
    if (device != NULL)
    {
        request->device = strdup (device);
        if (request->device == NULL)
        {
            fprintf (stderr, "%s", strerror(errno));
            exit (EXIT_FAILURE);
        }
    }
    else
        request->device = NULL;

    request->action = action;

    if (command != NULL)
    {
        request->command = strdup (command);
        if (request->command == NULL)
        {
            fprintf (stderr, "%s", strerror(errno));
            exit (EXIT_FAILURE);
        }
    }
    else
        request->command = NULL;

    request->reaction = reaction;
}

```

Figura 6.86: Funções suplementares para composição de mensagens

 Excerto de *idn/dis/di.h*

```

/* True if the last received message has field 'field' equal to 'value' */
#define idn_di_if_message(di, field, value) \
  (!strcmp (di->last_message-> ## field, value))

/* True if the last received message has a successful return status code. */
#define idn_di_success(di) \
  (atoi (di->last_message->action) == IDN_STATUS_SUCCESS)

/* True if the last received message has an unsuccessful return status code. */
#define idn_di_failure(di) \
  (atoi (di->last_message->action) != IDN_STATUS_SUCCESS)

/* Return if the last received msg. has an unsuccessful return status code. */

/* If the last received message has the field 'field' equal to 'value'
   then call the function 'do_this' passing the DI 'di' as argument. */

#define idn_di_response(di, field, value, do_this) \
do \
{ \
  if ( ! strcmp (di->last_message-> ## field, value)) \
  { \
    (do_this)(di); \
    return; \
  } \
} \
while (0)

/* If the last received message has the field 'field' equal to 'value'
   then do nothing and answer with action 'status' and command 'reply'. */

#define idn_di_standard_reply(di, field, value, status, reply) \
do \
{ \
  if ( ! strcmp (di->last_message-> ## field, value)) \
  { \
    idn_di_reply (di, status, reply); \
    return; \
  } \
} \
while (0)

/* If the last received message has the field 'field' equal to 'value'
   then return immediately from the current function. */

#define idn_di_skip(di, field, value) \
do \
{ \
  if ( ! strcmp (di->last_message-> ## field, value)) \
    return; \
} \
while (0)

/* If the last received message has the field 'field' not equal to 'value'
   then return immediately from the current function. */

#define idn_di_skip_not(di, field, value) \
do \
{ \
  if ( strcmp (di->last_message-> ## field, value)) \
    return; \
} \
while (0)

void
idn_di_build_request (idn_request_t *, char *, int, char *, void (*)() );

```

Figura 6.87: Macros suplementares para comparação de mensagens

Capítulo 7

Ensaaios e Resultados

Complementando a descrição do protótipo IDN, este capítulo apresenta as informações básicas para a operação do software. Através de ensaios que exemplificam sua aplicação em configurações práticas, essa parte do estudo oferece uma demonstração do seu funcionamento elementar, e do seu comportamento em condições reais. A partir dos resultados anotados, extraem-se algumas conclusões acerca da viabilidade e das limitações do modelo, bem como dos recursos tecnológicos atualmente disponíveis à sua construção.

7.1 Instruções de Configuração e Inicialização do GI

No eventual interesse pela reprodução dos experimentos descritos neste capítulo, convém registrar as instruções essenciais para a configuração e inicialização do sistema-piloto utilizado (as notas são válidas para a versão apresentada, especificamente o extrato da etiqueta *thesis_release* do repositório CVS). Principiando pelo software que implementa o gerenciador de interfaces no protótipo IDN, o primeiro passo consiste na construção do módulo executável a partir do código-fonte, organizado em uma coleção de arquivos, em sua maior parte examinados no capítulo anterior.

Para tal, deve-se invocar o utilitário *make* [93] no diretório principal, onde encontra-se o arquivo *Makefile* (Figura 7.1) contendo todas as regras para compilação automática do programa. Com base neste, o comando `make idn` cuida de gerar todos os objetos binários e

 Excerto de *idn/Makefile*

```
# Makefile - General Makefile to build the software
#
#          SRCSfile: Makefile,v $ $Revision: 1.39 $
#          IDN Project, SEL/EESC-USP.
#          Refer to idn.txt for further information.
#
# This Makefile relies on features of GNU make and GCC.
# Targets: idn (the IM), dis (the Dis), all and clean (as usual)

# The standard shell and our mighty compiler; no implicit rules

export

SEHLL    = /bin/sh
CC       = gcc
SUFFIXES =

# These flags are needed; user may add his own flags through CFLAGS

C_FLAGS = -Wall -D_GNU_SOURCE -ansi -g
LD_FLAGS = -lpthread

# The source files

sources = main.c list.c interface.c device.c connector.c instance.c \
          network.c log.c protocol.c router.c shell.c system.c \
          syscmd.c dispatcher.c options.c

# Objects and their own makefiles

objects = $(sources:.c=.o)
makefiles = $(sources:.c=.d)

# Default targets

all : idn clients

# Automatic prerequisite generation

%.d: %.c
    set -e; $(CC) -MM $(CPPFLAGS) $< \
    | sed 's/\($*\)\.o[ :]*/\1.o $@ : /g' > $@; \
    [ -s $@ ] || rm -f $@
include $(makefiles)

# Rule for objects

%.o : %.c
    $(CC) $(C_FLAGS) $(CPPFLAGS) $(CFLAGS) -c $<

# The server (Interface Manager)

idn : $(objects)
    $(CC) $(LD_FLAGS) $(LDLFLAGS) $(CFLAGS) $^ -o $@

# The clients (Device Interfaces)

clients:
    cd dis && $(MAKE)

# Clean all

.PHONY : clean
clean :
    rm -rf $(objects) $(makefiles) *~ idn \#*
    cd dis/ && $(MAKE) clean
```

Figura 7.1: Makefile para compilação automática do software

depois ligá-los no módulo final¹, correspondente ao arquivo *idn*, gerado no mesmo diretório ao término do processo.

¹O arquivo *Makefile* fornecido nessa versão experimental é ainda incompleto e não contém regras para instalação dos executáveis, arquivos de documentação etc.

Saída de `idn --help`

```
megryan:~/idn$ ./idn --help
Usage: idn [OPTION...]
IDN - Interactive Device Network (interface manager)

  -?, --help            Give this help list
  --usage              Give a short usage message
  -V, --version         Print program version
megryan:~/idn$
```

Figura 7.2: Invocação do módulo `idn` no terminal

O GI pode ser, então, invocado diretamente na linha de comando. Conforme a prática convencional válida para sistemas GNU/Linux, informações essenciais acerca do programa podem ser obtidas mediante a chamada do programa no *prompt* do interpretador de comandos do terminal, com os argumentos `--help`, `--usage` e `--version` (Figura 7.2).

Invocado sem quaisquer argumentos, o programa deve iniciar o GI. Entretanto, deve se estar atento para a possibilidade de o módulo *idn* falhar, tendo sua inicialização abortada caso o número de porta especificado para o *socket* TCP coincidir com o utilizado por algum outro processo em execução no sistema. O número 5555, arbitrariamente escolhido, tem sido freqüentemente utilizado como “porta de teste” em implementações experimentais; todavia, embora atualmente livre na plataforma utilizada, essa porta pode eventualmente conflitar com futuras especificações. Nesse caso, um novo valor deve ser selecionado no arquivo de cabeçalho *globals.h*, onde é conveniente observar que a faixa de 0 a 1023 (*well-known ports*) é controlada pela IANA (*Internet Assigned Numbers Authority*) para serviços padronizados na Internet; a faixa de 1024 a 49151 (*registered ports*) é reservada para serviços locais do sistema; a faixa de 49152 a 65535 (*private ports*) é livre e usualmente utilizada pelo sistema operacional para a criação de portas efêmeras (RFC 1700) [94] — no sistema Debian a lista das portas em utilização pode ser consultada em */etc/services*. Se qualquer alteração for necessária o software deve ser recompilado novamente através do comando `make`.

Se a chamada tiver sucesso, as informações produzidas no terminal devem ser análogas às aquelas mostradas na Figura 7.3. Todas as anotações geradas pelas funções de `log idn_log`, `idn_step` e `idn_system_log` são direcionadas para a saída padrão do processo, ao passo que a função `idn_system_failure` direciona seus dados para a saída de erro padrão. Em geral, ambos os canais correspondem ao terminal, mas podem ser redirecionados para um

Saída de `idn --help`

```

1 megryan:~/idn$ ./idn
2 >
3 > IDN 0.1 (LAMI, University of Sao Paulo)
4 > running in megryan.sel.eesc.sc.usp.br (143.107.235.7)
5 > which is a i686 Linux 2.2.17
6 >
7 > starting IDN shell
8 > starting system device
9 > waiting for the next network event
10 > new connection request
11 > client 1 connected from 127.0.0.1 (localhost)
12 > so we're done with this event
13 >
14 > waiting for the next network event

```

Figura 7.3: Iniciando o módulo `idn`

arquivo no caso de se desejar armazenar os dados para consulta posterior.

3-5 Após uma sucinta apresentação, o programa exibe informações coletadas a respeito da
 7-8 plataforma onde está sendo executado, incluindo o nome da estação, o endereço de rede
 Internet (IP), dados do *hardware* (CPU) e do sistema operacional. Depois, inicia-se o
IDN shell ² e, em seguida, também a ID do sistema, a qual, dentre outras providências,
 9-10 emite um pedido de conexão para a camada de transporte na porta do GI. Assim, quando
 atinge o monitor de eventos da rede, a linha principal do programa sequer bloqueia-se,
 posto que já se encontra lá a requisição do sistema. Atendendo a solicitação, a conexão do
 14 primeiro cliente é efetuada, após o que, o laço principal finalmente detém-se à espera de
 nova atividade no vetor de *sockets* monitorado.

1-5 Então, com o auxílio de um aplicativo *telnet* ordinário (Figura 7.4), a partir de qualquer
 estação na rede, é possível estabelecer uma conexão TCP com o IDN shell e, através desta,
 ganhar acesso a um conjunto de comandos que permitem inspecionar internamente as
 estruturas de dados do manipulador de interfaces e controlar a execução do algoritmo de
 encaminhamento.

7-8 Diretamente no *prompt* do *IDN shell*, o comando `ld` fornece uma lista de todos os
 dispositivos alocados no manipulador — logo após a inicialização do GI, a única entrada
 é a do dispositivo de sistema. O formato de cada linha de resposta é o esquematizado na
 Figura 7.5(a). O primeiro número à direita é a variável `id` da estrutura `idn_device_t`,
 que serve apenas como um identificador exclusivo do dispositivo, afim de distingui-lo na

² *IDN shell* refere-se à sub-aplicação do módulo `idn`; não confundir com *shell* do sistema GNU/Linux.

Saída de IDN shell

```

1 monaco@winona:~$ telnet megryan 5556
2 Trying 143.107.235.7...
3 Connected to megryan.sel.eesc.sc.usp.br.
4 Escape character is '^]'.
5 IDN Shell vr 0.1
6 idn>
7 idn> ld
8 1 [SYSTEM ] empty, srv 0, int 0 :
9 idn>
10 idn> lc
11 [SYSTEM ] 1 d, 0 i, 1 p, Regular <SYSTEM/SYSTEM>
12 idn>
13 idn> l
14
15 1. empty srv 0, int 0
16 [SYSTEM ]
17
18 idn>

```

Figura 7.4: Conectando-se ao IDN shell

Figura 7.5: Formato de saída do comando `lc`

listagem (internamente, todas as referências são feitas através de ponteiros). Na sequência, o símbolo entre colchetes é o nome do conector ao qual o elemento está ligado, e adjacente a este vem a indicação de que o *buffer* do dispositivo encontra-se vazio. Uma vez que não há instâncias, os últimos dois campos indicam que o dispositivo não tem um servo, e que não emitira qualquer interrupção. Finalmente, se o dispositivo portasse uma entrada ou saída pendente, a mensagem apareceria após o sinal de dois pontos.

10-11 Os dados referentes ao conector do sistema, e de todos os outros existentes, podem ser consultados mediante o comando `lc`. Nesta ordem, cada um dos campos tem o significado indicado na Figura 7.5(b): o nome do conector entre colchetes, o número de dispositivos e de instâncias associados, o indicador de programa, a condição do conector e, entre sinais

Saída de `idn`

```
megryan:~/idn$ ./idn
>
> IDN 0.1 (LAMI, University of Sao Paulo)
> running in megryan.sel.eesc.sc.usp.br (143.107.235.7)
> which is a i686 Linux 2.2.17
>
> starting IDN shell
> starting system device
> waiting for the next network event
> new connection request
> client 1 connected from 127.0.0.1 (localhost)
> so we're done with this event
>
> waiting for the next network event
> new shell connection
> system shutdown requested by user
> network interrupted
> so we're done with this event
>
> see you latter
megryan:~/idn$
```

Figura 7.6: Finalizando o GI

“<>”, os alvos explícito e implícito.

13-16

A estrutura do manipulador completa pode ser mais facilmente visualizada através do comando 1, que tem a seguinte forma: para cada conector, uma linha contendo seu nome entre colchetes é encimada pela lista de dispositivos a ele conectados, ao passo que na parte de baixo são mostradas as instâncias que eventualmente lá existirem.

Finalmente, a finalização do GI (Figura 7.6) pode ser solicitada a qualquer momento de duas maneiras: localmente no console, enviando um caracter `^C` (via teclado, usualmente através da combinação “Control-c”; ou ainda com o auxílio do comando de sistema `kill`) ou remotamente, mediante o comando `IDN shell kk`. Em ambos os casos, um sinal `SIGINT` é enviado ao processo, especificamente para a linha de execução principal. A função `idn.shutdown` registrada como capturador desse sinal é invocada assincronamente para alterar a *flag* (variável de indicação) de controle do laço principal, o que causa o *shutdown* do GI.

Ainda como nota complementar, que pode vir ser de utilidade prática em investigações experimentais que eventualmente sejam conduzidas a partir do software apresentado, é oportuno registrar uma medida da qual se valeu durante o desenvolvimento do sistema-piloto e cuja necessidade se fez notar em virtude da forma como o mecanismo servidor foi implementado. O detalhe a que se refere a nota diz respeito às especificações do protocolo TCP, em cujo modelo de transições existe um estado designado `TIME_WAIT`, o qual su-

cede a uma operação *active close* (encerramento ativo). Em termos sucintos, a condição é invariavelmente alcançada pela entidade de transporte que, havendo uma conexão estabelecida, requisita seu término mediante o envio de um segmento FIN à entidade parceira. Nesse caso, mesmo finda a conexão e liberada a porta por ela alocada, o serviço de transporte do sistema operacional é proibido de reutilizá-la por um tempo determinado pela constante MSL³, a fim de que eventuais datagramas remanescente da conexão extinta não se confundam com os da nova.

Infelizmente, devido ao modo como funciona a presente versão do protótipo IDN, isso acontece com o processo do GI durante a finalização (*shutdown*). O problema pode ser identificado com o auxílio do utilitário de sistema *netstat*, em geral disponível nas plataformas tipo unix.

4-5 A Figura 7.7 (a) mostra as dez primeiras linhas geradas pelo comando quando invocado na estação *megryan*, onde o programa *idn* está em execução. A saída informa que há quatro conexões TCP estabelecidas, fornecendo o endereço das duas extremidades. A primeira refere-se ao *socket* do GI (servidor, porta 5555), ao passo que a segunda mostra o *socket* do dispositivo do sistema⁴ (cliente, porta efêmera criada pelo sistema operacional). Cada uma delas aparece como a entidade parceira da outra, respectivamente, em um arranjo cruzado⁵.

4-5 O cenário após o encerramento do servidor (*idn*) pode ser visualizado na Figura 7.7 (b). Quando o processo termina, ambos os *sockets* entram em TIME_WAIT e, portanto, suas portas ficam indisponíveis para uso subsequente. Como resultado, qualquer tentativa de invocar o programa logo em seguida falha com a mensagem de erro: *idn: network.c: 67: idn_start_server: Address already in use*⁶ — a situação permanece inalterada até o instante em que as portas sejam liberadas pelo sistema, como registrado na Figura 7.7 (c).

³*Maximum segment lifetime* (tempo de vida máximo do segmento): o tempo que um datagrama IP pode sobreviver na Internet, após o qual é descartado.

⁴O identificador *localhost* é um sinônimo para o sistema local, qualquer que seja seu nome.

⁵As outras duas linhas mostram que há um par de conexões no serviço de login remoto seguro (ssh) a partir da estação *winona*. Não há qualquer relação com as anteriores.

⁶A linha 67 de *network.c*. é a mensagem de erro após a falha da chamada à função *bind*.

Saída de netstat

```

1 megryan:~$ netstat | head -n 10
2 Active Internet connections (w/o servers)
3 Proto Recv-Q Send-Q Local Address           Foreign Address         State
4 tcp        0      0 localhost:5555          localhost:1056          ESTABLISHED
5 tcp        0      0 localhost:1056          localhost:5555          ESTABLISHED
6 tcp        0      0 megryan.sel.eesc.sc:ssh winona.sel.eesc.sc:4963 ESTABLISHED
7 tcp        0      0 megryan.sel.eesc.sc:ssh winona.sel.eesc.sc:4952 ESTABLISHED
8 Active UNIX domain sockets (w/o servers)
9 Proto RefCnt Flags       Type       State      I-Node Path
10 unix  1      [ ]        STREAM     CONNECTED  8607  @00000000c
11 unix  1      [ ]        STREAM     CONNECTED  8963  @00000000e

```

(a)

Saída de netstat

```

1 megryan:~$ netstat | head -n 10
2 Active Internet connections (w/o servers)
3 Proto Recv-Q Send-Q Local Address           Foreign Address         State
4 tcp        0      0 localhost:5555          localhost:1056          TIME_WAIT
5 tcp        0      0 localhost:1056          localhost:5555          TIME_WAIT
6 tcp        0      0 megryan.sel.eesc.sc:ssh winona.sel.eesc.sc:4963 ESTABLISHED
7 tcp        0      0 megryan.sel.eesc.sc:ssh winona.sel.eesc.sc:4952 ESTABLISHED
8 Active UNIX domain sockets (w/o servers)
9 Proto RefCnt Flags       Type       State      I-Node Path
10 unix  1      [ ]        STREAM     CONNECTED  8607  @00000000c
11 unix  1      [ ]        STREAM     CONNECTED  8963  @00000000e

```

(b)

Saída de netstat

```

1 megryan:~$ netstat | head -n 10
2 Active Internet connections (w/o servers)
3 Proto Recv-Q Send-Q Local Address           Foreign Address         State
4 tcp        0      0 megryan.sel.eesc.sc:ssh winona.sel.eesc.sc:4963 ESTABLISHED
5 tcp        0      20 megryan.sel.eesc.sc:ssh winona.sel.eesc.sc:4952 ESTABLISHED
6 Active UNIX domain sockets (w/o servers)
7 Proto RefCnt Flags       Type       State      I-Node Path
8 unix  1      [ ]        STREAM     CONNECTED  8607  @00000000c
9 unix  1      [ ]        STREAM     CONNECTED  8963  @00000000e
10 unix  1      [ ]        STREAM     CONNECTED  60    @000000001
11 unix  1      [ ]        STREAM     CONNECTED  8964  /dev/log

```

(c)

Figura 7.7: Conexões ativas (netstat)

Durante o trabalho de desenvolvimento do software — especialmente na fase de testes — isso pode se revelar bastante incômodo posto que, a cada erro (ou solicitação do usuário) que causa o término do servidor, é preciso esperar pelo período de `TIME_WAIT` — que no sistema GNU/Linux é de 60 segundos — antes de reiniciar o programa. O artifício *ad hoc* adotado para esquivar-se da inconveniência foi um “*hacking*”⁷ no código-fonte do cerne do sistema operacional, alterando a constante para 2 segundos. O *patch*⁸ contendo as modificações para o *linux 2.2.18* é mostrado na Figura 7.8 — sua utilização fica como sugestão

⁷Jargão: intervenção *ad hoc* no código fonte de um programa.

⁸Arquivo de diferenças que pode ser aplicado ao programa original para impor as modificações, através do utilitário `patch`.

 Excerto de *idn/files/tcp_time_wait.diff*

```

--- kernel-source-2.2.18pre21/include/net/tcp.h Mon Aug  6 23:30:38 2001
+++ kernel-source-2.2.18pre21_hacked/include/net/tcp.h Mon Aug  6 23:36:36 2001
@@ -246,7 +246,7 @@
     */

#define TCP_TIMEOUT_LEN      (15*60*HZ) /* should be about 15 mins */
-#define TCP_TIMEWAIT_LEN (60*HZ) /* how long to wait to successfully
+#define TCP_TIMEWAIT_LEN (2*HZ) /* how long to wait to successfully
     * close the socket, about 60 seconds */
#define TCP_FIN_TIMEOUT (3*60*HZ) /* BSD style FIN_WAIT2 deadlock breaker */

@@ -274,7 +274,7 @@

/* TIME_WAIT reaping mechanism. */
#define TCP_TWICKILL_SLOTS      8      /* Please keep this a power of 2. */
-#define TCP_TWICKILL_PERIOD    ((HZ*60)/TCP_TWICKILL_SLOTS)
+#define TCP_TWICKILL_PERIOD    ((HZ*2)/TCP_TWICKILL_SLOTS)

/*
 * TCP option

```

Figura 7.8: Redução do intervalo de TIME_WAIT (linux 2.2.18)

provisória, embora o mais adequado seja modificar o mecanismo de conexão (*network.c*) para eliminar o problema, conforme melhor detalhado adiante.

7.2 Operação Elementar do Gerenciador de Interfaces

A fim de demonstrar os mecanismos básicos de comunicação através do GI, um cenário bastante simples pode ser arranjado utilizando-se novamente um aplicativo *telnet*, desta vez para simular a conexão de uma nova ID. Como exemplo, seja o caso em que o recém-chegado participante da RDI envia uma requisição ao dispositivo de sistema solicitando que este que lhe informe o nome provisório atribuído pelo GI ao seu conector.

- 5 A Figura 7.9 mostra o terminal *telnet* na estação *winona* onde, via teclado, uma mensagem IDNP/0.1 é composta (comando “name”) e em seguida enviada⁹ ao processo *idn* no outro extremo do canal TCP, na estação *megryan* (ambas as estações rodam sistema operacional Debian GNU/Linux na rede local das instalações do laboratório de pesquisa).
- 6-7 Logo após o correspondente processamento, retorna-lhe a resposta do sistema, a qual deve ser adequadamente retrucada pela ID a fim de encerrar o ciclo de operação. Conforme se nota, o nome automaticamente atribuído ao cliente é “2”, dado que se trata do segundo

⁹Convenientemente, a codificação utilizada na comunicação da ID é propositadamente compatível com a proporcionada pelo sistema *telnet* (linhas de texto ASCII terminadas por CR-LF).

Saída de telnet

```

1 monaco@winona:~$ telnet megryan 5555
2 Trying 143.107.235.7...
3 Connected to megryan.sel.eesc.sc.usp.br.
4 Escape character is '^]'.
5 IDNP/0.1 : SYSTEM 200 name
6 IDNP/0.1 : SYSTEM 200 2
7 IDNP/0.1 : SYSTEM 200 thanks

```

Figura 7.9: Simulação de uma ID via aplicação *telnet*

Saída de idn

```

1 megryan:~/idn$
2 >
3 > IDN 0.1 (LAMI, University of Sao Paulo)
4 > running in megryan.sel.eesc.sc.usp.br (143.107.235.7)
5 > which is a i686 Linux 2.2.19
6 >
7 > starting IDN shell
8 > starting system device
9 > waiting for the next network event
10 > new connection request
11 > client 1 connected from 127.0.0.1 (localhost)
12 > so we're done with this event
13 >
14 > waiting for the next network event
15 > new shell connection
16 > new connection request
17 > client 2 connected from 143.107.235.48 (winona.sel.eesc.sc.usp.br)
18 > so we're done with this event
19 >
20 > waiting for the next network event
21 > message from device 2: IDNP/0.1 : SYSTEM 200 name
22 > message arriving in an empty connector (interruption)
23 > self-evocation at 2 1, so chain outset
24 > 2 1 requests explicit target
25 > and evokes the new instance SYSTEM 2
26 > which resorts to implicit target
27 > referring to itself, so chain endpoint at SYSTEM 2
28 > chain establishment completed

```

Figura 7.10: Exemplo A (*idn*, parte 1)

conector estabelecido no sistema.

É interessante observar a reação correspondente no GI. Todo processo em seu interior pode ser acompanhado em tempo de execução ativando-se o modo de passo travado, acessível mediante o comando *stepmode* no *IDN shell*. Isso faz com que o programa pare em pontos predeterminados dentro do código (função *idn_step*), para que se possa inspecionar as estruturas de dados internas. Para liberá-lo e chegar ao próximo ponto de parada, envia-se o caracter ENTER (ASCII 13).

1-12 A Figura 7.10 transcreve registros gerados pelo GI referentes à parte inicial do processo — os sinais de ponto que aparecem à esquerda em algumas linhas mostram onde as paradas ocorrem. O trecho que interessa neste exemplo começa logo após a inicialização do GI e da conexão do *IDN shell*, quando a linha principal está bloqueada no monitor da rede.

14-20 O primeiro evento é a detecção da requisição de conexão do cliente, para o qual, um

Saída IDN shell

```
idn> 1

1. empty srv 0, int 0
[SYSTEM ]

2. empty srv 0, int 0
[2      ]

idn>
```

Figura 7.11: Exemplo A (*IDN shell*, parte 1, condição inicial)

dispositivo e um conector são, então, alocados. Assim que isso é realizado, a linha principal encontra-se novamente à espera de atividade na rede. Há então uma chance de observar o manipulador de interfaces através do *IDN shell*, cuja saída é mostrada na Figura 7.11.

21-22 Instantes mais tarde, o cliente envia sua requisição. Ao seu recebimento, a linha do GI desbloqueia-se e a função `idn_receive_message` é chamada para transferir a mensagem para o *buffer* do dispositivo correspondente. Uma vez que seu conector está vazio, o algoritmo de encaminhamento identifica corretamente que se trata de uma interrupção e, portanto, procede à evocação de uma cadeia completa de instâncias, para que estas efetuem o transporte da tarefa dentro do manipulador.

23-27 A primeira instância surge como uma auto-evocação no conector “2” e responde pelo início da cadeia. Nesse caso, encaminhamento explícito tem lugar e, portanto, a segunda instância é evocada no conector do sistema, em nome da primeira. A partir daí, somente o encaminhamento implícito pode ser escolhido para evocar a sucessora da segunda instância. Ocorre, entretanto, conforme já visto através do comando `ld`, que o alvo implícito do conector é ele próprio. Como procedimento recursivo, este (`idn_evoke`) termina quando uma instância evoca sua consecutiva no mesmo conector; tem-se aí o final da seqüência.

Nesse momento um ponto de parada permite que o manipulador seja novamente inspecionado. As duas instâncias presentes podem ser examinadas através do comando `li` do *IDN shell*, ilustrado na Figura 7.12 (a). Cada uma das linhas da listagem produzida na sua saída tem o formato explicado na Figura 7.12 (b).

Os dados gerados pelo programa podem ser melhor apreciados dentro do panorama completo do manipulador, transcrito na Figura 7.13 (a). Como o formato conciso da informação proporcionada pelo *IDN shell* pode apresentar-se algo criptográfico para efeito

Figura 7.12: Exemplo A (*IDN shell*, parte 1, cadeia estabelecida) Figura 7.13: Exemplo A (*IDN shell*, parte 1, manipulador completo)

de interpretação visual, a Figura 7.13 (b) ao lado fornece uma representação gráfica alternativa.

Conforme mostra o comando 1, ambos os conectores agora possuem cada qual uma instância¹⁰. No conector “2”, o dispositivo número 2 (alocado para a ID recém conectada) indica que é uma entrada disponível em seu *buffer* (*input*), e que a instância número 1 fora contratada para servi-lo (*srv 1*) no evento de uma interrupção (*int 1*); a mensagem em questão aparece logo em seguida. Corretamente, a instância número um, no mesmo conector, sinaliza que está a espera de uma entrada (*PI*), a qual, quando recebida, será referida à sua consecutiva (*Refer*) na cadeia em condições normais (*Regular*). Entre sinais de parênteses um par de números indica as instâncias a quem a tarefa será reportada e a qual será referida (1, 2), respectivamente na fase descendente e ascendente do ciclo de operação — nesse caso, a instância reporta-se a si mesma e refere-se à instância número 2.

¹⁰Devido a um erro no código fonte, o comando 1 lista os parâmetros da instância em uma ordem diferente da que produz *li*. Em particular, o primeiro mostra (*Rp, Rf*) (o correto), enquanto o segundo mostra (*Rf, Rp*).

A Figura ainda informa que a instância número 1 não reconhece nenhum dispositivo como mestre (ms 0) e que o código de *status* de resposta é nulo (100).

No diagrama, mantendo a convenção adotada em capítulos anteriores, os círculos na parte superior do diagrama respondem pelos dispositivos; os retângulos, pelos conectores; e os círculos na parte inferior indicam as instâncias presentes. Os arcos orientados correspondem aos ponteiros que integram a cadeia: os terminados em setas preenchidas são os ponteiros de referência; os terminados em setas não preenchidas, os ponteiros de reporte. A indicação ao lado do dispositivo 2 (anel espesso) informa que seu buffer contém uma entrada pendente, e o segmento pontilhado aponta para a serva-interruptida destinada a recebê-la — marcações correspondentes dão o estado da sua instância.

A outra instância, por sua vez, está vinculada ao conector do sistema, onde não tem mensagens a serem trocadas com seu dispositivo (SB) e está pronta para receber uma ordem (Ready) de sua parceira. Também em condição normal (Regular), a instância reporta-se àquela de número 1 no segundo semiciclo, mas deve fazer uma auto-referência no primeiro. No restante, seus parâmetros são os mesmos válidos para a primeira instância.

O instantâneo dos dados armazenados no manipulador de interfaces fora obtido logo após o estabelecimento da cadeia. A seguir, o GI deve chamar o despachador para cuidar das entregas pendentes, observando a ordem de processamento: *i*) receber entrada, *ii*) referir requisição, *iii*) dispensar cadeia obsoleta, *iv*) liberar saída, *v*) reportar resposta (seção 6.3.8, pág.150).

A seqüência é retomada na Figura 7.14, onde transcrevem-se os registros gerados pelo GI (à esquerda). A fim de ilustrar o processo que se segue, à cada ponto de parada um extrato do manipulador é obtido (à direita): na parte inferior, consta a saída do comando `li` no referido instante; acima deste, uma representação gráfica auxiliar que facilita a visualização do que ocorre internamente¹¹.

29-30

A listagem inicia mostrando o momento seguinte à evocação da cadeia, quando o despachador detecta que a instância interruptora possui uma entrada pendente. No gráfico,

¹¹A informação à direita corresponde ao estado em que o manipulador se encontra no momento da parada. Como regra, as linhas à esquerda iniciando com “.” indicam a alteração que será efetuada *a seguir*.

Figura 7.14: Exemplo A (*idn*, parte 2)

o triângulo preenchido e de vértice para baixo denota que a mensagem esperada já se encontra disponível no *buffer* do dispositivo. Logo, a função `idn.handle_pending_input`, examinada no capítulo anterior, é chamada para efetuar o recebimento. Isso feito, a cadeia de instâncias é deixada no estado mostrado logo abaixo.

31-32 O círculo com borda mais espessa indica que a mensagem fora transferida do dispositivo para a instância, conforme pode ser também verificado na saída do comando `li`, onde a instância 1 (primeira linha) mostra o membro `command` da estrutura contendo agora o comando enviado pelo usuário (“name”). Havendo recebido uma incumbência, a instância 2 (cliente *telnet*) tem agora um mestre; na mesma linha, nota-se também a mudança de `PI` (*pending input*) para `SB` (*stand by*): estando agora de posse de sua ordem, a interruptora está apta a referi-la à sua sucessora no ciclo de operação, fato esse que é detectado pelo despachador, levando a cadeia para a condição seguinte.

33-35 A mudança do anel destacado da instância 1 para a instância 2 corresponde ao prosseguimento da tarefa no ciclo ascendente. O arco orientado com seta preenchida informa que uma referência fora executada, e o triângulo não preenchido e de vértice para cima denota que a segunda instância possui agora uma saída pendente. Como acusa a saída do

Figura 7.15: Exemplo A (*idn*, parte 3)

IDN shell, a interruptora passa para o estado de espera; o comando está agora no *buffer* da outra instância. Continuando o processamento na seqüência predeterminada, o despachador não encontra cadeias obsoletas, mas detecta a pendência (PO) no passo seguinte, e toma a providência que lhe cabe.

36-45 O panorama passa a ser o da Figura 7.15. No diagrama, o destaque no anel da instância 2 é substituído pelo preenchimento no triângulo acima, denotando que a mensagem fora transferida para o dispositivo. O *IDN shell* mostra que o conteúdo do campo de comando fora suprimido e que o elemento está agora a espera de uma entrada (PI). Examinando a cadeia, o despachador (à esquerda) não encontra reportes pendentes, chegando assim ao final da seqüência de verificações de entrega. Contudo, como pelo menos uma fora realizada, todo o procedimento é re-executado desde o início mais uma vez. Nenhuma entrega remanescente sendo então encontrada, o despachador finaliza seu trabalho. O GI, então, chama *idn_send_message* para transmitir as saídas geradas às IDs que devem recebê-las.

46-54 A situação após o envio dos dados é a ilustrada pelo diagrama seguinte. O comando *li* não mostra qualquer alteração, posto que a última operação fora realizada no dispositivo e não na instância. Porém, o triângulo de vértice para baixo, não preenchido, denota que

a segunda instância está à espera de uma entrada.

O que ocorre a seguir merece atenção especial. Havendo transmitido uma mensagem especificamente para a ID do sistema, o processo entra em uma região crítica. Para que a linha do sistema (receptora) não concorra com a linha principal (emissora) no acesso à estrutura de dados do manipulador, ambas excluem-se mutuamente dessa atividade.

A providência pode ser vista nos registros do GI. Conforme descrito anteriormente, a linha principal bloqueia-se em um semáforo até que o sistema tenha terminado seu trabalho. Na figura, a mudança do identificador “>” para “|” significa que quem está gerando as saídas é a linha do sistema. Note-se que a mensagem aí recebida não é a originalmente enviada pelo cliente *telnet*, mas uma nova composição a partir do endereço do remetente (conector), mais os campos de ação e comando armazenados na instância. Após processar e executar o comando recebido, a linha do sistema envia uma resposta através do *socket* TCP e, imediatamente, libera o semáforo da linha principal. Havendo concluído seu trabalho, o sistema bloqueia-se novamente a espera de novos comandos, cedendo a vez à linha principal.

54-55 Esta outra, por sua vez, retomando a execução (Figura 7.16), volta ao monitor de eventos de rede onde, antes mesmo de se bloquear, encontra a mensagem do sistema. Detectada, a linha é suspensa no próximo ponto de parada, permitindo que o monitor seja inspecionado mais uma vez. O traço sobre a base do triângulo invertido na segunda instância denota que a mensagem esperada já fora recebida pelo servidor TCP do GI, mas ainda não fora armazenada no buffer do dispositivo — por essa razão, não há alteração dos dados fornecidos pelo comando do *IDN shell*.

57-58 Assim que a medida é tomada, entretanto, o *buffer* do dispositivo de sistema passa a estar disponível para leitura. A situação para essa instância é análoga àquela pela qual passara a instância interruptora no início do ciclo de operação: assim que o despachador é chamado, a entrada pendente é detectada e transferida para a instância (note-se o campo de comando na listagem do *IDN shell*).

59-60 A instância 2 agora está pronta para passar a ordem adiante. O fato de o ponteiro referido na correspondente estrutura apontar para a própria instância não implica em qualquer diferença no modo como a entrega é tratada: tudo se passa como se a ordem fosse

Figura 7.16: Exemplo A (*idn*, parte 4) Figura 7.17: Exemplo A (*idn*, parte 5)

referida à entidade sucessora. Esse procedimento hipotético é ilustrado na Figura 7.17.

61-63 O que de fato acontece é uma auto-referência no final da cadeia. A instância a executá-la passa a possuir uma mensagem a ser transferida para o dispositivo, conforme mostram o diagrama e a saída do comando `li`. Neste último, vale observar que agora a informação acerca da instância 2 alterou-se de `Refer` para `Report` (hipoteticamente passando pelo estado intermediário `wait`). Assim, depois de constatar a ausência de cadeias a serem desfeitas, o despachador detecta a saída pendente e dá prosseguimento ao seu atendimento, levando à configuração seguinte.

64-73 Havendo transferido novamente a mensagem da instância 2 para o dispositivo no mesmo conector, não há mais qualquer entrega a ser diligenciada. Retornando, assim, do despachador, o GI prepara-se para transmitir as saídas armazenadas.

74-79 Ao fazê-lo, como sempre deve acontecer, a linha principal bloqueia-se e o controle passa para a linha do sistema. Uma vez que o dispositivo de sistema tenha decidido não executar comandos recebidos de si mesmo, nada acontece exceto o retorno da mensagem para o GI.

81-85 Liberada (Figura 7.18), a linha principal vai receber o retorno do dispositivo junto ao monitor de eventos. As situações imediatamente antes e depois de transferi-la do dispositivo, são aquelas registradas pelo *IDN shell* e traduzida nos diagramas correspondentes.

86-90 A instância 2 está de posse de uma ordem pronta para ser entregue. Dado seu estado (`Report` na saída do *IDN shell*) — e não havendo referências, dispensas, saídas ou entradas pendentes — o despachador a seleciona para reportar-se. Tem início assim a segunda fase do ciclo de operação e a tarefa toma seu caminho de volta em direção ao começo da cadeia.

91-96 Isso feito, o arco orientado terminado em seta não preenchida aparece indicando que naquele ponto a ordem já fora entregue de volta ao elemento precedente na sequência ativa — acompanhando a evolução, na saída do comando `li` pode se ver que seu estado muda de `Report` para `Done`, sinalizando que a instância já cumprira sua missão.

97-107 O despachador chega novamente ao final da lista de verificações. Todavia, como pelo menos uma entrega fora efetivada, o laço se repete e é então que a saída pendente na instância interruptora é detectada. Quando, a seguir, esta é transferida para o primeiro dispositivo, o final do ciclo de operação é iminente. Este é o momento em que a ID

Figura 7.18: Exemplo A (*idn*, parte 6)

interruptora recebe sua esperada resposta. No que concerne à aplicação em processo, a operação (interação cliente-servidor) está concluída. Todavia, do ponto de vista da RDI, para que a transação seja realmente dada por terminada, é ainda preciso cumprir os ritos finais previstos no modelo.

108-111 Por essa razão, após a transmissão da mensagem (Figura 7.19), a instância volta a indicar uma entrada pendente. O arranjo permanece nesse estado até que a ID (nesse caso o cliente *telnet*) envie um retorno final.

Figura 7.19: Exemplo A (*idn*, parte 7)

112 Nessa versão do protótipo IDN, o comando embutido nessa última mensagem não é processado e sua chegada serve apenas como uma confirmação da interruptora de que a tarefa finalmente chegara de volta ao ponto de partida.

113-120 O que acontece a seguir é, como das outras vezes, o armazenamento da mensagem no *buffer* do dispositivo e posteriormente, quando o despachador entrar em ação, a sua transferência para a instância. A esta altura, o laço de verificação de entregas pendentes constata que a interruptora está apta a reportar-se à sua antecessora na cadeia.

121-125 De novo, isso é realizado segundo um procedimento hipotético (Figura 7.20), como se a instância houvesse se reportado à uma terceira ID. Observe-se, entretanto, que ao entregar a ordem a si mesma, a instância que já estava no estado final Done, é compelida ao estado de sobre-utilização: *over*. A informação do *IDN shell* e o sinal de ponto no diagrama denotam essa condição. Seguindo a busca por trabalhos pendentes, o despachador encontra nessa

Figura 7.20: Exemplo A (*idn*, parte 8)

instância o início de uma cadeia obsoleta.

126-142 A função `idn_dismiss` é então chamada para recursivamente eliminar toda a sequência. Finalmente, quando uma cadeia é desestabelecida o despachador identifica o final de um ciclo de operação.

Na implementação apresentada, dois detalhes merecem comentário adicional. O primeiro diz respeito à auto-referência que tem lugar no final da cadeia, quando a ID terminal recebe uma mensagem de si mesma. Nos ensaios realizados dentro da rede local, a questão não tem se mostrado tão relevante; no caso da operação através de redes de longa distância, por outro lado — como na aplicação-exemplo — onde a comunicação venha constituir o provável fator limitante do desempenho global do sistema, há razão em se considerar a eliminação de uma transmissão desnecessária. Isso pode ser feito alterando-se o algoritmo de encaminhamento para suprimir alguns passos intermediários no processo descrito. Conquanto a medida possa contribuir para a redução do tempo de resposta, introduzir uma

violação nas regras de funcionamento tem seu preço no que concerne à consistência do mecanismo — ponderações acerca dessa matéria ficam como sugestão para futuros trabalhos.

O outro detalhe tem relação com o extremo oposto, no início da cadeia. Na versão introduzida, o retorno final da ID interruptora não é encaminhado adiante, e sequer é processado pelo GI, a não ser para confirmar o término do ciclo de operação. De fato, o completamento do ciclo pode ser detectado antes disso, bastando verificar quando um dispositivo recebe uma mensagem da instância que interrompera previamente (é possível distingui-lo por meio da variável `interrupted` na estrutura do dispositivo). Todavia, neste caso, o retorno final proporciona um mecanismo que pode ser útil para implementação de transações — com respeito à propriedade de recuperabilidade (*recoverability* ou *durability*) — através da técnica 2PC (*Two-phase commitment*).

Um esquema desse tipo pode ser criado, estendendo-se o conjunto de valores definidos no protocolo IDNP/0.1 para o campo de ação, onde o código 200, tem sido utilizado para requisitar a execução imediata do comando. Um outro número poderia ser utilizado para solicitar a simulação de todo o processo, causando o estabelecimento da cadeia e o transporte da mensagem através de todas as IDs envolvidas. Cada uma delas, porém, em lugar de efetuar a operação, apenas registrariam sua condição de executá-la (com base na disponibilidade, prioridade, permissão de acesso etc.) — a reserva de recursos pode ser implementada também nessa etapa.

Quando a tarefa chegar de volta à ID interruptora, esta tem condições de verificar se todas as parceiras concordam em efetivar a operação. Nesse caso, o retorno gerado pela interruptora não determina o desestabelecimento da cadeia, mas causaria um segundo ciclo de operação pela mesma sequência de instâncias para, agora sim, ser processada como execução.

7.3 Utilização da API

Por meio do aplicativo *telnet* é possível simular a comunicação entre uma ID e o GI enviando e recebendo mensagens diretamente na linha do terminal. Para realizar os demais ensaios, contudo, onde se tem várias IDs interagindo ao mesmo tempo na RDI, é considera-

 Excerto de *idn/dis/example_1.c*

```

1 /* example_1.c - example #1 (minimal API utilization)
2 *
3 *      $RCSfile: example_1.c,v $ $Revision: 1.2 $
4 *      IDN Project, SEL/EESC-USP.
5 *      Refer to idn.txt for further information.
6 */
7
8 #include <stdlib.h>
9 #include "di.h"          /* 1. Include the header file. */
10
11 int
12 main (int argc, char **argv)
13 {
14     idn_di_t *di;         /* 2. Declare the DI. */
15
16     di = idn_new_di ();   /* 3. Create the DI. */
17     if (di == NULL)
18         return EXIT_FAILURE;
19
20     if (idn_di_start (di)) /* 4. Start the DI. */
21         return EXIT_FAILURE;
22
23     return EXIT_SUCCESS;
24 }

```

Figura 7.21: Implementação mínima de uma ID utilizando a API

velmente mais prático utilizar programas especializados. Para isso, construíram-se algumas aplicações bastante simples na forma de IDs. Sua apresentação, além de interessante para descrever os experimentos a serem apresentados, constitui um modo prático de ilustrar a utilização da API desenvolvida como parte do sistema-piloto¹².

O arquivo *example_1.c*, listado na Figura 7.21, fornece um exemplo de como o programador pode utilizar esse recurso para criar uma ID de funcionalidade mínima. Apenas quatro passos assaz simples são necessários.

- 9 • Passo 1: inclusão do arquivo de cabeçalho *di.h*.
- 14 • Passo 2: declaração de um ponteiro para uma estrutura do tipo *idn_di_t*.
- 26 • Passo 3: alocação de uma nova ID por meio da função *idn_new_di*.
- 20 • Passo 4: ativação da ID por meio da função *idn_di_start*.

Bastam essas linhas de código para produzir uma ID bastante rudimentar, mas capaz de conectar-se ao GI e interagir adequadamente com a RDI. Depois de gerado o módulo

¹²Todos os arquivos comentados nesta seção encontram-se no subdiretório *dis*, dentro do diretório principal de instalação do protótipo.

Saída de `example_1`

```

4 megryan:~/idn/dis$ ./example_1
5 << IDNP/0.1 : SYSTEM 200 name
6 > IDNP/0.1 : SYSTEM 200 2
7 < IDNP/0.1 : servant 200 Thank you, my lord

```

Figura 7.22: Inicialização de `example_1`

executável¹³, o programa pode ser invocado diretamente no terminal GNU/Linux. O resultado é mostrado¹⁴ na Figura 7.22.

5 O funcionamento do módulo é bastante simples. Assim que inicia, a linha do operador acessa o único comando no vetor de configurações e o envia na forma de uma mensagem para a linha principal, que está, então, bloqueada a espera de eventos. Esta, a seu turno cuida de encaminhá-la ao GI através do *socket* TCP. O símbolo “<<” indica o envio da interrupção.

6 Algum tempo depois, após a requisição haver sido processada pela receptora (o sistema, nesse caso), sua correspondente resposta é entregue à ID — o símbolo “>” denota uma mensagem recebida.

7 Finalmente, para cumprir o protocolo, a ID envia de volta o retorno final (uma mensagem de “agradecimento” padrão), confirmando o final do ciclo de operação, para que a cadeia formada seja descartada. Retornos são marcados com “<”.

Toda atividade gerada no interior do GI é mostrada através dos registros produzidos pelo módulo *idn*, listados na Figura 7.23. Daí em diante, a linha do operador é suspensa indefinidamente. Utilizando um aplicativo *telnet* para conectar-se novamente ao módulo *idn*, é possível enviar comandos para a ID. Sabendo que o nome provisório a ela atribuído pelo GI é “2”, a mensagem IDNP/0.1 : 2 200 hello pode ser transmitida via terminal, resultando no processo descrito nas saídas de cada um dos programas, transcritas na Figura 7.24.

Evidentemente, uma ID que apenas pergunta seu nome e passa a não fazer mais nada,

¹³O programa deve ser compilado e ligado com os arquivos-objeto que implementam as funções da API. Com o auxílio do arquivo *Makefile* fornecido, o comando `make dis` deve gerar todas as IDs citadas como exemplo.

¹⁴Normalmente, o módulo *di.o* produz um relatório de tudo que faz na saída padrão, em tempo de execução; para suprimir a geração de registros, o programa pode ser compilado com a macro `VERBOSE 0` definida.

Saída de idn

```

megryan:~idn$ ./idn
> IDN 0.1 (LAMI, University of Sao Paulo)
> running in megryan.sel.eesc.sc.usp.br (143.107.235.7)
> which is a i686 Linux 2.2.19
>
> starting IDN shell
> starting system device
> waiting for the next network event
> new connection request
> client 1 connected from 127.0.0.1 (localhost)
> so we're done with this event
>
> waiting for the next network event
> new shell connection
> new connection request
> client 2 connected from 127.0.0.1 (localhost)
> so we're done with this event
>
> waiting for the next network event
> message from device 2: IDNP/0.1 : SYSTEM 200 name
> message arriving in an empty connector (interruption)
> self-evocation at 2 1, so chain outset
> 2 1 requests explicitly target
> and evokes the new instance SYSTEM 2
> which resorts to implicit target
> referring to itself, so chain endpoint at SYSTEM 2
> chain establishment completed
> checking for pending inputs
> handling a pending input at 2 1
> checking for pending references
> handling pending reference of 2 1 to SYSTEM 2
> checking for dismissable chains
> checking for pending outputs
> handling pending output at SYSTEM 2
> checking for pending reports
> some deliveries this time: doing all over again.
> checking for pending inputs
> checking for pending references
> checking for dismissable chains
> checking for pending outputs
> checking for pending reports
> no deliveries this time: we're all done.
> looking for messages to be released
> transmitting message of SYSTEM 2 at device 1
> hold on: main thread suspended
> system got: IDNP/0.1 : 2 200 name
> executing: "name"
> system sent: IDNP/0.1 : SYSTEM 200 2
> go on: main thread released
> all releasable messages already transmitted
> so we're done with this event
>
> waiting for the next network event
> message from device 1: IDNP/0.1 : SYSTEM 200 2
> message arriving in a non-empty connector (return)
> checking for pending inputs
> handling a pending input at SYSTEM 2
> checking for pending references
> handling pending reference of SYSTEM 2 to SYSTEM 2
> checking for dismissable chains
> checking for pending outputs
> handling pending output at SYSTEM 2
> checking for pending reports
> some deliveries this time: doing all over again.
> checking for pending inputs
> checking for pending references
> checking for dismissable chains
> checking for pending outputs
> checking for pending reports
> no deliveries this time: we're all done.
> looking for messages to be released
> transmitting message of 2 1 at device 2
> all releasable messages already transmitted
> so we're done with this event
>
> waiting for the next network event
> message from device 2: IDNP/0.1 : servant 200 Thank (...)
> message arriving in a non-empty connector (return)
> checking for pending inputs
> handling a pending input at 2 1
> checking for pending references
> checking for dismissable chains
> checking for pending outputs
> checking for pending reports
> handling pending report of 2 1 to 2 1
> some deliveries this time: doing all over again.
> checking for pending inputs
> checking for pending references
> checking for dismissable chains
> completed chain starting at 2 1
> dismissing SYSTEM 2
> dismissing 2 1
> checking for pending outputs
> checking for pending reports
> some deliveries this time: doing all over again.
> ----- END OF CICLE -----
> checking for pending inputs
> checking for pending references
> checking for dismissable chains
> checking for pending outputs
> checking for pending reports
> no deliveries this time: we're all done.
> looking for messages to be released
> all releasable messages already transmitted
> so we're done with this event
>
> waiting for the next network event

```

Figura 7.23: Registros de *idn* para a invocação de *example_1*

Saída de telnet	Saída de example.1
<pre>monaco@teri:~\$ telnet localhost 5555 Trying 127.0.0.1... Connected to localhost. Escape character is '^]'. IDNP/0.1 : 2 200 hello IDNP/0.1 : 2 200 At your command, master</pre>	<pre>> IDNP/0.1 : 2 200 At your command, master < IDNP/0.1 : servant 200 At your command, master</pre>
(a)	(b)
Saída de idn	
<pre>> waiting for the next network event > message from device 3: IDNP/0.1 : 2 200 hello > message arriving in an empty connector (interruption) > self-evocation at 3 3, so chain outset > 3 3 requests explicitly target > and evokes the new instance 2 4 > which resorts to implicit target > referring to itself, so chain endpoint at 2 4 > chain establishment completed > checking for pending inputs > handling a pending input at 3 3 > checking for pending references > handling pending reference of 3 3 to 2 4 > checking for dismissable chains > checking for pending outputs > handling pending output at 2 4 > checking for pending reports > some deliveries this time: doing all over again. > checking for pending inputs > checking for pending references > checking for dismissable chains > checking for pending outputs > checking for pending reports > no deliveries this time: we're all done. > looking for messages to be released > transmitting message of 2 4 at device 2 > all releasable messages already transmitted > so we're done with this event > > waiting for the next network event > message from device 2: IDNP/0.1 : servant 200 At (...) > message arriving in a non-empty connector (return) > checking for pending inputs > handling a pending input at 2 4 > checking for pending references > checking for dismissable chains > checking for pending outputs > checking for pending reports > handling pending report of 2 4 to 3 3 > some deliveries this time: doing all over again. > checking for pending inputs > checking for pending references > checking for dismissable chains > checking for pending outputs > handling pending output at 3 3 > interruption from device 3 at 3 3 completed > checking for pending reports > some deliveries this time: doing all over again. > checking for pending inputs > checking for pending references > checking for dismissable chains > checking for pending outputs > checking for pending reports > no deliveries this time: we're all done. > looking for messages to be released > transmitting message of 3 3 at device 3 > all releasable messages already transmitted > so we're done with this event > > waiting for the next network event</pre>	<pre>> checking for pending references > checking for dismissable chains > checking for pending outputs > checking for pending reports > no deliveries this time: we're all done. > looking for messages to be released > transmitting message of 2 4 at device 2 > all releasable messages already transmitted > so we're done with this event > > waiting for the next network event > message from device 2: IDNP/0.1 : servant 200 At (...) > message arriving in a non-empty connector (return) > checking for pending inputs > handling a pending input at 2 4 > checking for pending references > checking for dismissable chains > checking for pending outputs > checking for pending reports > handling pending report of 2 4 to 3 3 > some deliveries this time: doing all over again. > checking for pending inputs > checking for pending references > checking for dismissable chains > checking for pending outputs > checking for pending reports > no deliveries this time: we're all done. > looking for messages to be released > transmitting message of 3 3 at device 3 > all releasable messages already transmitted > so we're done with this event > > waiting for the next network event</pre>
(c)	

Figura 7.24: Um ciclo de operação (*example.1*)

exceto responder às outras com uma mensagem fixa, não é por certo muito interessante. Alguns recursos extras proporcionados pela API podem ser utilizados para construir aplicações mais úteis. Um deles é a alteração do vetor de configurações, como mostrado na Figura 7.25 (a).

11-16

O programa *example.2* é essencialmente o mesmo listado anteriormente, a não ser pela inclusão de duas partes extras. No preâmbulo, tem-se a alocação estática de um vetor de re-

Excerto de *idn/dis/example_2.c*

```
1 /* example_2.c - example #2 (using the config vector)
2 *
3 *      $RCSfile: example_2.c,v $ $Revision: 1.1 $
4 *      IDN Project, SEL/EESC-USP.
5 *      Refer to idn.txt for further information.
6 */
7
8 #include <stdlib.h>
9 #include "di.h"
10
11 idn_request_t setup[3] =      /* 1. Create the new configuration vector. */
12 {
13     {IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "version", NULL},      /* Ask version.*/
14     {IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "name example", NULL}, /* Change name.*/
15     {NULL, IDN_STATUS_VOID, NULL},
16 };
17
18 int
19 main (int argc, char **argv)
20 {
21     idn_di_t *di;
22
23     di = idn_new_di ();
24     if (di == NULL)
25         return EXIT_FAILURE;
26
27     di->setup_requests = setup; /* 2. Replace the configuration vector. */
28
29     if (idn_di_start (di))
30         return EXIT_FAILURE;
31
32     return EXIT_SUCCESS;
33 }
```

(a)

Saída de <i>example_2</i>	Saída de IDN shell
<pre>monaco@megryan:~/works/idn/dis\$./example_2 << IDNP/0.1 : SYSTEIM 200 version > IDNP/0.1 : SYSTEIM 200 IDN 0.1 < IDNP/0.1 : SYSTEIM 200 IDN 0.1 << IDNP/0.1 : SYSTEIM 200 name example > IDNP/0.1 : SYSTEIM 200 example < IDNP/0.1 : SYSTEIM 200 example</pre>	<pre>idn> 1 1. empty srv 0, int 0 [SYSTEM] 2. empty srv 0, int 0 [example]</pre>

(b)

(c)

Figura 7.25: Utilizando o vetor de configurações

quisições com três elementos. O primeiro determina que uma mensagem deve ser enviada ao dispositivo de sistema (`IDN_SYSTEM_NAME`), requisitando a execução (`IDN_STATUS_SUCCESS`) do comando que informa a versão deste (`version`) — a instrução não especifica o que deve ser feito com a resposta (`NULL`), logo, esta é ignorada. O segundo elemento solicita a alteração do nome da ID para “example”. Por último, o terceiro elemento, iniciando com `NULL`, denota o final do vetor.

Depois, no corpo do programa, atribui-se o vetor ao respectivo membro da estrutura que representa a ID. Após compilada, a ID pode ser invocada no terminal produzindo

Excerto de *idn/dis/example_3.c*

```

15 idn_request_t setup[3] =
16 {
17     {IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "name example", NULL},
18     {NULL, IDN_STATUS_VOID, NULL},
19 };
20
21 void
22 random_number (idn_di_t *di)      /* 1. Define a function to process the command. */
23 {
24     int fd,n;
25     char *str;
26     fd = open ("/dev/random", O_RDONLY);
27     read (fd, &n, 1);
28     asprintf (&str,"%d", n);
29     idn_di_reply (di, IDN_STATUS_SUCCESS, str);
30     free (str);
31 }
32
33 void
34 attend (idn_di_t *di) /* 2. Define a new 'attend' function. */
35 {
36     idn_di_skip_not (di, action, "200");
37     idn_di_skip (di, address, "example");
38     idn_di_response (di, command, "random", random_number);
39     idn_di_reply (di, IDN_STATUS_PERMANENT_CLIENT_ERROR, "What do you mean ?");
40 }
41
42 int
43 main (int argc, char **argv)
44 {
45     idn_di_t *di;
46
47     di = idn_new_di ();
48     if (di == NULL)
49         return EXIT_FAILURE;
50
51     di->setup_requests = setup;
52
53     di->attend = attend; /* 3. Overwrite the DI's 'attend' function. */
54
55     if (idn_di_start (di))
56         return EXIT_FAILURE;
57
58     return EXIT_SUCCESS;
59 }

```

Figura 7.26: Implementação da funcionalidade da ID

a saída mostrada na Figura 7.25 (b). No *IDN shell*, o comando `lc` executado a seguir, mostra que o segundo conector tem agora o novo nome, como na Figura 7.25 (c).

Além de reconfigurar os parâmetros operacionais da ID, é interessante redefinir-lhe a funcionalidade, i.e. implementar o conjunto de funções a serem oferecidas à RDI. O excerto de código da Figura 7.26 mostra como isso é feito no programa *example_3.c*, o qual estende o anterior com alguns recursos.

18-25 A primeira providência é definir uma função para processar o comando em questão. Antes de retornar, esta deve preencher o *buffer* da ID com uma mensagem de resposta contendo o resultado do processamento. No exemplo, `random_number` obtêm um número

Saída de telnet

```
monaco@megryan:~/works/idn/dis$ telnet localhost 5555
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^J'.
IDNP/0.1 : example 200 wrong
IDNP/0.1 : example 300 What do you mean ?
IDNP/0.1 : example 100 never mind
IDNP/0.1 : example 200 random
IDNP/0.1 : example 200 134534259
IDNP/0.1 : example 200 oh yes
IDNP/0.1 : example 200 random
IDNP/0.1 : example 200 134534325
IDNP/0.1 : example 200 thanks again
```

Figura 7.27: Saída de *example_3*

aleatório a partir do tanque entrópico do sistema operacional¹⁵ e produz uma mensagem indicativa de sucesso.

33-40 Depois, é necessário criar uma nova função de atendimento. Um procedimento bem simples é fornecido pelo programa: mensagens cujo campo de ação não indiquem sucesso são ignoradas, assim como mensagens vindas da própria ID; o comando “random” invoca a única função implementada, ao passo que qualquer outro comando retorna uma mensagem de erro.

53 Finalmente, sobrescreve-se o procedimento predefinido na ID pela nova função de atendimento. Isso produz o resultado ilustrado na Figura 7.27. As linhas em destaque correspondem a três mensagens enviadas à ID através de um terminal *telnet*: primeiro um comando errado, seguido da resposta da RDI e do retorno final; depois dois comandos corretos.

Em tempo, para completar a apresentação das principais funções da API, o exemplo da Figura 7.28 mostra como sobrescrever o procedimento de atendimento ao operador para gerar a aplicação propriamente dita.

23-28 O programa *example_4.c* inicia definindo a função que gera interrupções. No exemplo, *ask* espera um segundo e envia uma requisição para a ID de nome *who*, requisitando-lhe o comando *what*. Quando a ID entrar em atividade, logo após a execução do vetor de configurações, a função será executada indefinidamente dentro de um laço.

¹⁵Presente desde Linux 1.3.30, o dispositivo */dev/random* é um gerador de números aleatórios retirados de um tanque entrópico, o qual é alimentado pelo ruído ambiental gerado pelas atividades do hardware; se o tanque estiver vazio, a leitura bloqueia até que haja ruído suficiente.

Excerto de *idn/dis/example_4.c*

```

21 char *who, *what;
22
23 void
24 ask (idn_di_t *di)      /* 1. Define a function to generate requests. */
25 {
26     sleep (1);
27     idn_di_request (di, who, IDN_STATUS_SUCCESS, what);
28 }
29
30 void
31 thanks (idn_di_t *di)   /* 2. Define a function to process the replies. */
32 {
33     idn_di_reply (di, IDN_STATUS_SUCCESS, "I got it");
34 }
35
36 int
37 main (int argc, char **argv)
38 {
39     idn_di_t *di;
40
41     if (argc < 3)
42     {
43         fprintf (stderr, "Usage: ask who what\n");
44         exit (EXIT_FAILURE);
45     }
46
47     di = idn_new_di ();
48     if (di == NULL)
49         return EXIT_FAILURE;
50
51     di->setup_requests = setup;
52
53     who = argv[1];
54     what = argv[2];
55
56     di->issue = ask;      /* 3. Overwrite the DI's 'issue' function. */
57     di->process = thanks; /* 4. Overwrite the DI's 'process' function. */
58
59     if (idn_di_start (di))
60         return EXIT_FAILURE;
61
62     return EXIT_SUCCESS;
63 }

```

Figura 7.28: Implementando a aplicação

30-34 Logo em seguida, a linha do operador bloquear-se-á até a chegada da resposta, quando, então, a função `thanks` será chamada para processá-la. Esta simplesmente produz uma mensagem como resposta — em aplicações mais elaboradas, a reação correspondente seria executada nessa chamada.

56-57 O corpo do programa é praticamente idêntico ao dos exemplos anteriores. Nele, sobrescrevem-se os dois procedimentos originalmente fornecidos pela ID.

As últimas duas IDs prestam-se bem a realização de um ensaio simples, porém ilustrativo. Conectando-se ambas (`example_3` e `example_4`) ao módulo *idn* tem-se uma pequena, porém funcional, RDI. A Figura 7.29 mostra ambas em operação.

Como nota adicional, observa-se que nessa versão do sistema piloto todas as funções que geram mensagens contendo resultados de execução de instruções têm-no feito diretamente

Figura 7.29: Uma RDI com duas IDs

no campo de comando. Desse modo, o próprio valor retornado (o número aleatório, por exemplo) é interpretado pelo receptor como uma função. Alternativamente, poder-se-ia acrescentar ao CMI do protocolo uma função nula para representar “processamento não especificado” e fornecer o resultado em si na sua lista de argumentos. Conquanto tal estratégia tenha se mostrado mais coerente durante a fase de reavaliação do sistema, com a maior parte do trabalho realizado, sua adoção tardia não provou-se praticável — permanece, portanto, como sugestão.

7.4 Execução de Programas

Valendo-se da facilidade proporcionada pela API, três IDs foram construídas para demonstrar como programas são executados pelo módulo *idn*. O primeiro implementa uma interface de operador (Figura 7.30 e Figura 7.31) na forma de uma ID. Basicamente, tudo que faz é ler linhas fornecidas via entrada padrão (teclado), convertê-las em mensagens IDNP e enviá-las para a RDI. O formato da entrada é uma cadeia de caracter contendo o destinatário, seguido pelo comando.

O segundo programa (Figura 7.32 e Figura 7.33) proporciona a funcionalidade de um equipamento muito rudimentar que, dado um número, responde com seu dobro. A peculiaridade do sistema é que tanto o número de entrada quanto o de saída são fornecidos por

 Excerto de *idn/dis/op.c*

```

/* op.c - an operator interface (reads from stdin)
 *
 *      $RCSfile: op.c,v $ $Revision: 1.9 $
 *      IDN Project, SEL/EESC-USP.
 *      Refer to idn.txt for further information.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <assert.h>
#include "di.h"
#include "log.h"
#include "protocol.h"
#include "globals.h"
#include "system.h"

int wait_flag;

/* The 'issue' function: read lines from the standard input. */
void
read_from_stdin (idn_di_t *di)
{
    char *command;
    char *addressee;
    int read_bytes, size;

    addressee = NULL;
    size = 0;
    read_bytes = getline (&addressee, &size, stdin);
    if (read_bytes < 0)
    {
        idn_system_failure ();
        raise (SIGABRT);
        return;
    }
    addressee[read_bytes-1] = '\0';
    strtok_r (addressee, " ", &command);
    idn_di_request (di, addressee, IDN_STATUS_SUCCESS, command);
    free (addressee);
}

/* Function to process the configuration request. */
void
get_name (idn_di_t *di)
{
    if (idn_di_failure (di))
    {
        fprintf (stderr, "%s\n", di->last_message->command);
        exit (EXIT_FAILURE);
    }
    di->name = strdup (di->last_message->command);
    if (di->name == NULL)
    {
        perror ("ops");
        exit (EXIT_FAILURE);
    }
    idn_di_reply (di, IDN_STATUS_SUCCESS, "Thank you very much");
}

```

 Figura 7.30: Listagem de *op.c* (parte 1)

 Excerto de *idn/dis/op.c*

```

/* Function to handle consults. */

void
reply(idn_di_t *di)
{
    if (idn_di_failure (di))
        return;
    idn_di_standard_reply (di, address, IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "OK");
    idn_di_standard_reply (di, command, "version", IDN_STATUS_SUCCESS, "OP 0.1");
    idn_di_skip (di, address, di->name);
    idn_di_reply (di, IDN_STATUS_PERMANENT_CLIENT_ERROR, "What do you mean ?");
}

/* Function process regular replies. */

void
process(idn_di_t *di)
{
    sigset_t sigmask;
    int signum;

    if (idn_di_success (di))
    {
        if (wait_flag && !(idn_di_if_message(di, address, IDN_SYSTEM_NAME)))
        {
            sigemptyset (&sigmask);
            sigaddset (&sigmask, SIGINT);
            printf ("... ***====> ABOUT TO ANSWER <====***\n");
            sigwait (&sigmask, &signum);
        }
        idn_di_reply (di, IDN_STATUS_SUCCESS, "Thanks");
        return;
    }
    idn_di_reply (di, IDN_STATUS_SUCCESS, "Oh, I see");
}

int
main(int argc, char **argv)
{
    idn_di_t *di;
    char *myname;

    idn_request_t setup[4];

    if (argc < 2)
    {
        fprintf (stderr, "Usage: op NAME [w]\n");
        exit (EXIT_FAILURE);
    }
    wait_flag = 0;
    if (argc == 3)
    {
        if (strcmp (argv[2], "w"))
        {
            fprintf (stderr, "%s: invalid argument %s\n", argv[0], argv[2]);
            exit (EXIT_FAILURE);
        }
        wait_flag = 1;
    }

    asprintf (&myname, "name %s", argv[1]);

    idn_di_build_request (&setup[0], IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, myname, get_name);
    idn_di_build_request (&setup[1], NULL, IDN_STATUS_VOID, NULL, NULL);

    di = idn_new_di ();
    if (di == NULL)
        return EXIT_FAILURE;

    idn_di_configure_handlers (di, reply, read_from_stdin);

    di->setup_requests = setup;
    di->process = process;

    if ( ! idn_di_start(di) )
        printf ("Connection ended by server\n");

    return EXIT_SUCCESS;
}

```

 Figura 7.31: Listagem de *op.c* (parte 2)

 Excerto de *idn/dis/dev.c*

```

/* dev.c - a dummy service provider (parla solo in italiano)
 *
 *          $RCSfile: dev.c,v $ $Revision: 1.8 $
 *          IDN Project, SEL/EESC-USP.
 *          Refer to idn.txt for further information.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "di.h"
#include "log.h"
#include "protocol.h"
#include "globals.h"
#include "system.h"

int wait_flag;

void
get_name (idn_di_t *di)
{
    if (idn_di_failure (di))
    {
        fprintf (stderr, "%s\n", di->last_message->command);
        exit (EXIT_FAILURE);
    }
    di->name = strdup (di->last_message->command);
    if (di->name == NULL)
    {
        perror ("ops");
        exit (EXIT_FAILURE);
    }
    idn_di_reply (di, IDN_STATUS_SUCCESS, "Grazie");
}

void
reply(idn_di_t *di)
{
    if (idn_di_failure (di))
        return;
    idn_di_standard_reply (di, address, IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "Va bene");
    idn_di_standard_reply (di, command, "version", IDN_STATUS_SUCCESS, "DEV 0.1");

    if (wait_flag)
    {
        printf ("... *****> ABOUT TO ANSWER <*****\n");
        getchar ();
    }
    idn_di_skip (di, address, di->name);

    idn_di_standard_reply (di, command, "uno", IDN_STATUS_SUCCESS, "due");
    idn_di_standard_reply (di, command, "due", IDN_STATUS_SUCCESS, "quattro");
    idn_di_standard_reply (di, command, "tre", IDN_STATUS_SUCCESS, "sei");
    idn_di_standard_reply (di, command, "quattro", IDN_STATUS_SUCCESS, "otto");
    idn_di_standard_reply (di, command, "cinque", IDN_STATUS_SUCCESS, "dieci");

    idn_di_reply (di, IDN_STATUS_PERMANENT_CLIENT_ERROR, "Non lo capisco");
}

```

 Figura 7.32: Listagem de *dev.c* (parte 1)

 Excerto de *idn/dis/dev.c*

```

    idn_di_reply (di, IDN_STATUS_PERMANENT_CLIENT_ERROR, "Non lo capisco");
}

int
main(int argc, char **argv)
{
    idn_di_t *di;
    char *myname;

    idn_request_t setup[4];

    if (argc < 2)
    {
        fprintf (stderr, "Usage: dev NAME [w]\n");
        exit (EXIT_FAILURE);
    }
    wait_flag = 0;
    if (argc == 3)
    {
        if (strcmp (argv[2], "w"))
        {
            fprintf (stderr, "%s: invalid argument %s\n", argv[0], argv[2]);
            exit (EXIT_FAILURE);
        }
        wait_flag = 1;
    }

    myname = NULL;
    asprintf (&myname, "name %s", argv[1]);

    idn_di_build_request (&setup[0], IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, myname, get_name);
    idn_di_build_request (&setup[1], NULL, IDN_STATUS_VOID, NULL, NULL);

    di = idn_new_di ();
    if (di == NULL)
        return EXIT_FAILURE;

    idn_di_configure_handlers (di, reply, NULL);

    di->setup_requests = setup;

    if ( ! idn_di_start(di) )
        printf ("Connection ended by server\n");

    return EXIT_SUCCESS;
}

```

 Figura 7.33: Listagem de *dev.c* (parte 2)

Saída de op	Saída de dev
<pre>monaco@megryan:~/works/idn/dis\$./op A << IDNP/0.1 : SYSTEM 200 name A > IDNP/0.1 : SYSTEM 200 A < IDNP/0.1 : A 200 Thank you very much C uno << IDNP/0.1 : C 200 uno > IDNP/0.1 : C 200 due < IDNP/0.1 : A 200 Thanks C cinque << IDNP/0.1 : C 200 cinque > IDNP/0.1 : C 200 dieci < IDNP/0.1 : A 200 Thanks C venti << IDNP/0.1 : C 200 venti > IDNP/0.1 : C 300 Non lo capisco < IDNP/0.1 : A 200 Oh, I see</pre>	<pre>monaco@megryan:~/works/idn/dis\$./dev C << IDNP/0.1 : SYSTEM 200 name C > IDNP/0.1 : SYSTEM 200 C < IDNP/0.1 : C 200 Grazie > IDNP/0.1 : A 200 uno < IDNP/0.1 : C 200 due > IDNP/0.1 : C 200 due < IDNP/0.1 : C 200 due > IDNP/0.1 : A 200 cinque < IDNP/0.1 : C 200 dieci > IDNP/0.1 : C 200 dieci < IDNP/0.1 : C 200 dieci > IDNP/0.1 : A 200 venti < IDNP/0.1 : C 300 Non lo capisco > IDNP/0.1 : C 300 Non lo capisco < IDNP/0.1 : C 300 Non lo capisco</pre>
(a)	(b)

Figura 7.34: Comunicação entre *op* e *dev*

extenso, e em italiano — somente os naturais de 1 a 5 são reconhecidos.

Ambos os programas necessitam que lhes seja fornecido um argumento obrigatório quando da sua invocação na linha de comando: o nome sob o qual a ID deve registrar-se durante o procedimento inicial de configuração. Além deste, um segundo argumento, opcional, se fornecido, faz com que o programa espere por um sinal do usuário antes de enviar um retorno para a RDI. Em *dev.c*, o sinal é um caracter ENTER; em *op.c*, um caracter ^C.

O Exemplo da Figura 7.34 transcreve a conversação entre as duas entidades através do GI, junto do qual *op* registra-se sob o nome *A* e *dev*, sob o nome *C*. A linha em destaque na listagem da esquerda, logo após as mensagens de configuração, é a entrada fornecida no terminal. Abaixo desta, aparece a mensagem de interrupção gerada pelo procedimento do operador, seguida pela resposta recebida e depois pelo retorno final. A correspondente requisição aparece novamente em destaque como uma consulta recebida no excerto da direita.

O arranjo presta-se como uma analogia ao sistema de comunicação serial utilizado como exemplo na seção 5.2 (pág. 76). Supondo que “uno”, “due”, “tre” etc. sejam funções oferecidas pela interface de instrumentação *dev*, é possível utilizar-se de um programa para oferecer à aplicação em *op* o acesso a um conjunto de funções mais adequadas — por

Excerto de *idn/dis/proxy.c*

```

/* proxy.c - a proxy for dev.c (english-italian translator)
 *
 *      $RCSfile: proxy.c,v $ $Revision: 1.8 $
 *      IDN Project, SEL/EESC-USP.
 *      Refer to idn.txt for further information.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "di.h"
#include "log.h"
#include "protocol.h"
#include "globals.h"
#include "system.h"

int wait_flag;

/* Function to process the configuration reply. */

void
get_name (idn_di_t *di)
{
    if (idn_di_failure (di))
    {
        fprintf (stderr, "%s\n", di->last_message->command);
        exit (EXIT_FAILURE);
    }
    di->nare = strdup (di->last_message->command);
    if (di->nare == NULL)
    {
        perror ("ops");
        exit (EXIT_FAILURE);
    }
    idn_di_reply (di, IDN_STATUS_SUCCESS, "Thanks");
}

/* Function to process the configuration reply. */

void
check_implicit (idn_di_t *di)
{
    if (idn_di_failure (di))
    {
        fprintf (stderr, "%s\n", di->last_message->command);
        exit (EXIT_FAILURE);
    }
    idn_di_reply (di, IDN_STATUS_SUCCESS, "Thanks again");
}

```

Figura 7.35: Listagem de *proxy.c* (parte 1)

exemplo, “one”, “two”, “tree” etc. Isto é feito interpondo-se uma terceira ID *B* entre *A* e *C*.

Utiliza-se, então, um programa como *proxy.c*, listado na Figura 7.35 e Figura 7.36. O código-fonte e a forma de invocação da ID é semelhante ao de *op* e *dev*, exceto pelo fato de que *proxy* toma dois argumentos obrigatórios da linha de comando, tendo o primeiro o mesmo significado: o nome de registro. O segundo argumento deve ser o nome de qualquer ID na rede, o qual será utilizado pelo programa para solicitar ao dispositivo de sistema a configuração do seu alvo implícito.

Assim, com *A* e *C* já conectados ao GI, *proxy* pode ser chamado no terminal da forma

 Excerto de *idn/dis/proxy.c*

```

/* Function to attend to the RDI. */

void
reply(idn_di_t *di)
{
    if (idn_di_failure (di))
        return;
    idn_di_standard_reply (di, address, IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, "Ok");
    idn_di_standard_reply (di, command, "version", IDN_STATUS_SUCCESS, "OP 0.1");

    if (wait_flag)
    {
        printf ("... ***====> ABOUT TO ANSWER <====***\n");
        getchar ();
    }

    idn_di_skip (di, address, di->name);

    idn_di_standard_reply (di, command, "one", IDN_STATUS_SUCCESS, "uno");
    idn_di_standard_reply (di, command, "two", IDN_STATUS_SUCCESS, "due");
    idn_di_standard_reply (di, command, "three", IDN_STATUS_SUCCESS, "tre");
    idn_di_standard_reply (di, command, "four", IDN_STATUS_SUCCESS, "quattro");
    idn_di_standard_reply (di, command, "five", IDN_STATUS_SUCCESS, "cinque");

    idn_di_standard_reply (di, command, "due", IDN_STATUS_SUCCESS, "two");
    idn_di_standard_reply (di, command, "quattro", IDN_STATUS_SUCCESS, "four");
    idn_di_standard_reply (di, command, "sei", IDN_STATUS_SUCCESS, "six");
    idn_di_standard_reply (di, command, "otto", IDN_STATUS_SUCCESS, "eight");
    idn_di_standard_reply (di, command, "dieci", IDN_STATUS_SUCCESS, "ten");

    idn_di_standard_reply (di, command, "Non lo capisco", IDN_STATUS_SUCCESS, "I don't understand");

    idn_di_reply (di, IDN_STATUS_PERMANENT_CLIENT_ERROR, "Sorry ...");
}

int
main(int argc, char **argv)
{
    idn_di_t *di;
    char *myname;
    char *implicit;

    idn_request_t setup[3];

    if (argc < 3)
    {
        fprintf (stderr, "Usage: proxy NAME IMPLICIT [w]\n");
        exit (EXIT_FAILURE);
    }
    wait_flag = 0;
    if (argc == 4)
    {
        if (strcmp (argv[3], "w"))
        {
            fprintf (stderr, "%s: invalid argument %s\n", argv[0], argv[3]);
            exit (EXIT_FAILURE);
        }
        wait_flag = 1;
    }

    myname = NULL;
    asprintf (&myname, "name %s", argv[1]);

    implicit = NULL;
    asprintf (&implicit, "implicit %s", argv[2]);

    idn_di_build_request (&setup[0], IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, myname, get_name);
    idn_di_build_request (&setup[1], IDN_SYSTEM_NAME, IDN_STATUS_SUCCESS, implicit, check_implicit);
    idn_di_build_request (&setup[2], NULL, IDN_STATUS_VOID, NULL, NULL);

    di = idn_new_di ();
    if (di == NULL)
        return EXIT_FAILURE;

    idn_di_configure_handlers (di, reply, NULL);

    di->setup_requests = setup;

    if ( ! idn_di_start(di) )
        printf ("Connection ended by server\n");

    return EXIT_SUCCESS;
}

```

Figura 7.36: Listagem de *proxy.c* (parte 2)

Figura 7.37: Invocação de proxy

mostrada na Figura 7.37 (a). O trecho mostra uma tentativa frustrada de iniciar o programa que fora abortada devido ao fato de o alvo implícito não existir. Logo, em seguida (em destaque) há uma chamada de sucesso. A situação do manipulador de interfaces nesse ponto pode ser vista na Figura 7.37 (b), onde listam-se quatro conectores e nenhuma instância. O destaque na saída de `lc` chama atenção para o alvo implícito de *B* apontando para *C*, gerando o programa *B : C*. Esse e todos os outros programas registrados podem ser visualizados com o comando `p`, também em destaque. Por último, o comando `c` fornece uma lista das cadeias estabelecidas — no caso, não há nenhuma.

A Figura 7.38 (a) mostra os registros do módulo *idn* no instante em que *A* envia uma interrupção endereçada a *B*. Imediatamente à detecção da mensagem, uma cadeia é estabelecida: auto-evocada, a instância 9 em *A* evoca em *B* a instância 10, que evoca em *C* a instância 11, que tenta evocar sua sucessora no mesmo conector, terminando assim a sequência. O resultado pode ser visualizado no *IDN shell*, cuja saída é transcrita na Figura 7.38 (b). O comando `li` lista as três instâncias e o comando `c` informa que estas formam uma cadeia *A – B – C*.

Daí em diante, desenvolve-se o ciclo de operação. Em vez de transcrever os registros gerados por *idn* — que constituem uma listagem consideravelmente extensa ao longo das suas 158 linhas e das 33 entregas realizadas pelo despachador — utiliza-se em seu lugar a representação gráfica adotada anteriormente. Para indicar precisamente qual ocorrência

Saída de idn	Saída de IDN shell
<pre> > waiting for the next network event > new shell connection > . message from device 2: IDNP/0.1 : B 200 one > message arriving in an empty connector (interruption) > self-evocation at A 9, so chain outset > A 9 requests explicit target > and evokes the new instance B 10 > which resorts to implicit target > and evokes the new instance C 11 > which resorts to implicit target > referring to itself, so chain endpoint at C 11 > . chain establishment completed </pre>	<pre> idn> li [A] 9 PI Refer Regular (10, 9) ms 0 100 [B] 10 SB Ready Regular (11, 9) ms 0 100 [C] 11 SB Ready Regular (11, 10) ms 0 100 idn> c chain: [A] 9 PI Refer : 100 [B] 10 SB Ready : 100 [C] 11 SB Ready : 100 idn> </pre>
(a)	(b)

Figura 7.38: Estabelecimento da cadeia $A - B - C$

Símbolo	Evento	Significado
!	INTERRUPÇÃO	chegada de uma interrupção, logo antes de haver sido detectada pelo despachador
+	RETORNO	chegada de um retorno, logo antes de haver sido detectado pelo despachador
↓	RECEBIMENTO	despachador lê a mensagem da ID e armazena no dispositivo
↑	TRANSMISSÃO	despachador retira a mensagem do dispositivo e envia para a ID
↓	ENTRADA	despachador transfere a mensagem do dispositivo para a instância
↑	SAÍDA	despachador transfere a mensagem da instância para o dispositivo
→	REFERÊNCIA	despachador transfere a mensagem de uma instância para sua sucessora
←	REPORTE	despachador transfere a mensagem de uma instância para sua sucessora
×	DISPENSA	despachador extingue uma cadeia obsoleta

Figura 7.39: Símbolos que representam eventos do despachador

gera a respectiva modificação na cadeia, os símbolos adicionados explicados na Figura 7.39 são empregados.

O processo que tem lugar pode ser indicado como na Figura 7.40. Nela, a sequência inicia pelo diagrama no canto superior esquerdo e segue no sentido vertical descendente, coluna por coluna. O primeiro evento (!) é a chegada da interrupção de A , causando a evocação da cadeia. Imediatamente a seguir, a mensagem é recebida (↓), deixando a primeira instância com uma entrada pendente no dispositivo ao qual serve. Em cada círculo, indica-se o conector e o número da instância¹⁶.

Assim que é chamado, o despachador detecta a pendência e realiza o recebimento da mensagem (↓), levando ao próximo estágio, onde há uma referência pronta para ser efetivada. Atendida (→), chega-se à situação seguinte. Pela ordem de verificação de

¹⁶Lembrando que o contador de instâncias é continuamente incrementado, os números de identificação 9, 10 e 11 refletem a pre-existência de cadeias extintas, oriundas das requisições de configuração das IDs envolvidas.

Figura 7.40: Exemplo B

entregas, o despachador não encontra cadeias obsoletas¹⁷, mas identifica a saída pendente e a executa (↑). Depois vem uma transmissão (⇑) para *B* e a espera pelo seu retorno. Quanto este chega (+), o processo continua desta forma até o final da seqüência, onde *A* marca o início de uma cadeia a espera da sua dispensa. A eliminação da cadeia (×)

¹⁷Em cada passo, o símbolo de evento indica apenas a ação realizada. Por exemplo, se pela ordem de verificação de pendências o despachador procurar sem sucesso por entradas (↓), referências (→), dispensas (×) e saídas (↑), e encontra apenas um reporte (←) pronto para ser atendido, então somente o último símbolo é anotado.

Figura 7.41: Execução da cadeia A-B-C

responde pelo último diagrama.

O resultado pode ser visto na Figura 7.41. Em negrito aparece a tarefa na fase ascendente do ciclo de operação: partindo de *A* (“one”), sendo convertida (“uno”) em *B* e seguindo, então, até *C*. O semi-ciclo complementar aparece sublinhado: deixando *C* (“due”), sendo novamente traduzida (“two”) em *B* e retornando finalmente até *A*.

Apenas para completar a descrição, é oportuno mostrar aqui o que acontece se, em meio ao ciclo de operação, uma das IDs envolvidas desconectar-se abruptamente — na prática a situação pode ser criada terminando o programa *proxy* antes que *B* receba a resposta de *C*. A Figura 7.42 (a) fornece a condição do manipulador logo antes da desconexão. Em destaque, nota-se a instância número 10 no conector “B” formando a cadeia *A* – *B* – *C*.

No instante posterior, a Figura 7.42 (b) permite conferir que o conector “B” fora removido, e a instância número 10, tornando-se órfã, fora adotada pelo conector do sistema. Conforme informa o comando *c*, a cadeia agora é *A* – *SYSTEM* – *C*. Ainda, a Figu-

Figura 7.42: Quebra abrupta de uma cadeia.

ra 7.42 (c) lista o registro do módulo *idn* referente ao evento de desconexão, enquanto a Figura 7.42 (d) mostra a resposta que *A* recebe mais tarde ao final do ciclo de operação.

Uma alteração que poderia ser experimentada é a inclusão de um parâmetro a mais na estrutura do conector (*idn_connector_t*) para indicar uma “marca d’água”: um número mínimo de dispositivos que deve existir naquele elemento, abaixo do qual este seja marcado para remoção. Na implementação, esse número é virtualmente 1, mas bem poderia ser zero; nesse caso, mesmo que IDs acidentalmente se desconectem, o conector seria mantido até

que aquela se reconecte, sem destruir programas e cadeias ativas.

É ainda possível considerar mais uma sofisticação do sistema de programação utilizando-se, em lugar de um alvo implícito fixo, uma regra para configuração dinâmica deste parâmetro, possivelmente em função da origem da interrupção, ou do conector precedente no programa. A exploração dessa possibilidade pode ser um tópico para futuros desenvolvimentos do protótipo.

7.5 Ciclos de Operação Concorrentes

A fim de descrever como o módulo *idn* implementa ciclos de operação concorrentes, é conveniente iniciar com uma visão clara do algoritmo de verificação de pendências do despachador, o qual pode ser representado pela Figura 7.43. O laço mais externo do fluxograma representa o laço principal do GI, onde o procedimento de espera, ao alto, é o monitor de eventos da rede. A linha de execução principal lá permanece bloqueada até a chegada de uma mensagem, que pode ser um um retorno ou uma interrupção — no primeiro caso, o procedimento padrão é a evocação de uma nova cadeia de instâncias para comportá-la.

Tanto em um quanto em outro caso, o próximo evento é a recepção da transferência da mensagem para o dispositivo (recepção). A partir desse ponto, adentra-se efetivamente ao domínio do despachador, representado pelo laço mais interno do fluxograma. Nele, atende-se, nesta ordem, no máximo uma pendência de cada um dos tipos: entrada, referência, dispensa, saída e reporte. Procede-se assim, iterativamente, até que na última repetição nenhuma entrega seja efetuada. Todas as mensagens armazenadas nos dispositivos são então transmitidas para as respectivas IDs e o laço principal retorna ao procedimento de espera.

Assim, para efeito de examinar os diversos cenários, no que concerne ao processamento de ciclos de operação concorrentes, deve se observar que a chegada de novas mensagens — não importa em que instante cheguem ao monitor de eventos — só são detectadas uma vez a cada ciclo completo. Enquanto isso, se forem recebidas pelo serviço de transporte, as mensagens permanecem no buffer do *socket* esperando para serem lidas.

Figura 7.43: Algoritmo do despachador

Para ilustrar como o esquema funciona, é interessante iniciar com um arranjo bem simples, onde duas IDs concorrem pelo acesso de uma terceira. Valendo-se do exemplo da seção anterior, seja o caso onde ambos os clientes *A* e *B* são interfaces de usuário fornecidas pelo programa *op*, e o servidor *C*, pelo programa *dev*. Supondo que *A* gere a primeira interrupção, e considerando-se a sequência de eventos que podem se suceder dependendo de quem produza a segunda, terceira etc. mensagens, a Figura 7.44 fornece uma representação esquemática de todas as combinações possíveis.

Partindo-se da raiz (topo) da árvore, onde *A* é a emissora da primeira interrupção, e seguindo de nó em nó, o completamento de ambos os ciclos de operação pode se dar segundo 13 caminhos alternativos. No caminho número 1, por exemplo, essencialmente não há concorrência: *D* emite uma interrupção depois de *A* haver enviado seu retorno final e, portanto, quando a primeira cadeia já houvera sido extinta.

Normalmente, quando não há interseção entre os ciclos de operação, a cada recebimento da mensagem pelo GI, corresponde exatamente uma transmissão (fluxograma da Figura 7.43), cada interrupção ou retorno que chega ao GI é entregue e despachado para

Figura 7.45: Manipulação de requisições pendentes (parte 1)

retorno de C , mais a requisição de D que ficara suspensa; daí em diante os ciclos seguem normalmente (Figura 7.46).

Um exemplo mais completo, onde todos os possíveis cenários de concorrência vêm a tona, é dado a seguir, tomando como base um programa $B : C$ acessado simultaneamente por dois clientes A e D . O cenário para esse caso é bem mais complexo, com os 109

Figura 7.46: Manipulação de requisições pendentes (parte 2)

caminhos alternativos mostrados na Figura 7.47¹⁸ — um a um, todos estes foram ensaiados no sistema-piloto.

¹⁸Para simplificação, os ramos são interrompidos a partir do ponto onde uma das duas cadeias é extinta.

Figura 7.47: Possíveis caminhos para três ciclos concorrentes

Seja, por exemplo, o ramo que parte do início da árvore e chega ao ponto final através da sequência A, B, D, C, C, B, B, A (que deve continuar com C, C, B, D). Nele, os ciclos se desenvolvem como mostrado na Figura 7.48. No passo em destaque (retângulo com borda dupla), o dispositivo no conector B está envolvido na cadeia $A - B - C$. Não obstante, uma vez que a ID está pronta para atender solicitações, a tarefa da cadeia $D - B - C$ pode lhe ser entregue. A partir daí, ou B ou C retorna primeiro, cada qual para a cadeia que a espera — as duas alternativas levam ao completamento correto do ciclo de operação.

A sequência mostrada corresponde ao segundo caso: C retorna primeiro, depois retorna novamente antes de B no próximo evento. Isso faz com que a tarefa na primeira cadeia encontre a ID em B ocupada e, portanto, nenhuma saída é transmitida (retângulo sólido). Logo mais, quando B responde, ambas as mensagens são liberadas (retângulo tracejado) e, daí em diante, se B retornar primeiro, ambos os ciclos seguem normalmente até o final.

Decorre das simplificações adotadas na implementação do protótipo e do protocolo de aplicação experimental que todas as IDs na rede são tratadas de forma indiferenciada, resultando em um modelo de recurso bastante rudimentar. Por exemplo, o despachador, em princípio, não faz qualquer distinção entre quais deles podem ser acessados por ciclos de operação concorrentes (sensores) e quais necessitam de reserva de uso exclusivo (atuadores). Assim, para ser aplicável na prática, o mecanismo IDN necessita ainda definir um modelo de recurso adequado. Conquanto não se tenha avançado nesse aspecto, sugere-se, para trabalhos seguintes, investigar um modelo genérico para a especificação da funcionalidade, onde os recursos possam configurar as características de cada função exportada, com base, por exemplo, nas propriedades de *reentrância* (possibilidade de ser acionada de forma independente em tarefas concorrentes), *idempotência* (acionamentos sucessivos da mesma função geram sempre um mesmo resultado) etc.

7.6 Replicação de Recursos

No exemplo mais simples de ciclos de operação concorrentes, onde duas IDs A e D disputam pelo acesso a uma terceira ID C (seção 7.5, pág.226), demonstrou-se como o gerenciador central armazena a mensagem da segunda entidade-cliente quando a servido-

Figura 7.48: Operações reentrantes (parte 1)

ra ainda está ocupado com o primeiro pedido. Contudo, se o serviço prestado pela ID C é freqüentemente requisitado, pode ser mais conveniente aumentar sua disponibilidade mediante a técnica de replicação, o que é feito ajuntando à RDI uma ou mais IDs suplementares idênticas à primeira (mesmo conjunto de instruções). Depois de conectadas, cada uma delas deve, então, mover-se de seu respectivo conector para aquele onde está a ID a ser replicada.

Para compor um cenário bem simples, seja o caso ilustrado na Figura 7.50 (a), que mostra o manipulador de interfaces através do *IDN shell*. O dispositivo 2, no conector

Figura 7.49: Operações reentrantes (parte 2)

“client-1”, o dispositivo 3, no conector “client-2” e o dispositivo 4, no conector “resources” são três IDs conectadas ao GI — as duas primeiras (programa *op*) representam clientes da terceira (programa *dev*).

Uma quarta ID é, então, conectada e a ela o GI atribui o dispositivo 5 e o conector de rótulo “16”. Esta é fornecida pelo programa *replica.c*, uma versão ligeiramente modificada do programa *dev.c*, onde o vetor de requisições fora alterado de forma que, em lugar do comando “name”, para registrar o nome do conector, o comando “join” é utilizado para transferir o dispositivo para um outro conector já existente.

Saída de IDN shell	Saída de IDN shell
<pre>idn> 1 1. empty srv 0, int 0 [SYSTEM] 2. empty srv 0, int 0 [client-1] 3. empty srv 0, int 0 [client-2] 4. empty srv 0, int 0 [resource] 5. empty srv 0, int 0 [5] idn></pre>	<pre>idn> 1 1. empty srv 0, int 0 [SYSTEM] 2. empty srv 0, int 0 [CLIENT-1] 3. empty srv 0, int 0 [CLIENT-2] 4. empty srv 0, int 0 5. empty srv 0, int 0 [RESOURCE] idn></pre>
(a)	(b)

Figura 7.50: Configuração de réplicas

A situação capturada pelo comando 1 é aquela imediatamente anterior à requisição ser emitida. No extrato seguinte, visto na Figura 7.50 (b), logo após a mensagem (IDNP/0.1 : SYSTEM join RESOURCE) haver sido enviada e processada, o manipulador apresenta-se rearranjado: ambos os dispositivos 4 e 5 estão agora em RESOURCE e o último conector, vazio, fora removido. A operação que leva o manipulador da condição inicial para a final pode ser acompanhada na Figura 7.51 (a seqüência se desenvolve a partir do canto superior esquerdo, na vertical descendente, coluna por coluna).

Agora, com essa configuração, um dos clientes endereça uma mensagem para o conector servidor, causando o estabelecimento de uma cadeia. Ao atingi-lo, a tarefa é entregue ao primeiro dispositivo livre no conector. Se antes de este produzir uma resposta, o outro cliente também enviar sua mensagem, esta formará uma segunda cadeia que transportará a tarefa até o próximo dispositivo livre. Nesse ponto o manipulador tem a estrutura esquematizada na Figura 7.52 — uma vez que cada dispositivo sabe quem é seu servo, a resposta tem seu caminho de volta garantido.

Além de servir em prol da disponibilidade, a replicação é também a técnica elementar no aprimoramento da confiabilidade de recursos e, embora não contemplada ainda no sistema-piloto, não é difícil estender o mecanismo IDN para explorar essa possibilidade.

Figura 7.51: Transferência de dispositivos

Sistemas de tempo real — domínio onde se insere a aplicação estudada — usualmente empregam *replicação ativa*¹⁹ como base para elevar a tolerância a faltas [53]. No protótipo, a estratégia pode ser experimentada adicionando-se um parâmetro a mais à estrutura `idn_connector.t` para indicar o número de réplicas exigidas. Desse modo, o procedimento

¹⁹Replicação ativa: a requisição é aplicada às entradas de n réplicas; uma das n saídas (ou uma combinação destas) produz a resposta final. Replicação passiva: a requisição é aplicada à entrada de uma réplica; falhando a resposta, a próxima réplica é consultada e assim sucessivamente.

Figura 7.52: Acessos simultâneos a um recurso replicado

`idn_handle_pending_output` pode ser modificado para transferir a mensagem não a um, mas a n dispositivos e aguardar pela chegada de todas as respostas. Ao serem recebidas por `idn_handle_pending_input`, também estendida, estas podem ser comparadas com base em critérios pré-estabelecidos (voto da maioria, média dos valores etc.) para gerar uma única resposta.

7.7 Avaliação de Desempenho

No que se refere à carga gerada pela sua operação sobre a plataforma onde é executado, o programa *idn* é relativamente pouco custoso para o sistema operacional, conforme pode-se inferir a partir dos dados da Figura 7.53 (a), onde transcreve-se a saída gerada pelo utilitário *ps*, o qual obtêm informações a cerca dos processos no sistema. Em virtude dos detalhes internos da atual implementação de *LinuxThreads*, cada uma das linhas de execução do programa (linha principal, linha do sistema e linha do *IDN shell*) aparece como um processo distinto na listagem (a segunda entrada é na verdade uma quarta linha de execução que responde pela linha de gerenciamento).

Sem atividade, a carga computacional é mínima, como mostra a porcentagem de utilização do processador (%CPU, TIME) e de memória (%MEM, VSZ, RSS). A Figura 7.53 fornece uma segunda listagem do comando *ps* obtida após 55 minutos de operação, com as

Saída de ps									
USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME COMMAND
monaco	778	0.0	2.4	1336	752	pts/3	S	16:46	0:00 ./idn
monaco	779	0.0	2.4	1336	752	pts/3	S	16:46	0:00 ./idn
monaco	780	0.0	2.4	1336	752	pts/3	S	16:46	0:00 ./idn
monaco	781	0.0	2.4	1336	752	pts/3	S	16:46	0:00 ./idn

(a)

Saída de ps									
USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME COMMAND
monaco	778	0.0	2.4	1336	752	pts/3	S	17:41	0:21 ./idn
monaco	779	0.0	2.4	1336	752	pts/3	S	17:41	0:00 ./idn
monaco	780	0.0	2.4	1336	752	pts/3	S	17:41	0:00 ./idn
monaco	781	0.0	2.4	1336	752	pts/3	S	16:71	0:00 ./idn

(b)

Figura 7.53: Utilização de recursos da plataforma

IDs *example_1* e *example_2* comunicando-se à taxa de um ciclo de operação por segundo. Para essa condição típica de utilização (aceitável para comunicação via Internet), o sistema trabalha por volta de 1% do tempo. Testes envolvendo números maiores de IDs na rede comunicando-se a taxas reduzidas demonstraram que o desempenho do sistema não se degrada apreciavelmente até por volta de 30 a 50 elementos, mas varia perceptivelmente em função da utilização da plataforma (memória disponível, outros processos em execução etc.).

O maior problema, todavia, revelou-se quando o sistema foi testado fora da carga de operação prevista. Na prática, este foi detectado pela primeira vez em um dos ensaios, onde um grupo de 8 IDs, operando em diferentes frequências, era conectado à RDI. Em determinadas circunstâncias, o arranjo exibia falhas esporádicas de comunicação entre alguns dos elementos. Mais tarde, observou-se que o mesmo acontecia invariavelmente quando a RDI era composta por um número muito elevado de IDs ou quando, mesmo em conjuntos menores, algumas delas trabalhavam em taxas mais elevadas de requisições, resultando daí que algumas IDs simplesmente não eram atendidas. O cenário, conhecido na literatura como *starvation*, alude bem propriamente à situação onde um cliente é fadado a “morrer de fome” enquanto outros se fartam do serviço oferecido pelo sistema.

Procurando pela sua causa, o limite de carga do GI foi medido experimentalmente por meio de um teste realizado em circunstância absolutamente além do cenário assumido.

Havendo apenas duas IDs na rede, e deixando que a ID cliente enviase interrupções tão rapidamente quanto pudesse gerá-las, observou-se que o mecanismo foi capaz de atender cerca de 1800 tarefas completas em 2 minutos, o que resulta em uma taxa máxima de 15 requisições por segundo (dados obtidos para a comunicação via rede local a 10Mbps entre duas estações i586, 166MHz), e uma carga de 10% do tempo.

Constatou-se, assim, que a limitação decorre da violação da hipótese de cobertura [53] levantada na seção 6.1.3. Pela suposição original de que o intervalo entre mensagens sucessivas é maior que o tempo de restabelecimento do GI, deveria haver a qualquer momento não mais do que uma mensagem a espera de atendimento do monitor de eventos. Entretanto, se estas são recebidas a uma taxa superior à estipulada (quando o fluxo total de mensagens geradas por cada ID se aproxima do limite de 15 requisições por segundo), então, mensagens pendentes são armazenadas pelo serviço de transporte. Isso, em razão do modo como a função `idn_next_network_event` é implementada, faz com que o sistema originalmente acionado por eventos passe a operar como um sistema acionado por tempo; há, então, uma ordem de prioridade implícita no vetor de *sockets*, onde as primeiras IDs a se conectarem têm precedência sobre as outras — quando coincide de estas emitirem requisições com maior frequência, as demais são preteridas.

A questão poderia ser solucionada alterando-se o escalonador de atendimento da referida função, cujo método atual, se não absolutamente impróprio, é sem dúvida pouco conveniente. Contudo, há outros problemas graves com seu projeto que justificam alterações mais fundamentais. Por exemplo, da maneira como programado, o sistema é vulnerável a um tipo de ataque DoS (*Denial of Service*)²⁰ muito simples: uma vez que a linha principal lê as mensagens diretamente do vetor de *sockets*, basta que um cliente maliciosamente demore-se em terminar sua transmissão para que todo o sistema fique congelado. Isso, associado ainda à inadequada finalização da conexão que leva a porta do servidor ao resguardo do `TIME_WAIT`, confirma que o programa necessita de modificações. De fato, como um servidor TCP concorrente, o laço principal do GI pode ser melhor implementado utilizando-se processos ou linhas de execução distintos para efetuar a leitura dos *sockets* e,

²⁰Negação de Serviço: exploração de falhas de segurança em sistemas de serviços que impede o acesso correto às suas funções.

só então, transferir os dados para a linha principal.

Em tempo, uma questão ora lembrada mas não aprofundada no trabalho, é como minimizar a vulnerabilidade do ponto crítico representado pelo elemento central; uma possibilidade é utilizar replicação ativa do GI em processadores distintos e conexões redundantes entre as IDs e as réplicas.

Fora da rede local, conforme esperado, constata-se que a comunicação constitui de fato o gargalo do desempenho. É de pouco interesse levantar índices nessa categoria, posto que essas medidas são praticamente determinadas pelas características da própria rede (tipo de canal, equipamentos de conectividade, etc.) e de condições momentâneas (demanda, modificações de configuração, congestão etc.), que influenciam na taxa de transmissão, latência, confiabilidade e outros parâmetros da comunicação.

Apenas como informação qualitativa, alguns dados experimentais foram obtidos com o auxílio de colaboradores que atenderam ao convite de executar uma série de testes de desempenho através da rede, a partir dos quais estimaram-se alguns indicadores usualmente empregados para descrever as características da comunicação fim-a-fim na Internet²¹. Nos gráficos apresentados a seguir, as estações indicadas são aquelas relacionadas na Figura 7.54 — para cada uma delas o número de *hops* (nós intermediários) até a estação-base *winona* fora determinado por meio do utilitário *tracerout*²², comumente disponível em plataformas GNU/Linux.

O gráfico da Figura 7.55 mostra os valores do indicador RTT (*round-trip time*, tempo de ida e volta) que estima o tempo que um datagrama leva para ir da origem ao destino, e de lá de volta para a origem (pacotes ICMP de 64 bytes). A diferença dos tempos entre a conexão na LAN, a um *hop* dentro do campus, via linha ADSL à 11 *hops* dentro do país e em de 20 *hops* através do link intercontinental, é de ordens de grandeza. A discrepância não se dá apenas em função da configuração geográfica mas também de fatores temporais, como mostra o gráfico da Figura 7.56, o qual indica os valores médios do RTT monitorados durante o período de um mês.

²¹Para tal, alguns pequenos programas em linguagem C foram escritos e distribuídos para os participantes.

²²O utilitário *traceroute* utiliza um artifício baseado em sucessivos incrementos campo TTL (*time-to-live* do cabeçalho IP) e mensagens ICMP para descobrir o caminho de um datagrama UDP através dos

Figura 7.54: Estações que tomaram parte nos ensaios

Figura 7.55: Valores do RTT para diversos caminhos na Internet

Conforme se observa, portanto, ainda que operando possivelmente nas condições mais
segmentos ponto-a-ponto da origem ao destino.

Figura 7.56: Valores de RTT entre duas estações durante 30 dias

Figura 7.57: Desempenho da comunicação através da Internet

O gráfico à esquerda compara os tempos de transmissão para mensagens de diversos tamanhos: L é o comprimento em bytes, T o tempo de transferência de 1 byte (curva aproximada a partir de várias amostragens). À direita, o gráfico compara o tempo de estabelecimento da conexão (E) com o tempo de transferência (T) de 100 bytes — em geral, o tempo de conexão é superior ao tempo de transferência de mensagens curtas.

favoráveis — i.e. sobre uma conexão de transporte persistente e através de mensagens curtas, conforme ilustra a Figura 7.57 (a) e Figura 7.57 (b) — a imprevisibilidade e inconstância da qualidade da comunicação restringem bastante a possibilidade de operação em tempo-real.

Capítulo 8

Conclusões

Redes de dispositivos telemáticos emergem como uma tecnologia promissora nesta primeira década do século, capturando as potencialidades oriundas dos recentes avanços na área dos sistemas de transmissão e processamento de informação. Com a Internet em destaque, sistemas tele-interativos de operação remota têm sido investigados em suas inúmeras aplicações, incluindo ensino à distância, telemedicina, instrumentação remota, tele-automação, sistemas CSCW¹, tele-manufatura, dentre outros.

Uma dentre as visadas possibilidades nesse terreno é o aproveitamento das capacidades de computação e comunicação dos dispositivos telemáticos para a construção de efetivos sistemas de compartilhamento de equipamentos em escala global — um conceito que tem inspirado trabalhos de pesquisa que vão desde a telemetria via Internet até redes de dispositivos autônomos inteligentes, culminando em ambiciosas concepções de ambientes colaborativos, onde realidade virtual e teleoperação se combinam em sofisticados mecanismos de telepresença nas quais pessoas, bases de dados, recursos computacionais e instrumentos interagem independentemente de restrições geográficas [95]. Destacam-se nesta frente, por exemplo, os *Collaboratories* (“Colaboratórios”, aglutinação de “colaboração” e “laboratório”, Wulf [96]), um projeto multi-institucional suportado por diversas organizações de pesquisa e ensino nos EUA, no qual objetiva-se a criação de uma infra-estrutura nacional para compartilhamento de recursos telemáticos, ligando universidades, indústrias e sistema

¹ *Computer-supported collaborative work*, trabalho colaborativo auxiliado por computador.

de ensino médio e fundamental.

Neste estudo, investigam-se os fundamentos da operação à distância em redes telemáticas através da revisão das atuais tendências no emergente campo do comando remoto via Internet, examinando-se os aspectos tecnológicos com relação ao seu desenvolvimento e aplicações. A partir de uma extensa revisão do assunto, abarcando tanto o cenário acadêmico quanto a iniciativa industrial, reconhece-se que a pesquisa nesse terreno apenas iniciara. Assim, não obstante o interesse que aceleradamente vem se concentrando na área em virtude do crescente reconhecimento de suas potencialidades, o estudo permite constatar a condição ainda incipiente que a caracteriza no que diz respeito ao âmbito prático. De fato, contrastando com o acelerado interesse nos primeiros protótipos operacionais, ainda que rudimentares, urge a questão acerca da viabilidade tecnológica da construção de efetivos sistemas de compartilhamento de recursos, especialmente no que diz respeito à experiência com a insatisfatória confiabilidade e desempenho permitidos pela atual tecnologia Internet.

No contexto assinalado, a contribuição proposta pelo trabalho oferece-se através de uma abordagem alternativa em relação as técnicas convencionais atualmente difundidas na área em foco, e cujas limitações levantadas motivaram a decisão de experimentar um modelo distribuído. A original arquitetura RDI, resultante dessa perspectiva, baseia-se no conceito de ambiente virtual cooperativo, mais precisamente, projetado como uma rede de dispositivos de tempo-real que comunicam-se segundo um modelo de interação com propriedades específicas.

Tendo, portanto, como orientação a aplicação prática, a investigação focaliza-se na implementação de um mecanismo experimental, desde a seleção das ferramentas, até a construção de um sistema-piloto. Submetendo, então, o protótipo a uma série de ensaios para avaliar seu comportamento em situações típicas, tem-se como objetivo verificar como as definições do modelo sugerido satisfazem as necessidades anotadas durante o desenvolvimento conceitual. Comparando-se os resultados com os que podem ser obtidos por meio de técnicas tradicionalmente empregadas é possível concluir que o mecanismo sugerido logra atender aos objetivos para o qual fora proposto.

8.1 Avaliação do Sistema

Os critérios considerados na avaliação do modelo estudado incluem os parâmetros normalmente de interesse na análise de sistemas de tempo-real distribuídos: flexibilidade, escalabilidade, desempenho e confiabilidade. O primeiro deles, essencial à generalidade da aplicação, decorre naturalmente da descentralização da funcionalidade, o conceito-chave do modelo RDI. A definição de uma interface de comunicação descomplicada e a concentração dos serviços elementares no gerenciador central possibilitam a criação de aplicações sofisticadas a partir de componentes modulares relativamente simples, inclusive através da configuração de dispositivos complexos a partir da participação combinada de IDs singulares em camadas funcionais, segundo programação fornecida ao GI. Ainda, o protocolo IDNP prevê o aprimoramento do modelo, reservando espaço para a implementação de novos recursos.

A API fornecida como exemplo — ou versões mais elaboradas que delas possam derivar — demonstra a facilidade de criação de aplicações em RDI. Embora tenha se utilizado como ilustração uma biblioteca de programação para sistema operacional GNU/Linux (na verdade, válida para qualquer sistema POSIX), suas funções, bastante simples, podem ser facilmente portadas para arquiteturas diversas, inclusive em sistemas embutidos microcontrolados. A idéia é que, mediante a especificação de um CMI (conjunto mínimo de instruções) e CIAs (conjunto de instruções da aplicação) adequados, conjuntamente com a disponibilização de IDs especializadas, seja possível criar pacotes de componentes prontos (*“off-the-shelf”*) para serem utilizados como *“kits”* de desenvolvimento. Em trabalhos futuros, além das correções que o protótipo necessita, seria interessante estender o protótipo IDN para obter um mecanismo operacional — para tal, resta ainda completar a rudimentar especificação do protocolo IDNP, e construir um pacote básico de desenvolvimento a partir da API fornecida como exemplo.

Em uma utilização prática, é possível conceber equipamentos inteligentes exportando sua funcionalidade através da Internet, solicitando e provendo serviços especializados na forma de dispositivos tele-interativos. Em uma instalação científica, por exemplo, instrumentos de aquisição de dados podem requisitar instruções de configuração disponíveis em

bancos de dados na rede, ou podem enviar amostragens de leituras para processamento estatístico em estações dedicadas. Do mesmo modo, em uma aplicação doméstica, um aparelho de gravação de vídeo pode consultar os horários de exibição de determinada apresentação diretamente do servidor da operadora de TV, assim como sensores de segurança em uma linha de produção industrial podem emitir ordens de suspensão ou reconfiguração do maquinário à detecção de anomalias durante o funcionamento. O leque de sugestões é sem dúvida extenso, mas o princípio geral é o compartilhamento de recursos, entre pessoas e equipamentos, e entre cada um destes entre si.

Tendo em vista um sistema de âmbito global, como no caso da Internet, a escalabilidade é também um parâmetro fundamental. Nesse aspecto, tanto o modelo RDI quanto o protótipo introduzido apresentam propriedades satisfatórias, principalmente em função do projeto modular. É possível até mesmo ampliar a idéia de uma RDI para uma rede de RDIs, onde vários GIs sejam interligados a fim de criar infra-estruturas mais complexas. Conquanto o sistema-piloto esteja sujeito a degradação de desempenho relacionada ao aumento do número de IDs na rede, as diversas sugestões de modificações registradas ao longo do texto devem fornecer um caminho para o aprimoramento do protótipo.

Finalmente, os últimos dois critérios, desempenho e confiabilidade, possivelmente respondam atualmente pelas maiores preocupações no terreno em exploração. Para ambos existe limitantes externos associados à tecnologia de comunicação que, com referência à Internet, são ainda campos necessitando de desenvolvimento urgente. Com efeito, o grande entusiasmo gerado pela aplicação em destaque por parte de um crescente número de usuários cada vez mais habituados às facilidades da Internet coexiste com uma sensível incredulidade na viabilidade de sua implantação — um receio justificável a partir da vivência na utilização diária da rede. Com efeito, enquanto admite-se em geral a teleoperação telemática em empregos bastante simples, tais como em ensino à distância e entretenimento; reais utilizações em sistemas científicos, industriais ou médicos, por outro lado, são ainda muitas vezes relegados à ficção científica.

Dentre os dois parâmetros, talvez o desempenho seja a meta menos desafiadora. É certo que, para muitas aplicações, as taxas de transmissão obtidas através de redes de longa distância deixam bastante a desejar. Contudo, para outra classe também extensa,

os resultados obtidos experimentalmente permitem concluir pela viabilidade de diversas utilizações do compartilhamento de equipamentos telemáticos via Internet, especialmente onde o tempo de resposta do recurso acessado é razoavelmente superior ao atraso médio na comunicação.

Uma das maneiras de classificar sistemas de tempo real é a que os distingue com base na magnitude da consequência de eventuais falhas. Se essa medida for da mesma ordem que os benefícios do sistema (execução de um experimento didático), estes são considerados flexíveis; se os danos forem ordens de magnitude superior ao benefício (controle de uma planta nuclear), o sistema é considerado rígido [53]. Segundo essa categorização, redes de dispositivos telemáticos sobre a Internet enquadram-se hoje no primeiro grupo.

A maior dificuldade, porém, não reside no elevado tempo de resposta, mas na sua imprevisibilidade. Em se tratando de uma rede tão heterogênea, a qualidade do canal de comunicação varia demasiadamente, tanto de ponto para ponto como em função do tempo. Assim, embora a camada de transporte seja capaz de oferecer um serviço satisfatório para aplicações não críticas, as variações nos atrasos de transmissão oscilam de frações de segundo até vários segundos, tornando praticamente inviável determinar uma faixa de operação razoável, se a constância do tempo de resposta for importante — é esperado que o panorama melhore com o rápido progresso na área das redes de computadores e com novas tecnologias de transmissão de dados em desenvolvimento. Em particular, uma vez que a operação de sistemas distribuídos de tempo-real dependem da comunicação também em tempo-real, sistemas que habilitem essa modalidade de transmissão sobre a Internet devem ser investigados — é importante, porém, manter a compatibilidade com padrões existentes, especialmente com padrões abertos, a fim de permitir a máxima interoperabilidade e propiciar o desenvolvimento de pesquisas nesse campo.

Face ao acelerado surgimento de novos padrões na esfera da telecomunicação digital, é temerário prever quais dentre eles tornar-se-ão efetivos a longo e mesmo a curto prazo. Todavia, vale notar que em destaque, hoje, encontram-se os canais de alta velocidade através de fibra ótica e sistemas ATM, indicando uma evolução das WANs em direção a um cenário de redes metropolitanas conectadas por linhas de elevada capacidade [97]. Atendida essa urgência na esfera da comunicação, a tecnologia dos dispositivos telemáticos

e a integração de aplicações via Internet estão prontos para impulsionar grandes avanços nessa frente.

Finalmente, a confiabilidade constitui definitivamente o problema mais grave. Na literatura, esse parâmetro é usualmente descrito em termos de quatro medidas: $R(\delta t)$ (*reliability*), probabilidade de que o sistema esteja provendo o serviço correto depois de um intervalo de tempo δt ; $S(\delta t)$ (*safety*), probabilidade de que o sistema não falhará catastróficamente dentro de um intervalo de tempo δt ; $A(t)$ (*availability*), probabilidade de que o sistema esteja pronto a prover o serviço no instante t ; $M(t)$ (*maintainability*), probabilidade de que o sistema esteja restabelecido no instante t após haver falhado em $t = 0$. Um quinto atributo, a segurança, diz respeito à habilidade do sistema em prevenir o acesso não autorizado [53].

Infelizmente, em nenhum deles a Internet tem reconhecida excelência; longe disso, a própria natureza da diversidade de tecnologias presentes em sua construção torna difícil garantir níveis de qualidade em âmbito global. Entretanto, se não é possível ainda vislumbrar uma rede mundial homogênea em termos de confiabilidade de serviço, em escala menor pode-se ao menos apontar para possíveis caminhos. Tendo em vista a criação de redes de dispositivos telemáticos para uso restrito entre instituições consorciadas, por exemplo, a instalação e manutenção de linhas de comunicação dedicadas é plenamente viável. Como exemplo particularmente interessante, investimentos nessa direção podem permitir a universidades e centros de pesquisa criarem infra-estruturas para o compartilhamento de recursos científicos, didáticos e de instrumentação.

8.2 Síntese das Contribuições

Com base nas considerações anotadas, é possível sintetizar as contribuições propostas no trabalho através das seguintes conclusões principais:

- No que concerne à viabilidade das redes de dispositivos telemáticos, e tendo como estudo-de-caso o compartilhamento da funcionalidade através de redes de longa distância, constata-se que, não obstante ainda precária em diversas qualidades, a tec-

nologia atualmente disponível já permite sua utilização efetiva em algumas aplicações simples. Torna-se, portanto, bastante encorajadora a exploração das potencialidades por ela habilitadas, não somente buscando o benefício imediato que já é possível, mas também a fim de desenvolver o conhecimento na emergente área.

- Em termos práticos, o principal obstáculo técnico à criação de sistemas de compartilhamento de dispositivos telemáticos, no momento, são as insuficientes qualidades da infra-estrutura de comunicação disponíveis, ao exemplo da Internet mundial, especialmente no que se refere ao desempenho e confiabilidade. No primeiro aspecto destaca-se a dificuldade para o desenvolvimento de aplicações de tempo-real, resultando daí a importância de pesquisas nesse terreno — convenientemente, vale notar que os rumos e o ritmo da evolução das novas tecnologias de comunicação digital apontam para um futuro bastante promissor. O mesmo vale para a questão da confiabilidade onde, além do fator tecnológico, cabem ações práticas como investimentos em sistemas de comunicação dedicados e políticas de gestão de recursos telemáticos, visando garantir níveis de qualidade satisfatórios.
- Ainda que desenvolvendo-se em um cenário sujeito a diversas dificuldades do ponto de vista da tecnologia atual, esta pesquisa antecipa-se ao advento de uma Internet avançada, tal qual pretendem consolidar os esforços em pauta na definição da próxima geração das tecnologias de transmissão e comunicação. Sob essa perspectiva, as observações e conclusões trazidas neste trabalho contribuem com subsídios para orientar o projeto de novos recursos visando uma rede capaz de viabilizar a aplicação em foco.

Assim, enquanto a evolução tecnológica não alcança plenamente as expectativas, é ainda assim possível, ao exemplo do que já principiara nos EUA e Europa, iniciar a implantação de sistemas-piloto para promover pesquisas nesse campo e contribuir para a efetiva exploração da tecnologia das redes de dispositivos telemáticos em compartilhamento de recursos também no Brasil.

Referências Bibliográficas

- [1] J. Vertut and P. Coiffet. *Teleoperation and Robotics Evolution and Development*, volume 3A of *Robot Technology*. Kogan Page Ltd, London, 1985.
- [2] M. W. Thring. Telechiric mining. In *2nd RO.MAN.SY*, pg. 459, Warsaw, 1976.
- [3] TERMITE. Terminology of Telecommunications, International Telecommunication Union - ITU, April 1993. <http://www.itu.int/search/wais/Termite/index.html>.
- [4] E. G. Johnsen and W. R. Corliss. Teleoperator and human augmentation. Technical Report SP5047, NASA Report, 1967.
- [5] H Kopetz. The production of intelligent products in developing countries. *Microelectronics Monitor*, (29):63–71, 1190.
- [6] Coke CSMU. Cs/csmu's coke machine. Computer Science Department, Carnegie Mellon University, 1998. <http://www.cs.cmu.edu/coke/>.
- [7] B. Zimmerman. Request for comments rfc-1288 - the finger user information protocol. RFC Editor, Internet Engineering Task Force - IETF, 1991. <http://www.ietf.org/rfc/rfc1288.txt>.
- [8] Coke RIT. Rit coke machine. Rochester Institute Technology, 1998. *finger* drink@csh.rit.edu.
- [9] Coke DIKU. Diku coke machine. Computer Science Department, University of Copenhagen, 1997. <http://amigos12.diku.dk/>.

- [10] Coke MRU. Acm/mru coke machine. Association for Computer Machinery, University of Missouri-Rolla, 1997. <http://acm.cs.umr.edu/soda/>.
- [11] Coke UWA. Cc/uwa's coke machine. Computer Club, University of Western Australia, 1997. <http://www.ucc.gu.uwa.edu.au/services/drink.html>.
- [12] Coke Caltech. Caltech's coke machine. California Institute of Technology, 1996. <http://www.ugcs.caltech.edu/walterfb/coke/coke.htm>.
- [13] Coke MIT. Mit's coke server. Massachusetts Institute of Technology, 1995. <http://www-swiss.ai.mit.edu/htbin/coke>.
- [14] Coke Columbia. Columbia's coke server. Columbia University, 1996. <http://www.cc.columbia.edu/cocacola/>.
- [15] M. Johnson. Cambridge university's internet tojan room coffee machine. Cambridge University, Trojan Room, 1994. <http://www.cl.cam.ac.uk/coffee/coffee.html>.
- [16] H. Hahn and R. Stout. *Dominando a Internet*. McGraw-Hill Ltda, São Paulo, Brasil, 1995.
- [17] J. Paris. The randon hall laundry server. Massachusetts Institute of Technology, 1996.
- [18] K. Goldberg, M. Mascha, S. Genter, N. Rothenberg, C. Sutter, and J. Wiegley. Beyond the web: Excavating the real world via mosaic - the Mercury project. In *Second International WWW Conference*, Chicago, USA, Oct. 1994.
- [19] K. Goldberg, M. Mascha, S. Genter, and N. Rothenberg. Desktop teleoperation via world wide web. In *1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan, 1995.
- [20] K. Goldberg and J. Santaromana. The telegarden. Ars Electronica Center, Austria, 1996. <http://telegarden.aec.at>.
- [21] B. R. Farzin and K. Goldberg. A minimalist telerobotic installation on the internet. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, Victoria, B. C. Canada, 12-17 Oct. 1998.

- [22] S. B. Goldberg, G. A. Bekey, Y. Akatsuka, and N. Bressanelli. Digimuse: an interactive telerobotic system for remote viewing of three-dimensional art objects. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, Victoria, B. C. Canada, 12-17 Oct. 1998.
- [23] M. J. Cox and J. E. F. Baruch. Robotic telescopes: An interactive exhibit on the world wide web. In *2nd International Conference of the World Wide Web*, Chicago, USA, Oct. 1994.
- [24] K. Taylor and J. Trevelyan. A telerobot on the world wide web. In *Robots for Australian Industries Conference*, pg. 108–120, Melbourn, Australia, Jul. 1995. Australia Robot Association.
- [25] K. Taylor and J. Trevelyan. Australia's telerobot on the web. In *International Symposium on Industrial Robots*, Singapore, Oct. 1995.
- [26] J. E. Borelli, F. J. Monaco, E. A. Neves, and A. Gonzaga. Aplicação do telediagnóstico para a investigação do estado patológico e monitoramento remoto de indivíduos portadores de desabilidades. In *III Congresso Iberoamericano de Comunicación Alternativa Y Aumentativa, I Congresso Iberoamericano de Tecnologías de Apoyo para la Discapacidad*, pg. 435–437, Madrid, 18-20 Out. 2000.
- [27] F. J. Monaco and A. Gonzaga. Toward distributed cooperative telemanufacturing over the internet. In *15th International Conference on CAD/CAM, Robotics & Factories of the Future*, pg. MT7–17, 1999.
- [28] B. Mason-Fossum and U. Thakkar. Primary school classroom and chickscope: Studying the egg in the classroom and using the internet. In *3rd Conference on Human Factors & Web*, Denver, Colorado, 12 jun 1997. US WEST Communications. Electronic proceedings at <http://www.uswest.com/web-conference/>.
- [29] C. Bohus, L. Crowl, B. A. Aktan, and M. H. Shor. Running control engineering experiments over the internet. In *IFAC 13th Trienal World Congress. vol. G*, pg. 25–33, San Francisco, California, Jun. 1996.

- [30] M. L. Ruz and V. F. Batlle. Low cost laboratory equipment for distance education of automatic control. In *IFAC 13th Trienal World Congress*, pg. 19–24, San Francisco, California, Jun. 1996.
- [31] ORNL. ORNL's telepresence connects researchers with remote microscopes. Oak Ridge National Laboratory, Press Release, 1996. http://www.ornl.gov/Press_Release/archive/960627-01.html.
- [32] J. M. Price. The emsl on-line (eol) project. Environmental Molecular Sciences Laboratory, 1996. <http://www.emsl.pnl.gov:2080/docs/emsl-online/index.htm>.
- [33] Shrikumar. Ipic web server. Real-Time & Database Systems Lab, University of Massachusetts, 1999. <http://www-ccs.cs.umass.edu/shri/iPic.html>.
- [34] R. Braden. Request for comments rfc-1122 - requirements for internet hosts – communication layers. RFC Editor, Internet Engineering Task Force - IETF, 1989. <http://www.ietf.org/rfc/rfc1122.txt>.
- [35] Mycal Labs. uwebserver based on iready technology. <http://www.mycal.net/wsweb>.
- [36] Matchbox webserver. Wearable Labs, Stanford University, 1999. <http://wearables.stanford.edu/hardware.html#webserver>.
- [37] RFC-854. Request for comments - RFC-854 - telnet protocol specification. RFC Editor, Internet Engineering Task Force - IETF, 1983. <ftp://ftp.isi.edu/in-notes/rfc854.txt>.
- [38] T. Berners-Lee, R. Cailliau, J-F. Groff, and B. Pollerman. World-wilde web: the information universe. In *Electronic Networking: Research, Applications and Policy*, volume 1, West Port, CT, 1992.
- [39] K. Taylor and B. Dalton. A new robotics nice. *IEEE Robotics & Automation Magazine*, 7(1):27, Mar. 2000.
- [40] R. Siegwart, C. Wannaz, P. Garcia, and R. Blank. Guiding mobile robots through the web. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, Victoria, B. C. Canada, 12-17 Oct. 1998.

- [41] F. Monaco and A. Gonzaga. Web-based system paradigms and the conceptual design of network applications. In *I Workshop on Web Engineering, 21th International Conference on Software Engineering*, Los Angeles, USA, 16-22 May 1999.
- [42] P. Saucy and F. Mondada. Khepontheweb: one year of access to a mobile robot on the internet. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, Victoria, B. C. Canada, 12-17 Oct. 1998.
- [43] D. Schulz, W. Burgard, and A. B. Cremers. Predictive simulation of autonomous robots for tele-operation systems using the world wide web. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, Victoria, B. C. Canada, 12-17 Oct. 1998.
- [44] J. G. Keramas. How will a robot change your life. *IEEE Robotics & Automation Magazine*, 7(1):27, Mar. 2000.
- [45] M. Stein. Painting on the world wide web: the pumapainting project. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, pg. 37-42, Victoria, B. C. Canada, 12-17 Oct. 1998.
- [46] J. Myers and M. Rose. Request for comments rfc-1939 - post office protocol - version 3. RFC Editor, Internet Engineering Task Force - IETF, 1996. <http://www.ietf.org/rfc/rfc1939.txt>.
- [47] J. Reynolds and J. Postel. Request for comments rfc-959 - file transfer protocol. RFC Editor, Internet Engineering Task Force - IETF, 1985. <http://www.ietf.org/rfc/rfc959.txt>.
- [48] w3c. The world wide web consortium, 1998. <http://www.w3.org/MarkUp/>.
- [49] T. Berners-Lee. Request for comments rfc-2396 - universal resource identifiers in www: A unifying syntax for the expression of names and addresses of objects on the network as used in the world-wide web. RFC Editor, Internet Engineering Task Force - IETF, 1994. <http://www.ietf.org/rfc/rfc2396.txt>.

- [50] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Request for comments rfc-2616 - hypertext transfer protocol - http/1.1. RFC Editor, Internet Engineering Task Force - IETF, 1999. <http://www.ietf.org/rfc/rfc2616.txt>.
- [51] WITS. The web interface for telescience. Jet Propulsion Lab NASA / Graham Technology Solutions, 2000. <http://mars.graham.com/wits>.
- [52] A. S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, 1992.
- [53] H. Kopetz and P. Veríssimo. *Distributed Systems*, chapter Real Time and Dependability Concepts. ACM Press, 2nd edition, 1993.
- [54] B. Dalton and K. Taylor. A framework for internet robots. In *Workshop Robots on the Web, IEEE/RSJ International Conference on Inteligente Robots and Systems*, pg. 37-42, Victoria, B. C. Canada, 12-17 Oct. 1998.
- [55] OMG. Welcoma to omg's corba website. Object Management Group, Inc., 2000. <http://www.corba.org>.
- [56] Sun Microsystems. Java technology. Sun Microsystems, Inc., 2001. <http://www.sun.com/java>.
- [57] C. Anderson, A. Brandit, H Hernandez, and M. Shamma. Message oriented middleware. School of Information Management and System, University of California at Berkeley, 2000. <http://sims.berkeley.edu/courses/is206/f97/GroupB/mom>.
- [58] Sun Microsystems. Jini network technology. Sun Microsystems, Inc., 2001. <http://www.sun.com/jini>.
- [59] F. J. Monaco and A. Gonzaga. Remote device command and resource sharing over the internet: a new approach based on a distributed layered architecture. *IEEE Transactions on Computer*, 2002. no prelo.
- [60] P. Laplante. *Real-Time Systems Design and Analysis*. IEEE Press, 1992.
- [61] P. Chan and R. Lee. *The Java Class Libraries*. Addison-Wesley, USA, 1996.

- [62] J. Gosling, Joy B., and G. Steele. *The Java Language Specification*. Addison-Wesley, USA, 1996.
- [63] L. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, USA, 1996.
- [64] K. Arnold and J. Gosling. *Programando em Java*. Makron Books, USA, 1997. Java-Soft, Sun Microsystems, Inc.
- [65] A. S. Tanenbaum. *Redes de Computadores*. Editora Campus, 1994.
- [66] Ö. Babaoglu and K. Marzullo. *Distributed Systems*, chapter Real Time and Dependability Concepts. ACM Press, 2nd edition, 1993.
- [67] S. Mullender. *Distributed Systems*, chapter Kernel Support for Distributed Systems. ACM Press, 2nd edition, 1993.
- [68] F. Schneider. *Distributed Systems*, chapter Replication Management Using State-Machine Approach. ACM Press, 2nd edition, 1993.
- [69] B.M. Leiner, V.G. Cerf, D.D. Clark, et al. A brief history of the internet, vr. 3.1. Internet Histories, Internet Society - ISOC, 1998. <http://www.isoc.org/internet-history/brief.html>.
- [70] W.R Stevens. *TCP/IP illustrated*, volume 1. Addison-Wesley Publishing, USA, 1994.
- [71] J. B. Postel. Request for comments rfc-791 - internet protocol. RFC Editor, Internet Engineering Task Force - IETF, 1981. <http://www.ietf.org/rfc/rfc791.txt>.
- [72] S. E. Dering and R. Hiden. Request for comments rfc-1883 - internet protocol, version 6 (ipv6) specification. RFC Editor, Internet Engineering Task Force - IETF, 1995. <http://www.ietf.org/rfc/rfc1883.txt>.
- [73] J. B. Postel. Request for comments rfc-768 - user datagram protocol. RFC Editor, Internet Engineering Task Force - IETF, 1980. <http://www.ietf.org/rfc/rfc768.txt>.
- [74] J. B. Postel. Request for comments rfc-793 - transmission control protocol. RFC Editor, Internet Engineering Task Force - IETF, 1981. <http://www.ietf.org/rfc/rfc793.txt>.

- [75] J. Postel. Request for comments rfc-821 - simple mail transfer protocol. RFC Editor, Internet Engineering Task Force - IETF, 1982. <http://www.ietf.org/rfc/rfc821.txt>.
- [76] T. Berners-Lee, R. Fielding, and H. Frystyk. Request for comments rfc-1945 - hypertext transfer protocol – http/1.0. RFC Editor, Internet Engineering Task Force - IETF, 1996. <http://www.ietf.org/rfc/rfc1945.txt>.
- [77] J. Tranter. *Linux Multimedia Guide*. O'Reilly & Associates, Inc., USA, 1996.
- [78] G. Hankins. Linux serial howto. Linux on-line documentation, 1997. <ftp://sunsite.unc.edu:/pub/Linux/docs/HOWTO/Serial-HOWTO>.
- [79] P. H. Baumann. Linux serial-programing howto. Linux on-line documentation, 1997. <http://sunsite.unc.edu/LDP/HOWTO/Serial-Programming-HOWTO.html>.
- [80] S. Loosemore, R. Stallman, R. McGrath, A. Oram, and U. Drepper. *The GNU C Library Reference Manual*. FSF, 1999. ISBN: 1-882114-54-X.
- [81] advantech. Andvantech co. ltda. Home Page, 1998. <http://www.advantech.com>.
- [82] E. G. Coffman, M. J. Elphick, and A. Soshani. System deadlocks. *Computing Surveys*, 3:67–78, jun 1971.
- [83] W. Weihl. *Distributed Systems*, chapter Transaction-Processing Techniques. ACM Press, 2nd edition, 1993.
- [84] Debian. Debian project, 2001. <http://www.debian.org>.
- [85] Linux. The linux home page at linux online. Linux Org., 2001. <http://www.linux.org>.
- [86] GPL. The gnu project and the free software foundation. Free Software Foundation, 1991. <http://www.gnu.org>.
- [87] GNU. The gnu c library, 2001. <http://www.glibc.org>.
- [88] Gcc home page. The GNU Project and the Free Software Foundation, 2001. <http://www.gnu.org/software/gcc/gcc.html>.

- [89] CVS. Cvs - the concurrent version system. CVS Home, 2001.
<http://www.cvshome.org>.
- [90] GPL. Gnu general public licence. Free Software Foundation, 1991.
<http://www.gnu.org/copyleft/gpl.html>.
- [91] W. R. Stevens. *Unix Network Programming*, volume 1. Prentice Hall, Inc., USA, 2nd edition, 1998.
- [92] A. Burns. Distributed hard real-time systems: What restrictions are necessary ? In H. Zedan, editor, *Proceedings of the 1989 Real-Time Symposium*. Elsevier Scientific, 1989.
- [93] *GNU make, A Program for Directing Recompilation*. Free Software Foundation Inc., USA. ISBN 1882114-81-7.
- [94] J. Reynolds and J. Postel. Request for comments rfc-1700 - assigned numbers. RFC Editor, Internet Engineering Task Force - IETF, 1994.
<http://www.ietf.org/rfc/rfc1700.txt>.
- [95] V.G. Cerf et al. *National Collaboratories: Applying Information Technology for Scientific Research*. National Academic Press, Washington DC, USA, 1993.
- [96] R.T. Kouzes, J.D. Myers, and W.A. Wulf. Collaboratories: Doing science on the internet. *IEEE Computer*, 29:40-46, oct 1996.
- [97] P. Veríssimo. *Distributed Systems*, chapter Real Time Communication. ACM Press, 2nd edition, 1993.