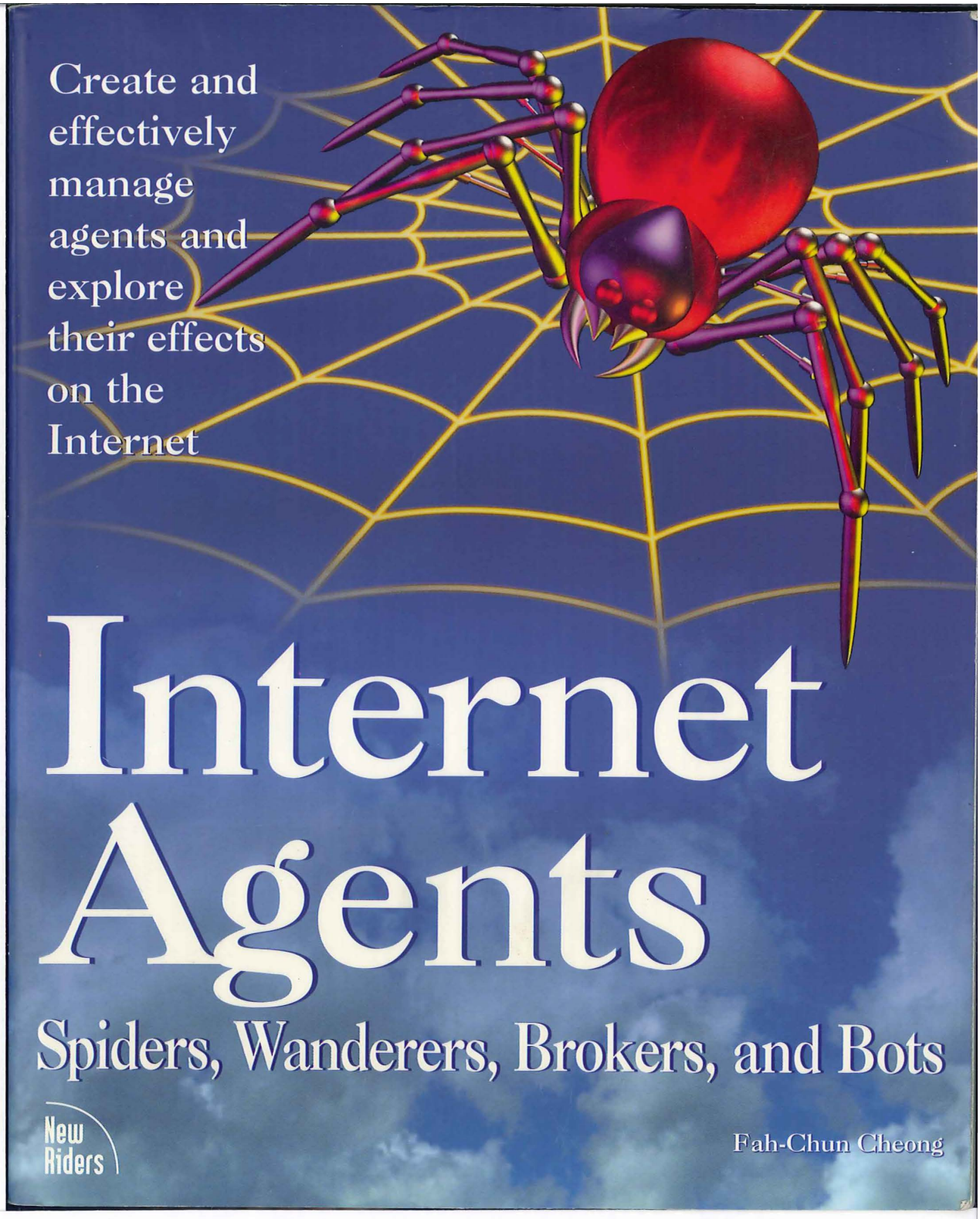
Internet Agents: Spiders, Wanderers, Brokers, and Bots

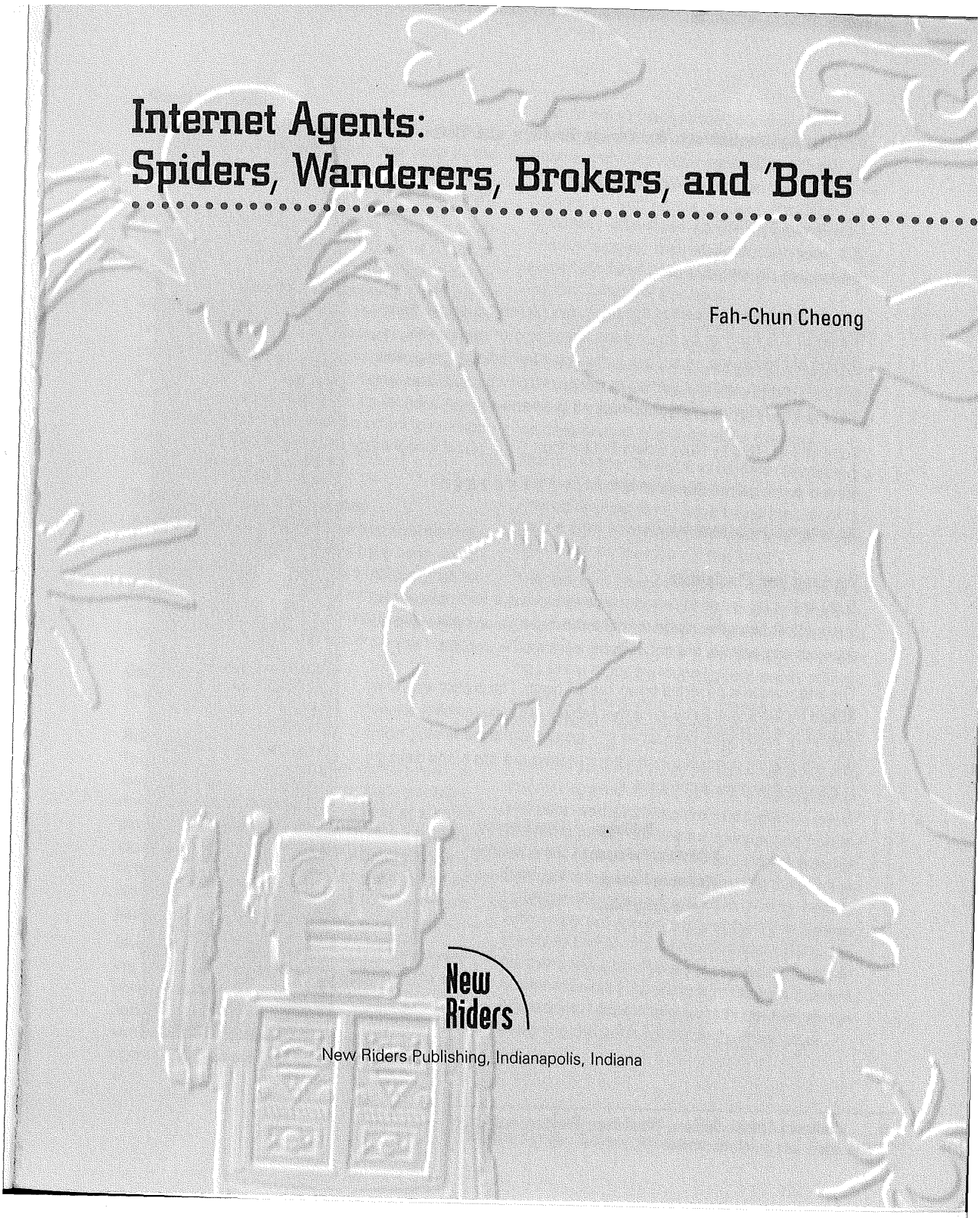
New Riders



IPR2025-00714
DATABRICKS EX1008 Page 2

Internet Agents: Spiders, Wanderers, Brokers, and 'Bots

Fah-Chun Cheong



New Riders

New Riders Publishing, Indianapolis, Indiana

IPR2025-00714
DATABRICKS EX1008 Page 3

Internet Agents: Spiders, Wanderers, Brokers, and 'Bots
By Fah-Chun Cheong

Published by:
New Riders Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

Copyright © 1996 by New Riders Publishing

Printed in the United States of America 1 2 3 4 5 6 7 8 9 0

CIP Data Available upon Request

Warning and Disclaimer

This book is designed to provide information about Internet agents. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an "as is" basis. The author and New Riders Publishing shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the disks or programs that may accompany it.

Publisher	Don Fowley
Publishing Manager	Jim LeValley
Marketing Manager	Ray Robinson
Managing Editor	Tad Ringo

Product Development Specialist
Julie Fairweather
Development Editor
Suzanne Snyder
Production Editor
Cliff Shubs
Copy Editors
Amy Bezek, Fran Blauw, Gail Burlakoff, Laura Frey, Lisa Wilson
Associate Marketing Manager
Tamara Apple
Acquisitions Coordinator
Tracy Turgeson
Publisher's Assistant
Karen Opal
Cover Designer
Jay Corpus
Cover Illustrator
Roger Morgan
Book Designer
Sandra Schroeder
Manufacturing Coordinator
Paul Gilchrist
Production Manager
Kelly D. Dobbs
Production Team Supervisor
Laurie Casey
Graphics Image Specialists
Jason Hand, Clint Lahnen, Laura Robbins, Craig Small, Todd Wente
Production Analysts
Angela D. Bannan
Bobbi Satterfield
Production Team
Heather Butler, Dan Caparo, Kim Cofer, Kevin Foltz, Erika Millen, Erich J. Richter, Christine Tyner, Karen Walsh
Indexer
Christopher Cleveland

IPR2025-00714
DATABRICKS EX1008 Page 4

About the Author

Fah-Chun Cheong consults with start-up companies around the San Francisco Bay Area in the application of agent technologies for electronic commerce on the World Wide Web and Internet.

Mr. Cheong received his B.S. in Electrical Engineering from The University of Texas at Austin in 1986, and his M.S. and Ph.D. degrees in Computer Science from the University of Michigan in 1988 and 1992, respectively. His Ph.D. research work is on the design and development of an experimental agent-oriented programming language and compiler system for heterogeneous distributed computing environments. He founded Agent Computing, Inc. in 1994, with a vision to develop innovative application-specific agent technologies for the Internet.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. New Riders Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Dedication

To my parents and sisters

Acknowledgments

This book might not have been written (well, at least not in 1995!) if Vinay Kumar had not invited me along to a dinner earlier this year at a sushi place in San Francisco with Jim LeValley, Publishing Manager for New Riders Publishing. I thank him for that and for the many interesting and insightful discussions on a variety of topics we have had over many cups of espresso.

A very big thank you to Kevin Hughes for reviewing drafts of this book. I am grateful to ex-colleagues at EIT and ex-EIT friends, especially Jeff Pan and Jim McGuire, for information in a variety of areas, most notably procurement agents, Web robots, and secure HTTP.

I would like to thank all the people on the Internet whose pioneering work in agents, spiders, wanderers, and Web robots has made an early book on this topic a possibility. Special thanks to all the authors of Web robots, spiders, and wanderers who have answered e-mail questionnaires on Internet agents; their insightful comments and responses have contributed much toward shaping the content of this book.

I am indebted to Roy Fielding for his libwww-perl and MOMspider source code, which, in a vastly simplified form, have now become the basis upon which WebWalker is built. Many thanks to Bruce Krulwich whose BargainFinder agent on the Web inspired the development of WebShopper for this book.

Martijn Koster has authored and maintained a number of marvelous Web pages on the net. Among his creations, I have found the List of Robots a comprehensive reference and an invaluable resource for much of this book.

The Stanford Libraries have proved invaluable to me on this project, as on others. I am extremely grateful that Stanford opens its Mathematics and Computer Science Library, and also the Engineering Library, to the surrounding community at large.

A very big thank you to the friendly, competent, and generally fantastic editorial staff at New Riders who prepared this book for publication. I am indebted to Jim LeValley for taking an interest in Internet agents, coming up with an initial plan for this book, and supplying me continuously with an unending stream of helpful sources and materials. I am especially thankful to Julie Fairweather for developing the book, coordinating the process to keep publication on schedule, and for helping with numerous screen-shots of the Web. Special thanks to Cliff Shubs for his excellent editing and his many thoughtful remarks on the book, and to Suzanne Snyder for helping with the development of the book. Many thanks go to Roger Morgan for designing the great spider on the front cover.

IPR2025-00714
DATABRICKS EX1008 Page 5

Contents at a Glance

Part I: Introduction

1 The World of Agents 3
2 The Internet: Past, Present, and Future 37
3 World Wide Web: Playground for Robots 61

Part II: Web Robot Construction

4 Spiders for Indexing the Web 81
5 Web Robots: Operational Guidelines 105
6 HTTP: Protocol of Web Robots 125
7 WebWalker: Your Web Maintenance Robot 153

Part III: Agents and Money on the Net

8 Web Transaction Security 185
9 Electronic Cash and Payment Services 205

Part IV: Bots in Cyberspace

10 Worms and Viruses 229
11 MUD Agents and Chatterbots 249

Part V: Appendices

A HTTP 1.0 Protocol Specifications 283
B WebWalker 1.00 Program Listing 293
C WebShopper 1.00 Program Listing 337
D List of Online Bookstores Visited by BookFinder 347
E List of Online Music Stores Visited by CDFinder 351
F List of Active MUD Sites on the Internet 355
G List of World Wide Web Spiders and Robots 375
Bibliography 387
Index 401

IPR2025-00714
DATABRICKS EX1008 Page 6

Table of Contents

Part I: Introduction 1

1 The World of Agents 3

What are Agents? ... 5
Agents and Delegation ... 6
Personal Assistants ... 6
Envoy Desktop Agents ... 8
New Wave Desktop Agents ... 8
Surrogate Bots ... 9
Internet Softbots ... 10
Agents and Coordination ... 12
Conference-Support Agents ... 12
Integrated Agents ... 13
Communicative Agents ... 15
Agents and Knowledge ... 17
Teaching Agents ... 17
Learning Agents ... 19
Common-Sense Agents ... 21
Physical Agents ... 23
Agents and Creativity ... 24
Creative Agents ... 24
Automated Design Agents ... 27
Agents and Emotion ... 27
Art of Animation ... 28
Artificial Intelligence ... 28
The Oz Project ... 28
Agents and Programming ... 30
KidSim ... 30
Oasis ... 31
Agents and Society ... 32
Control ... 33
Over Expectations ... 33
Safety ... 33
Privacy ... 33
Commercial Future of Agents ... 33
Product Suites ... 34
Mobile Computing ... 34
Concluding Remarks ... 35

IPR2025-00714
DATABRICKS EX1008 Page 7

2 The Internet: Past, Present, and Future **37**

Early Days of ARPAnet ... 38
Notable Computer Networks ... 39
Internet and NSFnet ... 41
NSF and AUP ... 42
Growth of the Internet ... 42
How Big is the Internet? ... 45
Internet Society, IAB, and IETF ... 55
Information Superhighway and the National Information Infrastructure ... 57

3 World Wide Web: Playground for Robots **61**

World Wide Web Development ... 62
Growth of the Web ... 62
Information Dissemination with the Web ... 62
Innovative Uses of the Web ... 65
Architecture of the World Wide Web ... 65
Web Clients ... 66
Web Servers ... 66
Web Proxies ... 67
Web Resource Naming, Protocols, and Formats ... 67
URI and URL: Universal Resource Identifier and Locator ... 67
Common URI Syntax ... 68
URLs for Various Protocols ... 69
Gopher and WAIS ... 69
HTTP: HyperText Transfer Protocol ... 69
Statelessness in HTTP ... 70
Format Negotiations ... 70
HTML: HyperText Markup Language ... 71
Level of HTML Conformance ... 71
HTML Tags ... 72
Forms and Imagemaps: Enhanced Web Interactivity ... 73
Fill-Out Forms ... 73
Clickable Images ... 73
Gateway Programming: Processing Client Input ... 74
Gateway Program Interaction ... 75
The Next Step: Agents on the Web ... 76
Early Commerce Agents ... 76
Web Agents of the Future? ... 78

Part II: Web Robot Construction **79**

4 Spiders for Indexing the Web **81**

Web Indexing Spiders ... 82
WebCrawler: Finding What People Want ... 84

IPR2025-00714
DATABRICKS EX1008 Page 8

Searching with WebCrawler 84
How WebCrawler Moves in Webspace 85
Lycos: Hunting WWW Information 89
Searching with Lycos 90
Lycos' Search Space 91
Lycos Indexing 92
How Lycos Moves in Webspace 92
Harvest: Gathering and Brokering Information 93
Searching with Harvest 94
Harvest Architecture 95
WebAnts: Hunting in Packs 99
WebAnts Motivation 100
WebAnts Searching and Indexing 100
Issues of Web Indexing 100
Recall and Precision 101
Good Web Citizenship 101
Performance 102
Scalability 102
Spiders of the Future 103

5 Web Robots: Operational Guidelines 105

Web Robot Uses 106
Web Resource Discovery 107
Web Maintenance 107
Web Mirroring 107
Proposed Standard for Robot Exclusion 108
Robot Exclusion Method 108
Robot Exclusion File Format 109
Recognized Field Names 109
Sample Robot Exclusion Files 110
The Four Laws of Web Robotics 110
I. A Web Robot Must Show Identifications 111
II. A Web Robot Must Obey Exclusion Standard 112
III. A Web Robot Must Not Hog Resources 113
IV. A Web Robot Must Report Errors 115
The Six Commandments for Robot Operators 115
I. Thou Shalt Announce thy Robot 116
II. Thou Shalt Test, Test, and Test thy Robot Locally 117
III. Thou Shalt Keep thy Robot Under Control 118
IV. Thou Shalt Stay in Contact with the World 119
V. Thou Shalt Respect the Wishes of Webmasters 119
VI. Thou Shalt Share Results with thy Neighbors 120
Robot Tips for Webmasters 121
Web Ethics 122

IPR2025-00714
DATABRICKS EX1008 Page 9

6 HTTP: Protocol of Web Robots 125

Understanding HTTP Operation 126
Messaging with HTTP 128
Message Headers 128
General Message Header Fields 129
Request Message 130
Method 130
Request Header Fields 133
Response Message 136
Status Codes and Reason Phrases 137
Response Header Fields 140
Entity 141
Entity Header Fields 141
Entity Body 146
Protocol Parameters 147
HTTP Version 147
Universal Resource Identifiers 147
Date/Time Formats 147
Content Parameters 148
Media Types 148
Character Sets 148
Encoding Mechanisms 149
Transfer Encodings 149
Language Tags 150
Content Negotiation 150
Access Authentication 151

7 WebWalker: Your Web Maintenance Robot 153

The Web Maintenance Problem 154
Web Infostructure 154
Past Approaches 154
Web Maintenance Spiders 155
WebWalker Operation 156
Processing Task Descriptions 156
Avoiding and Excluding URLs 156
Keeping History 157
Traversing the Web 157
Generating Reports 157
Is WebWalker a Good Robot? 157
WebWalker Limitations 158
WebWalker Program Installation 158
WebWalker Task File 159
Global Directives 159
Task Directives 160
Task File Format 161

IPR2025-00714
DATABRICKS EX1008 Page 10

WebWalker Usage Examples 161
Sample WebWalker Output 162
WebWalker Forms Interface 167
WebWalker Program Organization 169
External Library Calls 169
WebWalker Program Call-Graph 170
Configuration Section 172
Avoidance Package 174
History Package 175
Traversal Package 177
Summary Package 178
Growing into the Future 181

Part III: Agents and Money on the Net **183**

8 Web Transaction Security **185**

Concepts of Security 186
Privacy: Keeping Private Messages Private 187
Authentication: Proving You Are Who You Claim to Be 188
Integrity: Ensuring Message Content Remains Unaltered 189
Brief Tour of Classical Cryptography 189
The Role of NSA 190
Development of Data Encryption Standard (DES) 190
Development of Public-Key Cryptography 191
Problems with Secret Keys 191
Key Management 192
The RSA Alternative 192
Comparing Secret-Key and Public-Key Cryptography 193
Digital Signatures 194
How Digital Signatures Work 194
The Digital Signature Standard 197
Key Certification 197
Certifying Authority 197
Certificate Format 198
Two Approaches to Web Security 198
Secure Socket Layer (SSL) 200
Secure HTTP (S-HTTP) 201
Current Practice and Future Trend in Web Security 203

9 Electronic Cash and Payment Services **205**

Brief History of Money 206
Choice of Payment Methods 207
What is Digital Cash? 207
Digital Cashier's Check 208
Anonymous Digital Cash through Blind Signatures 210

IPR2025-00714
DATABRICKS EX1008 Page 11

Ecash from DigiCash 211
Ecash Security and Other Issues 213
Payment Systems on the Internet 214
U.S. Payment Systems Today 214
CyberCash Internet Payment Service 215
Information Commerce on the Internet 220
Economics of Information Commerce 220
First Virtual Payment System 222
The Future 225

Part IV: Bots in Cyberspace **227**

10 Worms and Viruses **229**

Short History of Worms 230
The First Worm 230
The Christmas Tree Worm 232
The Internet Worm 232
Anatomy of the Internet Worm 232
Method of Worm Attack 232
Method of Worm Defense 233
What Does the Worm Not Do? 234
Brief History of Viruses 234
Types of Viruses 236
Boot-Sector Infectors 236
File Infectors 236
PC Virus Basics 237
Viral Activation in the Boot Process 238
Step One: ROM BIOS Routines Execution 238
Step Two: Partition Record Code Execution 238
Step Three: Boot-Sector Code Execution 238
Step Four: IO.SYS and MSDOS.SYS System Code Execution 239
Step Five: COMMAND.COM Shell Execution 240
Step Six: AUTOEXEC.BAT Batch File Execution 240
Viral Activation During Normal Operation 240
Viral Replication 241
Viral Reinfection 241
Symptoms of Viral Infection 241
Major IBM PC Viruses 242
Pakistani Brain Virus 242
Friday the 13th Virus 243
Lehigh Virus 243
Dark Avenger 243
Michelangelo Virus 244
Stealth Techniques 244

IPR2025-00714
DATABRICKS EX1008 Page 12

Advanced Viral Techniques 245
Encrypted Virus 245
Multi-Encrypted Virus 246
Instructions Rescheduling 246
Mutation Engine 246
Armored Virus 246
Worms and Viruses Summarized 247

11 MUD Agents and Chatterbots 249

The Turing Test 250
Eliza: The Mother of All Chatterbots 251
A Conversation with Eliza 252
Eliza Internals 252
Parry: The Artificial Paranoia Agent 253
An Interview with Parry 254
Distinguishing Parry from Human Patients 255
MUDs: Virtual Worlds on the Internet 255
Inside MUDs 256
Sample MUD Interactions 256
TinyMUDs: Virtual Communities on the Internet 266
Sample TinyMUD Interactions 267
Social Interactions on TinyMUDs 268
Colin: The Prototypical MUD Agent 269
Colin's Information Services 270
Colin's Mapping Services 271
Colin's Miscellaneous Services 273
Julia: Chatterbot with a Personality 273
An Interaction with Julia 274
Inside Julia 274
The Loebner Prize Competition 275
Julia in the Loebner Prize Competition 276
CHAT: A Knowledgeable Chatterbot 278
Tricks of the Chatterbots 278
Eliza's Tricks 278
Parry's Tricks 279
Julia's Tricks 279
Closing Words 279

Part V: Appendices 281

A HTTP 1.0 Protocol Specifications 283

Notational Conventions 284
Augmented BNF 284
Basic Rules 285

IPR2025-00714
DATABRICKS EX1008 Page 13

HTTP 1.0 Grammar 286
1. HTTP Message 286
2. Request 286
3. Response 287
4. Entity 288
5. Protocol Parameters 289
6. Content Parameters 290
7. Access Authentication 290

B WebWalker 1.00 Program Listing 293

C WebShopper 1.00 Program Listing 337

D List of Online Bookstores Visited by BookFinder 347

E List of Online Music Stores Visited by CDFinder 351

F List of Active MUD Sites on the Internet 355

Doran's MUDlist 356
TYPE <unknown> (2) 356
TYPE aber (19) 356
TYPE almostIRC (1) 357
TYPE bsx (1) 357
TYPE circle (6) 357
TYPE darkmud (1) 358
TYPE dgd (6) 358
TYPE diku (57) 358
TYPE dum (3) 361
TYPE lp (111) 361
TYPE lp-german (3) 366
TYPE mare (1) 367
TYPE merc (13) 367
TYPE moo (20) 368
TYPE moo-portuguese (1) 369
TYPE muck (16) 369
TYPE mudwho (2) 370
TYPE muse (12) 370
TYPE mush (39) 371
TYPE oxmud (2) 373
TYPE silly (1) 373
TYPE talker (7) 373
TYPE teeny (2) 373
TYPE tiny (1) 374
TYPE uber (1) 374
Key 374

IPR2025-00714
DATABRICKS EX1008 Page 14

G List of World Wide Web Spiders and Robots 375

The JumpStation Robot ... 376
RBSE Spider ... 376
The WebCrawler ... 376
The NorthStar Robot ... 376
W4 (World Wide Web Wanderer) ... 377
The Fish Search ... 377
The Python Robot ... 377
HTML Analyzer ... 377
MOMspider ... 377
HTMLgobble ... 378
WWWW—the WORLD WIDE WEB WORM ... 378
WM32 Robot ... 378
Websnarf ... 379
The Webfoot Robot ... 379
Lycos ... 379
ASpider (Associative Spider) ... 379
SG-Scout ... 379
EIT Link Verifier Robot ... 380
NHSE Web Forager ... 380
WebLinker ... 380
Emacs W3 Search Engine ... 380
Arachnophilia ... 381
Mac WWWWorm ... 381
ChURL ... 381
Tarspider ... 381
The Peregrinator ... 381
Checkbot ... 382
Webwalk ... 382
Harvest ... 382
Katipo ... 382
InfoSeek Robot ... 383
GetURL ... 383
Open Text Corporation Robot ... 383
NIKOS ... 383
The TkWWW Robot ... 384
A Tcl W3 Robot ... 384
TITAN ... 384
CS-HKUST WWW Index Server ... 384
Spry Wizard Robot ... 384
Weblayers ... 385
WebCopy ... 385
Scooter ... 385
Aretha ... 385
WebWatch ... 385

IPR2025-00714
DATABRICKS EX1008 Page 15

Bibliography **387**

Chapter 1 387
Chapter 2 390
Chapter 3 391
Chapter 4 392
Chapter 5 393
Chapter 6 393
Chapter 7 394
Chapter 8 395
Chapter 9 396
Chapter 10 397
Chapter 11 398

Index **401**

IPR2025-00714
DATABRICKS EX1008 Page 16

part I

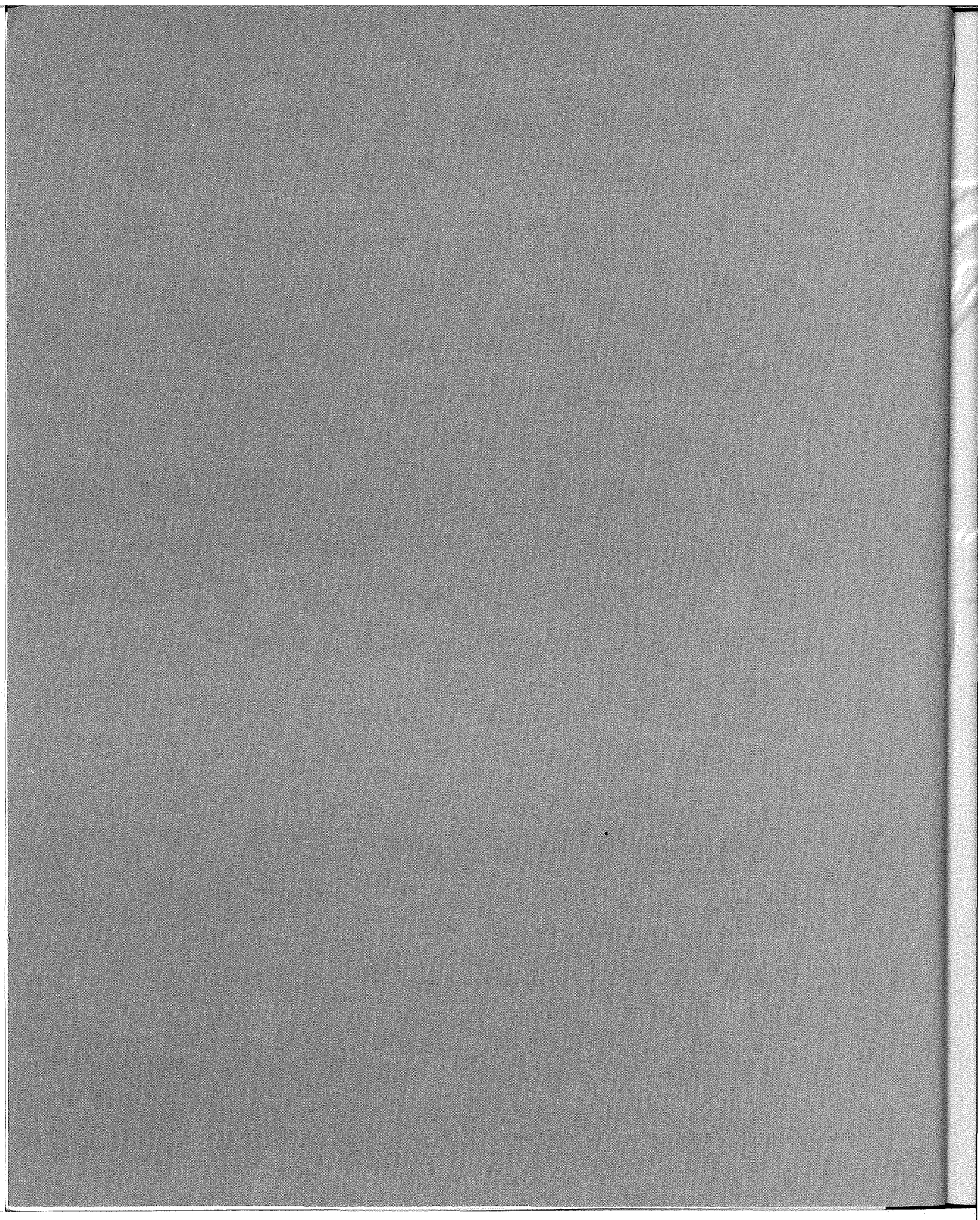
Introduction

1 The World of Agents 3

2 The Internet: Past, Present, and Future 37

3 World Wide Web: Playground for Robots 61

IPR2025-00714
DATABRICKS EX1008 Page 17



chapter 1

The World of Agents

Welcome to the world of agents. On the Internet, agents can take on many different forms and perform interesting functions. Some agents have been deployed on the Net and are in use daily. The following are some common types of agents on the Internet that you probably have already encountered:

IPR2025-00714
DATABRICKS EX1008 Page 19

- Web robots, spiders, and wanderers
- Web commerce agents
- Worms and viruses
- MUD agents and chatterbots

Web robots, *spiders*, and *wanderers* are programs that traverse the World Wide Web information space. They move from one Web document to another by referencing the hyperlinks embedded in the Web pages. Web robots speak the native HyperText Transfer Protocol (HTTP) of the World Wide Web, using it to retrieve Web documents from servers. They crawl on the Web to discover new resources, to index the Webspace for keyword searching, and to seek out dead links in Webspace for automated maintenance.

Web commerce agents are the automated Web shoppers, bargain-hunters, and smart online buyers for comparison-shopping and automated procurement. They are also the automated online catalogs and electronic sales representatives for manufacturers and retailers. But more importantly, they are playing the emerging roles of brokers, barterers, traders, and middlemen that promise to facilitate commerce on the Internet and on the Web in the near future.

Worms and *viruses* are malicious agents that replicate themselves in an elusive way to travel from machine to machine, network to network. In the past, they were often hand-carried by humans on floppy disks. But for the future, the Internet, with its decentralized global connectivity, is increasingly a vulnerable new medium of transport. Such undercover dark agents of society are considered harmful and extremely dangerous to the well-being of our global computing and communications infrastructure.

MUD agents and *chatterbots* are automatons from the world of Multi-User Dungeons or Multi-User Dimensions in cyberspace. MUD agents provide useful services to human players, such as answering inquiries and giving directions, through a typewritten natural-language interface. Chatterbots are conversational agents whose main job is to chat with human players. Unlike MUD agents, chatterbots are not specific to MUDs and can also exist outside of the MUD world.

The following chapters of this book examine various Web robots and spiders, and introduce some widely accepted operational guidelines for Web robots. Web commerce agents, although currently with few deployed examples, are introduced along with the World Wide Web. This book includes a chapter dedicated to discussing how one such spider for Web maintenance, WebWalker, can be constructed. This book also examines the operations of worms and viruses, as well as MUD agents and chatterbots.

The foundation technologies underlying Internet agents, such as transaction security, electronic cash, and payment services, are explained in detail along with examples of current commercial offerings on the Internet.

This book is about agents on the Internet, but for the sake of building a solid foundation for appreciating agents in general, the remainder of this chapter is dedicated to the pioneers of agent research who are currently busy constructing agents of the future. The next section discusses the concept of agents in general, and introduces a taxonomic agent framework for understanding various kinds of agents.

IPR2025-00714
DATABRICKS EX1008 Page 20

What are Agents?

The qualifying attributes of agenthood have for many years been the staple of lively philosophical discussions and the favorite subject of debates within the agent research community. Never before has a field of inquiry been so rich and diverse, yet fragmented, that its primary subject of inquiry remains shrouded in a perpetual rhetoric: *What exactly is an agent?*

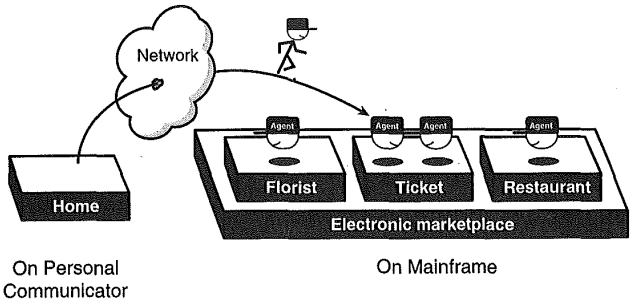
Simply put, agents can be considered personal software assistants with authority delegated from their users. Early visionaries such as Nicholas Negroponte (1970, 1989) and Alan Kay (1984) were among the first to recognize the value of software personal assistants. They spoke of the idea of employing agents in the interface to delegate certain computer-based tasks. More recently, several computer manufacturers have adopted this idea to illustrate their vision of the interface for the future; for example, videos produced by Apple (1988). In the words of Ted Selker (1994) from IBM's Almaden Research Center, "Agents are computer programs that simulate a human relationship, by doing something that another person could otherwise do for you."

The Telescript agent programming language technology developed by General Magic, a start-up company in the Silicon Valley, supports the deployment of software agents as personal delegates across the network. General Magic defines an agent as a piece of Telescript program that is sent across the network (White 1994). The Telescript program encapsulates the user's instructions for performing all kinds of tasks in electronic venues on the network, which are called "places." Electronic mailboxes, calendars, markets, and gathering points, for example, are all places.

As illustrated in figure 1.1, people who dispatch Telescript agents can think of these agents as electronic extensions of themselves, capable of gathering information resourcefully, negotiating deals, and performing transactions on their behalf. These Telescript agents can be customized for an individual user's preferences, and also are intelligent in the sense that they can have contingency plans. In other words, Telescript agents can assess themselves, as well as the conditions of their surrounding environment when situated in different places, and act accordingly, perhaps changing from an original course of action to an alternative plan.

Figure 1.1

A Telescript agent from General Magic.



IPR2025-00714
DATABRICKS EX1008 Page 21

The definition of agents, however, usually deviates from such a simple one as delegated software programs given above. Agent research has drawn upon the ideas and results produced by people from diverse disciplines, including robotics, software engineering, programming languages, computer networks, knowledge engineering, machine learning, cognitive science, psychology, computer graphics—even art, music, and film. From this diversity of perspectives, not one definition, but a rich set of views on agents, has emerged.

In addition to being understood as delegated software entities, agents can also be studied along other important dimensions, such as coordination, knowledge, creativity, and emotion. The programming and social aspects of agents are also important considerations. The remaining sections of this chapter explore the concept of agents along these various dimensions.

Agents and Delegation

Agents are primarily human-delegated software entities that can perform a variety of tasks for their human masters. This section examines their roles as personal assistants, desktop agents, surrogate bots, and softbots.

Personal Assistants

Pattie Maes, an assistant professor with the Massachusetts Institute of Technology Media Lab, has been working to create agents that reduce work and information overload for computer users (1994). She believes that as computers and networks begin to reach a larger populace, the current dominant metaphor of direct manipulation (Schneiderman 1988), which requires the user to initiate all tasks explicitly and to monitor all events, might not be the most convenient for many new, untrained users. She favors an alternative, complementary style of interaction called "indirect management," (Kay 1990) which engages the user in a cooperative process with a computer program known as the intelligent personal assistant.

Maes's work has resulted in agents that provide personalized assistance for a variety of tasks, including meeting scheduling, e-mail handling, electronic news filtering, and the selection of books, music, and other forms of entertainment. In the process of constructing such agents, Maes has identified the following two problems:

- **Competence.** How does an agent acquire the knowledge to decide when, with what, and how to help the user?
- **Trust.** How do you ensure that users feel comfortable delegating tasks to an agent?

According to Maes, both problems can be solved with a machine-learning approach, where the agent learns about its user's habits through interactions over time. Specifically, a learning agent gradually acquires its competence by the following:

- Observing and imitating the user
- Receiving positive and negative feedback from the user
- Receiving explicit instructions from the user
- Asking other agents for advice

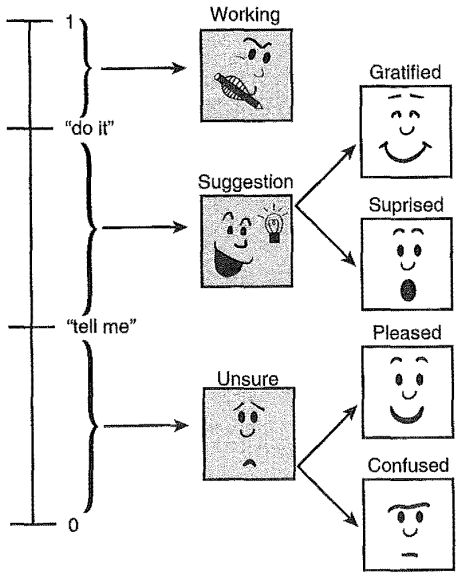
IPR2025-00714
DATABRICKS EX1008 Page 22

Over time, the agents become more helpful as they accumulate knowledge about how the user handles certain situations. Gradually, more tasks that initially were performed directly by the user can be taken care of by the agent. As shown in figure 1.2, Maes' agents use simple caricatures to convey their internal state to the user.

The user also is given time to gradually and incrementally build up a model of the agent's competencies and limitations. The particular learning approach adopted enables the agent to give explanations for its reasoning and behavior in language the user is familiar with. An example of this would be "I thought you might want to take this action because this situation is similar to this other situation we have experienced before." The user would have the opportunity to become more comfortable delegating tasks to the agents after using them for some time.

Figure 1.2

Simple caricatures convey agent "emotional" states to user.



IPR2025-00714
DATABRICKS EX1008 Page 23

Envoy Desktop Agents

The Envoy Framework has been proposed by researchers at Brown University's Institute for Research in Information and Scholarship (IRIS) as an open architecture for agents on the desktop (PYFLHCM 1992). The framework supports agents that operate in conjunction with existing desktop user applications and assists users with the more tedious, repetitive, and time-consuming tasks. Envoy agents help users with tasks such as the following:

- Sifting through incoming information
- Monitoring information sources continuously
- Searching data sources at regular intervals
- Delegating tasks now for future execution

In the Envoy Framework shown in figure 1.3, a user specifies a mission for the Envoy by interacting with an Envoy-aware application. These Envoy-aware applications are called *operatives* because they are responsible for actually carrying out missions on behalf of the user. As the user's representative, the Envoy would schedule, track, and dispatch all missions the user has specified, and handle all communications with the operatives.

When an operative completes an assigned mission, it notifies the Envoy, which in turn notifies the user through a set of Envoy-aware applications called *informers*. The mission results can be a brief message, a short report, or an interactive report viewable from the native application interface. At any time, the user can view a mission summary listing all active missions, as well as reports generated by operatives responsible for those missions.

A *bureau chief* on the local area network maintains a record of each user's Envoy, as well as all Envoy-aware applications in the environment. New operatives or informers in the environment must first register with the bureau chief.

New Wave Desktop Agents

In contrast to the Envoy Framework, Hewlett-Packard's New Wave Agent (HP 1989) is a more limited form of desktop integration that automates

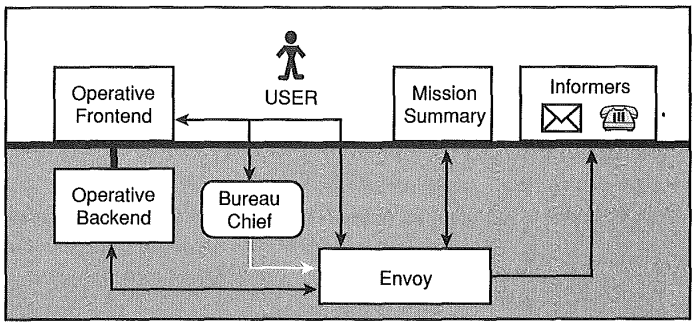


Figure 1.3

The Envoy Framework employs operatives, informers, and a bureau chief.

IPR2025-00714
DATABRICKS EX1008 Page 24

tasks users perform frequently. Application developers implement a defined set of protocols to make their applications agent-aware in the New Wave environment.

A New Wave user can specify routine tasks by demonstration. Say, for example, the user wants to start a database access application, download specific information into a spreadsheet, generate a graph from the spreadsheet data, copy the graph to a text document, and mail it to a group of users. All the user needs to do is turn on the recording feature and perform the desired sequence of actions interactively. The task is represented as a script document on the desktop and can be scheduled for execution using the calendar. The script also can be edited by the user if needed.

The integration of agent functionality into the desktop environment enables users to automate routine and repetitive tasks quite easily. Because tasks can be defined by example, the cognitive overhead of learning a scripting language is substantially reduced. A user needs only be sufficiently familiar with the language to make any necessary modifications to scripts. In addition, the calendar on the New Wave desktop provides an intuitive metaphor and convenient mechanism for scheduling agent tasks.

Surrogate Bots

Agents can relieve users of low-level administrative and clerical tasks, such as setting up meetings, sending out papers, locating information, tracking whereabouts of people, and so on. Research scientists at AT&T Bell Labs, Henry Kautz and Bart Selman, and MIT graduate student Michael Coen, have built and tested an agent system consisting of *surrogate bots* that addresses the real-world problem of handling the communication involved in scheduling a visitor to their laboratory at AT&T Bell Labs (1994).

Kautz, Selman, and Coen have identified the following issues as important for successful deployment of agents: reliability, security, and ease-of-use. Users should be able to assume that the surrogate bots are reliable and predictable, and human users should remain in ultimate control.

They approach the problem in a bottom-up fashion by first identifying specific tasks that are both feasible using current technology and also truly useful to the everyday users. After this, a set of software surrogate bots are designed, implemented, and tested with real users.

Visitor Scheduling Bots

The job of scheduling visitors is quite routine, but it consumes a substantial amount of the host's time. The normal sequence of tasks are as follows:

1. Announce the upcoming visit by e-mail.
2. Collect responses from people who would like to meet with the visitor.
3. Put together a schedule that satisfies as many constraints as possible.
4. Send out the schedule to participants.
5. Possibly reschedule people at the last minute due to unforeseen events.

In their agent system, a specialized surrogate bot, the *visitorbot,* handles the visitor scheduling. For each individual user, there is a *userbot* whose job

IPR2025-00714
DATABRICKS EX1008 Page 25

is to mediate communications between the human owner and the visitorbot. Figure 1.4 shows the user-interface created by a userbot in response to a message from the visitorbot.

The task-specific visitorbot specifies what information needs to be transferred or obtained but not how the communication should take place. It is the responsibility of each userbot to consider its owner's preferences and to accordingly determine the preferred mode of communication: graphics, voice, fax, or e-mail.

The userbot has its own graphical window containing buttons the user can press to change the preferred mode of communication, or to suspend processing of messages until a later time. The window also contains buttons labeled with all the different *taskbots* known to the userbot. When the user presses one of these buttons, the userbot sends a help request message to the appropriate taskbot, thereby initiating an interaction between the user and the selected taskbot.

Internet Softbots

Oren Etzioni and Daniel Weld, both professors at the University of Washington at Seattle, have the long term goal of developing an agent-based interface that enables naive users to locate, monitor, and transmit information across the net. For the past three years, they have led the Internet Softbot project, which focuses on the problems of designing and building a software robot capable of effectively exploring the Internet (1994).

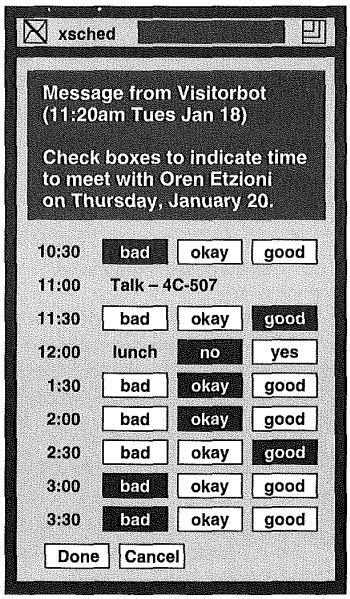


Figure 1.4

A window pops up to show a message from visitorbot.

IPR2025-00714
DATABRICKS EX1008 Page 26

The Internet Softbot uses a Unix shell and the World Wide Web to interact with a wide range of Internet resources. Softbot sensors are analogous to whiskers on a physical insect robot, and include Internet facilities such as archie, gopher, netfind, and others. Softbot effectors are analogous to the mechanical arms and legs on a physical robot, and include ftp, telnet, mail, and numerous file manipulation commands. The softbot is designed to incorporate new sensor and effector facilities into its repertoire of Internet-based tools as they become available.

According to Etzioni and Weld, the softbot supports a qualitatively different kind of human-computer interface. In addition to simply allowing the user to interact with the computer, the softbot behaves like an intelligent personal assistant. The user can make a high-level request, and the softbot uses search, inference, and knowledge to determine how to satisfy the request. Furthermore, the softbot is designed to be robust enough that it can tolerate and recover from ambiguity, omissions, and errors in human requests.

Softbot Planner

The planning component of softbot is called the *softbot planner*. It takes as input a logical expression which describes the user's goal in the form of a sentence in first-order predicate logic. For users unfamiliar with logical expressions, a graphical fill-in form that automatically translates to a softbot goal is available.

After searching a library of *action schemata* describing available information sources, databases, utilities, and software commands, a sequence of actions to achieve the goal is then generated. The softbot planner is able to decompose complex goal expressions into simpler components and solve them with divide-and-conquer techniques. Interactions between subgoals, which are usually problematic, are automatically detected and resolved.

The softbot planner relies on a logical model of the available Internet resources that tells it how these resources can be invoked or accessed, as well as the effect of doing so. Unlike traditional programs and scripts which are committed to a rigid flow of control determined by the programmer when the program was coded, the softbot planner synthesizes plans on demand when the program is run, based upon the user's goal. In the words of Etzioni and Weld, a softbot "is worth a thousand shell scripts."

Example Softbot Usage

With the Internet Softbot, for example, a user can quickly perform the task of "sending the budget memos to Mitchell at CMU" with ease (see fig. 1.5).

The softbot first disambiguates the reference to Mitchell at CMU by executing the command `finger mitchell@cmu.edu` and recording who the various Mitchells are at CMU. If necessary, it prompts the user to select the intended recipient. If it decides to send the memos, the softbot determines the correct e-mail address and reasons about the document format (for example, postscript if it contains figures and LaTeX source otherwise). Furthermore, if Mitchell is out of town (for example, as notified by reply e-mail from the "vacation" program), or if the memos are confidential (such as encrypted), it ensures delivery in a timely and secure manner.

IPR2025-00714
DATABRICKS EX1008 Page 27

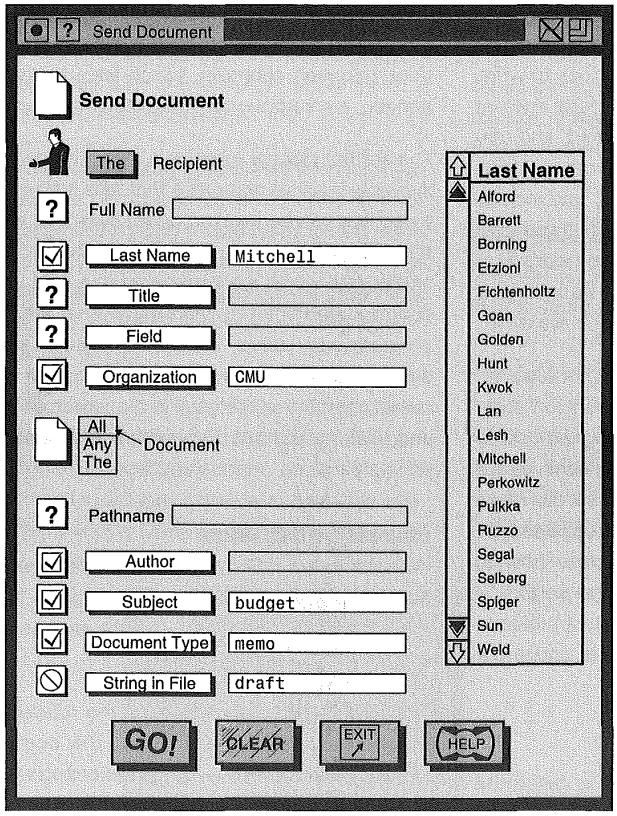


Figure 1.5

Softbot request form for sending a document.

Agents and Coordination

Agents can also facilitate work and coordinate tasks among people, machines, and other agents. This section describes conference-support agents such as GDS and M, and communicative agents based upon an agent communication language (ACL). The coordination, collaboration, and communication aspects of agents are emphasized.

Conference-Support Agents

Researchers at LUTCHI Research Centre at Loughborough University of Technology in England demonstrate that group support agents are viable for design tasks (ECJS 1994). They have constructed a Geographic Decision System (GDS) to provide multi-agent group support to design conferences.

IPR2025-00714
DATABRICKS EX1008 Page 28

GDS has a separation of function, which is achieved by partitioning the system into distinct components. A central communication bus serves as the backbone of the system, attached to which are the per-user presentation layers and dialog controllers, plus one of each of the following system-wide agents:

- Conference agent
- Group agent
- Application agent
- User agent
- Floor agent

The *conference agent* controls initialization of the system. It interacts with the person who starts the conference to learn about the other participants, their locations, and any applications to be shared. A presentation layer and a dialog controller are then created for each participant. The conference agent next invokes other agents and starts the appropriate applications along with their respective application interface modules. Throughout the conference, the conference agent allows newcomers to join, members to leave, and different applications to be shared.

A separate *group agent* supports the customization of group options. It might be undesirable, for example, that every participant does have the ability to end the conference.

The *application agent* provides external software application services to the group. An example of such an external software system could be the geographic information system, which might be useful to the group in the design of road systems. The application agent can intercept and modify messages from the dialog controllers to the application interface module by snooping on the communication bus.

The *user agent* intercepts all messages on the communication bus, which allows it to have master control of interaction with users. In other words, the user agent enables different members of the group to view data from different vantage points and to interact with it, and with one another, in different styles.

The *floor agent* works with the user agent to ensure that only one participant can enter data at a time. The floor agent understands different floor policies, such as moderated, first-come, or round-robin, and offers the capability to change the floor policy as needed.

Integrated Agents

Doug Riecken, a researcher from the AT&T Bell Laboratories, takes the position that it takes many integrated agents to create a software assistant (1994). In such an approach, many different reasoning processes, called a "society of agents," are integrated to realize a software assistant capable of performing a broad range of tasks. Riecken's efforts have resulted in the realization of *M*, a software assistant that helps the user classify, index, store, retrieve, explain, and present information in a desktop multimedia conferencing environment.

M's Architectural Design

M's architectural design is based upon the theory of integrating a variety of reasoning processes, or

IPR2025-00714
DATABRICKS EX1008 Page 29

agents, to form an intelligent assistant. Influenced by AI pioneer Marvin Minsky's "society of mind" theory (1985), M is built to accommodate the following types of reasoning capabilities:

- Spatial (based upon properties of space)
- Structural (based upon relationships between parts of some object)
- Functional (based upon the functional purpose of some object)
- Temporal (based upon properties related to time)
- Causal (based upon events, actions, and state changes in objects)
- Explanation-based (explanation of a situation through first principles)
- Case-based (solving new problem by analogy of stored solutions to old ones)

M accomodates these capabilities by integrating various subsystem components, including a spreading activation semantic network for realizing K-lines/polynemes, a rule-based system, a set of blackboards for realizing transframes and pronomes, a scripting system, a history logfile system, and an I/O system.

M's Operation

M was used at the AT&T Bell Labs within a virtual meeting room that supports multimedia desktop conferencing. Participants collaborate using pen-based computers and with voice through telephones. The goal of the software assistant is to classify and index the changing state of the virtual meeting room.

In this virtual place, each user is supported by a personalized assistant, and the world is composed of electronic documents, electronic ink, images, markers, white boards, copiers, staplers, and so on. The assistants attempt to recognize and define relationships among objects based upon actions applied by the user to the world and the resulting new states of the world.

From observing that a user annotates two adjacent documents by drawing a circle to enclose them together, for example, M can infer and explain a plausible relationship between the two documents. Essentially, M applies the following reasoning capabilities:

- Spatial reasoning to find out about the nearness of the two documents and the circle
- Structural and functional reasoning about the circle enclosing two documents
- Causal reasoning about the action of enclosing objects.

Riecken's underlying thesis is that an assistant for classifying and explaining actions applied to objects within a dynamic world should be functionally effective if it can simultaneously generate and test multiple domain theories in relation to a given goal.

When an event occurs, such as an individual annotating a document or moving a piece of paper, M's I/O system records who did what and archives it as an input record for processing. M attempts to generate and maintain simultaneous theories of the world by using a set of "blackboards" to which emerging theories of the world are posted. Thus, each blackboard serves as the working area to

IPR2025-00714
DATABRICKS EX1008 Page 30

expand and improve a given theory, and the set of blackboards are ranked based upon the strength of each theory.

According to Riecken, the integrated agents in M make possible a new framework for users to work together electronically. M improves the performance of participants in a virtual meeting room by allowing for added expressiveness while minimizing many computer-related actions.

Communicative Agents

Professor Michael Genesereth and his graduate student Steven Ketchpel at Stanford University have examined the practical issues of software interoperation from the viewpoint of an agent software architecture (GK 1994). They have coined the term *agent-based software engineering* to describe the approach of writing software applications as components called software agents.

These *software agents* interoperate by exchanging messages in a universally mandated agent communication language. Software agents differ from objects in object-oriented programming in that the meaning of an agent message is based upon a common language with agent-independent semantics, whereas the meaning of an object message can vary from one object to another.

Genesereth and Ketchpel have identified the following three issues that need to be addressed within the context of agent-based software engineering:

- What is an appropriate agent communication language?
- What is the best way to build agents capable of communicating in this language?
- What communication architectures are conducive to cooperation?

Agent Communication Language

Two popular approaches are used to design an agent communication language: a procedural approach or a declarative approach. In the procedural approach, communication can be thought of as the exchange of procedural directives. Individual commands, as well as entire programs, can be transmitted and executed at the recipient's end. Scripting languages, such as TCL, Apple Events, and Telescript, are based upon the procedural approach.

In the declarative approach, communication can be thought of as an exchange of declarative statements, such as definitions, assertions, or assumptions. The declarative approach, in the form of Agent Communication Language (ACL), was chosen by Genesereth and Ketchpel for their agent-based software engineering.

ACL was designed by researchers in the ARPA Knowledge Sharing Effort (NFFGPSS 1991). ACL is made up of three parts: a vocabulary, an inner language called Knowledge Interchange Format (KIF), and an outer language called Knowledge Query and Manipulation Language (KQML).

The vocabulary of ACL is listed in a large and open-ended dictionary of words appropriate for common application areas (Gruber 1991). KIF is a prefix version of first-order predicate calculus, capable of encoding simple data, constraints, rules, and quantified expressions, among other things. KQML is a

IPR2025-00714
DATABRICKS EX1008 Page 31

linguistic layer above KIF that provides contextual information for more efficient communications.

With a clear definition of the ACL, it is straightforward to write agent programs that abide by certain behavioral constraints in order to work together correctly. For the large number of existing legacy software, however, Genesereth and Ketchpel offer the following three approaches to agentification:

- Implement a *transducer*, which mediates between an existing program and other agents.
- Design a *wrapper* around an existing program to enable it to speak ACL.
- Rewrite the original program, as a last resort.

Agent Communication Architecture

Several architectures have been proposed for organizing agents to enhance collaboration. In the *contract-net approach* (DS 1983), agents in need of services distribute requests for proposals to other agents, who evaluate those requests and submit bids to the originating agents. The originators use the bids in deciding to whom to award contracts.

In the *specification sharing approach*, agents advertise their individual capabilities and needs. This information is then used to coordinate agent activities.

Finally, in the *federated system approach*, agents do not communicate directly with one another but instead rely on system programs called *facilitators* to handle all communications with other agents (see fig. 1.6).

Already, agent architecture has been put to use in concurrent engineering for application-level interoperation, as reported by Cutkosky (1993). The long-range vision for agent technology, according to Genesereth and Ketchpel, is one in which any system can interoperate with any other system without the intervention of human users or their programmers.

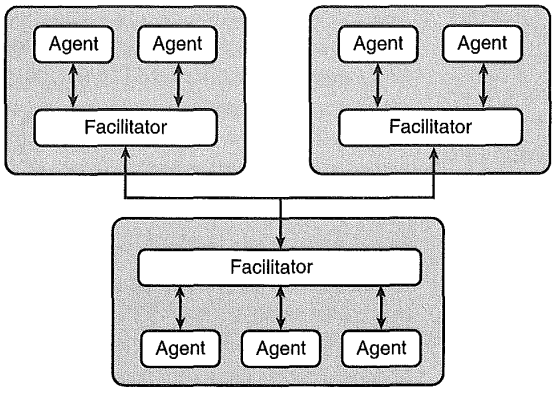


Figure 1.6

Federation of agents.

IPR2025-00714
DATABRICKS EX1008 Page 32

Agents and Knowledge

The knowledge component of agents serves many useful functions. This section discusses how agents can teach people new programming skills, learn about the calendar scheduling habits of human users, reason with common sense, and derive world knowledge from sensing the surrounding physical environment.

Teaching Agents

Ted Selker, manager of User Systems Ergonomics Research at IBM's Almaden Research Center, views agents as computer programs that simulate a human relationship. According to Selker, there can be two types of agents.

An *assistant-style agent* is one that builds a relationship with the user through a private interface. Using this interface, the agent can understand the user's needs to perform formerly complex or unknown tasks with computer-created macros. An *advisory-style agent*, on the other hand, is one that builds a user relationship with the explicit goal of educating the user.

Selker has built an advisory-style teaching agent called Cognitive Adaptive Computer Help (COACH) that helps users learn to program in the Lisp programming language (1994).

The COACH Agent

To use a computer language effectively, a student needs to understand both its syntax and its semantics.

The *syntax* includes language statements, as well as tokens such as keywords and acceptable variable names in Lisp. The syntax definition is used as a way to classify user progress and to guide instruction.

The *semantics* of the language includes learnable concepts in Lisp such as evaluation, iteration, stored variables, and so on. In addition, learnable concepts, all of which must be mastered to do a specific task, are further organized by COACH into basic sets. The COACH system also includes examples of these learnable concepts and a model of the particular student's understanding and ability to use each one. COACH has the user-interface shown in figure 1.7.

COACH watches the user's actions in order to build an adaptive user model of the user's experience and expertise. While the user is working on a task, aspects of the user's successes and failures are recorded. The system is proactive in that it can anticipate user needs and is capable of presenting help before it is requested. Both the user and the system can initiate help in a mixed-initiative interaction.

Several representations of language knowledge work together in COACH to create help for the user:

- Subject frames, which consist of knowledge about the domain
- Adaptive frames, which hold the recording of user experience relative to a domain
- Presentation rule sets, which embody a model of teaching
- A multi-level parser, which is the syntax definition of the domain

IPR2025-00714
DATABRICKS EX1008 Page 33

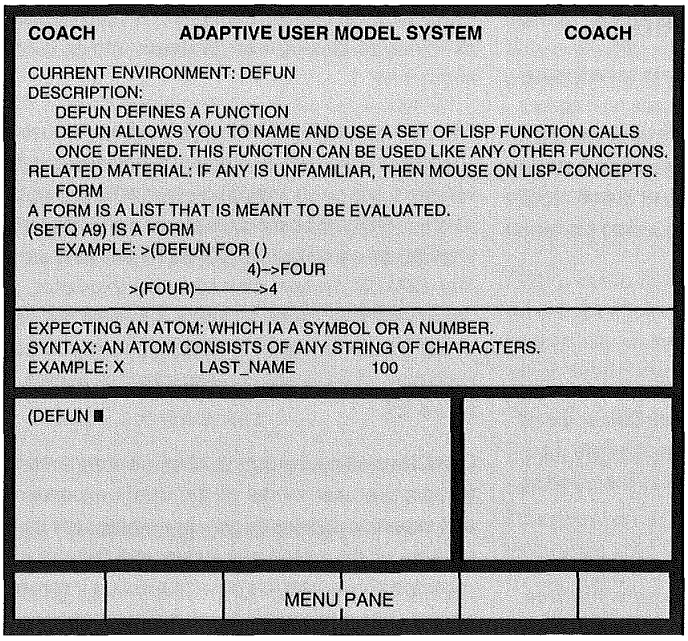


Figure 1.7

The COACH user-interface.

The defined network of relationships between the domain to be learned, the user's actions, and the state of the user model, forms the basis for selecting user help. The system chooses when to use example, description, and syntax style help depending upon the levels of user expertise, such as novice, intermediate, professional, or expert.

Field-Testing COACH

To test the hypothesis that an adaptive coaching paradigm improves productivity, a version of COACH without the automatic help and adaptiveness was created.

In a usability study conducted by Selker involving 19 programmers with no prior Lisp experience, users of the adaptive system wrote five times as many Lisp functions as those of the nonadaptive one. During the course, users of the adaptive system liked Lisp more than the other group, consulted the help screen more often, and rated COACH higher as a learning environment. Finally, at the end of the course, only 11 percent of the students from the adaptive group, as compared to a full two-thirds of students from the nonadaptive group, indicated they felt uncomfortable with Lisp (Selker 1994).

An adaptive teaching scenario concentrates on the user's individual needs by moving students toward an apprenticeship or learn-while-doing approach and away from syllabus-style classroom experience. An adaptive teaching paradigm is found to improve productivity.

IPR2025-00714
DATABRICKS EX1008 Page 34

The COACH system demonstrates that agent technology can successfully work in place of a human coach to give personalized instruction while a student is actually working out solutions.

Learning Agents

Researchers at Carnegie Mellon University (CMU), Mitchell, Caruana, Freitag, McDermott, and Zabowski, believe that machine learning plays an important role in future personal software assistants. They imagine a future where knowledge-based assistants "operate across the network as a kind of software secretary, providing services for work and home such as paying bills, making travel arrangements, submitting purchase orders, and locating information in electronic libraries" (MCFMZ 1994).

The Calendar Apprentice

The success of these agents will depend on knowing and learning about the particular user's habits and goals, and tailoring to them accordingly. The CMU researchers have built a calendar manager called Calendar Apprentice (CAP), which learns its user's scheduling preferences from experience—it is a learning apprentice that assists the user in managing a meeting calendar.

CAP provides an interactive editing and e-mail interface to an online calendar, and is capable of giving customized scheduling advice to each user. In approximately five user-years of experience (one user-year is equivalent to one user using CAP for one year), CAP has learned an evolving set of several thousand rules that characterize scheduling preferences for each of its users (JDMMZ 1991; DBMMZ 1992; MCFMZ 1994).

Traditionally, many programs provide simple parameters enabling users to explicitly customize the program's behavior. Text editors, for example, enable users to set default font types and sizes, while desktop window managers enable users to choose the default placement of icons and windows.

According to the CMU researchers, however, there are limits to this approach. Customizing an e-mail sorter to accommodate one's personal notion of an urgent message, for example, requires detailed articulation of a fairly subtle concept. Furthermore, even if users are willing to initially customize their assistants, they might be unwilling to continually update this knowledge. A message about a particular business contract, for example, might be quite urgent before an approaching deadline, but not necessarily as urgent after the deadline.

The approach adopted in CAP can be summarized as follows:

- Provide a convenient interface (see fig. 1.8) that enables the user to perform the task—an editing and e-mail interface to an online calendar, for example.
- Treat each user interaction as a training example of the user's habits. Each meeting scheduled by the user reflects preference for the duration, time, location, and so on, of this type of meeting.
- Learn general regularities from this training data and use this learned knowledge to increase the services offered. An assistant could, for example, provide interactive advice to the user or offer to negotiate specific meetings on the user's behalf.

IPR2025-00714
DATABRICKS EX1008 Page 35

08/25/1992 Immigration Course Aug 25 – Sept 11
08/26/1992 IC talk is 11:15 – 12:15

Time	Monday 8-24	Tuesday 8-25	Wednesday 8-26	Thursday 8-27	Friday 8-28
8:00					
8:30					
9:00					
9:30					
10:00					
10:30					
11:00			Immigration- Weh5409 SP: mitchell		
11:30					
12:00		Boclonek Weh5309			
12:30					
1:00					
1:30	Zabowski Weh5309 Reddy Simmon Weh5327		Harris Weh5309		
2:00					
2:30		Immigration- Weh5409 SP: unknown			
3:00	Edrc-Faculty Edrc-Conf-Rm			Adult Weh5309	Masuoka Weh5309
3:30					
4:00				Away ! ! !	
4:30					
5:00					
5:30					
6:00					

Time | 8-24 | 8-25 | 8-26 | 8-27 | 8-28

Duration: C-A[60] 30

Figure 1.8

The Calendar Apprentice user interface.

CAP Functionality

With CAP, users can edit the calendar by adding, deleting, moving, copying, and annotating meetings, and they can mark various calendar events as either tentative or confirmed. Other CAP commands instruct CAP to send e-mail meeting invitations or meeting reminders to the attendees as appropriate.

As time goes on, CAP learns the scheduling preferences of its user, and evolves gradually from a passive editing interface to a knowledgeable assistant capable of interacting more intelligently with the user and offloading the work of meeting negotiation from the user.

Currently, CAP learns rules that enable it to suggest the meeting duration, location, time, and date.

IPR2025-00714
DATABRICKS EX1008 Page 36

Each night, CAP automatically runs a learning algorithm to refine the set of rules it will use to provide advice on the following day. The learning algorithm is similar to ID3 (Quinlan 1986), which learns a decision tree from the most recent training data.

A *decision tree* organizes the problem of classifying an object into a series of questions about the object. Calendar meetings, for example, can be classified according to meeting location and based upon various "feature tests" at branch points. These feature tests can ask whether it is lunchtime or not (dining hall or conference room), as well as the attendee's department (EE building or CS building).

Field-Testing CAP

Results from field testing CAP within a small academic community at Carnegie Mellon University indicate that it is indeed possible for the system to learn rules that characterize scheduling preferences. The accuracy of learned advice varies significantly from feature to feature, and from user to user. It also is observed that the accuracy of CAP varies over time, reflecting the dynamic nature of the domain and the need for updating user-specific scheduling preferences. In particular, the periods of poorest performance correlate strongly with the semester boundaries in the academic year—when there are permanent scheduling changes.

Based upon CAP's performances, its creators at CMU conclude that "while rules learned by CAP are useful for providing interactive advice to be approved or overriden by the user, they are not sufficiently accurate to support autonomous negotiation of all meetings by the agent on the user's behalf" (MCFMZ 1994).

Rather than total automation of user workload, the CMU researchers foresee that a more likely scenario for practical software agents of the future is one of shared responsibility. Only the subset of situations for which the agent has high confidence will be handled autonomously, while difficult cases will always be referred to the user.

Common-Sense Agents

Douglas Lenat, principal scientist at Microelectronic and Computer Corporation (MCC), believes that agents need some common corpus of shared knowledge in order to communicate. According to Lenat, the past 20 years have witnessed numerous successes in which knowledge-based systems have been constructed and deployed. Amidst all these successes, however, there is constant failure as well. These systems cannot share knowledge and pool together their expertise and work together synergistically. In other words, these systems were *brittle* in the face of unanticipated situations (LGPPS 1990).

Lenat believes the primary impediment to achieving interesting agent behavior is lack of knowledge. He reasons that we would not need to work as hard to come up with clever algorithms, data structures, and architectures if we had a large database of knowledge to fall back on.

Backed by a 10-year, 25 million dollar grant in the Cyc project (as in enCYClopedia) that started in 1984, Lenat is boldly pioneering an attempt to assemble a massive knowledge base (on the order of tens of millions of axioms) spanning human consensus knowledge (LGPPS 1990; GL 1994).

IPR2025-00714
DATABRICKS EX1008 Page 37

Second Paradigm of Software Agents

In Lenat's view, there are two contrasting paradigms for software agents today. In the first paradigm, competence emerges from a large number of relatively simple agents integrated by some cleverly engineered architecture. An example of this first paradigm is SOAR (LNR 1987), whose forerunners were the early production systems like OPS5 (BFKM 1985).

In the second paradigm, competence emerges from the aggregate system possessing a large amount of useful knowledge. For real world tasks, this involves a dauntingly large amount of what might be called common-sense knowledge. In this second paradigm, the architecture is relatively unimportant. The archetype of this paradigm is Cyc, and its forerunners were the early expert systems.

The Cyc project intends to test seriously the second paradigm of software agents. Much of the constituent common-sense knowledge includes simple notions of time, space, causality, and events; human capabilities, limitations, goals, decision-making strategies, and emotions; familiarity with art, history, literature, and current affairs; and so forth.

The level of shared knowledge correlates directly to tasks performed by the intelligent agents. To be practical, Cyc has adopted the following maxim: "Share most of the meaning of most of the terms, most of the time" (GL 1994).

But how much shared knowledge is enough? The Cyc research so far seems to suggest that even relatively narrow tasks require a large fraction of common-sense knowledge to be shared. But fortunately, a wide range of tasks can use this same large body of shared knowledge.

Common Sense Knowledge in CYC

Lenat's approach is to express common-sense knowledge in a frame-based language (LGPPS 1990). The common-sense knowledge is represented by a more expressive predicate calculus (also called first-order logic) framework, which provides the following enhanced features:

- Defaults representation (allowing one to talk about unstated facts)
- Reification (allowing one to talk about propositions in the knowledge base)
- Reflection (allowing one to talk about the act of working on some problem)

In order to answer most queries, Cyc has to do some sophisticated inference. Rather than relying upon a single general mechanism (such as resolution) for problem solving, Cyc makes extensive use of specialized mechanisms that employ different algorithms and data structures for frequently used classes of inferences.

The bulk of the effort in building the knowledge base involves identifying, formalizing, and entering microtheories of various topics such as money, buying, shopping, and so on. Cyc researchers follow a process that begins with a statement, in English, of the theory. To achieve an axiomatization of the theory, the necessary Cyc concepts are identified and made precise. To test whether the topic has been adequately covered, stories dealing with the topic are represented in Cyc. Questions that a human should be able to answer after reading the story are then posed to Cyc.

Within the next two years, Lenat expects that most knowledge entry will take place by semiautomated

IPR2025-00714
DATABRICKS EX1008 Page 38

natural language understanding. Humans will then be able to "take the role of tutors rather than brain surgeons" in feeding knowledge to Cyc (GL 1994).

Physical Agents

Rodney Brooks, an associate professor with the MIT Artificial Intelligence Laboratory, believes in approaching intelligence in an incremental manner, with strict reliance on robots interfacing to the real world through perception and action at every step along the way.

Brooks offers his "physical grounding hypothesis," which states that to build a system that is intelligent, it is necessary to have its representation directly based upon the physical world. He observes that the real world is its own best model. In other words, the real world is always up to date and always contains every detail there is to be known. He believes that the trick is for autonomous agents in the form of physical robots to sense it appropriately and often enough.

The traditional notion of intelligent systems held by AI workers has been that of a central system, with perceptual modules as inputs and action modules as outputs. The traditional methodology decomposes intelligence into functional units whose combinations provide overall system behavior.

Brooks argues that "human-level intelligence is too complex and too little understood to be correctly decomposed into the right subpieces at the moment and that, even if we knew the subpieces, we still would not know the right interfaces between them."

Brooks prefers an alternative decomposition of an intelligent system along the orthogonal directions of behavior-generating modules, each of which individually connects sensing to action, without going through a central information processor. The advantage of this approach is that it gives an incremental path from very simple systems to complex autonomous intelligent systems. Furthermore, the coexistence and cooperation of these behavior-generating modules sets the stage for the emergence of more complex behaviors.

Brooks' research approach has resulted in a successful series of mobile robots with insect-level intelligence that operate without supervision in standard office environments (Brooks 1990, 1991).

The Genghis Robot

An example of Brook's mobile robots is Genghis, a six-legged robot weighing one kg that walks under Brooks' *subsumption architecture* and has a highly distributed control system (1989). The robot can successfully walk over rough terrain. Genghis is made up of 12 motors, 12 force sensors, six pyroelectric sensors, one inclinometer, and two whiskers. Genghis also is capable of following certain moving objects, such as human beings, using its pyroelectric sensors.

Genghis has no central control system. Instead, a subsumption architecture enables successive layers of behavior-generating modules to implement various aspects of Genghis' walking behavior. Genghis uses force measurements to comply with rough terrain and to lift its feet over obstacles, and it uses inclinometer measurements to selectively inhibit rough terrain compliance when appropriate. It uses whiskers to lift feet over obstacles and uses the directionality of infrared radiation to modulate the backswing of particular leg sets so that it follows a moving source of radiation.

IPR2025-00714
DATABRICKS EX1008 Page 39

The resulting control system in Genghis is elegant in its simplicity. It directly implements walking through very many tight couplings of sensors to actuators, without a centralized information processor. Genghis' capability to walk is thus an emergent behavior derived from the interaction of many diverse system components without the supervision of a centralized control system.

Agents and Creativity

Agents can be creative too. This section explores how agents can offer creative ideas in architectural styles, jazz music, mathematics, and mechanical shape design. Agents also can perform automated configuration design from a catalog of physical parts.

Creative Agents

Margaret Boden, a professor at the University of Sussex's School of Cognitive and Computing Sciences, has investigated the practical question of whether agent systems might help further human creativity (1994). She has examined how creativity in its various forms might be scientifically understood in terms of the computational resources involved.

Creativity involves coming up with something novel, new, and different. This new idea, in order to be interesting, must be intelligible. No matter how different the new idea is, it must be understood in terms of what was already known before. The potential role of agents as they relate to creativity includes suggesting, identifying, or even evaluating differences between familiar ideas and novel ones.

According to Boden, not all creativity can be understood as a novel combination of familiar ideas. Creative ideas are present in architecture, musical compositions, literary genres, mathematical theorems, and engineering inventions. Some creative ideas actually help open up a whole new set of conceptual spaces previously unthought of. This means that when exploring the implications of radical scientific theories or of new musical genres, simple combination juggling would not cut it. A structured, disciplined, and sometimes even systematic search for the promised meanings is necessary.

One way to start thinking about the whole enterprise of creativity is to consider the notion of conceptual spaces. A *conceptual space* is a mental terrain, a style of thinking (Boden 1991). It is defined by a set of constraints demarcating the boundaries and dimensions of the relevant domain. Many creative achievements result from exploring conceptual spaces in systematic and imaginative ways. Agents can help map, explore, and perhaps even guide in the transformation of conceptual spaces.

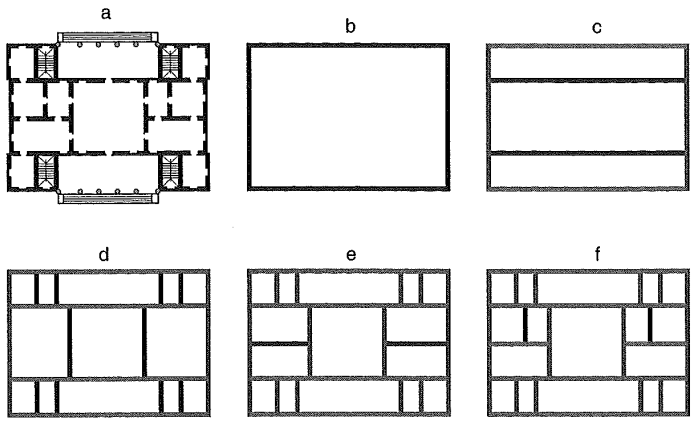
Architectural Styles

In the architectural domain, for example, computational work on architectural styles suggests some ways in which agents might help a human architect. The architectural style of Frank Lloyd Wright's Prairie House can be captured in a computer program (HF 1992). Similarly, the stylistic essence of a Palladian villa (see fig. 1.9) can be explicitly described with a computationally inspired "space grammar" that begins with a rectangle from which internal rectangles are recursively generated according to some prescribed rules (KE 1981). This process is illustrated in figure 1.10.

IPR2025-00714
DATABRICKS EX1008 Page 40

Figure 1.9

Palladian villa floorplan.



For a human architect or an architectural student who has little experience working on a particular architectural genre, an agent's timely advice or forbiddance on a piece of substructure design is especially valuable.

Jazz Dimensions

After the conceptual space of a specific domain is mapped, agents can explore it in interesting ways. In the jazz domain, for example, there are computer programs that help people improvise jazz (Hodgson 1990; Waugh 1992). These programs understand the various dimensions of the jazz musical space and can travel through it in many ways. If left to wander through the space by themselves, these programs improvise—on a given melody, harmony, and rhythm—by making random choices along many dimensions simultaneously. Working in this fashion, these programs often develop novel musical ideas that the professional jazz musicians find interesting and might want to explore further.

Mathematical Spaces

Agents can also guide in the transformation of conceptual spaces in surprising ways. The most well known example can be found in Douglas Lenat's program, Automatic Mathematician, whose transformations of the space of heuristics resulted in the discovery of two previously unknown theorems about prime numbers (1983).

Design Shapes

Researchers from the LUTCHI Research Centre at Loughborough University of Technology in England have studied how agents can assist design teams by providing support for *emergence*, a significant feature of the creative design process.

In particular, they have investigated the support of shape emergence in design communication, as well as how it can be handled by agents using pattern recognition methods. An emergent form displays characteristics not present in its source. The researchers' favorite example is the radical

IPR2025-00714
DATABRICKS EX1008 Page 41

transformation of the bicycle frame concept in the LotusSport bicycle, which uses a single-unit carbon fiber monocoque construction instead of the conventional steel-tube diamond frame (ECJS 1994).

Many psychological processes are involved in creative thinking, from combinational juxtaposition to the more complex exploratory-transformational reasoning to the highly unstructured emergent thinking. As we begin to understand more about the underlying computational aspects of such thought processes, we will be better equipped to build agents that could assist humans across a broader range of creative endeavours.

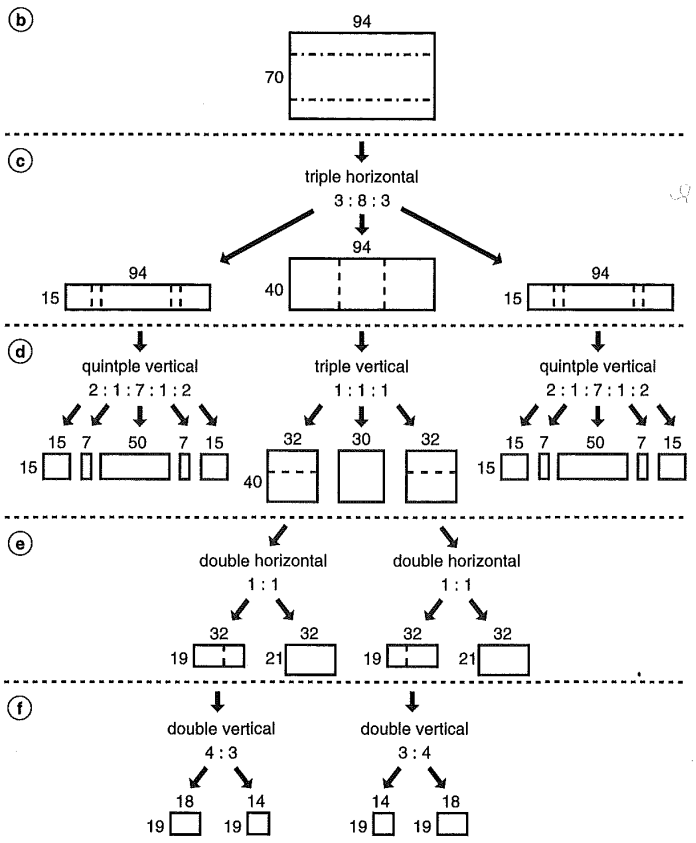


Figure 1.10

The process of generating a Palladian villa floorplan.

IPR2025-00714
DATABRICKS EX1008 Page 42

Automated Design Agents

Professor Bill Birmingham and graduate student Tim Darr of the University of Michigan have experimented with an approach of automated design-space exploration using the Automated Catalog-Design Service (ACDS) system (DB 1992; BDDWW 1993).

ACDS performs configuration design, where an artifact is designed by selecting parts from a catalog. The designer needs only to provide a high-level description of the design, including the functions to be performed, the interconnections of the components, and their specifications. In ACDS, the entire design space is reduced through a series of pruning operations until a set of feasible designs result.

ACDS is organized as a loosely-coupled network of different kinds of agents. It can self-organize based upon design specifications, such as the following:

- Catalog agents
- System agents
- Constraint agents

Catalog Agents

Each catalog agent represents a set of physical parts. ACDS can support thousands of catalog agents, each of which could be the product line of a component manufacturer. Catalog agents are able to choose whether to participate in a particular design.

System Agents

The system agent provides a graphical interface that enables the user to specify the design for presentation to the ACDS network. The system agent translates the high-level design specifications into the network's representation and broadcasts it to relevant agents. These design specifications are needed for creating any necessary constraint agents for the design.

Constraint Agents

Constraint agents maintain consistency throughout the network by enforcing design constraints. Each constraint agent ensures that the evolving design space conforms to the constraint it represents when evaluating proposed bids of their parts from catalog agents. Constraint agents can thus direct the pruning of part catalogs to satisfy any violated constraints. This process of removing infeasible parts, bidding, and pruning continues until all constraints are satisfied or a determination is made that no solutions exist.

Agents and Emotion

Though it might seem surprising at first, agents can have "emotions," too. This section explores the role of emotions in agents and discusses how emotion can help animate faceless software agents into cartoon-like effable characters (but not to the extent of *anthropomorphizing* them to human-level intelligence and capabilities), making them more lifelike.

Professor Joseph Bates of Carnegie Mellon University thinks emotions play an important role in the construction of believable agents (1994). He describes a believable agent not as one that has an honest or reliable character, but as one that provides an illusion of life in convincing ways so the audience wants to believe the agent is real.

IPR2025-00714
DATABRICKS EX1008 Page 43

According to Bates, believable agents are the interactive analog of believable characters discussed in the arts of fiction-writing and film-making. Emotion is the primary means of achieving this believability. An agent with demonstrated emotions helps people understand that the character really cares about its surrounding environment and that it truly has desires.

Art of Animation

Animation artists made great strides in advancing the state of the animation arts by constructing believable characters following the introduction of Disney's Mickey Mouse in the 1930's. Animation artists spoke of building characters "whose adventures and misfortunes make people laugh—and even cry" (TJ 1981).

According to Thomas and Johnson, two of Disney's nine earliest animators, "there is a special ingredient in (the arts of) animation that produces drawings that appear to think and make decisions and act of their own volition; it is what creates the illusion of life" (TJ 1981).

Artificial Intelligence

Many researchers in artificial intelligence (AI) have long sought to build robots or agents that seem to think, feel, and live. In addressing the 1985 American Association of Artificial Intelligence, Woody Bledsoe (1986), an AI pioneer at the University of Texas at Austin, spoke of his continuing dream to build a computer friend that could "understand, act, autonomously think, learn, enjoy, hate."

The AI researchers, in their search for the essential qualities of humanity, emphasize the computational aspects of re-creating capabilities such as reasoning, learning, and problem-solving on the computer. On the other hand, animation artists seek to reproduce life forms from nothing more than simple line drawings, inks, and celluloids that move frame by frame. The practical requirements of producing hundreds of thousands of such drawings forced animators to use extremely simple imageries, and to seek and abstract precisely that which is crucial.

Bates argued that, as a result, although the scientists might have been more effective in re-creating life with the help of a computer, it is the artists who have come closest to capturing the essence of humanity. The insights of character animators in their artistic inquiry might thus be key to building computational models of interactive agents that are believable.

The Oz Project

The Oz Project at Carnegie Mellon University is an experiment in how the work of programmers and animators could be combined to create visible, human-like entities with which humans could eventually work or play.

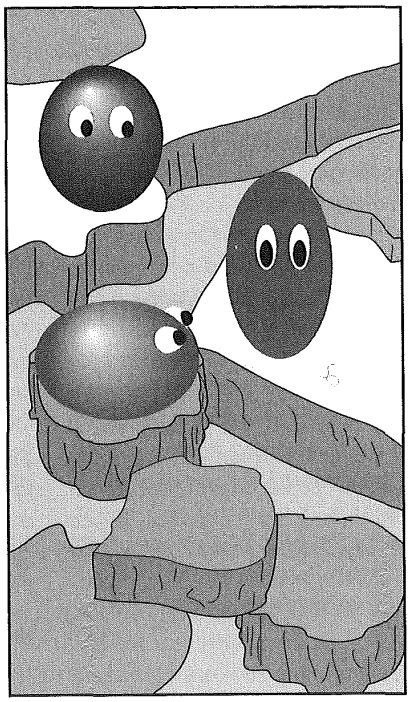
Bates is leading the Oz Project group to build a small, simulated world containing several real-time, interactive, self-animating creatures called Woggles.

The Woggles have names like Bear, Shrimp, and Wolf. As shown in figure 1.11, each Woggle is animated as a 3D oval or egg-shaped spherical entity with a pair of eyes, using the principles of traditional animation. At each moment, several Woggles are often seen moving, jumping, and gesturing socially on the screen.

IPR2025-00714
DATABRICKS EX1008 Page 44

Figure 1.11

Woggles have names like Bear, Shrimp, and Wolf.



In using emotions to construct believable agents, the Oz Project researchers needed to devise an internal representation of emotion inside the agent that was consistent with the appearance of its definite emotional state. The researchers developed a goal-directed, behavior-based architecture for action (Brooks 1986; LB 1983; Maes 1989). This action architecture is then coupled with a module for generating, representing, and expressing emotion (OCC 1988; BLR 1992).

The action system uses a minimalist notion of goals to manipulate a dynamically changing set of behaviors. The agents appraise surrounding events that occur with respect to their goals. This enables an agent to arrive at a clearly defined emotional state and to produce a definite emotional reaction to the event.

When a Woggle fails to reach an important goal, for example, and thinks the failure was caused by the action of another Woggle, it enters the angry state. In Woggles, each emotion is mapped in a personality-specific way to a behavioral feature of the Woggle. In this way, the emotional state of each Woggle can be made externally visible through its characteristic behaviors. In a fear state, for example, a Woggle whose fear is mapped to the aggressive feature behaves accordingly.

IPR2025-00714
DATABRICKS EX1008 Page 45

The believability of an interactive agent depends on the appearance of reactivity, emotions, goals, and situated social competence, among other things. In order to present a convincing illusion of life in agents, Bates suggests AI researchers should attempt a methodological emphasis on the emotional dimension of agents.

Agents and Programming

The programming aspects of agents are an important consideration, too. This section describes not so much the nuts and bolts of agent programming with traditional programming languages, but more the higher-level programming support offered for programming agents. KidSim handles agent programming without the use of a programming language, and Oasis offers explicit support for programming with distributed agents on a network of machines.

KidSim

David Smith, Allen Cypher, and Jim Spohrer from Apple Computer's Advanced Technology Group view the question of how to instruct agents as an end-user programming problem, currently an unsolved one in computer science. They believe computer scientists have not made programming easy enough for most people. They cite as evidence the fact that only a tiny fraction of computer users are able to program, although most can follow a recipe, give directions, make up stories, or plan trips—mental activities similar to those involved in programming (1994).

After observing that most computer users are proficient with some kind of editor or editor-like applications (such as drawing packages or painting programs), Smith, Cypher, and Spohrer decided to make programming as easy as editing. They have developed KidSim, for "Kids' Simulations," which is a toolkit that enables children to build symbolic simulations. The key idea in KidSim is the way in which children specify the behavior of agents, accomplished by combining two powerful techniques—graphical rewrite rules and programming by demonstration—into KidSim to improve the end-user's ability to program agents (SCS 1994).

Simulation Toolkit

In KidSim, kids can modify the programming of existing simulation objects and define new ones from scratch. A KidSim simulation primarily consists of the following components:

- A game board, divided into finite squares like a checkerboard
- A clock, whose time is divided into discrete (as versus continuous) ticks
- One or more simulation objects representing agents
- A copy box, the source of new simulation objects
- A rule editor, for defining and modifying rules

The game board represents the simulation microworld. The clock starts and stops a simulation. The clock can be run backward to undo changes, encouraging kids to experiment and take chances. The copy box is a container that automatically makes copies of simulation objects placed inside it.

IPR2025-00714
DATABRICKS EX1008 Page 46

In KidSim, the active objects in simulations are agents. During each clock tick, agents move around on the game board and interact with one another. Agents have their own visual appearance, characteristic properties (such as name, age, height, hunger, fear, and so on) and rules of behavior.

Graphical Rewrite Rules

The behaviors of agents are specified with graphical rewrite rules using the rule editor. A *graphical rewrite rule* is a transformation of a game board region from one state to another. It consists of a "before" part and an "after" part. Each part is a small scene that might occur during the simulation run. A rule matches if its "before" part is the same as some area of the game board at some moment in time. When a rule matches, KidSim transforms the corresponding region of the game board to the scene in the "after" part of the rule using a recorded program.

Programming by Demonstration

This recorded program is obtained from *programming by demonstration*, a technique in which the user puts the system in "record" mode and continues to operate the system in the normal way. The user's actions are then faithfully recorded in an executable program and can be replayed later as needed (Smith 1977; Cypher 1993). KidSim uses graphical rewrite rules as visual reminders of recorded actions, thus solving the problem of users trying to understand what the agents are supposed to do.

As reported by Smith, Cypher, and Spohrer in 1994, the KidSim approach appears to solve the end-user programming problem for some types of simulations. Perhaps we can derive from this that a solution to the general end-user programming problem probably lies some distance further down the same path.

Oasis

While in graduate school at the University of Michigan, the author had the opportunity to design a new programming language, called Oasis (Object and Agent Specification and Implementation System), for experimentation with agent-oriented programming (Cheong 1992a).

Oasis Agents

Oasis explicitly supports the concept of agents in its model of computation. In Oasis, agents are coarse-grained, computational entities that are dynamically created by the Oasis runtime system. Oasis agents are implemented as Unix processes that are distributed across the heterogeneous network of workstations. A collection of agents can thus cooperate among themselves to effect computations in a parallel distributed fashion on the network.

Each agent supports multiple threads of control using a non-preemptive scheduler. These threads can synchronize among themselves on condition variables specified by the programmer in the agent program. The threads facility enables an agent to accept and initiate multiple remote procedure calls concurrently.

Oasis Objects

Oasis objects are information nuggets that are dynamically created by agents during computation. Oasis objects do not have an identity in the traditional sense of object-oriented databases. They can

IPR2025-00714
DATABRICKS EX1008 Page 47

thus be freely transferred, traded, or replicated among the agents during cooperative computations without the constraints and programming hassle of maintaining consistency through locking. Objects no longer needed are automatically recycled through a garbage collector which, unlike traditional garbage collectors, does not require the use of runtime tags. The Oasis runtime system provides full support for automatic marshaling of objects, including complex user-defined objects with pointers, for remote procedure calls.

Oasis Compiler System

The Oasis compiler system generates native machine code but is network transparent in the sense that Oasis programmers need not be aware of workstation heterogeneity in the computing environment. In other words, the Oasis programmer does not have to maintain separate versions of binary code for different machine architectures. This is possible because the generation of native code at individual target machines is delayed until just before the agent program is actually run.

Oasis has been used to program a group of agents that cooperatively solve the Traveling Salesman Problem (in about 100 lines of Oasis code). The solution proceeds in a parallel distributed fashion on a cluster of workstations, with respectable speedups on different problem sizes.

The Oasis compiler generates native code for four different processors: Sparc, Mips, PowerPC, and 680x0. Its runtime system has been ported to several Unix platforms, including Sun-OS on Sparcstations, Ultrix on Decstations, Aix on PowerPC's, and Nextstep on Nextstations. The Oasis source code is publicly available by FTP from the University of Michigan at the following address (Cheong 1993b):

```
ftp://ftp.eecs.umich.edu/software/oasis/
```

Agents and Society

Donald Norman, an Apple Fellow at Apple Computer, foresees that the major difficulties with agents in our society are that people might not be comfortable with the autonomous actions of agents.

Norman observes that a distinguishing feature of the new crop of agents, as compared with mechanical robots of an earlier era, is that they now possess computational power, when previously they were simply servo mechanisms and control devices. According to Norman (1994), agents now:

> [H]ave Turing-machine powers, they take over human tasks, and they interact with people in human-like ways—perhaps with a form of natural language, perhaps with animated graphics or video. Some agents have the potential to form their own goals and intentions, to initiate actions on their own without explicit instructions or guidance, and to offer suggestions to people. Thus, agents might set up schedules, reserve hotel and meeting rooms, arrange transportation, and even outline meeting topics, all without human intervention. Moreover, today's agents are simple in comparison to those that are being planned.

IPR2025-00714
DATABRICKS EX1008 Page 48

Control

Indeed, it is important that people feel in control of their computational systems as a result of these added powers. They must be comfortable with actions performed for them by their agents. This can be accomplished in part through a better understanding of the underlying agent technology and in part through confidence in the system. Some people will always want to know the actions of their agents.

Over Expectations

Norman cautions that the added computational power of agents easily can foster an overblown expectation in people's mind of their exaggerated capabilities. People have a tendency to *anthropomorphize*, to see human attributes in anything that is the least bit intelligent. When fueled by the enthusiasm of technology visionaries who sees far into the future and amplified by the inclination of researchers to show their agents in human form, people naturally, but falsely, build on expectations of human-like intelligence, understanding, and actions in such personified agents.

Safety

Safety plays a part in the feeling of control, as does the issue of privacy. Agents should not do things that jeopardize the physical, mental, and financial well-being of human users. This can be tricky given that malicious agents in the form of computer worms and viruses can arrive unannounced and wreak havoc on the system.

Privacy

The question of privacy is an even more complex topic. The idea of autonomous, intelligent agents having access to one's personal records, correspondence, and financial activities can be somewhat disconcerting. Moreover, with embedded agents in e-mail messages, it might be difficult to safeguard one's privacy from the action of foreign agents collecting a recipient's private information and transferring it back to the senders.

Agent technology promises deliverance to computer users, relieving them from the complexity of command languages and the tedium of direct manipulation with intelligent, agent-guided interactions. Agents also can enhance human performance by making people appear smarter, or hide complexity by automating actions that you do not know how or prefer not to do.

Along with such promises comes the potential for social mischief, loss of privacy, and technological alienation from feelings of loss of control. But all these problems can be solved, though, provided enough consideration is given in the early design stages of intelligent systems of which agents are a part.

Commercial Future of Agents

Much of the agents discussed in the preceding sections exist only in universities and research labs. They have not made it to the commercial mainstream, yet. But according to Irene Greif (1994),

IPR2025-00714
DATABRICKS EX1008 Page 49

Director of Workgroup Technologies at Lotus, two industry trends could influence the evolution of agent technology and push agents out of the labs and into the mainstream of PC software:

- The move toward suites of internetworking desktop products
- The growing population of mobile users

Product Suites

A suite is a set of desktop applications that has been integrated to reduce the cost of software ownership and to improve individual productivity. Greif expects that agents will make an impact on suite products through "task-oriented" conversations with the users.

User interfaces today, for example, converse with users in a stylized fashion in the form of dialog boxes. This communication will become more powerful if they can converse about richer database structures, such as explicitly represented models of tasks in the form of task descriptions, which are similar to work process descriptions used by workflow agents (MWFF 1992). Greif envisions that the next significant step in the user interface will be a move away from conversing through forms to conversing about task descriptions. When this happens, the interaction between users and agents will become more like a collaboration through explicit data structures that represent tasks.

Mobile Computing

In the area of mobile computing, agents will add a new richness to the user interface. As people change their locations and work environments more frequently, they will continue to expect the same level of support from mobile computing, despite the vastly different capacity of the connectivity model. Greif envisions a personalized agent that understands where you are, what you are doing, and how you can best be reached. It is a new kind of agent in that instead of finding and doing things for a user in the network, it actually is interacting with other agents on the user's behalf.

To illustrate, consider the following example of a mobile user who is accessible only by pager and wants to read news articles about certain companies. Greif explains:

> It might not make sense for any of these articles to be forwarded to her when she only has her pager and can't read anything. However, if her calendar shows that she's on her way to visit the XYZ Co., it might be worth sending a message to her pager that there is a news item about that company. From an icon in the pager screen, she should be able to easily send a request back to her agent to have the full article faxed to the hotel before her breakfast meeting.

Most agents find something, take an action, and then move on. The interesting thing to note here is that, in this case, the agent might have to deliver the same piece of information several times, and in different formats (as a brief note to her pager and the full article to the hotel computer).

IPR2025-00714
DATABRICKS EX1008 Page 50

Concluding Remarks

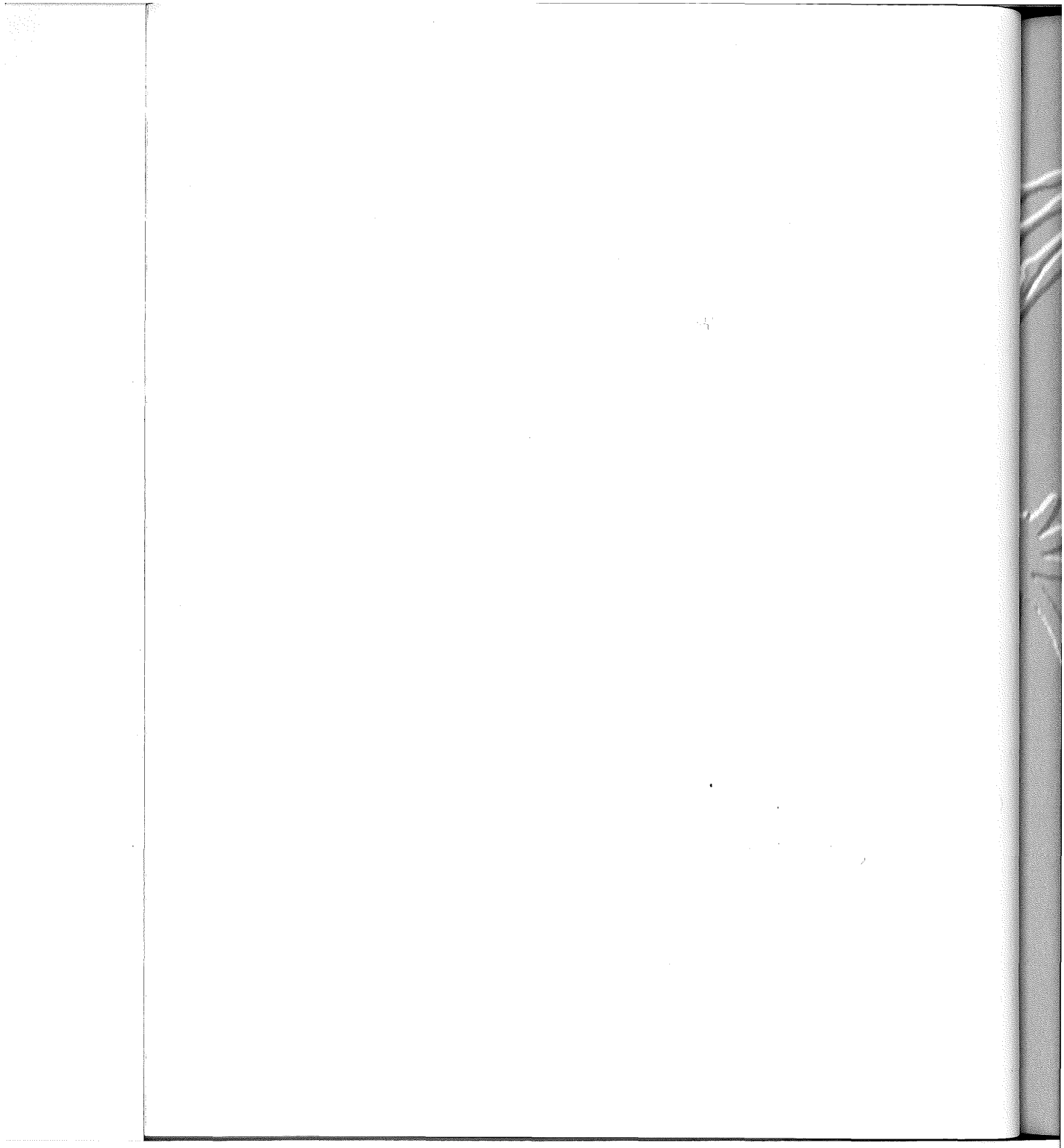
I do not have a separate category for intelligent agents. I think it would not do justice to agents described here to contemplate using a separate category of agents called *intelligent agents*, and to use this category for the purpose of taxonomic classification by including some in the category but excluding others.

Agents display their intelligence differently; some by being creative, some by being crafty and elusive (worms and viruses), some by being helpful (personal assistants and surrogate bots), and still others by being resourceful in their own ways (COACH). In addition, agents can use different means to achieve intelligence; some adopt heuristics (softbots), some others use constraints (ACDS), some depend on knowledge databases (Cyc), and yet others learn from experience (Calendar Apprentice).

I consider all of the agents described here as intelligent, but to different degrees. Whether they possess insect-level intelligence or command Cyc-style encyclopedic world knowledge, it does not really matter. A colony of simple autonomous insects sometimes can display more intelligent behavior than a complex omnicient robot. In my opinion, intelligence is simply too vague a term at the current state of the art in agent research to even be considered a useful taxonomic category for classifying and understanding the wide variety of agents in the world.

The following chapters take you on a tour to visit agents on the Internet: the Web robots, spiders, and wanderers; the Web shoppers and bargain-hunters; worms and viruses; as well as MUD agents and chatterbots. But first, the Internet, then the Internet agents.

IPR2025-00714
DATABRICKS EX1008 Page 51



chapter 2

The Internet: Past, Present, and Future

This chapter tracks the development history of networks in the U.S. that have led to the Internet as we know it today: the network of networks. We have relied on Peter Denning's excellent article for materials that relate to the first 20 years in the history of ARPAnet (Denning 1989).

In the 1950s, the U.S. Department of Defense was worried about the ability of U.S. forces to survive a nuclear first strike. Survival depended critically on

IPR2025-00714
DATABRICKS EX1008 Page 53

the durability of the communications network. Traditional telephone networks that used circuit switching technology were considered too fragile for the purpose. The Rand Corporation, a defense contractor, undertook a series of studies and came up with the recommendation that the communications network should be based upon a packet switching technology. With packet switching, instead of using fixed point-to-point connections between any pair of machines for communications, messages are divided into packets. These packets are independently routed between intermediate computers until they reach their final destination, whereupon the message is reassembled for final delivery.

About the same time, experiments were conducted around the world to investigate the new packet switching technology, which promised tremendous flexibility and reliability in connecting computers at various sites. The first packet-switching network was implemented at the National Physical Laboratory in England. It was quickly followed by ARPAnet in the U.S. in 1969.

Early Days of ARPAnet

In 1968, against the backdrop of the Cold War with Russia, the Defense Department's Advanced Research Projects Agency (ARPA) commissioned the Bolt Beranek and Newman (BBN) company to build the first Interface Message Processors (IMPs).



MPs are dedicated network controlling computers that translate between messages and packets.

By the end of 1969, BBN had delivered the first four IMPs along with a packet-switching network protocol called the *Network Control Protocol* (NCP). The first IMP was installed at UCLA in the fall of 1969. By 1970, the first packet-switched computer network in the U.S. was created, with four operating nodes connecting UCLA, U.C. Santa Barbara, Stanford University, and the University of Utah. This was the beginning of the ARPAnet. If any one link in the network failed, packets could still be routed via the remaining links, thus providing the needed fault tolerance and reliability. By 1971, there were 15 nodes on ARPAnet. By 1973, ARPAnet had grown to 37 nodes (Denning 1989).



Electronic mail very quickly became the major source of traffic on ARPAnet, although it was not mentioned among the original goals of ARPAnet.

The first public demonstration of ARPAnet was held in 1972, arranged by Robert Kahn of BBN, at the first International Conference on Computer Communications in Washington, D.C. It soon became clear that research networking was growing rapidly and that the ARPAnet needed to connect to other networks. A working group, chaired by Vinton Cerf of UCLA, was established to study the creation of a common protocol for internetwork communications.

In 1973, the newly renamed Defense Advanced Research Project Agency (DARPA) began a research program to investigate techniques and technologies for connecting various types of packet-switched networks together. This was called the Internetting

IPR2025-00714
DATABRICKS EX1008 Page 54

project, and the main internet that resulted from it was called the Internet (Cerf 1992).

In 1974, Vinton Cerf and Robert Kahn released the Internet Protocol (IP) and the Transmission Control Protocol (TCP), which define the way data are passed among machines in a packet-switched network. The first physical implementation of the Internet involved four networks: a packet satellite network, a packet radio network, the ARPAnet, and an Ethernet at the Xerox Palo Alto Research Center (QCM 1994).

Notable Computer Networks

Toward the end of the 1970s, various "community networks" began to emerge (QH 1986; QCM 1994). Notable examples include the following:

- CSNET, which connects computers in the computer science research community
- BITNET, which connects IBM machines in computing centers
- USENET, which connects Unix sites by UUCP or other means
- FidoNet, which connects MS-DOS PCs by phone lines
- Various internal corporate networks, for example, IBM VNET, DEC Easynet, Xerox Internet

By the late 1970s, the ARPAnet was serving a select number of research centers. However, not all universities had network connections. The University of Wisconsin discerned a need and decided to create a network for increased collaboration among computer science researchers. The Computer Science Research Network (CSNET) was thus formed in January 1981, funded in large part by the National Science Foundation. Most CSNET hosts didn't use TCP/IP; instead, many were connected by modems and phone lines and used dial-up protocols that permitted essentially one service: e-mail.

Vinton Cerf had suggested connecting ARPAnet and CSNET via a gateway using the TCP/IP protocols. It also was suggested that CSNET could exist as a collection of several independent networks sharing a gateway to the ARPAnet (Moore 1994a). This marked the beginning of the Internet as a collection of independent, free-standing networks that came to an agreement on how to communicate with each other. By 1982, researchers on CSNET could communicate with sites within CSNET and ARPAnet by e-mail with equal ease. In a limited sense, the Internet had taken a first step to becoming "the network of networks."

In May 1981, BITNET (Because It's Time Network) was formed. BITNET has a tendency to link computer centers together and was created to connect IBM mainframes at the City University of New York. BITNET was built using the Network Job Entry (NJE) protocol and software native to the IBM VM/370 operating system.

BITNET uses the Listserv mechanism for providing news services. *Listserv* was a program originally designed to act as a mailing-list server whose function is to distribute e-mail to users on a mailing list. Listserv can thus be considered a rudimentary form of Internet during the early evolution of the Internet. It's somewhat like the USENET newsgroup

IPR2025-00714
DATABRICKS EX1008 Page 55

concept. The difference is that the readers of a particular newsgroup need to first subscribe to the appropriate Listserv, and that news articles are sent directly via e-mail rather than broadcast throughout the network.

USENET (Users' Network) is not a physical network. It began in 1980 as a medium of communications between users of two machines, one at the University of North Carolina, the other at nearby Duke University. USENET newsgroups were invented to capture the flavors of both the ARPAnet-style mailing lists as well as the bulletin board services. The early USENET news distribution mechanism depended on Unix-to-Unix-Copy (UUCP) for transport of news articles over telephone links using a simple "flooding" algorithm (Horton 1983). When USENET became much larger, a more efficient protocol for delivering and accessing news, that is, the Network News Transfer Protocol (NNTP), was adopted (KL 1986). In addition to UUCP, USENET news can be carried on BITNET, as well as the larger Internet.



USENET has enjoyed immense popularity since its inception, reaching a large constituency and growing rapidly to encompass 2,000 machines in 1986.

FidoNet was invented in 1983 to connect personal computers running MS-DOS via modems and phone lines (see fig. 2.1). It was designed by Tom Jennings of San Francisco as an imitation of UUCP and USENET to link together Fido bulletin boards that had recently sprung up across the nation. The Fido protocols offer similar functions as that of UUCP but are completely different internally and more efficient. It allows users to send e-mail to each other and to create discussion groups just like USENET and BITNET. Starting in 1987, FidoNet

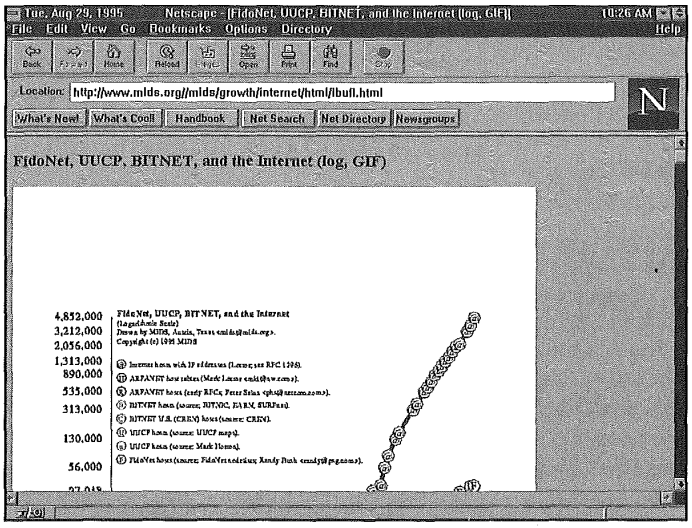


Figure 2.1

FidoNet, UUCP, BITNET, and the Internet. (Courtesy of MIDS, Austin, Texas)

IPR2025-00714
DATABRICKS EX1008 Page 56

could share traffic with USENET after the Unix-to-Unix-Copy (UUCP) software originally developed for the Unix platform was ported to MS-DOS.



Interestingly, the name Fido is not an acronym but the common pet name for a family dog.

In 1989, BITNET and CSNET merged to become the Corporation for Research and Education Networking (CREN). But CSNET was subsequently retired when the NSFnet regional networks subsumed its functions.

Internet and NSFnet

The first pieces of the Internet began around 1980 when DARPA began converting machines attached to its research networks to the new TCP/IP protocols. The transition to Internet technology was completed in January 1983 when DARPA mandated that all computers connected to the ARPAnet use TCP/IP. At around the same time, the original ARPAnet was split into two networks: ARPAnet for continued research and MILNET for military operations.

To encourage adoption of the new protocols, DARPA had decided to make an implementation of the TCP/IP available at a low cost. At that time, the computer science departments at most universities were running a version of Unix available from the University of California at Berkeley as part of its Berkeley Software Distribution. DARPA funded BBN to implement the Internet protocols under Unix and Berkeley to integrate them with its distribution. With TCP/IP networking support built into BSD 4.2 Unix, DARPA could reach over 90 percent of the university computer science departments in the U.S. As the Internet grew, the original method of naming nodes became unwieldy. A hierarchical naming system that allowed each *domain* to select its internal address was introduced in 1984.

In 1984, the National Science Foundation started connecting its supercomputing centers with a high-bandwidth network called the NSFnet. The first NSFnet was built by the Cornell Theory Center and the National Center for Supercomputing Applications (NCSA). The NSFnet started out as a 56 Kbps network in 1986, primarily serving the NSF's six supercomputer centers. In its lifetime (1986 to 1995), NSFnet had undergone several iterations over the implementation of its backbone, upgrading to higher speed at each stage: in 1986 (DS-0, 56 Kbps), 1988 (T-1, 1.544 Mbps), and 1990 (T-3, 45 Mbps).

Merit, a non-profit network corporation based in Michigan, began managing the NSFnet backbone in July, 1988, after working in partnership with IBM and MCI to deliver the initial T1 backbone to NSFnet, which connected 13 sites. The NSFnet had become the backbone for the potpourri of networks known collectively as the Internet.

Beginning in 1986, the National Science Foundation (NSF) supplied seed money to support the mid-level regional networks that provided extensive connectivity for campus networks at educational institutions, government agencies, and commercial businesses. The NSFnet had thus played a key role in further accelerating the already rapid growth of the Internet.

In 1990, after 20 years of service, the ARPAnet was officially retired. ARPAnet's role was, for all

IPR2025-00714
DATABRICKS EX1008 Page 57

practical purposes, supplanted by the NSFnet. In the same year, NSF created the Advanced Network Services Inc. (ANS), a non-profit corporation jointly owned by Merit, IBM, and MCI. In 1990, ANS took over the operation of the NSFnet backbone, which by then was already operating at T3 speeds (45 Mbps) using circuits provided by MCI and router technology from IBM. By the end of 1991, all NSFnet backbone sites were connected to the new ANS-provided T3 backbone (Merit 1992).

NSF and AUP

The NSF and the ANS were very generous in sharing the network backbone. The NSFnet services are available to any Internet user as long as NSF's acceptable-use policy (AUP) is adhered to. The acceptable-use policy basically states the following general principle:

> NSFnet Backbone services are provided to support open research and education in and among US research and instructional institutions, plus research arms of for-profit firms when engaged in open scholarly communication and research. Use for other purposes is not acceptable.

In particular, the AUP states the following as unacceptable use:

- Use for for-profit activities, unless covered by the General Principle or as a specifically acceptable use
- Extensive use for private or personal business

In other words, nearly anyone can use the NSFnet backbone as long as it is not used for profit or used extensively for private or personal business.

In 1990, the Federal Networking Council, as part of the governing body of the Internet, made a radical policy change. It no longer required organizations that wanted to join the Internet to seek sponsorship by a U.S. government agency. This event marked the start of the "commercialization" of the Internet (Moore 1994b).

In 1992, in extending ANS's contract to run NSFnet, NSF considered itself a customer of ANS. As a result, the limitations outlined by the acceptable-use policy applied only to traffic from the NSF (Moore 1994b). The expectation was that different organizations on the Internet would formulate their own acceptable-use policies regarding their portions of the Internet. For all practical purposes, the floodgate had finally opened for commercial use of the Internet.

Growth of the Internet

NSFnet performance statistics have been collected, processed, and reported by the Merit Network since 1988. In December 1994, the numbers contained in Merit's statistical reports began to decrease, as NSFnet traffic began to migrate to the new NSF network architecture.

In the new architecture, traffic is exchanged at interconnection points called *NAPs* (*Network Access Points*). Each NAP provides a neutral interconnection point for network service providers.

IPR2025-00714
DATABRICKS EX1008 Page 58

On April 30, 1995, the NSFnet Backbone Service was successfully transitioned to the new network architecture, signaling the end of the NSFnet project.

By any measure, the growth of the Internet has been impressive. As illustrated in table 2.1, Merit recorded a 134 percent growth (from 6031 to 14,121) in networks configured for traversal of the NSFnet backbone from July 1992 to July 1993. Mark Lottor, formerly at SRI but now at Network Wizards, used the ZONE program to determine the approximate number of Internet hosts and domains (see RFC 1296 (Lottor 1992)). His Internet Domain Survey of July 1993 shows 79 percent growth in Internet hosts in the year from July 1992. His October 1993 report shows 81 percent growth in hosts (see table 2.2) and 55 percent in domains from October 1992 to October 1993 (see table 2.3).

Table 2.1 History of NSFNet Growth by Networks

Date	Total Nets	Total Non-US
Jul 88	217	9
Jan 89	384	34
Jul 89	650	99
Jan 90	1,233	250
Jul 90	1,727	436
Jan 91	2,338	693
Jul 91	3,086	1,012
Jan 92	4,526	1,496
Jul 92	6,031	2,133
Jan 93	9,117	3,413
Jul 93	14,121	5,827
Jan 94	23,494	9,869
Jul 94	36,153	15,362
Jan 95	46,318	19,637

Table 2.2 Growth of Internet Hosts

Date	Hosts
Aug 81	213
Aug 83	562
Oct 85	1,961
Dec 87	28,174
Oct 89	159,000
Oct 90	313,000
Oct 91	617,000
Oct 92	1,136,000
Oct 93	2,056,000
Oct 94	3,864,000
Jan 95	4,852,000

IPR2025-00714
DATABRICKS EX1008 Page 59

Table 2.3 Growth of Internet Domains

Date	Domain
Jul 88	900
Jul 89	3,900
Oct 90	9,300
Jul 91	16,000
Oct 92	18,100
Oct 93	28,000
Oct 94	56,000
Jan 95	71,000

What do all these growth rates for networks, hosts, and domains mean? For one, the slowest growth seems to be in domains, which probably means that organizations join the net more slowly, but increase their host counts rapidly after they are connected (Quarterman 1993). For another, the fastest growth is in the number of networks configured for traversal of the NSFnet backbone (presumably a large fraction of all IP networks on the Internet), which probably indicates the important role played by NSFnet in connecting many previously isolated networks.

Most of these measures of Internet growth show sustained exponential growth (note the vertical scale is logarithmic). According to Matrix Information and Directory Service (MIDS) (see fig. 2.2), averaging across these figures gives us a rough count of approximately 100 percent annual growth (Quarterman 1993).

Another way to gauge the growth of Internet is by the volume of traffic. Table 2.4 shows the growth in traffic volume on the NSFnet backbone.

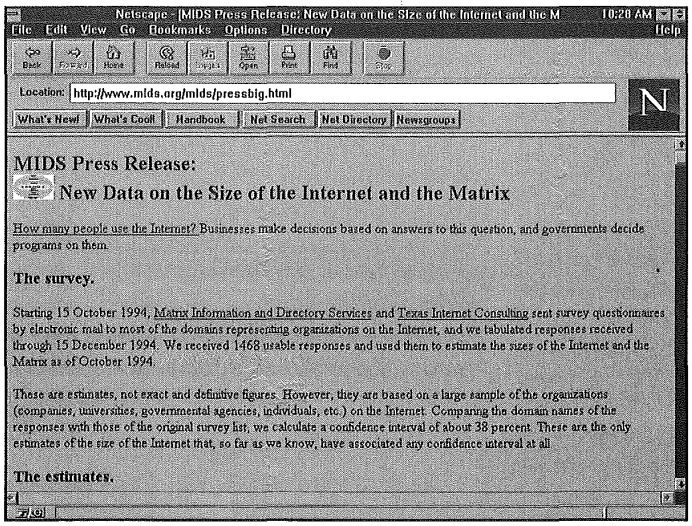


Figure 2.2

A press release on the MIDS home page.

IPR2025-00714
DATABRICKS EX1008 Page 60

Table 2.4 NSFnet Byte Traffic History (in billions of bytes)

Month	1991	1992	1993	1994	1995
Jan	NA	2,256	4,782	8,609	13,196
Feb	NA	2,371	5,015	9,303	9,790
Mar	1,268	2,761	6,053	11,226	11,218
Apr	1,402	2,848	6,219	11,587	5,316
May	1,442	3,061	5,845	12,187	NA
Jun	1,244	3,274	6,195	12,466	NA
Jul	1,594	3,373	6,389	12,764	NA
Aug	1,484	3,200	6,631	13,385	NA
Sep	1,769	3,315	7,022	14,990	NA
Oct	1,879	3,903	8,468	17,232	NA
Nov	1,959	4,651	8,483	17,781	NA
Dec	1,956	4,372	8,283	16,313	NA

How Big is the Internet?

According to Tony Rutkowski, Executive Director of the Internet Society, a commonly used method of estimating the total number of Internet users is to multiply the number of host computers by 10. In January 1995, for example, the ZONE program identified close to 5 million hosts, which is equivalent to about 50 million users.

A more detailed breakdown of such a measure of Internet users is provided by MIDS, a company that conducts ongoing investigations about the size, shape, and other characteristics of the Internet and other networks. Combining and processing data from a variety of sources, MIDS estimated the size of the Internet as of October 1994 to be as such:

- 7.8 million users of 2.5 million computers (MIDS calls this the core Internet) that can provide interactive services such as remote login, file transfer or World Wide Web
- 13.5 million users of 3.5 million computers that can use the interactive services supplied by the core Internet; for example, people who can use Mosaic or Lynx to browse the World Wide Web
- 27.5 million users who can exchange electronic mail with other users on the Internet, as well as other networks

IPR2025-00714
DATABRICKS EX1008 Page 61

The following figures provide further details on the distribution of the Internet by geography, both international (see fig. 2.3 and table 2.5) and U.S. (see table 2.6), as well as by top-level domain names (see table 2.7).

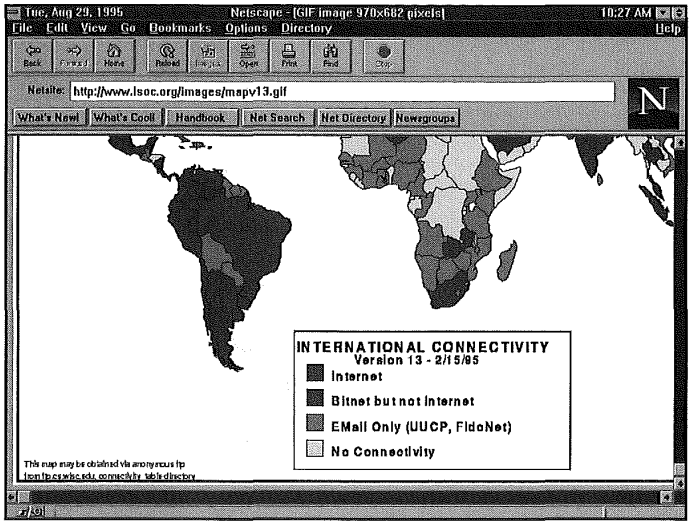


Figure 2.3

Global Connectivity Map. (Courtesy of Internet Society).

Total Code	Initial Country	Nets	Connection
DZ	Algeria	3	Apr 94
AR	Argentina	27	Oct 90
AM	Armenia	3	Jun 94
AU	Australia	1,875	May 89
AT	Austria	408	Jun 90
BY	Belarus	1	Feb 95
BE	Belgium	138	May 90
BM	Bermuda	20	Mar 94

Table 2.5

NSFNET International Connections and Nets

IPR2025-00714
DATABRICKS EX1008 Page 62

Total Code	Initial Country	Nets	Connection
BR	Brazil	165	Jun 90
BG	Bulgaria	9	Apr 93
BF	Burkina Faso	2	Oct 94
CM	Cameroon	1	Dec 92
CA	Canada	4,795	Jul 88 *
CL	Chile	102	Apr 90
CN	China	8	Apr 94
CO	Colombia	5	Apr 94
CR	Costa Rica	6	Jan 93
HR	Croatia	31	Nov 91
CY	Cyprus	25	Dec 92
CZ	Czech Republic	459	Nov 91
DK	Denmark	48	Nov 88
DO	Dominican Republic	1	Apr 95
EC	Ecuador	85	Jul 92
EG	Egypt	7	Nov 93
EE	Estonia	49	Jul 92
FJ	Fiji	1	Jun 93
FI	Finland	643	Nov 88
FR	France	2,003	Jul 88 *

continues

IPR2025-00714
DATABRICKS EX1008 Page 63

Table 2.5, Continued
NSFNET International Connections and Nets

Total Code	Initial Country	Nets	Connection
PF	French Polynesia	1	Oct 94
DE	Germany	1,750	Sep 89
GH	Ghana	1	May 93
GR	Greece	105	Jul 90
GU	Guam	5	Oct 93
HK	Hong Kong	95	Sep 91
HU	Hungary	164	Nov 91
IS	Iceland	31	Nov 88
IN	India	13	Nov 90
ID	Indonesia	46	Jul 93
IE	Ireland	168	Jul 90
IL	Israel	217	Aug 89
IT	Italy	506	Aug 89
JM	Jamaica	16	May 94
JP	Japan	1,847	Aug 89
KZ	Kazakhstan	2	Nov 93
KE	Kenya	1	Nov 93
KR	South Korea	476	Apr 90
KW	Kuwait	8	Dec 92
LV	Latvia	22	Nov 92
LB	Lebanon	1	Jun 94

IPR2025-00714
DATABRICKS EX1008 Page 64

Total Code	Initial Country	Nets	Connection
LI	Liechtenstein	3	Jun 93
LT	Lithuania	1	Apr 94
LU	Luxembourg	59	Apr 92
MO	Macau	1	Apr 94
MY	Malaysia	6	Nov 92
MX	Mexico	126	Feb 89
MA	Morocco	1	Oct 94
MZ	Mozambique	6	Mar 95
NL	Netherlands	406	Jan 89
NC	New Caledonia	1	Oct 94
NZ	New Zealand	356	Apr 89
NI	Nicaragua	1	Feb 94
NE	Niger	1	Oct 94
NO	Norway	214	Nov 88
PA	Panama	1	Jun 94
PE	Peru	44	Nov 93
PH	Philippines	46	Apr 94
PL	Poland	131	Nov 91
PT	Portugal	92	Oct 91
PR	Puerto Rico	9	Oct 89
RO	Romania	26	Apr 93

continues

IPR2025-00714
DATABRICKS EX1008 Page 65

Table 2.5, Continued

NSFNET International Connections and Nets

Total Code	Initial Country	Nets	Connection
RU	Russian Federation	405	Jun 93
SN	Senegal	11	Oct 94
SG	Singapore	107	May 91
SK	Slovakia	69	Mar 92
SI	Slovenia	46	Feb 92
ZA	South Africa	419	Dec 91
ES	Spain	257	Jul 90
SZ	Swaziland	1	May 94
SE	Sweden	415	Nov 88
CH	Switzerland	324	Mar 90
TW	Taiwan	575	Dec 91
TH	Thailand	107	Jul 92
TN	Tunisia	19	May 91
TR	Turkey	97	Jan 93
UA	Ukraine	60	Aug 93
AE	United Arab Emirates	3	Nov 93
GB	United Kingdom	1,436	Apr 89
US	United States	28,470	Jul 88 *
UY	Uruguay	1	Apr 94
UZ	Uzbekistan	1	Dec 94

IPR2025-00714
DATABRICKS EX1008 Page 66

Total Code	Initial Country	Nets	Connection
VE	Venezuela	11	Feb 92
VN	Vietnam	1	Apr 95
VI	Virgin Islands	4	Mar 93
93	Total	50,766	

* Merit began managing the NSFNET backbone in July, 1988.

Table 2.6

NSFnet Networks by U.S. States, May 1995

State	Code	Total Nets
Alabama	AL	260
Alaska	AK	26
Arizona	AZ	186
Arkansas	AR	70
California	CA	4,832
Colorado	CO	696
Connecticut	CT	463
Delaware	DE	23
Florida	FL	770
Georgia	GA	445
Hawaii	HI	127
Idaho	ID	56
Illinois	IL	577

continues

IPR2025-00714
DATABRICKS EX1008 Page 67

State	Code	Total Nets
Indiana	IN	347
Iowa	IA	147
Kansas	KS	70
Kentucky	KY	82
Louisiana	LA	198
Maine	ME	103
Maryland	MD	1,178
Massachusetts	MA	2,005
Michigan	MI	540
Minnesota	MN	867
Mississippi	MS	109
Missouri	MO	303
Montana	MT	37
Nebraska	NE	156
Nevada	NV	40
New Hampshire	NH	175
New Jersey	NJ	1,208
New Mexico	NM	142
New York	NY	2,152
North Carolina	NC	677
North Dakota	ND	21
Ohio	OH	1,233

Table 2.6, Continued

NSFnet Networks by U.S. States, May 1995

IPR2025-00714
DATABRICKS EX1008 Page 68

State	Code	Total Nets
Oklahoma	OK	136
Oregon	OR	593
Pennsylvania	PA	919
Rhode Island	RI	147
South Carolina	SC	240
South Dakota	SD	15
Tennessee	TN	353
Texas	TX	1,341
Utah	UT	141
Vermont	VT	68
Virginia	VA	1,964
Washington	WA	972
Washington DC	DC	744
West Virginia	WV	46
Wisconsin	WI	280
Wyoming	WY	28
Military, Asia	AA	10
Military, Europe	AE	92
Military, Pacific	AP	46
Military Unspecified	AX	8
(Unknown)	XX	6

IPR2025-00714
DATABRICKS EX1008 Page 69

Table 2.7 Host Distribution by Top-Level Domain Name, January 1995

com 1,316,966	it 30,697	gr 4,000	cn 569	li 27
edu 1,133,502	at 29,705	cl 3,054	ve 529	gb 27
uk 241,191	es 28,446	tr 2,643	bm 474	zw 19
gov 209,345	za 27,040	ru 1,849	in 359	am 19
de 207,717	dk 25,935	si 1,773	ph 334	jr 18
ca 186,722	be 18,699	th 1,728	ec 325	pa 17
mil 175,961	kr 18,049	my 1,606	kw 220	mo 12
au 161,166	tw 14,618	sk 1,414	id 177	dz 10
org 154,578	il 13,251	ee 1,396	uy 172	kz 7
net 150,299	hk 12,437	ar 1,262	pe 171	fj 5
jp 96,632	cz 11,580	co 1,127	eg 161	aq 4
fr 93,041	pl 11,477	hr 1,090	bg 144	md 3
nl 89,227	hu 8,506	int 904	lt 121	gl 3
se 77,594	mx 6,656	br 800	cy 88	fo 3
fi 71,372	ie 6,219	cr 798	pr 82	sa 2
ch 51,512	pt 5,999	lu 614	jm 76	gn 2
no 49,725	sg 5,252	lv 612	zm 69	by 2
us 37,615	su 4,963	ro 597	tn 57	az 1
nz 31,215	is 4,735	ua 574	ni 49	

IPR2025-00714
DATABRICKS EX1008 Page 70

Internet Society, IAB, and IETF

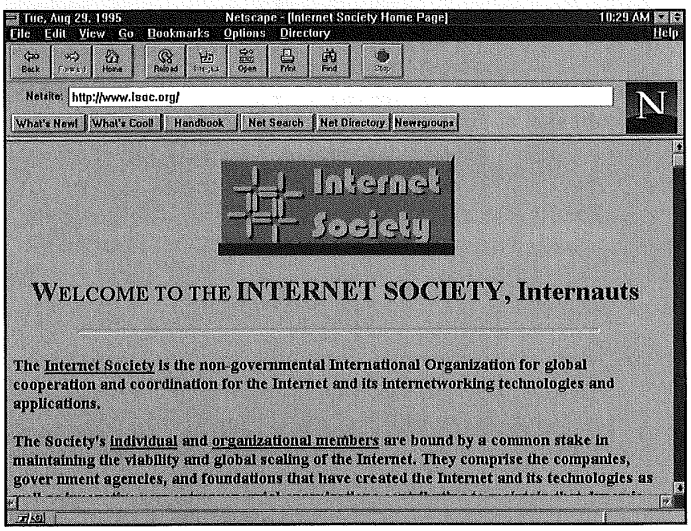
The Internet Society is an international organization for global cooperation and coordination for the Internet and its associated internetworking technologies and applications. Its principal purpose is

> [T]o maintain and extend the development and availability of the Internet and its associated technologies and applications—both as an end in itself, and as a means of enabling organizations, professions, and individuals worldwide to more effectively collaborate, cooperate, and innovate in their respective fields and interests.

The Internet Society (ISOC) (see fig. 2.4) was formed by a number of people with long-term involvement in the Internet Engineering Task Force (IETF) (see fig. 2.5). In 1990, it appeared that long-term support for the standards-making activity of the IETF, which had come primarily from research supporting agencies of the U.S. Government (notably ARPA, NSF, NASA, and DOE), might need to be supplemented in the future. As a result, one of its principal rationales was to provide an institutional home for and financial support for the Internet Standards process.

Figure 2.4

The Internet Society's home page.



IPR2025-00714
DATABRICKS EX1008 Page 71

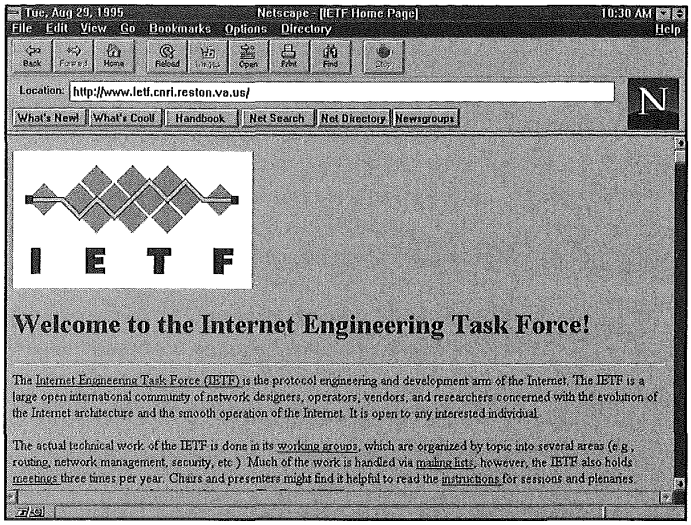


Figure 2.5

The IETF home page.

The Internet Society was announced in June 1991 at an international networking conference in Copenhagen and brought into existence in January 1992. In June 1992, at the annual meeting of the Internet Society, INET'92, in Kobe, Japan, the Internet Activities Board proposed to associate its activities with ISOC and was renamed the Internet Architecture Board (IAB).

The IAB is considered a technical advisory group of the ISOC. It is chartered to provide oversight of the architecture of the Internet and its protocols. Historically, the IETF and its sister organization, the Internet Research Task Force, had been considered two arms of the IAB. At the technical and developmental level, the Internet is made possible through creation, testing, and implementation of Internet Standards. These standards are developed by the Internet Engineering Task Force.

The IETF is a loosely self-organized group of people who make technical and other contributions to the engineering and evolution of the Internet and its technologies. The actual technical work of the IETF is done in its working groups, which are organized by topic into several areas (for example, routing, network management, and security).

The IETF produces a set of working documents, each called an RFC (Request for Comment). Some of these RFCs pass through the IAB Standards Process (Chapin 1992) to become Internet Standards. Internet Standards exist for all the basic TCP/IP protocols.

According to Vinton Cerf, the highest ISOC goal was to "keep the Internet going." Among the high priority activities associated with that goal was to provide support for the Internet Standards process carried out by the Internet Engineering Task Force.

IPR2025-00714
DATABRICKS EX1008 Page 72

Information Superhighway and the National Information Infrastructure

The first year of the Clinton administration saw the creation of the U.S. Advisory Council on the National Information Infrastructure as a new branch under the Commerce Department. Signed into President Clinton's executive order of 1993 was the national goal of creating an "information superhighway," the National Information Infrastructure (NII) which

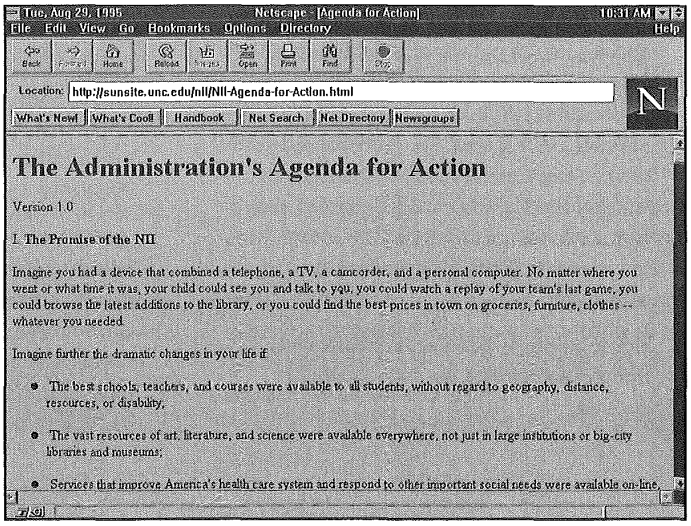
> [S]hall be the integration of hardware, software, and skills that will make it easy and affordable to connect people with each other, with computers, and with a vast array of services and information resources.

The following executive summary was excerpted from the U.S. Federal government's NII Agenda for Action (see fig. 2.6). It mentions nine goals for the NII that bear striking resemblances to what the Internet can offer today, but there are important differences as well.

> The National Information Infrastructure: *Agenda for Action Executive Summary*
>
> All Americans have a stake in the construction of an advanced National Information Infrastructure (NII), a seamless web of communications networks, computers, databases, and consumer electronics that will put vast amounts of information at users' fingertips. Development of the NII can help unleash an information revolution that will change forever the way people live, work, and interact with each other:

Figure 2.6

The Federal government's Agenda for Action home page.



IPR2025-00714
DATABRICKS EX1008 Page 73

➜ People could live almost anywhere they wanted, without foregoing opportunities for useful and fulfilling employment, by "telecommuting" to their offices through an electronic highway.

➜ The best schools, teachers, and courses would be available to all students, without regard to geography, distance, resources, or disability.

➜ Services that improve America's health care system and respond to other important social needs could be available online, without waiting in line, when and where you needed them.

Private sector firms already are developing and deploying that infrastructure today. Nevertheless, there remain essential roles for government in this process. Carefully crafted government action will complement and enhance the efforts of the private sector and assure the growth of an information infrastructure available to all Americans at reasonable cost. In developing our policy initiatives in this area, the Administration will work in close partnership with business, labor, academia, the public, Congress, and state and local government. Our efforts will be guided by the following principles and objectives:

➜ Promote private sector investment, through appropriate tax and regulatory policies.

➜ Extend the "universal service" concept to ensure that information resources are available to all at affordable prices. Because information means empowerment—and employment—the government has a duty to ensure that all Americans have access to the resources and job creation potential of the Information Age.

➜ Act as a catalyst to promote technological innovation and new applications. Commit important government research programs and grants to help the private sector develop and demonstrate technologies needed for the NII, and develop the applications and services that will maximize its value to users.

➜ Promote seamless, interactive, user-driven operation of the NII. As the NII evolves into a "network of networks," government will ensure that users can transfer information across networks easily and efficiently. To increase the likelihood that the NII will be both interactive and, to a large extent, user- driven, government must reform regulations and policies that may inadvertently hamper the development of interactive applications.

➜ Ensure information security and network reliability. The NII must be trustworthy and secure, protecting the privacy of its users. Government action will also ensure that the overall system remains reliable, quickly repairable in the event of a failure and, perhaps most importantly, easy to use.

IPR2025-00714
DATABRICKS EX1008 Page 74

➜ Improve management of the radio frequency spectrum, an increasingly critical resource.

➜ Protect intellectual property rights. The Administration will investigate how to strengthen domestic copyright laws and international intellectual property treaties to prevent piracy and to protect the integrity of intellectual property.

➜ Coordinate with other levels of government and with other nations. Because information crosses state, regional, and national boundaries, coordination is critical to avoid needless obstacles and prevent unfair policies that handicap U.S. industry.

➜ Provide access to government information and improve government procurement. The Administration will seek to ensure that Federal agencies, in concert with state and local governments, use the NII to expand the information available to the public, ensuring that the immense reservoir of government information is available to the public easily and equitably. Additionally, Federal procurement policies for telecommunications and information services and equipment will be designed to promote important technical developments for the NII and to provide attractive incentives for the private sector to contribute to NII development.

The time for action is now. Every day brings news of change: new technologies, like hand-held computerized assistants; new ventures and mergers combining businesses that not long ago seemed discrete and insular; new legal decisions that challenge the separation of computer, cable, and telephone companies. These changes promise substantial benefits for the American people, but only if government understands fully their implications and begins working with the private sector and other interested parties to shape the evolution of the communications infrastructure.

The benefits of the NII for the nation are immense. An advanced information infrastructure will enable U.S. firms to compete and win in the global economy, generating good jobs for the American people and economic growth for the nation. As importantly, the NII can transform the lives of the American people—ameliorating the constraints of geography, disability, and economic status—giving all Americans a fair opportunity to go as far as their talents and ambitions will take them.

Is the Internet the information superhighway that America is seeking? I think so. Since its inception as ARPAnet in 1969 and over the course of past twenty-five years, the Internet has demonstrated remarkable resilience, innovative adaptability, and spontaneous cooperation when faced with various challenges brought on by both changes in technology as well as its rapid growth.

IPR2025-00714
DATABRICKS EX1008 Page 75

I believe that the following NII issues can all be satisfactorily addressed and fully accommodated by the Internet—not in its present form, but in an advanced version of the Internet as it continues to evolve into the future:

- Private sector investment
- Universal availability
- Technology innovation
- Seamless interactivity
- Security and reliability
- Resource management
- Intellectual property rights
- Coordination
- Government information access

IPR2025-00714
DATABRICKS EX1008 Page 76

chapter 3

World Wide Web: Playground for Robots

In the past couple of years, the World Wide Web has completely reshaped the Internet. The Web has transformed the Internet from an exclusive country club frequented by the "well-connected" and privileged few, to a huge public arena visited daily by people from all walks of life. It has done so by introducing graphical user interfaces to facilitate access to the Internet, allowing users to experience sights and sounds in an intuitive style of navigation. The World Wide Web has opened the Internet to the masses.

IPR2025-00714
DATABRICKS EX1008 Page 77

World Wide Web Development

The precursor of the World Wide Web was a small, home-brewed personal hypertext system developed at CERN, Geneva's European Laboratory for Particle Physics, for keeping track of personal information on a distributed project. The positive experience prompted development of what became the World Wide Web. In March 1989, Tim Berners-Lee at CERN began circulating a proposal to build a "hypertext system" for easy sharing of information among geographically separated teams of researchers in the High Energy Physics community.

In October 1990, development on the World Wide Web was started and the project began to take shape. By Christmas of 1990, access to hypertext files and Internet news articles was demonstrated with the line-mode and graphical NeXTStep browsers. Before the end of 1991, the CERN newsletter announced the Web to the World. Other early browsers for the World Wide Web include Viola (Pei Wei, U.C. Berkeley), Mosaic (Marc Andreesen, Illinois NCSA), Cello (Thomas Bruce, Cornell University), as well as Lynx in full-screen character mode (Lou Montulli, University of Kansas).

Growth of the Web

Over time, the Web became immensely popular in part because of the browsers that made it easy for everyone on the Internet to roam, browse, and contribute to the Web information space. In April 1993, there were 62 registered Web servers on the Internet. By April 1994, the number of registered Web servers had grown to 829. By May 1994, the number increased to 1,248 (BLCLNS 1994).

The growth in World Wide Web traffic on the Internet is equally impressive. Since its start, World Wide Web traffic has grown at twice the rate of general Internet expansion. In 1994, World Wide Web traffic over the NSFnet, measured in bytes, grew an astounding 15-fold (1,500 percent)! Figure 3.1 plots the monthly traffic volume across the NSFnet T3 backbone from January 1993 through April 1995.

By July 1994, the Web had outgrown CERN's capability to deal with it as a single research laboratory dedicated to High Energy Physics. CERN began to transfer the Web project to a new group called the W3 Organization, a joint venture between CERN and MIT based in Cambridge, Massachusetts, for further development (see fig. 3.2). Between late 1994 and early 1995, this development venture blossomed into a collection of organizations and expertise called the World Wide Web Consortium.



Currently, all "official" Web-related research and developments are undertaken or coordinated by the W3 Consortium.

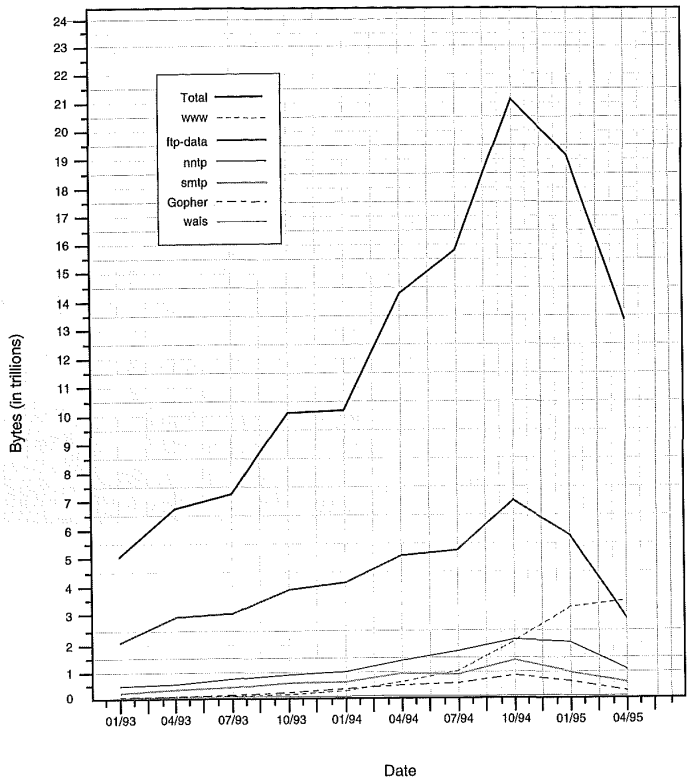
Information Dissemination with the Web

The Web originally was conceived as a convenient way to disseminate information within an organization (BLCLNS 1994). The Web behaves like a

IPR2025-00714
DATABRICKS EX1008 Page 78

Figure 3.1

Monthly traffic in bytes across the NSFnet T3 backbone.



networked repository of information that pools together useful knowledge, allowing collaborators at remote sites to share their ideas, as well as information on all aspects of a common project. Figure 3.3 illustrates the Web information space of a typical research center.

As a tool for information distribution, the Web can provide users and customers with resources previously available only to manufacturers, suppliers, and distributors. The Web has become very popular over the past two years as a new medium of expression on the Internet due in part to its capability to provide a flexible and extensible way to interact with users over the Internet for a variety of purposes.

Information residing on the Web can be smoothly reshaped by alterations in hypertext links to represent the state of new knowledge in a constantly changing environment. Furthermore, the highly scalable design of the Web requires no centralized administration of information. These properties have

IPR2025-00714
DATABRICKS EX1008 Page 79

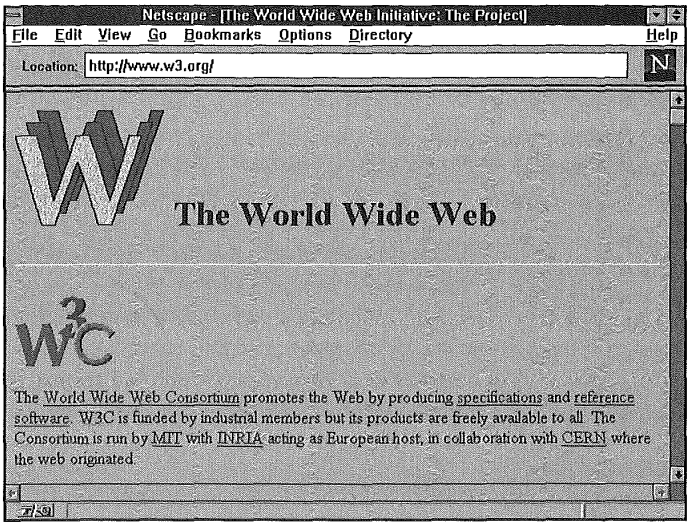


Figure 3.2

The W3C Home Page.

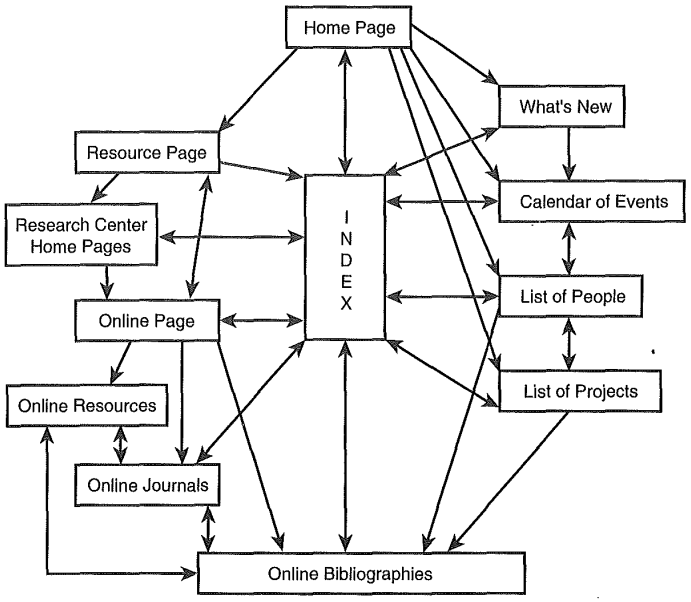


Figure 3.3

Web Information Space of a Research Center.

IPR2025-00714
DATABRICKS EX1008 Page 80

helped the Web to expand rapidly from its origins at CERN to the Internet, irrespective of boundaries of nations or disciplines.

Innovative Uses of the Web

Over the short span of a couple years, the Web has evolved to fulfill a great number of diverse needs on the Internet. It's a powerful medium for advertising and for delivery of online electronic catalogs and product information. It's also an important vehicle for setting up virtual storefronts in cyberspace. These virtual storefronts can be used for distributing software electronically, for browsing multimedia art galleries, for taking orders on various goods and services, for publishing electronic newspapers, or for "netcasting" radio and video programs. In short, the Web now has become a place of communications and learning, a new marketplace, and an exciting show ground for new information technologies.

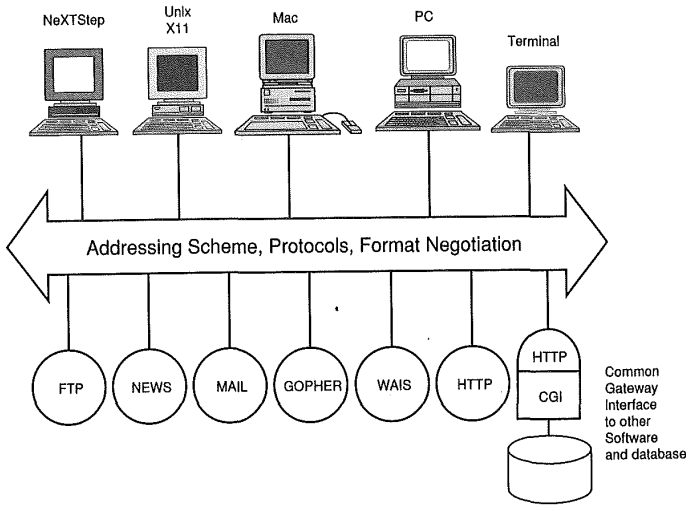
Architecture of the World Wide Web

The World Wide Web organizes, transmits, and retrieves information of all types by using a combination of hypertext, graphics, and multimedia technologies, unified in a set of naming conventions, network protocols, and document formats, and realized by using a client-server architecture.

The World Wide Web architecture is illustrated in figure 3.4 and is designed to be highly scalable. Its content is the universe of network-accessible information, which the Web originators have termed "an embodiment of human knowledge."

Figure 3.4

Client-Server Architecture of World Wide Web.



IPR2025-00714
DATABRICKS EX1008 Page 81

Besides the Web, there are other information systems like Gopher (AMLJTA 1993) and WAIS (DKMSSWSG 1990) that use a similar client-server architecture. These systems, however, play distinct roles and have different purposes. Gopher, which is sort of like a Web without full hypertext capability, uses a menu system that allows information to be organized in a hierarchy of directories. WAIS provides no navigation facilities and uses indexing exclusively to transport users into the desired location of the information space. Using the analogy of a book as an information space, Gopher often is described as its table of contents, WAIS the index pages, and World Wide Web the hypertext body where the bulk of the contents reside.

A body of software realizes the Web in a concrete form. This software architecture is composed of the following components that interoperate over the Internet:

- Clients that allow users to navigate the Web or even interact with the server in interesting ways
- Servers that allow Internet sites to publish information or export data to the world
- Proxies that facilitate communications and provide access control for sites that must rely on an intermediary host for communication with the Internet (for example, sites behind a firewall)

Web Clients

A World Wide Web client program runs on a desktop computer and is capable of accessing different Web servers distributed across the Internet. With an interactive Web browser client, users can view hypermedia documents by following information links in the Web information space. The first prototype of a Web client is a hypertext browser/editor on NeXTStep, written by Tim Bernes-Lee at CERN. Currently, a multitude of commercial browsers, such as Netscape Navigator, are available on a variety of client platforms such as PC/Windows, Mac, Unix/X11.

In addition to providing basic browsing functions, Web clients can solicit user input through an onscreen fill-out form. This capability allows bidirectional information flow and is useful for enhanced interactivity. Web clients that also provide editing functions further allow online Web document construction in a dynamic environment.

Web Servers

A World Wide Web server program usually runs on a multitasking workstation that is powerful enough to handle multiple requests from clients from all over the Internet. The most common request is to "GET" a Web page for display on the client browser. The two most popular server software packages are from CERN (written by Tim Bernes-Lee) and NCSA (written by Rob McCool), and they are simply called CERN and NCSA. Favorite platforms for Web servers include various flavors of Unix, as well as Windows/NT. The Web pages that the client views reside on a file system and have addresses that reflect the directory path that leads to the file.

Besides serving hypertext documents, a Web server also has the capability to act as a gateway to other software or information sources such as a relational database. Using the Common Gateway Interface, the Web server invokes a program script that takes information provided by the client

IPR2025-00714
DATABRICKS EX1008 Page 82

(usually from a fill-out form that appeared in the client browser), processes it according to instructions in the script, and returns a Web page result to the client.

Web Proxies

A *proxy* actually is a Web server that usually runs on a *firewall* machine (that is, a machine that functions like a security barrier between the larger Internet and a smaller local area network within an organization). The proxy acts as an intermediary between Web clients inside the firewall and Web servers out on the Internet. When the proxy receives a request from an internal machine behind the firewall, it sends the request out to some Web server on the Internet and waits for the response. When the reply comes back from the Internet, it passes the result back to the internal client.

A proxy also can be used to cache Web documents, which is useful when multiple clients within an organization (not necessarily behind a firewall) make requests for the same Web pages. The proxy will store the result of first requests and simply pass on the stored Web page for subsequent requests, substantially reducing network response time for the clients. The proxy also can store pre-loaded popular Web pages for use in caching.

Web Resource Naming, Protocols, and Formats

The World Wide Web incorporates the idea of a boundless information world in which all objects have a reference by which they can be retrieved. Despite the many different protocols in existence, the World Wide Web implements a universal addressing system, the Universal Resource Identifier (URI), to make object referencing in this world possible. Various protocols and access algorithms are encoded as specific Universal Resource Locators (URL), conforming to the general URI addressing scheme.

Although the World Wide Web architecture encompasses many other preexisting Internet protocols (see fig. 3.4), the native and primary network protocol used between World Wide Web clients and servers is the HyperText Transfer Protocol (HTTP). HTTP enables World Wide Web clients and servers to communicate efficiently, providing performance and features not otherwise available.

World Wide Web also defines a HyperText Markup Language (HTML), which is a document format that every World Wide Web client is required to understand. It is used for the transmission and representation of basic items such as text, list, and menus, as well as various styles of inputs in a fill-out form.

URI and URL: Universal Resource Identifier and Locator

The Web is designed to include objects that can be accessed by using any number of protocols that are either already in existence, being invented for the Web, or to be invented in the future. To abstract the idea of a generic object, the Web uses the concepts of a universal set of objects, and of a universal set of names and addresses of objects. A Universal Resource Identifier (URI) (BL 1994) is a

IPR2025-00714
DATABRICKS EX1008 Page 83

member of this universal set of names and addresses. URIs are strings used as addresses of objects on the Web, which could be documents, menus, or images.

Access instructions for an individual object under a given protocol are encoded into an address string. A *Universal Resource Locator* or *URL* (BLMM 1994) is a form of URI that expresses an address that maps onto an access algorithm using network protocols.

Both URIs and URLs are integral to the architecture of the World Wide Web. They allow for easy addressability of an object anywhere on the Internet, which is essential for the Web architecture to scale and for the Web information space to be independent of network and server topology.

Common URI Syntax

Although the syntax for the rest of the URL might vary depending on the particular scheme selected, URL schemes that involve an IP-based protocol connecting to a specified host on the Internet use a common syntax for the scheme-specific data, which conforms to the following URI specification:

scheme`://`*user*`:`*password*`@`*host*`:`*port*`/`*url-path*

Some or all of the parts, such as *user*`:`*password*`@`, `:`*password*, `:`*port*, and `/`*url-path*, might be excluded. The scheme-specific data starts with a double slash to indicate that it complies with the common Internet scheme syntax. The URL of the main page of the World Wide Web project, for example, is as follows:

```
http://www.w3.org/hypertext/WWW/
```

Where:

- The prefix `http` indicates the addressing scheme and defines the interpretation of the rest of the string
- The address `www.w3.org` identifies the HTTP server to be contacted
- The substring `hypertext/WWW/` identifies the document object to be accessed on the `www.w3.org` server

By default, the World Wide Web server listens to TCP port 80. The URI syntax, however, allows alternative ports to be specified. To designate an alternative port, for example, 8000, where an experimental Web server has been set up to listen on, the following URI is used:

```
http://www.w3.org:8000/experiment/test
```

Different network protocols use different syntaxes where appropriate. A small amount of common syntax, however, is enforced by URI to retain in the common model various forms and features usually encountered in many information systems. Hierarchical forms, for example, are useful for hypertext, where a large compound document can be split into many smaller interlinked documents. The common URI syntax reserves the forward slash character as a way of representing a hierarchical name space.

For query purposes, the question mark character is used as a separator between the address of an object and a query operation applied to it. In all cases, the client passes the path string to the server uninterpreted. A search on a text database, for example, might look like this:

```
http://www.my.com/AboutUs/Index/Phonebook?john
```


IPR2025-00714
DATABRICKS EX1008 Page 84

A reference to a particular part of a document might look like the following, where the fragment identifier string `#smith` is not sent to the server, but is retained by the client and used when the whole document has been retrieved:

```
http://www.my.edu/admin/people#smith
```

URLs for Various Protocols

URIs are universal. They encode members of a universal set of network addresses. A new URI scheme can be readily designed for any new network protocol that has some concept of objects. One can form an address for any object by specifying the set of protocol parameters necessary to access the object. If these protocol parameters for accessing the object are encoded into a concise string, with a prefix to identify the protocol and the encoding, one has a new URI scheme, also known as a Universal Resource Locator (URL). There are schemes for the following:

- HyperText Transfer Protocol (for example, `http://www.w3.org/hypertext/WWW`)
- Gopher protocol (for example, `gopher://gopher.micro.umn.edu/`)
- Wide Area Information Servers (for example, `wais://munin.ub2.lu.se:210/academic_e-mail_conf`)
- File Transfer Protocol (for example, `ftp://rtfm.ai.mit.edu/pub/usenet-by-group/news.answer/ftp-list`)
- Electronic mail address (for example, `mailto:webmaster@w3.org`)
- Usenet news (for example, `news://comp.infosystems.www.misc`)
- Reference to interactive sessions (for example, `telnet://downwind.sprl.umich.edu:3000`)
- Local file access (`file://localhost/etc/rc.local`)

Gopher and WAIS

Gopher and WAIS are two other information systems similar to WWW. Gopher is a hierarchical, menu-based, campus-wide information system that also provides a simple text search mechanism by means of a master index located on the Veronica server. The WAIS protocol is largely influenced by the z39.50 protocol used for networking library catalogs, and provides more sophisticated search capabilities using a master index.

HTTP: HyperText Transfer Protocol

HTTP is an Internet protocol for accessing Web servers (BLFN 1995). HTTP adopts a readable text-based style, similar to that of the File Transfer Protocol (FTP) and Network News Transfer Protocol (NNTP) that have been used on the Internet for many years. HTTP is not so much a protocol for transferring hypertext, as the name might suggest, but more a protocol for transferring information with the efficiency necessary for making hypertext jumps. The data transferred can be anything: for example, plain text, hypertext, images, audio, or video.

IPR2025-00714
DATABRICKS EX1008 Page 85

HTTP is a simple request and response protocol layered on top of TCP. There are essentially four steps to an HTTP transaction:

1. **Connect.** When a user clicks on a hyperlink, the client goes out to the Internet to locate the server machine specified in the URL, and attempts to establish connection with the server.
2. **Request.** Each HTTP request from the client begins with an operation code, called the *method*, followed by the URL of an object. The "GET" method retrieves the document URL. The "PUT" method updates the Web document, possibly with the help of a client editor. The "POST" method attaches a new document to the Web, or submits a filled-in form to the server for processing.
3. **Response.** The Web server attempts to fulfill the client's request and returns the result. A three-digit status code tells the client how the response was understood and attended to.
4. **Close.** The server terminates the connection after performing the requested action. Both client and server software must handle instances of unexpected or premature closings (for example, triggered by the Stop button on most browsers, or caused by machine crashes).

The entire process of an HTTP transaction can be observed from the status bar of most browsers. Using the Netscape Navigator, for example, you see the following:

```
Connect: Contacting http://www.w3.org...
Connect: Host contacted. Waiting for reply...
Transferring data...
Document: Done.
```

Statelessness in HTTP

HTTP is stateless, as evidenced by the fact that a network connection is made and broken for each HTTP operation. HTTP runs over a TCP connection that is held only for the duration of a single operation. When a user browses the Web, document objects are retrieved in succession from one, but sometimes multiple, servers on the Internet. The stateless model is simple and efficient because a hyperlink from one object could lead to an object that resides anywhere, maybe on the local server or some remote server.

Being a stateless protocol, HTTP does not understand the concept of a *session* (logical grouping of multiple consecutive transactions) and has no provision for remembering what has gone on before with particular client-server pairs. As far as the server is concerned, each HTTP request is handled anew and carries no history or knowledge from past transactions with the client. In cases where the server needs to track client interactions over several HTTP transactions, various gateway programming tricks have been invented to retain state variables in the server or to pass them around back and forth between client and server (DC 1995).

Format Negotiations

HTTP is capable of format negotiations. In addition to simply transferring HTML documents, HTTP can be used to retrieve documents in an unbounded and extensible set of formats. The client first sends a list of formats that it can handle, and the server replies with data in any of those formats that it can produce.

IPR2025-00714
DATABRICKS EX1008 Page 86

According to the original Web developers (BLCLNS 1994), this type of negotiation has the advantage of allowing proprietary formats to be used between consenting programs, without the need for standardization of those formats. Furthermore, this negotiation system introduces a hook for transporting future formats that have yet to be invented. Currently, this negotiation system is used for natural languages such as French or Japanese where available, as well as for compression forms such as x-compress or x-gzip.

When objects are in transit over the network, information about them (meta-information) is transferred in HTTP headers. By adopting an extension of the Multi-purpose Internet Mail Extensions (MIME) (BF 1993) for use in the set of headers, the Web developers made a design decision to facilitate integration of hypermedia mail, news, and information access.

By further adopting the convention that unrecognized HTTP headers and parameters are ignored, it has been easy to try new ideas on working production servers. This has allowed the protocol definition to evolve in a controlled way by the incorporation of tested ideas.

HTML: HyperText Markup Language

HTML is a common basic language for the interchange of hypertext (BLC 1995; Raggett 1995). It describes the structure and organization of a document. HTML is designed to be simple so that it can be easily produced by both people and programs.

The idea behind HTML is to format information online for efficient electronic distribution, search, and retrieval in such a way that it is independent of the appearance details of the document. This greatly expedites the writing and production of documents.

Conventional word processing formats dictate the appearance of documents when displayed, and thus do not interoperate across different word processors. HTML does not dictate, but merely suggests, appropriate presentations for documents. By focusing only on document structure, and not on final appearance, HTML allows Web browsers free rein to interpret and display an HTML-formatted document to the best of their capabilities.

Level of HTML Conformance

HTML is a markup language defined according to the Standard Generalized Markup Language (SGML) (Goldfarb 1990), an international standard (ISO 8879) for text information processing. A valid HTML document can be parsed by an SGML parser provided the SGML declarations also include a Data Type Definition (DTD) for HTML. HTML is an evolving standard with the following levels of conformance:

- **Level 0.** The minimum set of elements making up an HTML document that all browsers recognize. It includes a core set of simple structure elements such as headings, paragraphs, hyperlinks, bulleted lists, ordered list, and menu, all of which are useful when structuring online documents.
- **Level 1.** Level 0 features plus character formatting and inline images.

IPR2025-00714
DATABRICKS EX1008 Page 87

- **Level 2.** Level 0 and Level 1 features plus Form interface for the entry of data by users.
- **Level 3.** Level 0, 1, 2 features plus extensions for tables, figures, and mathematical formulas, stylesheets, and other features for control of layout.



Currently, many browsers support a subset of the more advanced (and possibly not yet standard) HTML features, in addition to those of the more basic levels. This is due in part to intense competition among browser manufacturers for market share.

HTML Tags

HTML documents consist of a set of tags that specify the logical structure of the document, as well as suggestion of how it could be displayed. Most HTML elements are identified in a document as a start tag, which gives the element name and attributes, followed by the content, followed by the end tag (see fig. 3.5). As in the following tag, tags define the start and end of headings, paragraphs, lists, character highlighting, and hyperlinks:

```
<HTML>
<HEAD>
<TITLE> Sample HTML Example </TITLE>
</HEAD>

<BODY>
<H1> This is H1 Header </H1>
<H2> This is H2 header </H2>
<H3> This is H3 header </H3>
<H4> This is H4 header </H4>
<P>     This is a paragraph. End tags are not
        strictly needed for paragraphs, but they
        are allowed.

<P> Here is an unordered list:
<UL>
<LI> First item in an unordered list.
<LI> Second item in an unordered list.
</UL>

<P> Here is an ordered list:
<OL>
<LI> First item in an ordered list.
<LI> Second item in an ordered list.
</OL>

<P> You can include character highlighting in a
    paragraph:
          e.g. <I> italics </I> or <B> bold</B>.

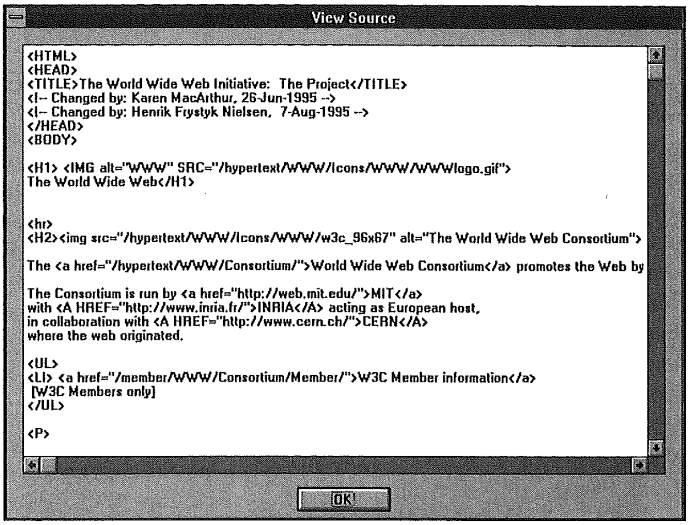
<P> This is a hypertext link to the
<A href="http://www.w3.org/"> W3C home page </A>.
</BODY>
</HTML>
```

HTML exists as a markup language independent of HTTP or World Wide Web. HTML can be used in hypertext e-mail (as "text/html" MIME content type), news, and anywhere hypertext structure is needed. There is no requirement that file contents of Web pages be stored in HTML. Servers can store file contents in other formats or in variations on HTML that include extra information of local interest only. Upon requests from clients, HTML documents can be generated on-the-fly.

IPR2025-00714
DATABRICKS EX1008 Page 88

Figure 3.5

HTML Document Source.



Forms and Imagemaps: Enhanced Web Interactivity

The power of the World Wide Web lies in its expressiveness. As originally conceived, the Web implements a distributed information space whose sole means of user interaction is hypertext navigation through mouse point-and-click. Fill-out forms and imagemaps are recent technical developments that substantially enrich the expressiveness of the Web by providing enhanced interactivity.

Fill-Out Forms

The HTML Form interface allows document creators to define HTML documents containing forms to be filled out by users. Features of a fill-out form include radio buttons, check boxes, menus (pull-down or otherwise), and text input, all of which are designed to accept user inputs. When a user fills out the form and presses a button indicating the form should be "submitted," the information on the form is sent to a server for processing. The server usually prepares an HTML document using the information supplied by the user and returns it to the client for display. Details are described in the next section on gateway programming.

Clickable Images

Imagemaps, originally invented in May, 1993, by Kevin Hughes, then of Honolulu Community

IPR2025-00714
DATABRICKS EX1008 Page 89

College, introduced interactive graphics to the World Wide Web. Imagemaps enable the use of clickable images, which are to graphics what hypertext links are to Web documents. In an imagemap, different parts of the image are linked to different places in the Web information space. When a user clicks on a certain part of an image, the corresponding Web document linked to that part will be fetched as though a normal hypertext link has just been activated. Imagemaps thus combine the freedom of graphics design with the navigational power of hypertext documents. The use of imagemap is best illustrated in the Virtual Tourist Web page, shown in figure 3.6, where a clickable world map (visible if you were to scroll down) provides an easy way for users to navigate different parts of the world.

Gateway Programming: Processing Client Input

The Common Gateway Interface (CGI) is key to providing advanced Web interactivity. It allows the Web to be connected to other software and databases by essentially functioning as a gateway. Using CGI, the user can execute a program remotely on the server, search for specific items in a database, or exchange information through an interface with other software. Gateway programs can be used to execute filter programs that generate HTML documents (from other native formats) on-the-fly, or to extract inventory and pricing information from commercial databases upon request.

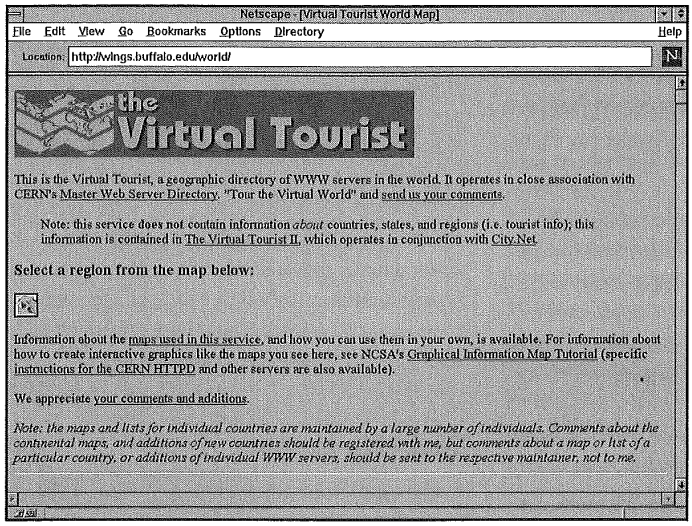


Figure 3.6

The Virtual Tourist Imagemap.

IPR2025-00714
DATABRICKS EX1008 Page 90

Mastery of gateway programming requires knowledge of either a scripting language (such as Perl or TCL) or a programming language (such as C/C++), an understanding of how input arguments are specified and extracted for processing, and the ability to properly format the output for display on client browsers.

Gateway programs usually are deposited under a directory called `/cgi-bin/` on most Unix machines, if using the CERN or NCSA Web servers. The pathname to the gateway program `/cgi-bin/qs` on the server `www.secapl.com`, for example, is included as part of its URL, as in the following:

```
http://www.secapl.com/cgi-bin/qs
```

Gateway Program Interaction

A link to the gateway program can be created in a Web page by embedding the gateway program's URL in an HTML anchor. The following are ways to invoke a gateway program (other than simply clicking on a specific hypertext link with an embedded gateway program URL):

- Clicking on a specific location within an imagemap
- Returning an ISINDEX query box
- Submitting a fill-out form

Figure 3.7 illustrates how a gateway program can be triggered using a fill-out form. The following describes the detailed action sequence:

1. The client requests the URL `http://www.secapl.com/cgi-bin/qs` from the `www.secapl.com` server after the user clicks on the corresponding hyperlink in a Web page.
2. The server returns the requested URL `http://www.secapl.com/qs`, which is an HTML form (see fig. 3.8).
3. The user fills out the form with `msft` and clicks on the submit button.
4. The client sends the resulting filled-out form, now identified by URL `http://www.secapl.com/qs?msft`, to the server for processing.
5. The server executes gateway program `/cgi-bin/qs` using `msft` as input argument.
6. The formatted output, identified by URL `http://qs.secapl.com:85/cgi-bin/qsyy` is passed from the server back to the client and displayed as a Web page on the user's screen.

There are specific books that provide more comprehensive coverage of gateway programming (DR 1995).

IPR2025-00714
DATABRICKS EX1008 Page 91

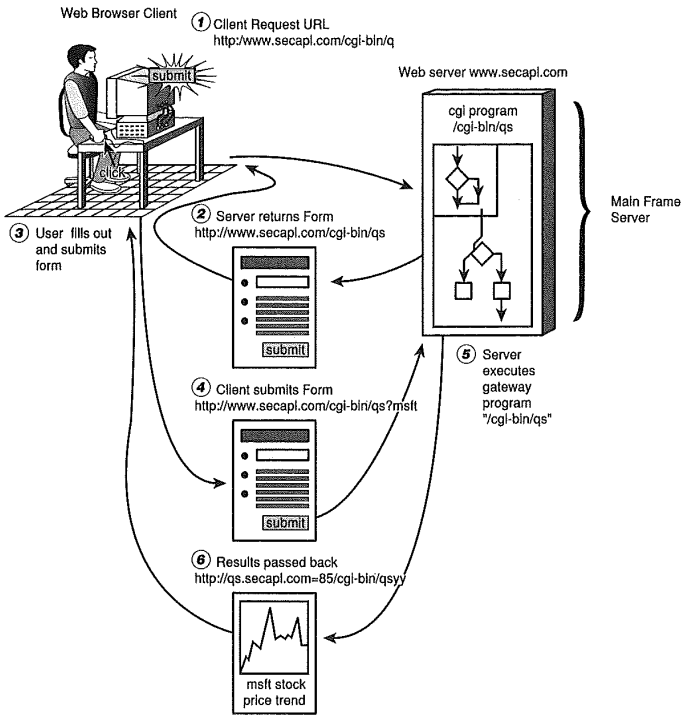


Figure 3.7

Invocation of a Gateway Program using Forms.

The Next Step: Agents on the Web

Most of the commercial and entrepreneurial efforts had focused on the Web being a facilitator of electronic commerce, usually consummated in secure transactions involving digital cash or some other form of payment schemes over the Internet. Many such applications require the client and server to authenticate each other and exchange sensitive information confidentially. Current HTTP implementations have only modest support for the cryptographic mechanisms appropriate for such transactions, but interesting developments are underway (see Chapters 8, "Web Transaction Security," and 9, "Electronic Cash and Payment Services").

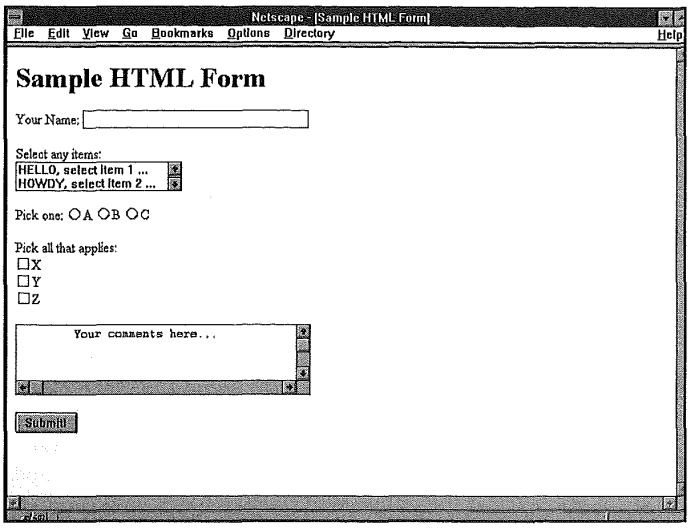
Early Commerce Agents

The World Wide Web and Mosaic have been considered the "killer apps" for the Internet because they have made possible the innovative use of the medium on a grand scale. It is believed that the next step in the evolution of the Internet is to consider the Web as a natural platform for introducing

IPR2025-00714
DATABRICKS EX1008 Page 92

Figure 3.8

HTML form to be filled-out.



innovative "commerce agents" of all types that can provide many interesting services such as bargain-hunting, mortgage-rate locking, bartering, brokering, and stock tracking.

Virtual storefronts on the Web, complete with online catalogs and automated ordering services, already implement a rudimentary form of sales agents. In response, various bargain-finding and procurement agents came into being on the Web relying on the use of such facilities. For example, the BargainFinder agent developed by Dr. Bruce Krulwich, a research scientist at Andersen Consulting, is one such early prototype of commerce agents. BargainFinder has been deployed on the Web as a service since July 1995 and can be found at Andersen Consulting's Web site at `http://bf.cstar.ac.com/bf/`. BargainFinder allows users to search for CDs and compare prices among nine compact disc sources on the Internet.

However, the BargainFinder service has run into problems. BargainFinder was not welcome at some of the online CD stores (especially those that charge higher prices for their CDs!). In fact, three out of the nine stores that BargainFinder visits have taken actions to prevent BargainFinder from accessing their online inventory database. There are also problems with BargainFinder trying to cope with some of the more difficult data formats used by a few online CD stores. To make Web shopping technology more readily accessible, I have developed a WebShopper agent that is freely available to the public. WebShopper can be configured to run as either BookFinder or CDFinder, from anywhere on the Internet. As their names imply, BookFinder shops for books and CDFinder shops for CDs on the Web.

IPR2025-00714
DATABRICKS EX1008 Page 93

WebShopper is quite similar to BargainFinder; the major difference being that WebShopper does not perform price comparisons for the user as does the BargainFinder. Instead, WebShopper explores various online sources (whose URLs and related search parameters are specified in a WebShopper task file), and simply collects final results of the search for particular books or CDs. The user can then perform comparisons based not just on the price alone, but also on other relevant terms of sale like freight charges and refund policies. The complete listing of WebShopper agent code can be found in Appendix C. Sample task files used by BookFinder and CDFinder to shop for books and CDs can be found in Appendices D and E.

Web Agents of the Future?

It is likely that a new breed of sophisticated commerce agents will come to dominate the Internet of the future, with no less impact as when compared with what Mosaic does to the Web. It also is likely that the arrival of these Internet "killer bots" will radically transform the face of the Web, upset the established marketplace and institutions, and challenge people with new ways of interacting with the medium.

At the phenomenal rate of commercialization that the Internet and the Web is currently experiencing, I would not be surprised to find in the near future digital versions of any of the following commerce agents on the Web: travel agents, insurance agents, real estate and mortgage brokers, stock brokers, manufacturers' agents, or even specialty headhunters of the literary, theatrical, sports, and talent agents genre! No one knows for sure which way future events will be played out. There have been speculations, conjectures, grand visions, and more importantly, concrete plans. It is my hope that most readers will find the rest of this book useful and informative as a gentle introduction to the wonderful world of agents and related technologies. For the few who harbor greater ambitions, it is further hoped that this book shall lead you down the path of constructing interesting agents for the Internet and the Web.

The next few chapters begin the journey on Web robots, which are agents that roam the Web with the goal of automating specific tasks related to the Web. Specifically, Chapter 4, "Spiders for Indexing the Web," introduces the use of *spiders* and *wanderers* for discovering Web resources. Chapter 5, "Web Robots: Operational Guidelines," examines Web robots in general and explores issues of interest and offers guidelines to both robot writers and Webmasters. Chapter 6, "HTTP: Protocol of Web Robots," provides an in-depth treatment of the latest version 1.0 of the HTTP protocol whose mastery is required for Web robot construction. Finally, Chapter 7, "WebWalker: Your Web Maintenance Robot," illustrates the detailed construction of one such Web robot, called the WebWalker.

IPR2025-00714
DATABRICKS EX1008 Page 94

part II

Web Robot Construction

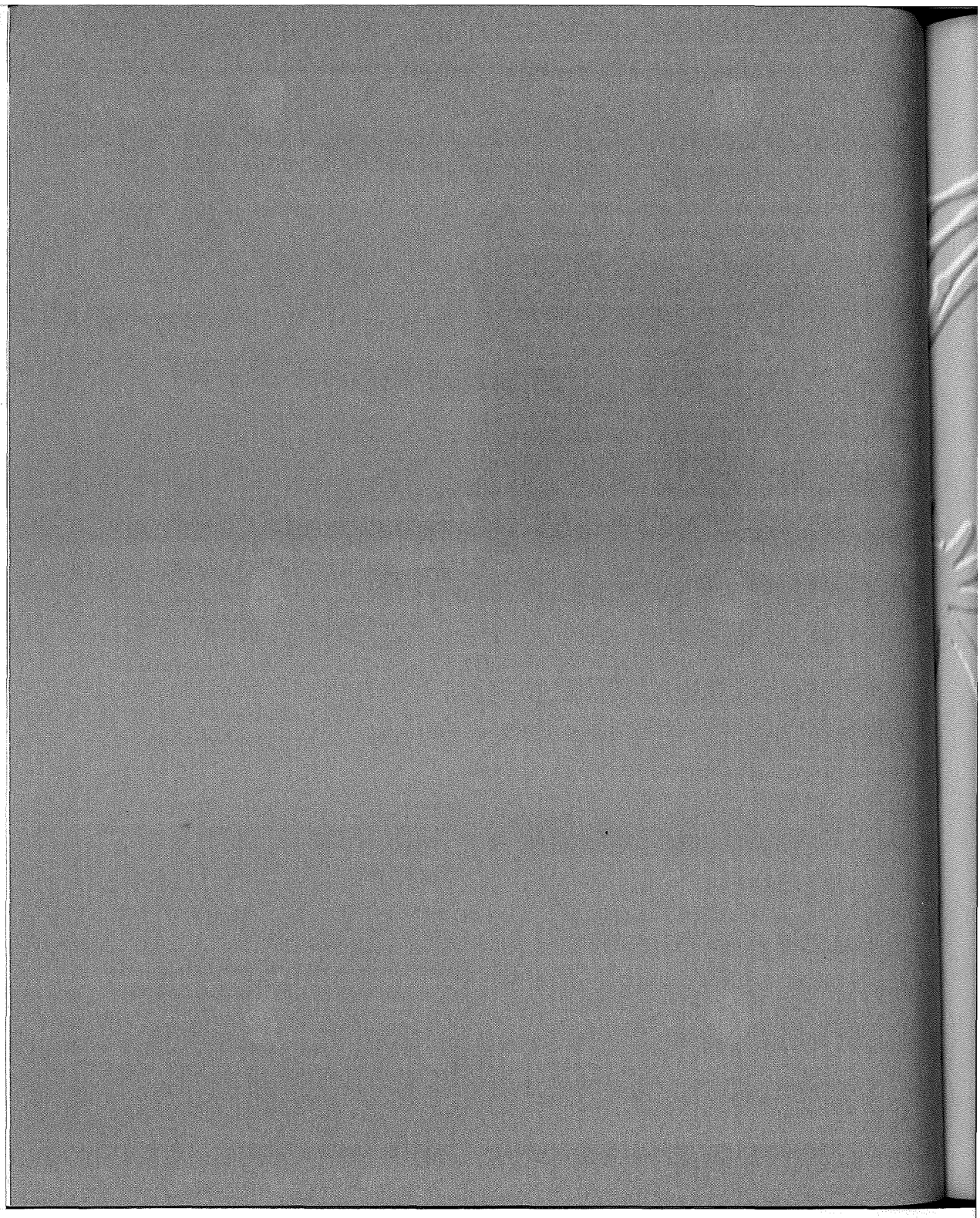
4 Spiders for Indexing the Web 81

5 Web Robots: Operational Guidelines 105

6 HTTP: Protocol of Web Robots 125

7 WebWalker: Your Web Maintenance Robot 153

IPR2025-00714
DATABRICKS EX1008 Page 95



chapter 4

Spiders for Indexing the Web

The World Wide Web is decentralized, dynamic, and diverse; navigation is difficult, and finding information can be a challenge. The reason for this challenge is that users of the World Wide Web usually navigate to find resources by following hypertext links. As the Web continues to grow, users must traverse more links to find what they are looking for, making it impractical to just wander the Web searching for information. Users, therefore, have come to depend on search engines to help them find online resources.

IPR2025-00714
DATABRICKS EX1008 Page 97

There are a number of different search engines available on the Web, each using a different method to build its underlying database. On one end of the spectrum are search engines that rely entirely on individual servers to provide self-indexing information, such as Martijn Koster's Aliweb (Archie-like indexing for the Web) (1994). This approach requires people to write index files in a specific format and store these files on their servers. Many (and apparently most) server managers have not proven willing or able to make the required effort. As a result, databases produced by this method are invariably far from complete.

On the other end of the spectrum are proactive engines, which use *Web robots* such as WebCrawler and Lycos to index large portions of the Web. Web robots, also called *spiders* or *wanderers*, are software programs that traverse the World Wide Web information space by following hypertext links and retrieving Web documents by standard HTTP protocol. Web robots require no centralized decision making and no participation from individual Web site administrators—that is, the *Webmasters*—other than their compliance with the protocols that make the Web operate in the first place. Engines of this class tend to build more complete databases than those that rely on the voluntary efforts of cooperative Webmasters.



Even the most comprehensive engines (such as Lycos) do not provide full indexing of the entire Web due to resource constraints. But by starting from the corpus of information that the spiders have discovered, and *recalled* by the search engine based upon user query, users can usually navigate much easier on the Web to find the *precise* information specific to their needs.

This chapter examines Web indexing spiders in general but focuses on two of the better known ones, the WebCrawler and Lycos spiders, that have come to dominate the Web. This chapter also discusses more advanced information gathering and dissemination architectures, such as Harvest and WebAnts, into which spiders of the future can be nicely integrated to work in a distributed and cooperative fashion.

Web Indexing Spiders

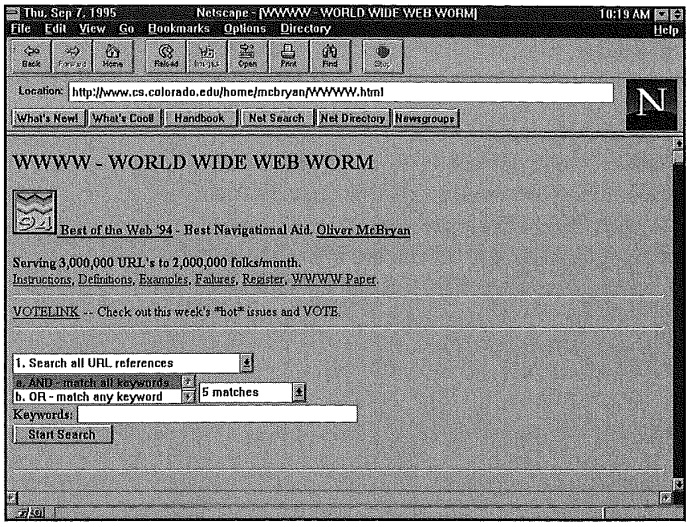
There are a variety of spiders that "crawl" around in the Web to collect information about what they find. Spiders make use of hyperlinks embedded in Web pages to automatically traverse the Web, moving from one HTML document to another by referencing the URL anchor. The information collected by spiders can be used for a variety of purposes, such as building an index for assisting users with keyword-oriented searching.

The World Wide Web Worm by Oliver McBryan, the first widely used spider, was made available in March 1994. It was an early ancestor to the newer species of spiders on the Web today. It builds an index of titles and URLs from its collection of over 100,000 Web documents and still provides the user with a search interface (shown in fig. 4.1) to its database.

IPR2025-00714
DATABRICKS EX1008 Page 98

Figure 4.1

World Wide Web Worm home page.



The World Wide Web Worm, as well as early spiders such as the Jumpstation (by Jonathan Fletcher), does not index the *content* of documents. Rather, only the HTML document titles and headers, as well as anchor text information outside the documents, are indexed.

These early spiders eventually became eclipsed by a subsequent generation of spiders that provide more powerful databases by indexing the full contents of documents. The Repository Based Software Engineering (RBSE) Spider arrived on the scene in February 1994 and was the first spider that indexed documents by content (Eichmann 1994). It was followed closely by Brian Pinkerton's WebCrawler, which began operation in April 1994.

According to Pinkerton, the reason for going to full-content indexing is that indexing by titles alone might not be adequate. Titles are an optional part of an HTML document, and 20 percent of the documents do not have them. In addition, basing an index only on titles omits a significant fraction of documents from the index. Furthermore, titles don't always reflect the content of a document. Therefore, by indexing both titles and content, spiders such as the WebCrawler and Lycos capture more of what people want to know.

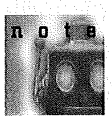
Web spiders often are criticized for being inefficient and wasteful of valuable Internet resources, even though they try to be *good citizens* on the Web (see Chapter 5, "Web Robots: Operational Guidelines," for more information on becoming citizens).

The triumph of the current generation of spiders over the earlier is mainly an issue of building high-quality, content-based indexes of Web documents. Given the phenomenal growth rate of the Web, the current generation of spiders will be hard-pressed

IPR2025-00714
DATABRICKS EX1008 Page 99

to keep up. A new generation of spiders will eventually supercede the current spiders by focusing on the increasingly important issues of performance and scalability.

These spiders of the future will use sophisticated information gathering and dissemination architecture, such as that offered by Harvest (BDHMS 1994) and WebAnts. On the horizon are a whole new generation of stronger, faster, and smarter spiders that can better survive the rapid growth of a dynamic and massive Web.



A list of information resources about other interesting spiders can be found in Appendix G, "List of World Wide Web Spiders and Robots," or Martijn Koster's up-to-date *List of Robots* (1994c).

WebCrawler: Finding What People Want

The WebCrawler project was started by Brian Pinkerton at the University of Washington in Seattle. WebCrawler is a resource discovery tool for the World Wide Web that provides a fast way of finding resources by maintaining an index of the Web that can be queried for documents about a specific topic.

WebCrawler was announced and made available to the world in April 1994 with an initial database containing information on Web documents from 6,000 servers. It answers over 6,000 queries per day and is updated weekly. In 1995, WebCrawler was acquired by America Online and is now operated as a public service available free to the Web community. As of this writing, WebCrawler has a content index of about 100 MB that holds information on over 150,000 different documents that it has explored. In addition, WebCrawler knows of the existence of over 1,500,000 unique documents it has not visited.

WebCrawler is capable of performing the following functions:

- Building indexes of the Web
- Automatically navigating on demand

Ordinary users can access the centrally maintained WebCrawler Index using a Web browser such as Mosaic or Netscape Navigator. Privileged users can run the WebCrawler client itself, automatically searching the Web on demand, but this feature is not available to the general public. The WebCrawler uses an incomplete breadth-first traversal to create an index and relies on an automatic navigation mechanism to find the rest of the information. Both document titles and document content are indexed using a vector space model (Salton 1989).

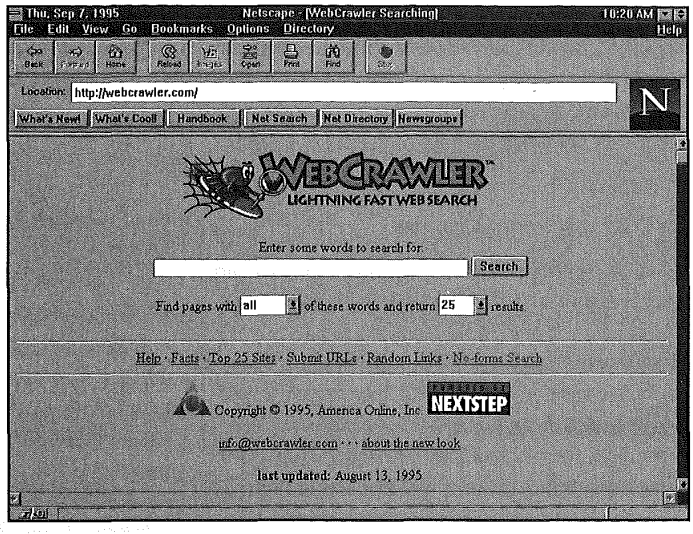
Searching with WebCrawler

The database built by WebCrawler is available through the following search page on the Web (see fig. 4.2).

IPR2025-00714
DATABRICKS EX1008 Page 100

Figure 4.2

The WebCrawler search form.



How WebCrawler Moves in Webspace

WebCrawler accesses the Web one document at a time, making local decisions in the Webspace about how best to proceed next. Unlike other centralized approaches to indexing and resource discovery, such as Aliweb (Koster 1994) and Harvest (BDHMS 1994), WebCrawler operates using only the infrastructure that makes the Web work in the first place: the ability of clients to retrieve documents from servers.

The WebCrawler design has the following characteristics:

- ➜ It uses a content-based, full-text indexing system to provide a high-quality index. In a Web robot, there is no additional network load imposed by full-text indexing; the load occurs only at the server.
- ➜ It uses a breadth-first search strategy to create a broad index, spreading the load among servers and ensuring that every server with useful content has at least several pages represented in the index.
- ➜ It tries to include as many Web servers as possible. It does so in a friendly manner, such as not overloading Web servers with rapid-fire requests. It also respects the *Robot Exclusion Standard* (see Chapter 5), which is a way for Webmasters to communicate to compliant robots which areas of the Web are off-limits.

The discovery of new documents is important in the WebCrawler design due to the dynamic nature of the Web information space. WebCrawler discovers new documents by learning their identities in the form of Uniform Resource Locators (URLs). WebCrawler starts with a known set of documents, examines the outbound links from them, follows

IPR2025-00714
DATABRICKS EX1008 Page 101

one of the links that leads to a new document, then repeats the whole process. In other words, WebCrawler simply explores the Webspace as a large directed graph using a graph traversal algorithm that performs the following sequence of actions over and over:

1. Discovers a new document
2. Marks the document as having been retrieved
3. Deciphers any outbound links
4. Indexes the content of the document

WebCrawler Architecture

As illustrated in figure 4.3, the WebCrawler software architecture is made up of the following four components:

- **The search engine.** This directs the WebCrawler's activities and is responsible for deciding which new documents to explore and for initiating their retrieval.
- **The agents.** These are responsible for retrieving the documents from the network at the direction of the search engine.
- **The database.** This handles the persistent storage of the document metadata, the links between documents, and the full-text index.
- **The query server.** This implements the query service provided to the Internet.

WebCrawler's Search Engine

The WebCrawler search engine determines which documents and what types of documents to visit. Non-indexable files, such as pictures, sounds, PostScript, or binary data, are not retrieved. In addition, erroneously retrieved files are ignored during the indexing step. This sort of file-type discrimination is applied to both indexing and real-time search modes.

The search engine uses different discovery strategies when running the WebCrawler in indexing mode and when running it in real-time search mode. In indexing mode, the goal is to build an index of as much of the Web as possible within limited storage space. WebCrawler believes that the Web documents used to build the index should come from as many different servers as possible. It uses a modified breadth-first algorithm to ensure that

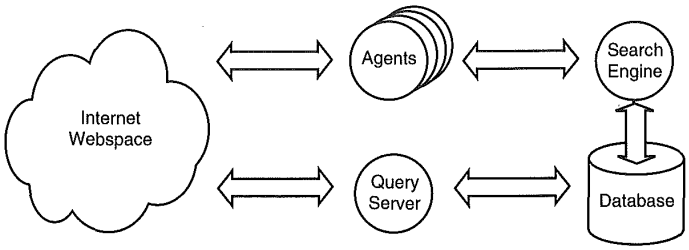


Figure 4.3

WebCrawler Software Architecture.

IPR2025-00714
DATABRICKS EX1008 Page 102

every server has at least one document represented in the index. These steps show how the algorithm works:

1. When a document on a new server is found, that server is placed on a list of servers to be visited right away.
2. One document from each of the new servers is retrieved and indexed before visiting any other documents.
3. When all known servers have been visited, indexing proceeds sequentially through a list of all servers until a new one is found, at which point the process repeats.

In real-time search mode, where the goal is to find documents that are most similar to a user's query, the WebCrawler uses a different search algorithm. The intuition behind the algorithm is that following links from documents that are similar to what the user wants is more likely to lead to relevant documents than following any link from any document. According to Pinkerton, this intuition roughly captures the way people navigate the Web; they find a document about a topic related to what they are looking for and follow links from there.

The algorithm works like this:

1. WebCrawler runs the user's query against its index to first come up with an initial list of similar documents.
2. From the list, the most relevant documents are noted, and any unexplored links from those documents are followed.
3. As new documents are retrieved, they are added to the index, and the query is re-run.
4. The results of the query are sorted by relevance, and new documents near the top of the list become candidates for further exploration.
5. The process is iterated either until the WebCrawler has found enough similar documents to satisfy the user or until a time limit is reached.

Searching for and finding documents by navigating from within other *similar* documents was first demonstrated and proven to work with the Fish search developed by Debra and Post at Eindhoven University of Technology (DP 1994). The Fish search offers a client-based search tool that is integrated with the Mosaic browser. The Fish search is reminiscent of schools of fish moving in the direction of food, hence its name. In the Fish search, each URL corresponds to a fish. After a document is retrieved, the fish spawns a number of children depending on whether the document is relevant and how many URLs are embedded in the document. The fish dies after following a number of links without finding any more relevant documents. Searches can be conducted by keywords, regular expressions, or by relevancy ranking with external filters. The WebCrawler extends this concept to initiate the search using the index, and to follow links in an intelligent order.

When people navigate, they choose links based on the *anchor text* (words that describe a link to another document) and tend to follow a directed path to their destination. When WebCrawler navigates and sees multiple links in a document, it evaluates each link for relevance based upon the similarity of the anchor text to the user's query. But anchor texts usually are short and do not adequately convey the

IPR2025-00714
DATABRICKS EX1008 Page 103

much needed relevance information as well as the full document text. To help the situation, Pinkerton noted that a thesaurus could be used to expand the anchor text.

WebCrawler's Agents

Agents are invoked by the search engine for the purpose of retrieving Web documents. Because waiting for servers and the network creates a search bottleneck, agents run in separate processes, and the WebCrawler employs up to 15 agents in parallel. For each new Web document to be retrieved, the search engine finds a free agent, and asks the agent to retrieve the URL representing the document. The agent either responds to the search engine with an object containing the document content or an explanation of why the document could not be retrieved. After the agent has responded, it becomes free again and may be given new work to do.

The agent program uses the CERN WWW library (libWWW), which supports access to several types of content through different protocols, including HTTP, FTP, and Gopher. As a practical matter, running agents in separate processes helps isolate the main WebCrawler process from memory leaks and errors in the agent and in libWWW.

WebCrawler's Database

The WebCrawler's database holds both the full-text index and the representation of the Web as a graph. The database is stored on disk and is updated as documents are added. To protect the database from system crashes, updates are made under the scope of transactions that are committed every few hundred documents.

WebCrawler uses NeXTStep's IndexingKit to build its full-text index, which is inverted to make queries fast: looking up a word produces a list of pointers to documents that contain that word. More complex queries are handled by combining the document lists for several words with conventional set operations. The index uses a vector-space model for handling queries (Salton 1989).

Words from a document are run through a "stop list" to prevent common words from being indexed, and they are weighted by their frequency in the document divided by their frequency in a reference domain. Words that appear frequently in the document and infrequently in the reference domain are weighted most highly, while words that appear infrequently in either are given lower weights. This type of weighting is commonly called *peculiarity weighting.*

The remainder of the database stores data about servers, documents, and links. Entire URLs are not stored; instead, they are broken down into objects that describe the server and the document. A link in a document is simply a pointer to another document. Each object is stored in a separate Btree on disk: documents in one, servers in another, and links in the last. Separating the data in this way allows the WebCrawler to scan the list of servers quickly to select unexplored servers or the least recently accessed server.

WebCrawler's Query Server

The query server implements the WebCrawler search service available via an HTML search form on the Web. This simple interface is powerful and can find related documents with ease. The query model it presents is a simple vector-space query

IPR2025-00714
DATABRICKS EX1008 Page 104

model based on the full-text database described earlier. Users enter keywords as their query, and the titles and URLs of documents containing some or all of those words are retrieved from the index and presented to the user as an ordered list sorted by relevance. In this model, relevance is the sum (over all words in the query) of the product of the word's weight in the document and its weight in the query divided by the number of words in the query.



The WebCrawler is a useful Web searching tool. It does not place an undue burden on individual servers while building its index. WebCrawler adopts the standard for robot exclusion standard (see Chapter 5) and identifies itself as *WebCrawler* in the HTTP User-Agent request header field when traversing the Web.

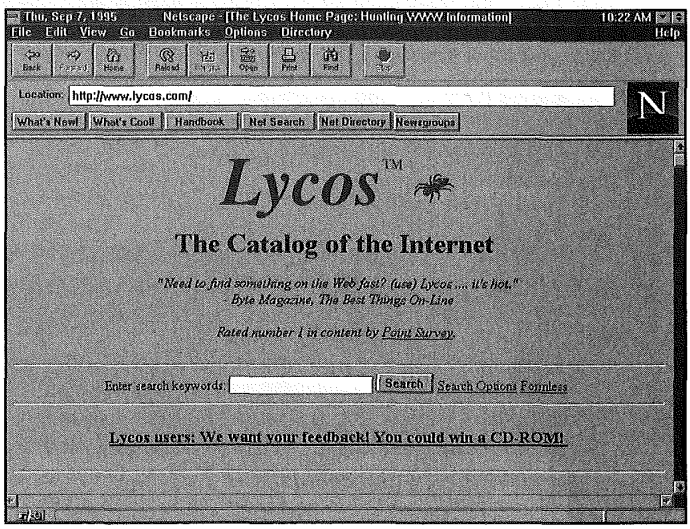
Lycos: Hunting WWW Information

The Lycos project was headed by Dr. Michael Mauldin of the Center for Machine Translation at Carnegie Mellon University as an experiment in "best-first-search" within the Web information space. The Lycos home page is shown in figure 4.4.

According to Mauldin, the word Lycos came from the arachnid family Lycosidae, which are relatively large ground spiders that catch their prey by pursuit rather than in a web. These spiders also are noted for their speed and are especially active at night. Lycos lives up to its name by continuously "hunting" its prey (Web pages on servers) for information. The search results are then merged with the catalog on a weekly basis.

Figure 4.4

The Lycos home page.



IPR2025-00714
DATABRICKS EX1008 Page 105

The Lycos spider is a fairly recent spider that was announced to the world in August 1994. It helps users locate Web documents containing specific user-supplied keywords. Due to the comprehensiveness of its database, Lycos quickly became very popular with Web users who needed to conduct full-content searches over the space of documents formed by the Web. By mid-July of 1995, Lycos accumulated the following:

- 5,077,834 unique URLs
- 1,177,750 documents (a total of 8,703,484,067 bytes)
- 3,900,084 unexplored URLs with descriptions
- 1,834,323,446 bytes of Lycos summaries
- 1,078,127,917 bytes of inverted index

The Lycos database has grown rapidly, from 634,000 references in August 1994 to over 5.6 million unique URLs in August 1995. Lycos thus offers a huge database to locate documents matching any given query.

Searching with Lycos

The search interface provides a way for users to find documents that contain references to a keyword, and to examine a document outline, keyword list, and an excerpt (see fig. 4.5). The result of a sample search using Lycos to find relevant Web pages on Ebola can be seen in figure 4.6.

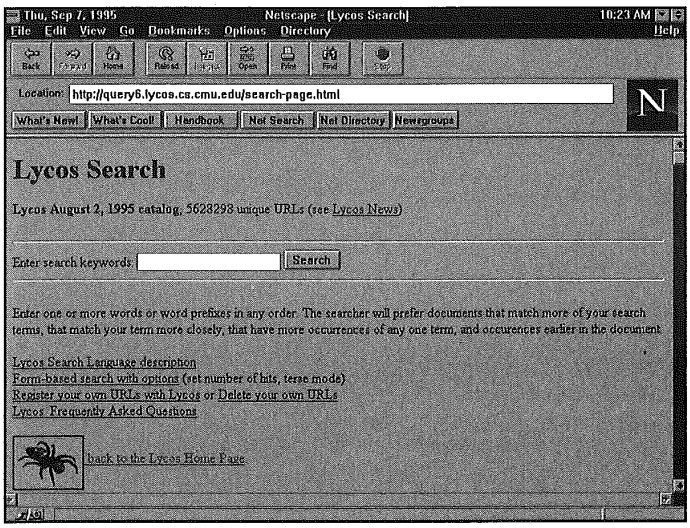


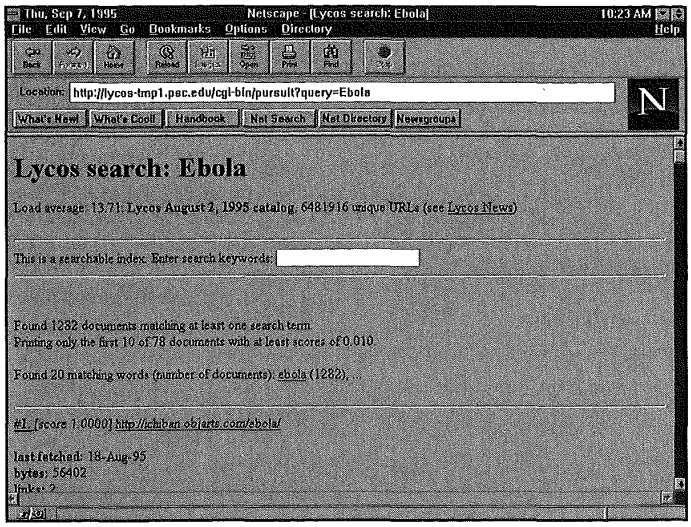
Figure 4.5

The Lycos search form.

IPR2025-00714
DATABRICKS EX1008 Page 106

Figure 4.6

A Lycos search for Ebola returns different match information.



Lycos' Search Space

Lycos defines the Webspace to be any documents in the following spaces:

- HTTP space
- FTP space
- Gopher space
- WAIS databases
- Usenet news
- Mailto space
- Telnet services
- Local file space

Lycos can retrieve documents that it has not searched by using the text in the parent document as a description for the unexplored links (the highlighted text from each HTML hyperlink anchor is associated with the URL for that anchor). Lycos does not, however, search and index ephemeral, time-varying, or infinite virtual spaces. Therefore, Lycos ignores the following spaces:

Lycos also ignores files that start with "/dev/tty" or end with these extensions: AU, AVI, BIN, DAT, DVI, EXE, FLI, GIF, GZ, HDF, HQX, JPEG, LHA, MAC, MPEG, PS, TAR, TGA, TIFF, UU, UUE, WAV, Z, or ZIP.

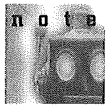
IPR2025-00714
DATABRICKS EX1008 Page 107

Lycos Indexing

To reduce the amount of information that needs to be stored, Lycos extracts the following pieces of information from each document that it retrieves:

- Title
- Headings and subheadings
- 100 most important words
- First 20 lines
- Size in bytes
- Number of words

The 100 important words are selected using the *Tf*IDf* weighting algorithm, which considers word placement and frequencies, among other factors. Words, for example, are scored by how far into the document they appear. Thus, hits in the title or first paragraph are scored higher.



In a collection of *N* documents, the *term frequency* (Tf) is the number of occurences of particular terms in the collection, and the *document frequency* (Df) is the number of documents in the collection in which particular terms occur. The idea of an *inverse document frequency* (IDf) is to measure how good particular terms are as a document discriminator—that is, to distinguish the few documents in which they occur from the many from which they are absent. A typical IDf factor is given by log(N/Df).

In the Tf*IDf weighting algorithm, the basic idea is that the best indexing terms are those that occur frequently in individual documents but rarely in the remainder of the collection. The importance, or *weight*, of a term is thus defined as the product of multiplying Tf, the term frequency, by *inverse document frequency* (IDf). In other words, weight = Tf × IDf = Tf × log(N/Df).

How Lycos Moves in Webspace

The Web-wandering component of Lycos originally was derived from a program called Longlegs, written by John Leavitt and Eric Nyberg at Carnegie Mellon University.

Lycos uses an innovative, probabilistic scheme to skip from server to server in Webspace. This avoids overloading any one server with a barrage of requests, and also allows Lycos to give preference to URLs deemed more informative. The basic steps of the algorithm are as follows:

1. When a URL resource is fetched, Lycos scans its contents for new URL references, which it adds to an internal queue.
2. To choose the next URL to explore, Lycos makes a random choice among the HTTP, Gopher, and FTP references on the queue based upon preferences.

Lycos prefers to seek out popular documents, that is, those that have multiple links into them. Lycos also has a slight preference for shorter URLs, which generally are top-level directories and documents closer to the "root" of the hierarchy (December 1994).

IPR2025-00714
DATABRICKS EX1008 Page 108

According to Mauldin, the Lycos philosophy is to keep a finite model of the Web that enables subsequent searches to proceed more rapidly. The idea is to prune the "tree" of documents and to represent the clipped ends with a summary of the documents found under that node. The 100 most important words lists from several documents can be combined to produce a list of the 100 most important words in the set of documents.



Lycos currently maintains an index database to a huge collection of Web documents that is probably the largest among all known spiders. Lycos complies with the standard for robot exclusion, and identifies itself as "Lycos" by setting the HTTP User-Agent field in the request header. In this way, Webmasters can tell when Lycos has hit their server.

Harvest: Gathering and Brokering Information

Harvest is an integrated suite of customizable tools that provides a scalable, customizable architecture for gathering, indexing, caching, replicating, and accessing Internet information, which includes the Web as well (BDHMS 1994). The philosophy behind the Harvest system is that it gathers information about Internet resources and customizes views into what is "harvested."

The creators of the Harvest system recognize three types of problems with most current Internet information systems:

- Most World Wide Web robots use expensive object retrieval protocols to gather indexing information and do not coordinate information gathering among themselves. Each World Wide Web robot gathers all the information it needs, without trying to share overlapping information with other robots.
- Little support exists for customizing how different information formats and index/search schemes are handled.
- Internet data and indices often become very popular and cause serious network and server bottlenecks.

According to Professor Michael Schwartz of the University of Colorado at Boulder, who is team leader for the project, Harvest can address the problem of how to make effective use of Internet information in the face of rapid growth in data volume, user base, and data diversity.

Harvest provides a very efficient means of gathering and distributing index information (with *Gatherers*), and supports the easy construction of many different types of indexes customized to suit the peculiarities of each information collection (with *Brokers*). In addition, Harvest also provides caching and replication support to alleviate bottlenecks.



Harvest was deployed on the Internet in November 1994, and can be reached at `http://harvest.cs.colorado.edu`, as shown in figure 4.7.

IPR2025-00714
DATABRICKS EX1008 Page 109

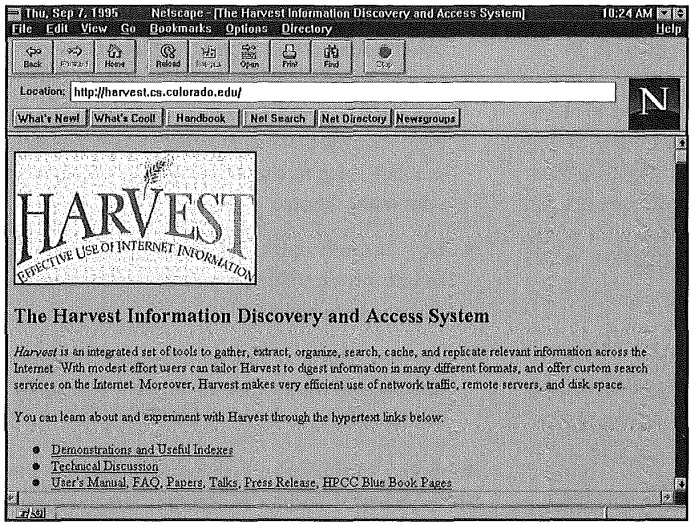


Figure 4.7

The Harvest home page.

Harvest enables Internet users to locate and summarize information stored in many different formats on machines around the world. The Harvest system interoperates with multiple information resources, including the World Wide Web. Harvest now has the capability, for example, to locate thousands of technical reports from around the world on a particular topic and then summarize the contents of each report.



Harvest is not a spider; it is more than that. A spider can be a component, called a *Broker*, in the Harvest architecture.

Searching with Harvest

With the help of a spider to collect Web pages, Harvest can index the Web information space. For example, the Harvest WWW Home Pages Broker, accessible through an HTML forms interface shown in figure 4.8, currently holds content summaries of more than 45,000 Web home pages. It uses WAIS as its backend searching and indexing engine.

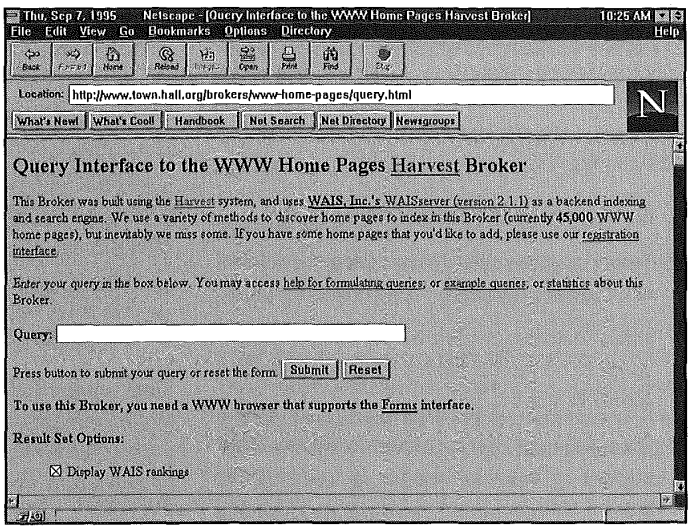
The Harvest Home Pages Broker has a very flexible and powerful interface, providing Boolean search queries based on author, keyword, title, or URL reference. For example, searching for Ebola on the Harvest Home Pages Broker returns 12 matches.

Although the Harvest database of World Wide Web documents is currently not as extensive as that of other spiders, it has great potential for efficiently collecting a large amount of them.

IPR2025-00714
DATABRICKS EX1008 Page 110

Figure 4.8

The Harvest WWW Home Pages Broker Query Interface.



Harvest Architecture

In contrast to the individual gathering efforts in the current generation of Web spiders (see fig. 4.9), the Harvest architecture offers a big improvement. In the Harvest architecture, both the information-gathering efforts, as well as gathered results, can be shared.

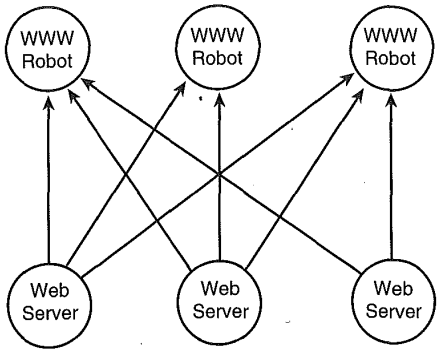


A Harvest Gatherer collects indexing information, while a Harvest Broker provides an incrementally indexed query interface to the gathered information.

As illustrated in figure 4.10, Harvest offers a flexible scheme consisting of Gatherers and Brokers

Figure 4.9

Uncoordinated information gathering by Web robots.



IPR2025-00714
DATABRICKS EX1008 Page 111

that can be arranged in various ways. This flexibility enables efficient use of network and server resources.

The Harvest architecture consists of the following subsystems:

- Gatherer collects indexing information
- Broker provides a flexible interface to gathered information
- Index/Search subsystem allows the information space to be flexibly indexed and searched in a variety of ways
- Object Cache stores contents of retrieved objects to alleviate access bottlenecks to popular data
- Replicator mirrors index information of Brokers to alleviate server bottlenecks

Harvest Gatherer

The Gatherer provides an efficient and flexible way to collect indexing information. It solves two major problems that plague most current Web indexing systems:

- Data collection inefficiencies
- Duplication of implementation effort

Most current indexing systems cause excessive load on remote sites and generate excess network traffic. Retrieving via HTTP/Gopher/FTP requires heavyweight operations, like forking separate

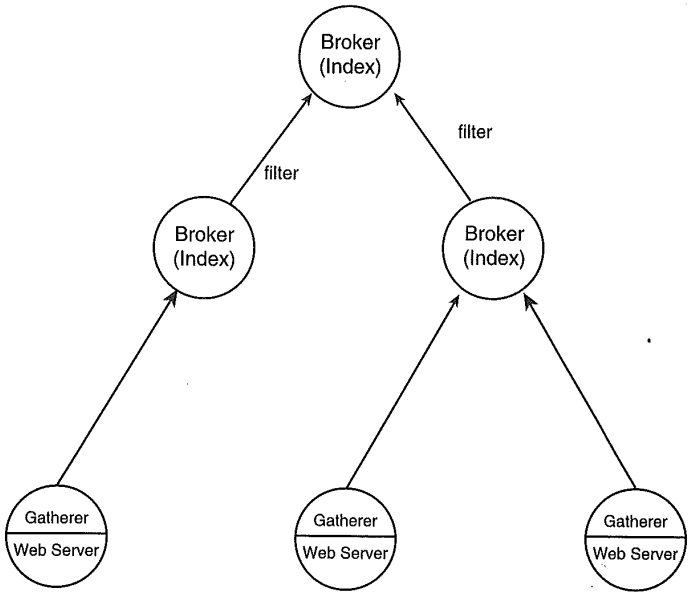


Figure 4.10

The Harvest approach to information gathering.

IPR2025-00714
DATABRICKS EX1008 Page 112

processes for each object, and entire objects often are retrieved when only a small part of the information actually is needed (for example, retaining only HTML anchors in an index).

Although the Gatherer can access an information Provider from across the network using the native HTTP, Gopher, or FTP protocols, this arrangement is primarily useful for interoperating with systems that do not run the Harvest software. The following are two important ways for Gatherers to achieve efficient use of network and server resources:

- A Gatherer can be run at the Provider site, saving a great deal of server load and network traffic.
- A Gatherer can feed information to many Brokers, saving repeated gathering costs.

The Harvest Gatherer provides efficient data collection through Provider site-resident software optimized for indexing. The Gatherer scans objects periodically, maintains a cache of indexing information (so that separate traversals are not required for each request), and allows a Provider's indexing information to be retrieved in a single stream (rather than requiring separate requests for each object). It minimizes network traffic by pre-filtering the contents and sending only incremental updates of indexing information in compressed form over the network.

The Gatherer avoids duplication of implementation efforts by providing enough flexibility to allow different indexes to be built. It uses a customizable content extraction system that allows users to customize what data are gathered, whether data are gathered locally (which is more efficient but requires site cooperation), or remotely (which allows data to be gathered via the standard HTTP/Gopher/FTP protocols). The Gatherer extracts information in different ways depending on the file types. It can, for example, find author and title lines in Latex documents, and symbols in object code.

Harvest Broker

The Broker provides an indexed query interface to gathered information. Periodically, the Broker retrieves information from one or more Gatherers or other Brokers, and incrementally updates its index. The Broker's interface is independent of the indexer, and can be customized to include new indexers with minimal effort. The Broker also can be configured to expire and re-collect information at varying intervals from the specified Gatherers.

The Broker collects objects directly from another Broker using a bulk transfer protocol. The Broker keeps track of the unique identifiers and time-to-live's for each indexed object. When a query or update is received, it invokes the Index/Search Subsystem.



A Broker can collect information from many Gatherers, to build an index of widely distributed information.

Brokers also can retrieve information from other Brokers, in effect cascading indexed views from one another, using the Broker's query interface to filter/refine the information from one Broker to the next.

Harvest provides a distinguished Broker instance called the Harvest Server Registry (HSR), which registers information about each Harvest Gatherer,

IPR2025-00714
DATABRICKS EX1008 Page 113

Broker, Cache, and Replicator in the Internet. The HSR is useful when searching for an appropriate Broker and when constructing new Gatherers and Brokers, to avoid duplication of effort. It can also be used to locate Caches and Replicators.

Harvest Index/Search Subsystem

Harvest defines a general Broker-Indexer interface that can accommodate a variety of back-end search engines to accommodate diverse indexing and searching needs. The backend is required to support Boolean combinations of attribute-based queries, and incremental updates. A variety of different backends can thus be used inside a Broker. Currently, Harvest supports WAIS, Glimpse, and Nebula; they all are optimized for different uses.

Glimpse supports space-efficient indexes and flexible interactive queries. Glimpse uses pointers to occurrence blocks of adjustable sizes, instead of pointing to the exact occurrence. It can thus achieve very space efficient indexes, typically 2–4 percent the size of the data being indexed, compared with 100 percent in the case of WAIS. As a concrete example, indexing the Computer Science technical reports from 280 sites around the world requires 9 GB with a standard WAIS index but only 270 MB using Glimpse. Glimpse also supports fast and incremental indexing, as well as queries involving Boolean combinations of keywords, regular expression pattern matching, and approximate matches.

In contrast to Glimpse, Nebula focuses on providing fast searches and complex standing queries at the expense of index size. Each object in Nebula is represented as a set of attribute/value pairs. Nebula supports the notion of a view, which is defined by standing queries against the database of indexed objects. This allows information to be filtered based upon query predicates, effectively constraining the search to some subset of the database. Within the scope of a view that contains computer science technical reports, for example, a user may search for networks without matching information about social networks. Because views exist over time, it is easy to refine and extend them, and to observe the effect of query changes interactively.

Harvest Object Cache

To alleviate bottlenecks that arise from accessing popular data, Harvest implements an Object Cache that stores the content of HTTP, Gopher, and FTP objects that have been retrieved. The Object Cache runs as a single, event-driven process. For ease of implementation, the Cache spawns a separate process to retrieve FTP files, but retrieves HTTP and Gopher objects itself. The Cache separately manages replacement of objects on disk and objects loaded in its virtual address space. It also keeps all metadata for cached objects in virtual memory, to eliminate access latency to the metadata.

Multiple Object Caches can be arranged hierarchically for scalability. The Object Cache allows sites to customize hierarchical relationships between caches at multiple levels of the network (for example, at a campus, regional, and backbone network). Different caching parameters, such as timeouts, maximum object size, cache storage size in memory and disk, as well as caching policies, also can be customized.

IPR2025-00714
DATABRICKS EX1008 Page 114

Harvest Replicator

The Harvest Replicator provides a weakly consistent, replicated wide-area file system for mirroring the information that the Brokers have. This alleviates bottlenecks that arise from heavy demand on particular servers. Each file system occasionally "floods" its closest neighboring file systems with complete state information to ensure consistency, and to allow its neighbors to detect updates that for some reason have failed to propagate. The weak form of consistency used in the Replicator is called eventual consistency; if all new updates ceased, the replicas eventually converge.

The Replicator also can be used to divide the gathering process among many servers (for example, allowing one server to index each U.S. regional network) by distributing the partial updates among the replicas. The Replicator also allows sites to customize the degree of replication, topology of updates, and the frequency of updates.

WebAnts: Hunting in Packs

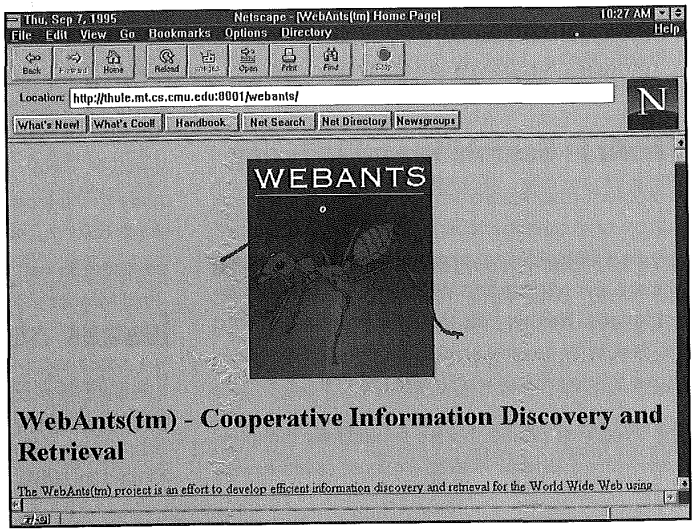
The WebAnts project is a new experiment headed by John Leavitt of Carnegie Mellon University to investigate the distribution of information collection tasks to a number of cooperating processors. It aims to create cooperating explorers (called *ants*) that share the work of finding things on the Web without duplicating each other's efforts. Leavitt noted that the origin of the metaphor derives from similarity between this and the manner in which biological ants leave chemical trails to sources of food and cooperate in the harvesting.



WebAnt is currently under development. As of this writing, it has not been deployed on the Web yet. But stay tuned and watch the WebAnts home page (see fig. 4.11).

Figure 4.11

The WebAnts home page.



IPR2025-00714
DATABRICKS EX1008 Page 115

WebAnts Motivation

According to Leavitt, the development of WebAnts was motivated by the following considerations:

- Information discovery on the Web is rapidly becoming too large a task for a single explorer agent. Not only will the local portion of the network sustain considerable traffic during exploration, such an exploration will consume progressively more time as the Net grows.
- The reliance on a single site for such services would create a bottleneck and does nothing to solve the problem related to the fan-in, fan-out nature of information discovery. Instead, it exacerbates the problem and makes that one site a bottleneck for all users.
- It is undesirable for multiple explorers to examine the same sites. If exploring the Web alone is a problem, having a number of non-cooperating, and therefore redundant, explorers is worse. Not only does it cause unnecessary load on the servers, it also fails to provide a reasonable service to the user.

WebAnts Searching and Indexing

A problem most users face in searching for information on a specific topic is that the user cannot rely on a single-search engine because it does not explore everything and could be a performance bottleneck. Neither can the user merely combine the results of several search engines together because this inevitably yields repeated hits.

The WebAnts project hopes to address these issues with a cooperative Web explorer, called an *ant*. Unlike spiders, ants are designed to share results with other ants without duplication of efforts. WebAnts has a clear preference for using explorer-based schemes over those that require cooperation from each information server (such as Martijn Koster's Aliweb). The WebAnts model can be used for purposes of searching and indexing the Web.

WebAnts Searching

For searching purposes, different ants may be directed based upon each others' results. When one ant finds a document that satisfies the search criteria, it can share the references from that document with other ants that are not currently exploring hits of their own. As each ant explores a document, other ants would know about it so that they do not have to examine the same document. This allows information to be gathered more effectively.

WebAnts Indexing

For indexing purposes, cooperation among ants allows each indexer to conserve resources by distributing the indexing load between different explorers. Each index server would provide all the information gathered by one of the ants during exploration. When querying, a user could restrict the query to the local ant or allow it to propagate to the entire colony. This reduces the bottleneck effect.

Issues of Web Indexing

The indexes built by Web robots save users from following long chains before they find a relevant document, thus saving Internet bandwidth (Pinkerton 1994). Brian Pinkerton calculates that if

IPR2025-00714
DATABRICKS EX1008 Page 116

the WebCrawler indexes 40,000 documents and gets 5,000 queries a day, and if each query means the user will retrieve just three fewer documents than she otherwise would have, then it will take about eight days for the WebCrawler's investment in bandwidth to be paid back.

The following subsections discuss Web indexing issues of recall and precision, good Web citizenship, performance, and scalability.

Recall and Precision

The capability of spiders to find useful information is usually measured in two ways: recall and precision (Salton 1989). *Recall* measures what fraction of the relevant documents are retrieved by the query, whereas *precision* indicates how well the retrieved documents match the query.

Recall is the proportion of relevant documents retrieved; that is, the number of relevant documents retrieved divided by the total number of relevant documents in the indexed collection. If, for example, an index contained 10 documents, 5 of which were about elephants, then a query for "elephants and ivory" that retrieved 4 relevant documents about elephants (but two non-elephant documents) would have a recall of 4/5 or 0.8.

Precision is defined as the proportion of retrieved documents—the number of relevant documents retrieved divided by the total number of documents retrieved—that are relevant. Using the same example as in the previous paragraph, the precision would be calculated as 4/(4+2) or 0.66.



A good indexing scheme aims for high recall and precision. A large proportion of the useful documents should be retrieved, and at the same time a large proportion of the extraneous documents should be rejected.

Both WebCrawler and Lycos have adequate recall. Finding enough relevant documents is not the problem. Instead, precision suffers because these systems give many false positives. Documents returned in response to a keyword search need only contain the requested keywords and might not be what the user is looking for. As a practical solution, assigning weights to documents returned by a query would help the user focus on the more relevant documents, but it would not completely eliminate irrelevant documents.

Good Web Citizenship

Webmasters can advise robots by specifying which documents are worth indexing in a special "robots.txt" document on their server (Koster 1994a). This type of advice is valuable to Web robots, and increases the quality of their indexes. In fact, some Webmasters have gone so far as to create special overview pages for Web robots to retrieve and include.

Both the WebCrawler and Lycos try hard to be good citizens on the Web. Although some poorly designed Web robots have been known to operate in a depth-first fashion, retrieving file after file from a single site, both WebCrawler and Lycos are conscientious of their traversal order so as not to overload any one particular server. WebCrawler

IPR2025-00714
DATABRICKS EX1008 Page 117

searches the Web in a breadth-first fashion, while Lycos uses a probabilistic scheme to skip from server to server. This avoids the problem of hitting any one server with a long string of requests.

Furthermore, when searching for something more specific among a relevant set of documents at a particular site, WebCrawler limits its search speed to one document per minute and sets a ceiling on the number of documents that can be retrieved from the host before query results are reported to the user.

Performance

As the Web continues its phenomenal growth, there comes a point where being just good citizens on the Web might not be enough to offset the load placed on network and server resources by indexing spiders.

Harvest is designed to ease the strain on servers, as well as on overall network traffic. The Harvest researches have compared the performance of Harvest with methods of native protocol access as used in all current spiders (BDHMS 1994). In their experiments, they have observed the following measured results:

- Harvest reduced HTTP/Gopher/FTP server load by a factor of 4 while extracting indexing information.
- Harvest reduced server load by a factor of 6,600, while delivering indexing information to remote indexers.
- Harvest reduced network traffic by a factor of 59.
- Harvest reduced index space requirements by a factor of 43.

Although the current generation of spiders is useful as a tool for indexing the Web, there needs to be a more efficient way to conduct Web exploration. The Harvest performance measurements have shown just how much room there is for future improvements in speed.

Scalability

Other than the issue of performance, any spiders that attempt to index the entire Web must face the ultimate challenge: scalability. Not only must every document in the Web be retrieved, but some portion of each document must be saved as a way of summarizing its contents for later retrieval. Different indexing schemes save different information from documents, but the problem facing every indexing spider is the same: *How to manage such a vast amount of information.*

The trade-off becomes one of quality of index versus coverage of documents. Saving more information per document reduces the number of documents than can be covered, and vice versa. One way to avoid the fatal trade-off is to distribute the resource load, as is done in WebAnts, where an army of cooperating ants share the work load so that each ant indexes only a small portion of the Web.

Indexing the Web in parallel with ants also reduces indexing time, a factor that is already becoming a problem for some spiders that need to explore a large number of Web documents. In addition, distributing the search among the ants eliminates the need for gigabytes of storage in one place to keep the

IPR2025-00714
DATABRICKS EX1008 Page 118

indexing information. Each cooperating ant needs only provide as much storage as is comfortable.

The problem of scalability is only endemic to the current generation of indexing spiders, which operate in a lone-ranger fashion. Spiders of the future, probably better called *crawlers* to distinguish them from the current spiders, will be scalable by working in a cooperative fashion.

Spiders of the Future

Although many early spiders could successfully crawl through Webspace in 1994, the rapid increase in the amount of information on the Web since then made this same crawl increasingly difficult. Only a few resourceful spiders, such as WebCrawler and Lycos, can accumulate enough Web documents to survive and dominate the Web. Meanwhile, the rest of the spiders, starved of information due to inadequate resources, slowly became extinct.

One problem that any indexing spider must eventually deal with is that the size of an index will grow proportionally to the size of the Web. The storage, retrieval, and distribution of information on this scale will no doubt prove a compelling challenge. Advanced information gathering and distribution architectures like Harvest and WebAnts can help spiders, or other crawlers such as ants, become more efficient and effective in their Web indexing efforts by sharing both their work load and results.

In the not-too-distant future, we can expect to see stronger, faster and smarter crawlers on the Web, supported by more efficient distributed information architecture, such as Harvest and WebAnts, that can address the important performance and scalability issues. The promise for the future is that systems like Harvest and WebAnts will provide users with increasingly effective means to locate information on the Web.

IPR2025-00714
DATABRICKS EX1008 Page 119



chapter 5

Web Robots: Operational Guidelines

World Wide Web robots, also called *spiders or wanderers*, are programs that traverse the World Wide Web by recursively retrieving pages hyperlinked by Uniform Resource Locators (URLs). They are viewed as special kinds of agents whose goal is to automate specific Web-related tasks—for example, retrieving Web pages for keyword indexing or maintaining Web information space at local sites. Although this book is about various kinds of Internet agents and their underlying technologies, the focus is really on understanding Web robots.

IPR2025-00714
DATABRICKS EX1008 Page 121



Spiders, wanderers, Web worms, fish, crawlers, walkers, and ants all mean one thing: Web robots, which are programs that traverse the World Wide Web information space by following hypertext links and retrieving Web documents by standard HTTP protocol. All these names are misleading, giving the false impression that the Web robot itself actually *moves*. In reality, the Web robot never leaves the machine where the program is run and is entirely different from the infamous Internet Worm of 1989 (Seeley 1989; Spafford 1989).

But do you really need to create yet another Web robot? There are already many of them out there in the public domain. Your needs probably can be fulfilled with one of the existing Web robots. Even if you do decide to construct a new Web robot after all, it does not have to be built entirely from scratch. The source code (usually in Perl) of quite a few public domain Web robots, such as that of Roy Fielding's MOMspider robot, are freely available for modifications. It usually is safer and more economical to go with a proven solution that already is fine-tuned for operation than it is to create new solutions from scratch.

A fairly detailed and comprehensive collection of robots on the Web, derived from the *List of Robots* which Martijn Koster (1994a) actively maintains at:

`http://web.nexor.co.uk/mak/doc/robots/active.html`

which is expanded to include other information, is available as Appendix G at the back of this book.

This chapter starts by describing major uses of Web robots and explaining how to bar specific Web robots from visiting specific portions of the Web space, by means of the widely adopted *Standard for Robot Exclusion* (Koster 1994b). The remainder of this chapter provides specific guidelines of acceptable Web robot behavior (*Four Laws of Web Robotics*), outlines the responsibility and vigilance expected of Web robot operators *(Six Commandments for Robot Operators*), offers some tips to Webmasters who suspect their servers may be under attack by a Web robot, and concludes with a discussion of Web ethics.

The Four Laws of Web Robotics and Six Commandments for Robot Operators described in this chapter are inspired by Martijn Koster's *Guide for Robot Writers* (1994c), which is based on a consensus of the various WWW newsgroups and mailing lists on acceptable and expected behaviors of Web robots and their operators.

Web Robot Uses

The earliest Web robot, Matthew Gray's World Wide Web Wanderer, was first deployed in June 1993 to measure the growth of the World Wide Web by discovering and counting the number of Web servers on the Net. As of this writing, the number of different Web robots has grown to more than 40 (see Appendix G).

Excluding the more recent BargainFinder type of application-specific Web commerce agents (described in Chapter 3), almost all known Web robots to date have been deployed for one or more of the following purposes:

- Web resource discovery
- Web maintenance
- Web mirroring

IPR2025-00714
DATABRICKS EX1008 Page 122

Web Resource Discovery

Web resource discovery is concerned with the problem of finding useful information on the Web. The rich, decentralized, dynamic, and diverse nature of the Web has made casual Web surfing enjoyable, but has made serious navigation aimed at finding specific information extremely difficult. People have thus increasingly relied on search engines to help locate online information. These search engines have depended on Web robots, often called spiders, to automatically traverse the Web to bring in Web documents for keyword indexing. It is perhaps the most exciting problem tackled by the current generation of Web robots.

As was discussed previously in Chapter 4, "Spiders for Indexing the Web," the two most prominent resource discovery Web robots in operation today are Brian Pickerton's WebCrawler robot and Michael Mauldin's Lycos spider. Both of which actively maintain a full-content index to a huge collection of Web documents, currently numbering in the millions. Both spiders continuously traverse the Web to keep their index database up-to-date. A keyword-oriented search facility to their index databases is made available to users by means of a front-end query interface and a corresponding back-end search engine.

Web Maintenance

A major difficulty in maintaining a Web information structure is that hypertext references to other Web pages might become outdated when the target Web page is deleted or moved, resulting in what are called *dead links*. Currently, there is no automated mechanism for proactively notifying Web document owners the moment hyperlinks in their Web pages become obsolete.

Some servers log failed HTTP requests caused by dead links, along with URL information of the specific Web page that refers to it in the first place (while returning an HTTP response code of "301 Moved Permanently" to the client). Such information in the server log files can then be scanned and processed at regular intervals to generate a list of Web pages with the corresponding dead links that they contain. However, this post-mortem style of solution is not quite practical because document owners in the real world are seldom notified this way.

A more workable solution seems to be that offered by a class of Web robots known as the Web maintenance spiders. They assist Web document owners and Webmasters maintain their portions of the Web information structure by automatically traversing the relevant branches of the local Web space periodically and checking for dead links. Roy Fielding's MOMspider, as well as its younger and simpler WebWalker cousin (to be discussed later in Chapter 7), are examples of Web maintenance robots. In addition, Web maintenance spiders can also perform checks for document HTML compliance, document style conformance, as well as other lesser known document content processings.

Web Mirroring

Mirroring is a common technique for setting up replicas of an information structure. For example, mirroring an FTP site involves copying its entire FTP file directory recursively and reproducing it on a different machine over the network. Popular FTP sites on the Internet are often mirrored in different parts of the world, for load sharing as well as for redundancy in case of failures. Mirroring can also yield faster or cheaper local, or even offline, access.

IPR2025-00714
DATABRICKS EX1008 Page 123

Robots that mirror Web information structures include, for example, HTMLGobble, Tarspider, and Webcopy. Mirroring the Web introduces an added complication not found in mirroring FTP sites, in that mirrored Web pages need to be rewritten to reflect changes in hyperlink references. Hyperlinks that used to point at original Web pages must now point to newly copied Web pages. Also, relative links that point to pages that have not been mirrored must be expanded into absolute links, so that they continue to point at original Web pages (and not at non-existent Web pages at the mirror site!).

The current generation of Web mirroring robots cannot detect and do not understand Web document changes. The unnecessary transfer of unchanged Web documents wastes valuable network resources. It can thus be expected that sophisticated mirroring robots of the future must also perform some amount of document revision control and management.

Proposed Standard for Robot Exclusion

In 1993 and 1994, robots sometimes visited Web servers where they were not welcome for various reasons. Sometimes these reasons were robot-specific—for example, certain robots swamped servers with rapid-fire requests or retrieved the same files (or the same sequence of files) repeatedly. Other situations are server-specific and there are cases where Webmasters have found robots getting caught in parts of the Web they were not meant to traverse—for example, very deep virtual trees generated by server programs on-the-fly, duplicated information, temporary information, or invocations of Common Gateway Interface program scripts with side-effects (such as voting). In a few cases, certain Web robots are simply not welcome as a matter of policy due to conflicting interests—for example, some online CD stores would like to bar the price-shopping BargainFinder agent from searching their Web sites.

These incidents indicate a need for an operational mechanism for Web servers to identify to robots that portions of their Web are out of bounds and should not be accessed in an automated fashion. The *Standard for Robot Exclusion* proposed by Martijn Koster (1994b) is an attempt to address such a need with a simple operational solution.

Robot writers are urged to implement this practice. You can find some sample Perl code in this Web page:

`http://web.nexor.co.uk/mak/doc/robots/norobots.pl`

Robot Exclusion Method

The method used to exclude robots from a Web server is for the Webmaster to create a file on the server that specifies an access policy for robots. This file, called the *robot exclusion file*, must be accessible with HTTP from a local URL with the standard path /robots.txt. The contents of the robot exclusion file describe the nature of the constraints and are detailed in the next section.

This approach was chosen because it can be implemented easily on any existing Web server. A Web robot can find the access policy from the robot exclusion file (whose URL path is /robots.txt) with only

IPR2025-00714
DATABRICKS EX1008 Page 124

a single document retrieval. Although the Webmaster can specify many constraints in the robot exclusion file, it is still up to individual Web robots to check for the existence of the file in the first place, to retrieve it and to adhere to its specified constraints.

According to Koster, a possible drawback of this single-file approach is that the robot exclusion file can be maintained only by the Webmaster and not the individual document maintainers at the site. This problem can be resolved easily by a local maintenance procedure that constructs the single robots.txt file from a number of other files. (The procedure for this is outside the scope of the proposed standard.)

Robot Exclusion File Format

The file consists of one or more records. Each record is of the following form, on a line terminated by a carriage return (CR), or a line-feed (LF), or a combination of carriage return followed by line-feed (CR/LF):

field:value

The field name is case insensitive. There can be optional spaces around the value. Blank lines (lines that contain no records but are terminated with CR, LF, or CR/LF) are ignored.

Comments are allowed in the robot exclusion file for annotation purposes. A comment line begins with a # character; any preceding spaces and the remainder of the comment line up to the line terminator(s) are discarded.

The presence of an empty robot exclusion file that contains nothing basically is meaningless and should be treated as if it were not there. In this case, all Web robots would consider themselves welcome at that site.

Recognized Field Names

Records with unrecognized field names are ignored. The following are the recognized field names defined in the standard:

- **User-Agent.** The value of this field identifies the robot in question. If there are multiple consecutive User-Agent records, then more than one robot shares the identical access policy (specified in the immediately following sequence of Disallow records). Each User-Agent record, or each block of consecutive User-Agent records as the case may be, must be followed by at least one Disallow record (to be described next).

  The robot should be liberal in interpreting the value of the User-Agent field. A case-insensitive substring match of the value without version information is recommended. The following are some examples of popular user agents:

```
User-Agent: Mozilla/1.1N    # Netscape browser
User-Agent: WebCrawler/2.0  # Web searcher
User-Agent: MOMspider/1.00  # Web maintainer
```

  If the value is *, the record describes the default access policy for any Web robot that has not matched with any of the other records. There must not be more than one record whose value is * in the robot exclusion file.

IPR2025-00714
DATABRICKS EX1008 Page 125

➜ **Disallow.** The value of this field specifies a partial string describing the prefix portion of the URL that is not to be visited. This can be a full path name or a partial path name. Any URL that begins with this value will not be retrieved. For example, the following line disallows both: `/home/index.html` and `/homeSweetHome.html`:

```
Disallow: /home
```

Whereas, this line disallows `/home/index.html` but allows `/homeSweetHome.html`.

```
Disallow: /home/
```

An empty value permits all URLs to be retrieved. At least one Disallow record must be present under each block of consecutive User-Agent records (for multiple robots sharing the same access policy), or under each User-Agent record (for a single robot with unique access policy). A Disallow record cannot be present without at least one User-Agent record preceding.

Sample Robot Exclusion Files

The following robot exclusion file specifies that no robots should visit any URL starting with `/cyberworld/map/` (directory of infinite virtual space), `/cgi-bin/` (directory of executable Common Gateway Interface scripts), or `/tmp/` (directory of temporary files soon to disappear):

```
User-Agent: *
Disallow: /cyberworld/map/ # Virtual space
Disallow: /cgi-bin/        # CGI scripts
Disallow: /tmp/            # Temporary files
```

The following robot exclusion file specifies that no robots should visit any URL starting with `/cyberworld/map/`, except the robot called *cybermapper*:

```
User-Agent: *            # Bar all robots...
Disallow: /cyberworld/map/

User-Agent: cybermapper  # ...except cybermapper
Disallow:
```

The following robot exclusion file example indicates that no robots should visit this site further:

```
User-Agent: *            # All robots go away!
Disallow: /
```

The Four Laws of Web Robotics

The aspiring creators of future Web robots would be wise to heed the advice proffered by seasoned Webmasters and other Web robot experts, which has been summarized in *the Four Laws of Web Robotics*. These laws codify the expected and accepted behavior of robots, and are listed in table 5.1.

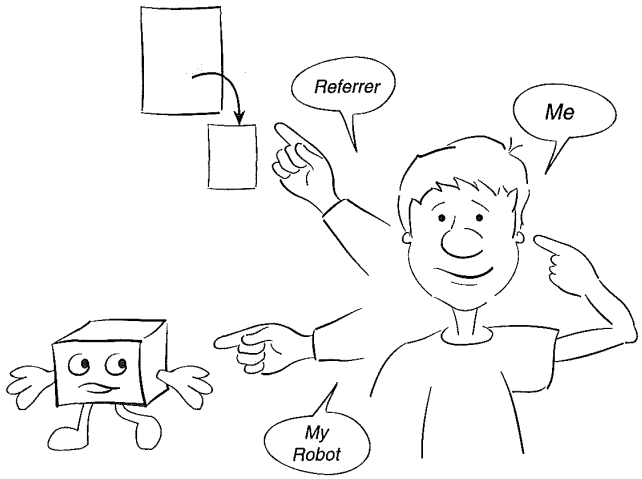
If you are building a new Web robot, you are strongly urged to design your robot program in such a way that all four laws of Web robotics are adhered to. The following subsections explain each law in detail.

IPR2025-00714
DATABRICKS EX1008 Page 126

Table 5.1

The Four Laws of Web robotics.

I.	A Web Robot Must Show Identifications
II.	A Web Robot Must Obey Exclusion Standard
III.	A Web Robot Must Not Hog Resources
IV.	A Web Robot Must Report Errors



I. A Web Robot Must Show Identifications

Webmasters want to know which robots are accessing their sites and who is operating the robots so they will know who to contact in case of trouble. In many cases, Webmasters also want to find out how others came to know of their sites. A Web robot can accommodate Webmasters by identifying itself (with User-Agent field), its operator (with From field), and the Web page referrer (with Referrer field).

Web Robot Self Identification

Web clients can identify themselves by means of the User-Agent fields supported in HTTP request headers. For example, the Netscape browser calls itself Mozilla, as in the following example:

```
User-Agent: Mozilla/1.1N
```

A Web robot can use the User-Agent field to state its name and provide a version number, as in the following example:

```
User-Agent: Terminator/1.0
```


IPR2025-00714
DATABRICKS EX1008 Page 127

This User-Agent field enables Webmasters to set Web robots apart from human-operated interactive Web browsers.

Robot Operator Identification

HTTP supports a From field in the request headers, allowing a Web robot to identify its human operator. An e-mail address is often used for identification here, as in the following example:

```
From: joe.robomaster@roboland.com
```

The From field enables Webmasters to contact the robot operator in case of problems. The robot operator can thus respond to Webmasters under a more amicable atmosphere than if he or she has been hard to track down.

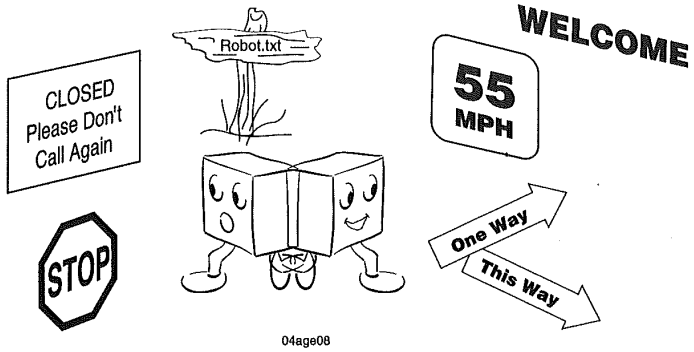
Web Page Referrer Identification

Webmasters often wonder how people came to learn of the existence of their Web sites. When accessing a particular Web page, it is possible and often helpful for a Web robot to identify to the Web server the parent document that hyperlinks to the Web page. This parent document is called the Web page referrer. HTTP supports a Referer field for purpose of identifying the parent document. It is informative, for example, for the Webmaster to know that the Web page currently being accessed is referred to by a paid listing with some Web advertising service, as shown in the example here:

```
Referer: http://www.referRus.com/launchpad.html
```

II. A Web Robot Must Obey Exclusion Standard

The *Standard for Robot Exclusion* was proposed by Martijn Koster (1994b) as a simple way for Web servers to communicate to Web robots which portions of their Web space are off-limits, and to what robots. Details of the standard were examined in a previous section in this chapter. To be considered good citizens on the Web, and for not getting trapped in infinite virtual Web spaces, all self-respecting Web robots must follow this standard.



04age08

IPR2025-00714
DATABRICKS EX1008 Page 128

III. A Web Robot Must Not Hog Resources

Web robots consume a great deal of resources. To minimize its impact on the Internet, a Web robot should keep the following in mind:

- **Request HEAD where possible.** HTTP supports a HEAD request method that retrieves only header information from Web documents, without the main body of HTML text. This incurs far less overhead than a full GET request, which retrieves entire documents and includes both headers and bodies. This feature comes in handy for Web robots to verify the existence and integrity of links in a document without necessarily retrieving all of their hyperlinked contents.
- **Specify what is needed.** HTTP provides an Accept field in its request header for a Web robot to specify to the server what kinds of data it can handle. A robot that is designed to analyze text information only, for example, should specify the following:

  Accept: x-text

  Specifying what is needed can save considerable network bandwidth because Web servers will not bother to send data that the Web robot cannot handle and might have to discard anyway.
- **Retrieve only what is needed.** URL suffixes also provide ample hints as to what type of data can be found at the other end of the link. If a link



refers to a file with the extension "ps", "zip", "Z", or "gif", for example, and the robot is equipped to handle only text data, it should not bother asking for its content from the server. After all, non-text files are fairly low-value artifacts for the purposes of indexing and querying. Although using URL suffixes is not the preferred way to do things (the recommended way is to use the Accept field in the HTTP request header), there is an enormous installed base out there that currently uses this method (all the FTP sites, for example).

Web robots always risk wandering off the Web into infinite virtual spaces. It is, therefore, imperative for Web robots to be given a list of places to avoid before embarking on a journey into Webspace. For example, URLs that begin

IPR2025-00714
DATABRICKS EX1008 Page 129

with "news:" (NEWS gateway) and "wais:" (WAIS gateway) should be filtered out in order to avoid exploring them. The robot should also pay attention to subpage references (A HREF="#abstract", for example) and not retrieve the same page more then once.

- **Retrieve at opportune times.** On some systems, there might be preferred times of access when the machines or networks are only lightly loaded. A Web robot planning to make many automatic requests to one particular site should be made aware of the site's preferred time of access.

- **Check all URLs carefully.** The Web robot should not assume that all HTML documents retrieved from the servers will be error-free. While scanning for URLs, the robot should be wary of things such as the following, which misses a matching double quote:

```
A HREF="http://somehost.somedom/doc
```

  Also, many Web sites do not use trailing slashes (/) on URLs for directories, which means that a naive strategy of concatenating names of URL subparts can result in malformed names.

- **Check all results thoroughly.** The robot should check all results thoroughly, including the status code. If a server constantly refuses to serve a number of documents, listen to what it is saying—the server might not serve documents to robots as a matter of policy.

- **Never loop or repeat.** There is always the danger of a robot getting caught in some infinite loop in the Web without the slightest idea of what has happened. To avoid this situation, the robot should keep track of all the places it has visited. It also should check to make sure that the different host addresses are not on the same machine. (For example, `web.nexor.co.uk` and `hercules.nexor.co.uk` are aliases of the same machine, which also is known by its IP address, 128.243.219.1.)

- **Retrieve in moderation.** Although Web robots can handle hundreds of documents per minute, a heavily used and multi-accessed server might not keep up. What is more, putting the server under a heavy load almost certainly will arouse the ire of many Webmasters, especially those who are less tolerant of robots.

  Robots are advised to rotate queries between different Web servers in a round-robin fashion or to "sleep" for a short period of time between requests. Retrieving one document per minute at any one particular Web server is much better than overloading it with retrieving one document per second. One document every five minutes per Web server is better still. After all, what's the rush?

- **Skip query interfaces.** Some Web documents are searchable (using the ISINDEX facility in HTML, for example) while others contain forms or are themselves dynamic documents. It is not advisable for robots to follow these links and hope

IPR2025-00714
DATABRICKS EX1008 Page 130

to get somewhere. An HTML textual analysis of the Web document, to be performed by the robot, can help determine whether any of the above cases apply.

IV. A Web Robot Must Report Errors

When a robot is traversing the Web, it might come across dangling links that point at Web pages that are obsolete, nonexistent, or inaccessible. This could be the result of the Webmaster having moved the page in question to a different location. He or she might have moved the page to a different machine or placed it under a different directory, for example. It also could be that the file in question has been renamed or that the Web server (or even the Domain Name server) has temporarily been out of service.

In all such cases, the Web robot should send an error-reporting e-mail to the address defined in the "mailto" link or the Webmaster of the site.

The Six Commandments for Robot Operators

Unleashing a Web robot on the Internet consumes substantial computational and network resources. Potential robot operators are strongly urged to reconsider their plans and to refrain from such an action until other cheaper alternatives have been fully exlored. Specifically, robot operators are urged to consider the following issues:



- The operational costs of a Web robot, in terms of computational and network resources consumed, as well as some level of vigilance and responsiveness on the part of the robot operator, must be weighed against its intended benefits.
- Sufficient computational resources and data storage capacity are required to cope with the potentially voluminous results—the Web is simply too huge for any one robot to cover.

Table 5.2 lists the six commandments for robot operators. The following subsections explain the six commandments in detail; read them carefully if you're planning on operating a Web robot.

IPR2025-00714
DATABRICKS EX1008 Page 131

I.	Thou Shalt Announce thy Robot
II.	Thou Shalt Test, Test, and Test thy Robot Locally
III.	Thou Shalt Keep thy Robot Under Control
IV.	Thou Shalt Stay in Contact with the World
V.	Thou Shalt Respect the Wishes of Webmasters
VI.	Thou Shalt Share Results with thy Neighbors

Table 5.2

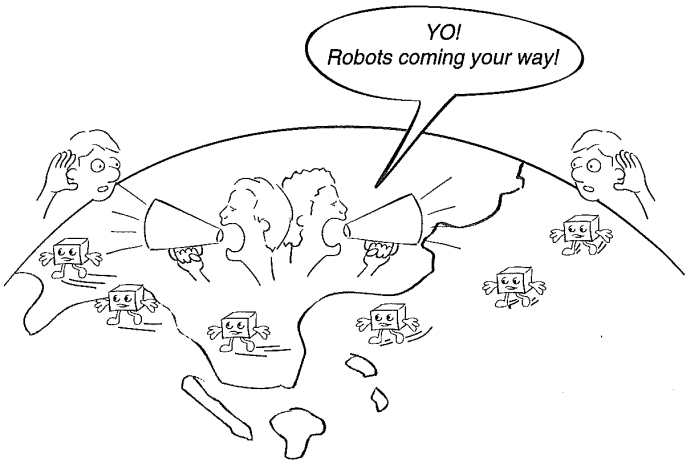
The Six Commandments for Robot Operators

I. Thou Shalt Announce thy Robot

For better communications, you should announce your robot prior to launching it on the Web; you should notify the world, perhaps the target Web sites, but most definitely the local system administator.

Notify the World

If Webmasters know that a robot is coming, they can keep an eye out for it and not be caught by surprise. A robot that benefits the entire net will be welcome and tolerated longer than one that services a smaller community.



IPR2025-00714
DATABRICKS EX1008 Page 132

Before writing or launching a robot, you should announce your intention by posting a message to the following USENET newsgroup:

```
comp.infosystems.www.providers
```

Or by sending an e-mail message to this address:

```
robots@nexor.co.uk
```

Include a brief description of the problem to be solved by the Web robot. It is possible that someone already might have been working on a similar robot, or one already might exist but is not listed.

Notify Target Sites

If your robot is targeted at a select few sites, it is professional courtesy to contact and inform the Webmasters directly.

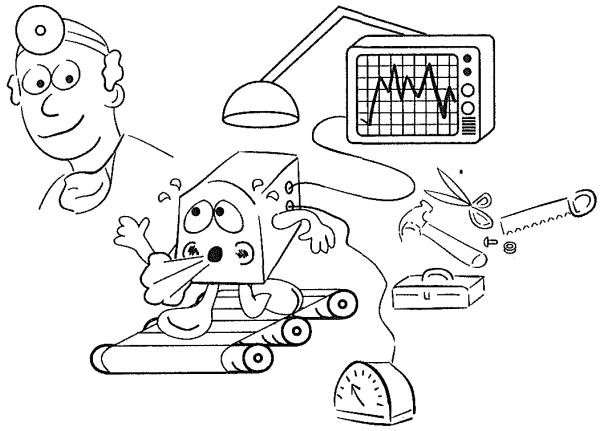
Notify Local System Administrator

Tell the local system administrator or network provider what resources or services might be used, such as increased network traffic and greater disk space utilization, when operating the robot. This way, if something goes wrong, the system administrator has been forewarned and won't have to rely on information about the robot and any resulting problems from second-hand sources.

II. Thou Shalt Test, Test, and Test thy Robot Locally

For testing purposes, you should start a number of Web servers locally to check the newly created robot. Do not try testing on remote servers before getting the bugs out of a robot. When going offsite for the first time, the robot should stay close to home. Have it start from a page with local URLs.

After completing a small test run, you should analyze the robot's performance and results. This practice helps you arrive at an estimate of how the operation would scale up to perhaps tens of thousands of documents. It soon becomes obvious if the workload might not be manageable; as a result, you can scale down the scope of the effort.



IPR2025-00714
DATABRICKS EX1008 Page 133

III. Thou Shalt Keep thy Robot Under Control

It is vital that the operator know what the robot is doing, and that the robot remain under control at all times. To accomplish this goal, follow these guidelines:

- **Log all activities.** Provide ample logging in order to track where the robot has been on the Web. To monitor the progress of the robot and keep it under control, it helps to collect useful information and to compile statistics, such as the following:

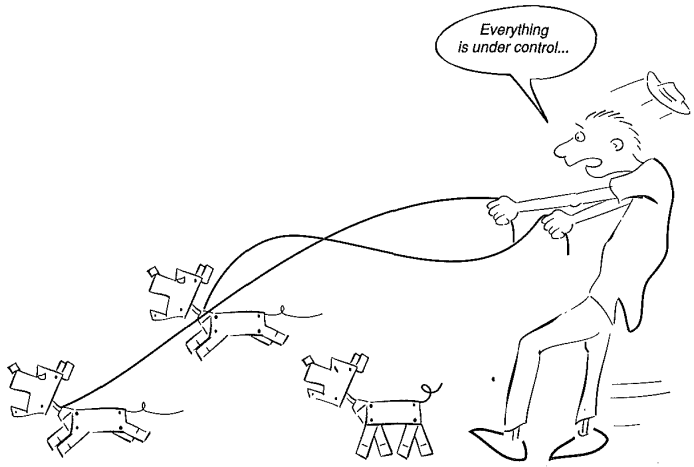
  Hosts recently visited

  Number of successes and failures

  Sizes of recently accessed files

  As was previously noted, the robot needs to know where it has been on the Web in order to prevent looping. Also, an updated estimate of the disk space requirement from time to time provides useful feedback to the operator and helps prevent a disk space crunch.

- **Provide guidance.** Design robots that can be guided easily. Commands that suspend or cancel the robot, or make it skip the current host, for example, can be very useful. For this to happen, the robot must be robust operationally—the robot needs to be checkpointed frequently during operation to ensure that the cumulative results are not lost if the robot fails.



IPR2025-00714
DATABRICKS EX1008 Page 134



IV. Thou Shalt Stay in Contact with the World

When you are running a Web robot, make sure that Webmasters can easily contact and start dialoging with you. If your robot's actions cause problems, you could be the only one who can fix it quickly. If possible, stay logged in to the machine that is running your robot so Webmasters can use *finger* or *talk* to contact you. In other words, don't go on vacation after unleashing your robot onto the Web.

The robot should be run only in your presence. Suspend the robot's operation when you are not going to be there—during weekends or after work, for example. Although it might be better for the performance of your machine to run your robot overnight, be considerate of others and the performance overhead of other machines.

V. Thou Shalt Respect the Wishes of Webmasters

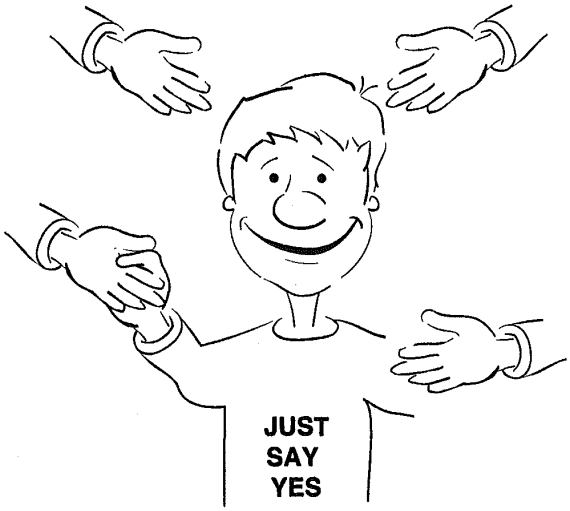
During operation, your robot will visit hundreds of sites. It probably will upset a number of Webmasters along its course. You must be prepared to respond quickly to their inquiries and tell them what your robot is doing.

If your Web robot does upset some Webmasters, instruct the robot to visit only their home pages and not go beyond. In many situations, it may be wise for the robot to pass over the complaining sites altogether.



It is not a good idea to evangelize to Webmasters, hoping to convert them to your cause and open up their Web sites to your robot. They are probably not in the least bit interested.

IPR2025-00714
DATABRICKS EX1008 Page 135



If your Web robot encounters technical barriers that Webmasters have devised to bar it from accessing their site, you should not try to make your Web robot go around them. Even though you might prove to Webmasters that it is difficult or impossible to limit access on the Web, you most likely will end up making enemies.

VI. Thou Shalt Share Results with thy Neighbors

You should archive and keep as much of the Web pages as you can store. You also should make the results accessible to the Internet community. After all, the effort to accumulate these documents has consumed considerable Internet-wide resources, and it is only fair to give something back in return. More specifically, you should do the following:

- **Share raw data.** The raw results consisting of retrieved Web pages should be made available to the Internet community, either through FTP or World Wide Web, in one form or another. This sharing of data enables interested people on the Internet to make use of the data in other interesting ways without having to duplicate the collection effort using another Web robot.
- **Publish polished results.** The Web robot is created and operated for a specific purpose; perhaps to build a specialized database or to gather some statistics. If these processed results are made available to the Web community in a polished form, people will be more appreciative of the robot's value and thus become more tolerant of its presence on the Web despite the increased network load. In addition, this is definitely a good way to get in touch with people of similar interests.

IPR2025-00714
DATABRICKS EX1008 Page 136



Robot Tips for Webmasters

If you are a Webmaster and you or your users are experiencing unusually sluggish response from your Web server, a Web robot might be attacking it with rapid-fire requests. To determine if a certain Web robot is indeed the culprit, and to find out more about it, here are some definite steps you can take:

1. Check your Web server logs carefully for signs of rapid-fire requests by paying close attention to time-stamps of multiple consecutive HTTP requests coming from the same machine address. Study the log for HTTP access request patterns to determine if indeed the sluggishness problem is caused by some offending Web robots. The HTTP request header fields User-Agent and From might reveal useful information about the Web robot and identify its operator.
2. Check Martijn Koster's *List of Robots* (or Appendix G of this book) to discover if the offending Web robot, identified in the HTTP User-Agent request header field, is one that is already known. Learn more about the culprit as needed, perhaps using Web search engines such as the WebCrawler or Lycos.
3. Find out more about the robot operator, identified in the HTTP From request header field, by means of *finger* or *rusers* over the Internet. The robot operator might also have published a Web page about himself and, more importantly, his Web robot project!

IPR2025-00714
DATABRICKS EX1008 Page 137

4. Raise the alarm in newsgroups among Webmasters, if needed, by posting to `comp.infosystems.www.providers` on USENET. You might not be alone. Chances are that there is already a thread of discussions on the topic between numerous other Webmasters facing the exact same problem.

The problem might have a simple solution: specify an entry in the robot exclusion file to exclude the offender. For example, the following entry added to the exclusion file tells the Web robot identified as NastyBot/1.0 to go away:

```
User-Agent: NastyBot/1.0  # Robot go away!
Disallow: /               # Off-limits!
```

If a Web robot is misbehaving, however, chances are that the robot creator would not also have properly implemented the robot exclusion standard. Do not get upset over it. It is also probably not wise to retaliate with the Web equivalent of a mail-bomb, which is to trap the robot into retrieving large amounts of data (perhaps a gigabyte-size HTML document generated on-the-fly) in the hope that it would choke. This would waste valuable network bandwidth and might not accomplish anything if the offending Web robot is robust, or simply smart.

It is perhaps better to try to get in contact with the robot operator and to engage in a constructive dialog, explaining clearly the problem that occurred at your Web site. You might also consider suggesting that the robot operator read *Guidelines for Robot Writers* or, perhaps, this chapter.

After your problem has been solved, you are strongly encouraged to share the experience with other Webmasters, robot builders, and robot operators in the Web community. This would save numerous other Webmasters from duplicating your efforts trying to investigate the similar problems caused by the same offending Web robots.

Web Ethics

Web ethics is an important concept for robot writers, robot operators, and Webmasters to understand. In 1942, Isaac Asimov stated his Three Laws of Robotics:

1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
2. A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Asimov's First Law of Robotics captures an essential insight: An intelligent agent should not slavishly obey human commands—its foremost goal should be to avoid harming humans. After all, society will reject autonomous agents unless there is some credible means of making them safe in the first place. But of course all this is quite abstract; the Web robots we're dealing with aren't going to chase anyone to kill them with superstrong pinchers at the ends of accordian-like arms!

Oren Etzioni and Daniel Weld, both professors at the University of Washington in Seattle who have

IPR2025-00714
DATABRICKS EX1008 Page 138

done extensive work with software robots, define a *softbot* as an agent that interacts with a software environment by issuing commands and interpreting the environment's feedback. In many respects, the softbot is very similar to a Web agent. It therefore is quite interesting to study Etzioni and Weld's formulation of a collection of *softbotic laws* (patterned after Isaac Asimov's Laws of Robotics) to govern such softbot agents (Etzioni and Weld, 1994):

- **Safety.** The softbot should not make destructive changes to the world.
- **Tidiness.** The softbot should leave the world as it first found it.
- **Thrift.** The softbot should limit its use of scarce resources.
- **Vigilance.** The softbot should refuse client actions with unknown consequences.

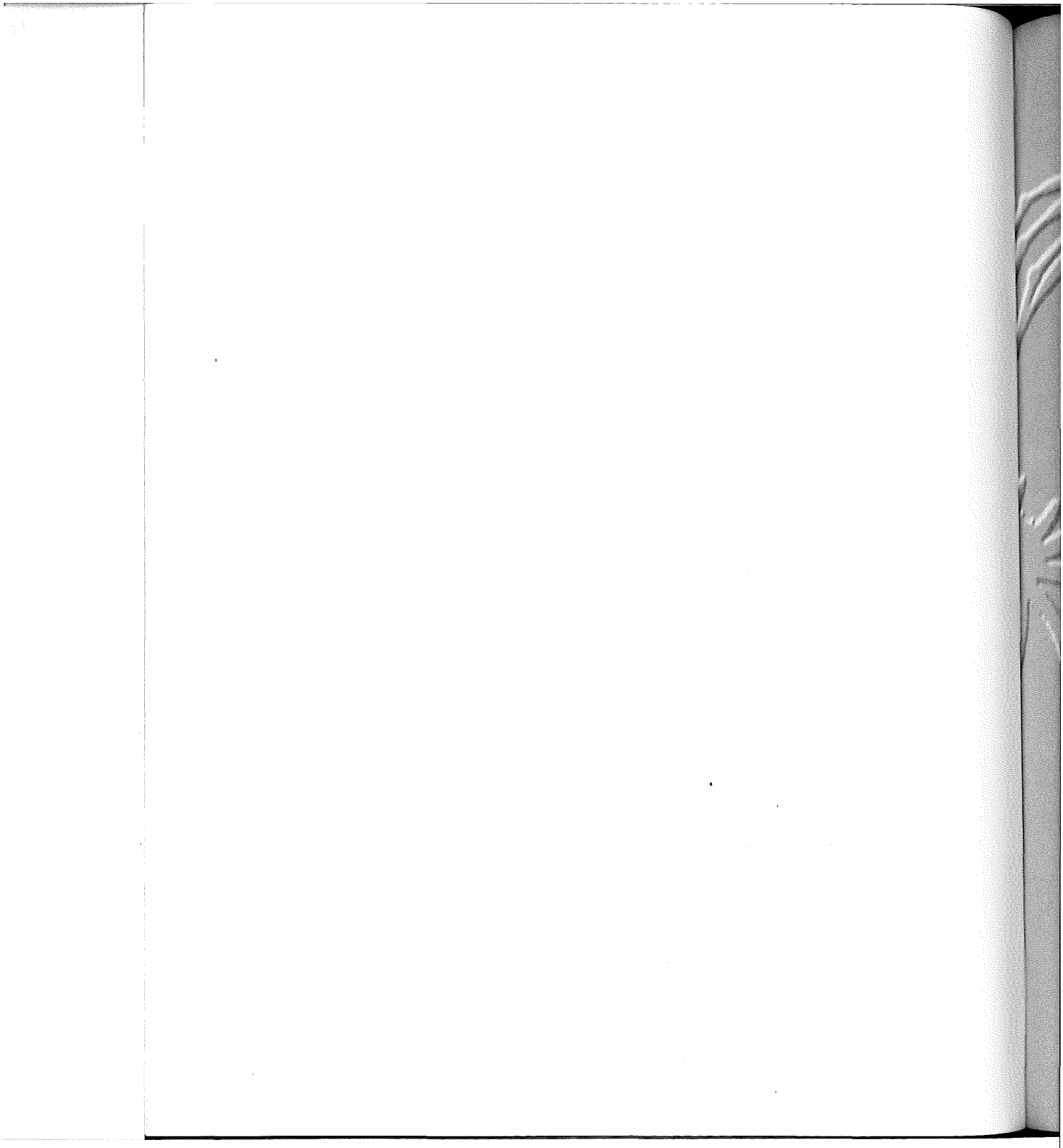
The laws of softbotics operate at a higher level when compared with the four laws of Web robotics described previously; you can probably detect some interesting commonalties that underlie the ethical aspects for all agents.

Similarly, Professor David Eichmann of the University of Houston, creator of the RBSE spider, offers his formulation of a code of conduct governing a general class of *service agents* (1994), which also includes Web robots:

- **Identity.** Agent activities should be readily discernible and traceable back to its operator.
- **Openness.** Information generated should be made accessible to the community in which the agent operates.
- **Moderation.** The rate and frequency of information acquisition should be appropriate for the capacity of the server and network so as not to create an overload situation on valuable computational and network resources.
- **Respect.** Agents should respect constraints placed on them by server administrators.
- **Authority.** Agents' services should be accurate and up-to-date.

According to Eichmann, a balance should be struck between the concerns of openness, moderation, and respect—all of which limit a service agent's scope and activities—and the concern of authority, which tends to broaden them.

IPR2025-00714
DATABRICKS EX1008 Page 139



chapter 6

HTTP: Protocol of Web Robots

The *Hypertext Transfer Protocol* (HTTP) is an application-level protocol for distributed, hypermedia information systems. HTTP has been in use since 1990 by the World Wide Web community on the Internet. The HTTP 1.0 specification aims to remain compatible with most of the existing HTTP server and client programs implemented prior to November 1994.

IPR2025-00714
DATABRICKS EX1008 Page 141

HTTP is a generic, stateless, object-oriented protocol that can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods. A feature of HTTP is the typing and negotiation of data representation, enabling systems to be built independently of the data being transferred.

Practical information systems require retrieval, search, front-end update, and annotation. HTTP enables an open-ended set of methods to be used to indicate the purpose of a request. It builds on the discipline of reference provided by the Universal Resource Identifier (URI) (BL 1994)—as a Universal Resource Locator (URL) (BLMM 1994) or as a Universal Resource Name (URN)—for indicating the resource on which a method is to be applied. Messages are passed in a format similar to that used by Internet Mail (Crocker 1982) and the Multipurpose Internet Mail Extensions (MIME) (BF 1993).



MIME is a freely available specification that offers a way to interchange text in languages with different character sets, and multimedia e-mail, among many different computer systems that use Internet mail standards.

HTTP also is used for communication between user agents and various gateways, enabling hypermedia access to existing Internet protocols, such as SMTP (Postel 1982), NNTP (KL 1986), FTP (PR 1985), Gopher (AMLJTA 1993), and WAIS (DKMSSWSG 1990). HTTP is designed to enable such gateways, through proxy servers, without any loss of the data conveyed by those earlier protocols.

HTTP is an important protocol for Web robots and key to their operations. This chapter examines the HTTP 1.0 specifications in detail. The information on HTTP 1.0 is based upon the Internet draft of HTTP 1.0 authored by Tim Bernes-Lee, Roy Fielding, and H. Frystyk Nielsen and submitted to the IETF Working Group in March 1995 (available at `http://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-http-v10-spec-03.txt`). A syntax grammar summary of HTTP 1.0 can be found in Appendix A.

Understanding HTTP Operation

The HTTP protocol is based on a request/response paradigm. A requesting program (called a *client*) establishes a connection with a receiving program (called a *server*) and sends a request to the server. A given program can be a client or a server. The use of these terms refers only to the role being performed by the program during a particular connection, rather than to the program's purpose in general.

The request transmitted to the server is in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The server responds with a status line (including its protocol version with a success or error code), followed by a MIME-like message containing server information, entity meta information, and possible body content. This entire process is covered in more detail later in this chapter.

On the Internet, communication generally takes place over a TCP/IP connection. The default port is TCP 80 (RP 94), but other ports can be used. The

IPR2025-00714
DATABRICKS EX1008 Page 142

HTTP 1.0 protocol also can be implemented on top of any other protocol on the Internet, or on other networks.

For most implementations, the client establishes the connection prior to each request, and the server closes it after sending the response. This is not a feature of the protocol, however, and is not required by the HTTP 1.0 specification. Both clients and servers must be capable of handling the premature closing of the connection by either party, which could be caused by user action, automated timeout, or program failure. The closing of the connection by either or both parties always terminates the current request, regardless of its status.

Table 6.1 explains the terminologies associated with the World Wide Web that will be used for the remainder of this chapter.

Table 6.1 World Wide Web Terms

Term	Definition
Connection	A virtual circuit established between two parties for the purpose of communication.
Message	A structured sequence of octets transmitted through the connection as the basic component of communication.
Request	An HTTP request message.
Response	An HTTP response message.
Resource	A network data object or service that can be identified by a URI.
Entity	A particular representation or rendition of a resource that can be enclosed within a request or response message. An entity consists of meta information (in the form of entity headers) and content (in the form of an entity body).
Client	A program that establishes connections for the purpose of sending requests.
User agent	The client program that is closest to the user and that initiates requests on behalf of the user.
Server	A program that accepts connections in order to service requests by sending back responses.
Origin server	The server on which a given resource resides or is to be created.

continues

IPR2025-00714
DATABRICKS EX1008 Page 143

Table 6.1, Continued

Term	Definition
Proxy	An intermediary program that usually runs on a firewall machine to other servers. (A firewall machine functions as a security barrier between the larger Internet and a smaller local area network within an organization.) A proxy server accepts requests from other clients and services them either internally or by passing them on (with possible translation). A caching proxy is a proxy server with a local cache of server responses.
Gateway	A proxy that services HTTP requests by translating them into protocols other than HTTP. The reply sent from the remote server to the gateway is likewise translated into HTTP before being forwarded to the user agent.

Messaging with HTTP

HTTP messages consist of requests from client to server and responses from server to client. These messages can be either *full* requests and responses or *simple* requests and responses.

Full requests and full responses use the generic message format of RFC 822 (Crocker 1982) for transferring entities. Both messages can include optional header fields (or simply "headers") and an entity body. A *null line* (a line with nothing preceding the *carriage return line feed*, or CRLF) separates the entity body from the headers. A full request looks like the following:

```
Method SP URI SP HTTP-Version CRLF
*( General-Header
 ¦ Request-Header
 ¦ Entity-Header )
CRLF
[ Entity-Body ]
```

A full response looks like the following:

```
HTTP-Version SP Status-Code SP Reason-Phrase CRLF
*( General-Header
 ¦ Response-Header
 ¦ Entity-Header )
CRLF
[ Entity-Body ]
```

Simple requests and responses do not allow the use of any header information and are limited to a single `GET` request method. The client is denied the benefit of content negotiation, and the server cannot identify the media type of the returned entity. A simple request looks like the following:

```
GET SP URI CRLF
```

A simple response is merely an optional entity body.

Message Headers

HTTP header fields include general header, request header, response header, and entity header fields. Each header field consists of a name followed by a colon (:) and the field value. Header fields can be

IPR2025-00714
DATABRICKS EX1008 Page 144

extended over multiple lines by preceding each extra line with at least one *linear white space* (LWS).

The order in which header fields are received is not significant.

Comments can be included in HTTP header fields by surrounding the comment text with parentheses.

General Message Header Fields

A few header fields apply in general terms to both request and response messages but do not apply to the communicating parties or to the entity being transferred. No general header field is required; however, they all are strongly recommended when their use is appropriate.

Additional general header fields can be implemented by the extension mechanism; applications that do not recognize those fields should treat them as entity header fields.

Date (When Was the Message Originated?)

The Date header represents the date and time at which the message was originated. The field value is an HTTP date. The following is an example:

```
Date: Tue, 15 Apr 1995 07:45:20 GMT
```

For most purposes, the default date can be assumed to be the current date at the receiving end. Because the date—as it is believed to be by the origin—is important for evaluating cached responses, however, origin servers should always include a Date header.

Clients should only send a Date header field in messages that include an entity body, as in the case of the `PUT` and `POST` requests; even then it is optional.

Forwarded (By Which Proxy Server?)

Proxies use the Forwarded header to indicate the intermediate steps between the user agent and the server (on requests) and between the origin server and the client (on responses). The header is intended to trace transport problems and to avoid request loops.

A message, for example, is sent from a client on dip.eecs.umich.edu to a server at `www-cis.stanford.edu port 80`, through an intermediate HTTP proxy at `agent.com port 8000`. The request received by the server at `www-cis.stanford.edu` would have the following Forwarded header field:

```
Forwarded: by http://agent.com:8000/ for
dip.eecs.umich.edu
```

Multiple Forwarded header fields are allowed in an HTTP message header and should represent each proxy that has forwarded the message.

Message-ID (How Are Messages Identified?)

The Message-ID field in HTTP gives the message a single, unique identifier that can be used to identify the message (not its contents) for "much longer" than the expected lifetime of that message.

Although not required, the address specification format typically used within a Message-ID consists of a string that is unique at the originator's machine, followed by the required at (@) character and the fully qualified domain name of that machine. The following is an example:

```
Message-ID: <9505031836.AA00266@agent.com>
```


IPR2025-00714
DATABRICKS EX1008 Page 145

The value of the Message-ID is composed using the time, date, and process id on the host `agent.com`. This method, however, is only one of many possible methods for generating a unique Message-ID. Recipients of a message should consider the entire value opaque.

MIME-Version (Is This Message MIME-Compliant?)

HTTP is not a MIME-conformant protocol. HTTP 1.0 messages, however, might include a single MIME-Version header field to indicate what version of the MIME protocol was used to construct the message. MIME 1.0 is the default for use in HTTP 1.0.

Use of the MIME-Version header field should indicate that the message is in full compliance with the MIME protocol, as defined in (BF 1993).

Current versions of HTTP 1.0 clients and servers unfortunately use this field indiscriminately, and thus receivers must not take it for granted that the message is in full compliance with MIME.

Request Message

A World Wide Web client can make requests to a World Wide Web server to begin an operation. A request message from a client to a server includes the following information within the first line (the request line):

- The method to be applied to the resource requested
- The identifier of the resource
- The protocol version in use

An HTTP request has two valid formats: the newer full request or the older simple request (for backward compatibility with HTTP 0.9).

If an HTTP 1.0 server receives a simple request, it must respond with an HTTP 0.9 simple response. An HTTP 1.0 client capable of receiving a full response should never generate a simple request.

Method

The Method token indicates the method to be performed on the resource identified by the request URI. The method is case-sensitive and extensible. The following is a list of currently specified methods:

- `GET`
- `HEAD`
- `POST`
- `PUT`
- `DELETE`
- `LINK`
- `UNLINK`

The list of methods accepted by a specific resource can be specified in an Allow entity header. The client, however, is always notified through the return code of the response whether or not a method is currently allowed on a specific resource, because this can change dynamically. Servers should return the status code `405 Method Not Allowed` if the method is known by the server but not allowed for the requested resource, and `501 Not Implemented` if the method is unknown or not implemented by the server.

IPR2025-00714
DATABRICKS EX1008 Page 146

The following sections describe the set of common methods for HTTP 1.0. Although this set can be expanded easily, additional methods cannot be assumed to have the same meaning for separately extended clients and servers. In order to maintain compatibility, the semantic definition for extension methods must be registered with the Internet Assigned Numbers Authority (IANA) (RP 1994).



The IANA is the central coordinator for the assignment of unique parameter values for Internet protocols. The IANA is chartered by the Internet Society and the Federal Network Council to act as the clearinghouse to assign and coordinate the use of numerous Internet protocol parameters.

GET (Retrieving Contents of a Resource)

The GET method retrieves information (in the form of an entity) that is identified by the request URI. If the request URI refers to a data-producing process, the produced data is returned as the entity in the response, not the source text of the process (unless that text happens to be the output of the process).

The GET method becomes a conditional GET method when the request message includes an If-Modified-Since header field. A conditional GET method requests that the identified resource be transferred only if it has been modified since the date given in the If-Modified-Since header. If the resource has not been modified since the If-Modified-Since date, the server returns a `304 Not Modified` response.

The conditional GET method is intended to reduce network usage by enabling cached entities to be refreshed without requiring multiple requests or transferring unnecessary data.

HEAD (Retrieving Only Header Information)

The HEAD method is identical to GET except that the server must not return any entity body in the response. The meta information contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request.

HEAD can be used for obtaining meta information about the resource identified by the request URI without transferring the entity body. The HEAD method is often used for testing hypertext links for validity, accessibility, and recent modification.

POST (Posting to a Resource)

The POST method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the request URI in the request line. POST is designed to allow a uniform method to cover the following functions:

- Annotating existing resources
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles
- Providing a block of data (usually a form) to a data-handling process
- Extending a database through an append operation

The actual function performed by the POST method is determined by the server and is usually dependent on the request URI. The posted entity is

IPR2025-00714
DATABRICKS EX1008 Page 147

considered to be subordinate to that URI in the same way that a file is subordinate to the directory containing it, a news article is subordinate to a newsgroup in which it is posted, or a record is subordinate to a database.

The client can apply relationships between the new resource and other existing resources by including Link header fields. The server can use the link information to perform other operations as a result of the new resource being added. For example, lists and indices might be updated. The origin server can also generate its own or additional links to other resources.

A successful `POST` does not require that the entity be created as a resource on the origin server or made accessible for future reference. That is, the action performed by the `POST` method might not result in a resource that can be identified by a URI. In this case, either `200 OK` or `204 No Content` is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

If a resource has been created on the origin server, the response should be `201 Created`. This response should contain the allocated URI, all applicable Link header fields, and an entity (preferably of type text/html) that describes the status of the request and refers to the new resource.

PUT (Creating or Modifying a Resource)

The `PUT` method requests that the enclosed entity be stored under the supplied request URI. If the request URI refers to an already existing resource, the enclosed entity should be considered a modified version of the resource residing on the origin server. The `200 OK` response should be sent back after successful completion of the request.

If the request URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server must inform the user agent through the `201 Created` response.

The fundamental difference between the `POST` and `PUT` requests is reflected in the different meaning of the request URI. The URI in a `POST` request identifies the resource that will handle the enclosed entity as an appendage. That resource can be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations.

In contrast, the URI in a `PUT` request identifies the entity enclosed with the request. The requestor of a `PUT` knows which URI is intended, and the receiver must not attempt to apply the request to some other resource. If the receiver desires that the request be applied to a different URI, it must send a `301 Moved Permanently` response; the requestor can then make its own decision regarding whether or not to redirect the request.

With `PUT`, the client can create or modify relationships between the enclosed entity and other existing resources by including Link header fields. As with `POST`, the server can use the Link information to perform other operations as a result of the request. The origin server can generate its own or additional links to other resources.

The origin server defines the actual method for determining how the resource is placed, and what

IPR2025-00714
DATABRICKS EX1008 Page 148

happens to the resource's predecessor. If version control is implemented by the origin server, the Version and Derived-From header fields should be used to help identify and control revisions to a resource.

DELETE (Getting Rid of a Resource)

The `DELETE` method requests that the origin server delete the resource identified by the request URI. This method can be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. The server should not indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

A successful response would be any of the following:

- `200 OK` if the response includes an entity describing the status
- `202 Accepted` if the action has not yet been enacted
- `204 No Content` if the response is OK but does not include an entity

LINK (Establishing Relationships with Other Resources)

The `LINK` method establishes one or more link relationships between the existing resource identified by the request URI and other existing resources. The difference between `LINK` and other methods allowing links to be established between resources is that the `LINK` method does not allow any entity body to be sent in the request and does not result in the creation of new resources.

UNLINK (Breaking Relationships with Other Resources)

The `UNLINK` method removes one or more link relationships from the existing resource identified by the request URI. These relationships might have been established using the `LINK` method or by any other method supporting the Link header. The removal of a link to a resource does not imply that the resource ceases to exist or becomes inaccessible for future references.

Request Header Fields

The request header fields allow the client to pass additional information about the request (and about the client itself) to the server. All header fields are optional and conform to the generic HTTP header syntax.

Although additional request header fields can be implemented by the extension mechanism, applications that do not recognize those fields should treat them as entity header fields.

Accept (Acceptable Media Ranges)

The Accept header field can be used to indicate a list of media ranges that are acceptable as a response to the request. An asterisk (*) is used to group media types into ranges, with */* indicating all media types and type/* indicating all subtypes of that media type. The set of ranges given by the client should represent what types are acceptable given the context of the request. The following example should verbally be interpreted as, "If you

IPR2025-00714
DATABRICKS EX1008 Page 149

have audio/basic, send it; otherwise send any audio type."

```
Accept: audio/*; q=0.2, audio/basic
```

The parameter q is used to indicate the quality factor, which represents the user's preference for that range of media types. Its default value is q=1.

If at least one Accept header is present, a quality factor of 0 is equivalent to not sending an Accept header field containing that media-type or set of media-types. If no Accept header is present, then it is assumed that the client accepts all media types. This is equivalent to the client sending the following accept header field:

```
Accept: */*
```

A more elaborate example is

```
Accept: text/plain; q=0.5, text/html, text/x-dvi;
q=0.8; mxb=100000, text/x-c
```

Verbally, this would be interpreted as, "Text/html and text/x-c are the preferred media types, but if they do not exist, then send the entity body in text/x-dvi if the entity is less than 100,000 bytes; otherwise, send text/plain." Here, the parameter mxb gives the maximum acceptable size of the entity body (in decimal number of octets, defaults to infinity) for that range of media types.

It must be emphasized that the Accept field should only be used when it is necessary to do the following:

- Restrict the response media types to a subset of those possible
- Indicate qualitative preference for specific media types
- Indicate the acceptance of unusual media types

Accept-Charset (Preferred Character Sets)

The Accept-Charset header field can be used to indicate a list of preferred character sets other than the default US-ASCII and ISO-8859-1. This field allows clients capable of understanding more comprehensive or special-purpose character sets to signal that capability to a server that is capable of representing documents in those character sets. An example follows:

```
Accept-Charset: iso-8859-5, unicode-1-1
```

The value of this field should not include US-ASCII or ISO-8859-1 because those values are always assumed by default.

Accept-Encoding (Acceptable Encodings)

The Accept-Encoding header field is similar to Accept, but it lists the encoding-mechanisms and transfer-encoding values that are acceptable in the response. An example of its use follows:

```
Accept-Encoding: compress, base64, gzip, quoted-
printable
```

The field value should never include the identity transfer-encoding values (7bit, 8bit, and binary) because they actually represent no encoding. If no Accept-Encoding field is present in a request, it must be assumed that the client does not accept any encoding-mechanism except for the identity transfer-encodings.

Accept-Language (Preferred Natural Languages)

The Accept-Language header field is similar to Accept, but it lists the set of natural languages that are preferred as a response to the request. Languages are listed in the order of their preference to

IPR2025-00714
DATABRICKS EX1008 Page 150

the user. The following example would mean "Send me a Danish version if you have it, or else a British English version."

```
Accept-Language: dk, en-gb
```

If the server cannot fulfill the request with one or more of the languages given, or if the languages only represent a subset of a multi-linguistic entity body, it is acceptable to serve the request in an unspecified language.

Authorization (Credentials of User Agent)

A user agent that wants to authenticate itself with a server (usually, but not necessarily, after receiving a `401 Unauthorized` response), may do so by including an Authorization header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested. The following is an example:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

If a request is authenticated and a realm specified, the same credentials should be valid for all other requests within this realm.

From (Originator of This Request)

If given, the From header field should contain an Internet e-mail address for the human user who controls the requesting user agent. Here is an example:

```
From: webmaster@w3.org
```

This header field may be used for logging purposes and as a means for identifying the source of invalid or unwanted requests. The interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsibility for the method performed. In particular, Web robot agents should include this header so that the responsible operator of the Web robot can be contacted if problems occur on the receiving end.

The Internet e-mail address in this field does not have to correspond to the Internet host that issued the request. When a request is passed through a proxy, for example, the original issuer's address should be used.

If-Modified-Since (Has the Resource Been Modified Since?)

The If-Modified-Since header field is used with the `GET` method to make it conditional. If the requested resource has not been modified since the time specified in this field, a copy of the resource is not returned from the server; instead, a `304 Not Modified` response is returned without any entity body. An example of the field follows:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

The purpose of this feature is to allow efficient updates of local cache information with a minimum amount of transaction overhead. The same functionality can be obtained, though with much greater overhead, by issuing a `HEAD` request and following it with a `GET` request if the server indicates that the entity has been modified.

Pragma (Server Directives to Apply)

The Pragma header field is used to specify directives that must be applied to all servers along the request chain (where relevant). The directives typically specify behavior that prevents intermediate

IPR2025-00714
DATABRICKS EX1008 Page 151

proxies from changing the nature of the request. HTTP 1.0 only defines meaning for the no-cache directive:

```
Pragma: no-cache
```

When the no-cache directive is present, a caching proxy must forward the request toward the origin server even if it has a cached copy of what is being requested. This allows a client to insist upon receiving an authoritative response to its request. It also allows a client to refresh a cached copy that has become corrupted or is known to be stale.

Pragmas must be passed through by a proxy even when they have significance to that proxy. This is necessary in cases when the request has to go through many proxies, and the pragma might affect all of them. It is not possible to specify a pragma for a specific proxy; however, any pragma-directive not relevant to a gateway or proxy should be ignored.

Referer (Document That Referred This URI)

The Referer field allows the client to specify, for the server's benefit, the address (URI) of the document (or element within the document) from which the request URI was obtained. This allows a server to generate lists of back-links to documents for interest, logging, optimized caching, and so on. It also allows obsolete or mistyped links to be traced for maintenance. Here's an example:

```
Referer: http://info.cern.ch/hypertext/DataSources/
Overview.html
```

If a partial URI is given, it should be interpreted relative to the request URI.

User-Agent (Client Program That Originated the Request)

The User-Agent field contains information about the user agent originating the request. This information is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. Although it is not required, user agents should always include this field with requests.

The field can contain multiple tokens specifying the product name, with an optional slash and version designator, and other products that form a significant part of the user agent. By convention, the products are listed in order of their significance for identifying the application. The following is an example:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

Product tokens should be short and to the point. The User-Agent field can include additional information within comments that are not part of the value of the field.

Response Message

After receiving and interpreting a request message, a server responds in the form of an HTTP response message. A simple response should only be sent in response to an HTTP 0.9 simple request or if the server only supports the more limited HTTP 0.9 protocol.

IPR2025-00714
DATABRICKS EX1008 Page 152

If a client sends an HTTP 1.0 full request and receives a response that does not begin with a status line, it should assume that the response is simple and parse it accordingly. Note that the simple response consists only of the entity body and is terminated by the server closing the connection.

The first line of a full response message (that is, the status line) consists of the following:

- The protocol version
- A numeric status code
- The associated textual phrase

Because a status line always begins with the protocol version (HTTP 1.0), the presence of that expression is considered sufficient to differentiate a full response from a simple response. Although the simple response format can allow such an expression to occur at the beginning of an entity body (and thus cause a misinterpretation of the message if it was given in response to a full request), the likelihood of such an occurrence is negligible.

Status Codes and Reason Phrases

The server returns a 3-digit status code, plus a short textual description of the status code, as a result of attempting to understand and satisfy client request. The first digit of the status code defines the class of response, as shown in table 6.2.

HTTP status codes are extensible and should be registered with the IANA. The classes of 2*xx* successful status code are presented in table 6.3.

Table 6.2 Classes of HTTP Response Code

Digit	Type	Description
1*xx*	Informational	Not used, but reserved for future use.
2*xx*	Successful	The action was successfully received, understood, and accepted.
3*xx*	Redirection	Further action must be taken to complete the request.
4*xx*	Client Error	The request contains bad syntax or cannot be fulfilled.
5*xx*	Server Error	The server failed to fulfill an apparently valid request.

IPR2025-00714
DATABRICKS EX1008 Page 153

Table 6.3 2xx Successful

Status Code	Explanation
`200 OK`	The request has been fulfilled and an entity corresponding to the requested resource is being sent in the response.
`201 Created`	The request has been fulfilled and resulted in a new resource being created.
`202 Accepted`	The request has been accepted for processing, but the processing has not been completed.
`203 Provisional Information`	The returned meta information in the entity header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy.
`204 No Content`	The server has fulfilled the request, but there is no new information to send back.

The class of 3*xx* redirection status code are presented in table 6.4.

Table 6.4 3xx Redirection

Status Code	Explanation
`300 Multiple Choices`	The requested resource is available at one or more locations and a preferred location could not be determined through content negotiation.
`301 Moved Permanently`	The requested resource has been assigned a new permanent URI, and any future references to this resource must be done using the returned URI.
`302 Moved Temporarily`	The requested resource resides temporarily under a different URI.
`303 Method`	This code is obsolete.
`304 Not Modified`	If the client has performed a conditional GET request and access is allowed, but the document has not been modified since the date and time specified in the If-Modified-Since field, the server shall respond with this status code and not send an entity body to the client.

IPR2025-00714
DATABRICKS EX1008 Page 154

The classes of 4*xx* client error status code are presented in table 6.5.

Table 6.5 4xx Client Error

Status Code	Explanation
`400 Bad Request`	The request had bad syntax or was inherently impossible to be satisfied.
`401 Unauthorized`	The request requires user authentication.
`402 Payment Required`	This code is not currently supported.
`403 Forbidden`	The request is forbidden for some reason that remains unknown to the client.
`404 Not Found`	The server has not found anything matching the request URI.
`405 Method Not Allowed`	The method specified in the request line is not allowed for the resource identified by the request URI.
`406 None Acceptable`	The server has found a resource matching the request URI, but not one that satisfies the conditions identified by the Accept and Accept-Encoding request headers.
`407 Proxy Authentication Required`	This code is reserved for future use.
`408 Request Timeout`	The client did not produce a request within the time that the server was prepared to wait.
`409 Conflict`	The request could not be completed due to a conflict with the current state of the resource.
`410 Gone`	The requested resource is no longer available at the server and no forwarding address is known.

IPR2025-00714
DATABRICKS EX1008 Page 155

The classes of 5*xx* server error status code are presented in table 6.6.

Table 6.6 5xx Server Errors

Status Code	Explanation
`500 Internal Server Error`	The server encountered an unexpected condition that prevented it from fulfilling the request.
`501 Not Implemented`	The server does not support the functionality required to fulfill the request.
`502 Bad Gateway`	The server received an invalid response from the gateway or upstream server it accessed in attempting to complete the request.
`503 Service Unavailable`	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
`504 Gateway Timeout`	The server did not receive a timely response from the gateway or upstream server it accessed in attempting to complete the request.

HTTP applications are not required to understand the meaning of all registered status codes. Applications are required, however, to understand the class of any status code (as indicated by the first digit) and to treat the response as being equivalent to the *x*00 status code of that class.

If an unknown status code of 421 is received by the client, for example, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents are encouraged to present the entity returned with the response to the user because that entity is likely to include human-readable information that will explain the unusual status.

Response Header Fields

The response header fields allow the server to pass additional information about the response that cannot be placed in the status line. These header fields are not intended to give information about an entity body returned in the response, but about the server itself.

Although additional response header fields can be implemented by means of the extension mechanism, applications that do not recognize those fields should treat them as entity header fields.

IPR2025-00714
DATABRICKS EX1008 Page 156

Public (Non-Standard Methods Supported by Server)

The Public header field lists the set of non-standard methods supported by the server. This field informs the recipient of the server's capabilities regarding unusual methods. The field value should not include the methods predefined for HTTP 1.0. The following is an example of its use:

```
Public: OPTIONS, MGET, MHEAD
```

This header field applies only to the current connection. If the response passes through a proxy, the proxy must either remove the Public header field or replace it with one applicable to its own capabilities.

Retry-After (When to Retry Again)

The Retry-After header field can be used with `503 Service Unavailable` to indicate how long the service is expected to be unavailable to the requesting client. The value of this field can be either a full HTTP date or an integer number of seconds (in decimal) after the time of the response. Two examples of its use follow:

```
Retry-After: Mon, 02 Jan 1995 15:00:00 GMT
```

```
Retry-After: 120
```

In the latter example, the delay is 2 minutes.

Server (Server Program Handling the Request)

The Server header field contains information about the software being used by the origin server program handling the request. The field is analogous to the User-Agent field. The following is an example:

```
Server: CERN/3.0 libwww/2.17
```

If the response is being forwarded through a proxy, the proxy application must not add its data to the product list. Instead, it should include a Forwarded field.

WWW-Authenticate (Challenge to the Client)

The WWW-Authenticate header field must be included as part of a `401 Unauthorized` response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the request URI.

Entity

Full request and full response messages can transfer an entity within some requests and responses. An entity consists of entity header fields and usually an entity body. In this section, both the sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

Entity Header Fields

Entity header fields define optional meta information about the Entity body or about the resource identified by the request (where no body is present). The recognized entity header fields are listed as follows:

- Allow
- Content-Encoding
- Content-Language
- Content-Length
- Content-Transfer-Encoding

IPR2025-00714
DATABRICKS EX1008 Page 157

- Content-Type
- Derived-From
- Expires
- Last-Modified
- Link
- Location
- Title
- URI
- Version

Other header fields are allowed but cannot be assumed to be recognizable by the recipient. Unknown header fields should be ignored by the recipient and forwarded by proxies.

Allow (Methods Applicable to Requested URI)

The Allow header field lists the set of methods supported by the resource identified by the request URI. It informs the recipient of valid methods associated with the resource. It must be present in a response with status code `405 Method Not Allowed`. An example of use is the following:

```
Allow: GET, HEAD, PUT
```

This field has no default value; if left undefined, the set of allowed methods is defined by the origin server at the time of each request.

If a response passes through a proxy that does not understand one or more of the methods indicated in the Allow header, the proxy must not modify the Allow header.

Content-Encoding (How Are Contents Encoded?)

The Content-Encoding header field is used as a modifier to the media type. Its value indicates what additional encoding mechanism has been applied to the resource. Its value also indicates what decoding mechanism must be applied to obtain the media type referenced by the Content-Type header field.

The Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type. An example of its use follows:

```
Content-Encoding: gzip
```

The Content-Encoding is a characteristic of the resource identified by the request URI. Typically, the resource is stored with this encoding and is only decoded before rendering at the user agent.

Content-Language (List of Natural Languages Intended)

The Content-Language field describes the natural language(s) of the intended audience for the enclosed entity. Note that this might not be equivalent to all the languages used within the entity.

Content-Language allows a selective consumer to identify and differentiate resources according to the consumer's own preferred language. If, for example, the body content is intended only for a Danish audience, the appropriate field is this:

```
Content-Language: dk
```

If no Content-Language is specified, the default is that the content is intended for all language

IPR2025-00714
DATABRICKS EX1008 Page 158

audiences. This can mean that the sender does not consider it to be specific to any natural language, or that the sender does not know for which language it is intended.

Multiple languages can be listed for content that is intended for multiple audiences. For example, a rendition of the Treaty of Waitangi, presented simultaneously in the original Maori and English versions, would call for this:

```
Content-Language: mi, en
```

However, just because multiple languages are present within an entity does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as A First Lesson in Latin, which is clearly intended to be used by an English audience. In this case, the Content-Language should only include en.

Content-Language can be applied to any media type—it should not be considered limited to textual documents.

Content-Length (Size of Entity)

The Content-Length header field indicates the size of the entity body (in decimal number of octets) sent to the recipient. In the case of the `HEAD` method, it is the size of the entity body that would have been sent had the request been a `GET`. An example follows:

```
Content-Length: 2395
```

Although it is not required, applications are strongly encouraged to use this field to indicate the size of the entity body to be transferred, regardless of the media type of the entity.

Content-Transfer-Encoding (How Are Contents Encoded for Transfer?)

The Content-Transfer-Encoding (CTE) header indicates what (if any) type of transformation has been applied to the entity to safely transfer it between the sender and the recipient. This differs from the Content-Encoding in that the CTE is a property of the message, not of the original resource.

Because all HTTP transactions take place on an 8-bit clean connection, the default Content-Transfer-Encoding for all messages is binary. However, HTTP can be used to transfer MIME messages which already have a defined CTE. An example follows:

```
Content-Transfer-Encoding: quoted-printable
```

Many older HTTP 1.0 applications do not understand the Content-Transfer-Encoding header. However, future HTTP 1.0 applications are required to understand it upon receipt. Gateways to MIME-compliant protocols are the only HTTP applications that would generate a CTE.

Content-Type (Media Type of the Entity)

The Content-Type header field indicates the media type of the entity body sent to the recipient. In the case of the `HEAD` method, it is the media type that would have been sent had the request been a `GET`. An example follows:

```
Content-Type: text/html; charset=ISO-8859-4
```

The Content-Type header field has no default value.

Derived-From (Which Version Derives This Entity?)

The Derived-From header field indicates the version tag of the resource from which the enclosed

IPR2025-00714
DATABRICKS EX1008 Page 159

entity was derived before modifications by the sender. This field is used to help manage the process of merging successive changes to a resource, particularly when such changes are being made in parallel and from multiple sources. Here's an example use of the field:

```
Derived-From: 3.1.2
```

The Derived-From field is required for PUT requests if the entity being put was previously retrieved from the same URI and a Version header was included with the entity when it was last retrieved.

Expires (When Does the Entity Expire?)

The Expires field gives the date and time after which the entity should be considered stale. This allows information providers to suggest the volatility of the resource. Caching clients (including proxies) must not cache this copy of the resource beyond the date given, unless its status has been updated by a later check of the origin server.

The format is an absolute date and time. An example of its use follows:

```
Expires: Thu, 01 Dec 1994 23:00:00 GMT
```



Applications are encouraged to be tolerant of bad or misinformed implementations of the Expires header. In particular, recipients might want to recognize a delta-seconds value (any decimal integer) as representing the number of seconds after receipt of the message that its contents should be considered expired. Likewise, a value of zero (0) or an invalid date format can be considered equivalent to an expires immediately.

Last-Modified (When Was the Resource Last Modified?)

The Last-Modified header field indicates the date and time at which the sender believes the resource was last modified. The exact meaning of this field is defined in terms of how the receiver should interpret it; if the receiver has a copy of this resource that is older than the current date given by the Last-Modified field, that copy should be considered stale.

Here's a example of its use:

```
Last-Modified: Tue, 04 Apr 1995 07:39:26 GMT
```

The exact meaning of this header field depends on the implementation of the sender and the nature of the original resource. For files, it might be just the file system "last-mod" date. For virtual objects, it might be the last time the internal state changed. In any case, the recipient should only know (and care) about the result—whatever gets stuck in the Last-Modified field—and not worry about how that result was obtained.

Link (How Do Other Resources Relate to This Entity?)

The Link header provides a means for describing a relationship between the entity and some other resource. An entity can include multiple Link values. Links at the meta information level typically indicate relationships like hierarchical structure and navigation paths. The Link field means the same as the <LINK> element in HTML (BLC 1995).

Relation values are not case-sensitive and might be extended within the constraints of the sgml-name syntax. There are no predefined link relationship values for HTTP 1.0. The title parameter can

IPR2025-00714
DATABRICKS EX1008 Page 160

be used to label the destination of a link such that it can be used as identification within a human-readable menu. Examples of usage include:

```
Link: <http://www.cern.ch/TheBook/chapter2>;
rel="Previous"

Link: <mailto:timbl@w3.org>; rev="Made"; title="Tim
Berners-Lee"
```

The first example indicates that the entity is previous to chapter 2 in a logical navigation path. The second indicates that the publisher of the resource is identified by the given e-mail address.

Location (Where to Locate the Resource)

The Location header field is an earlier form of the URI header and is considered obsolete. HTTP 1.0 applications, however, should continue to support the Location header to properly interface with older applications. The purpose of Location is identical to that of the URI header, except that no variants can be specified and only one absolute location URL is allowed. An example follows:

```
Location: http://info.cern.ch/hypertext/WWW/
NewLocation.html
```

URI (Entity's Resource Origin)

The Title header field indicates the title of the entity. Here's an example of the field:

```
Title: Hypertext Transfer Protocol — HTTP/1.0
```

This field is to be considered the same as the <TITLE> element in HTML (BLC 1995).

The URI header field can contain one or more Universal Resource Identifiers (URIs) by which the resource origin of the entity can be identified. This field is required for the 201, 301, and 302 response messages and can be included in any message that contains resource meta information.

Any URI specified in this field can be either absolute or relative to the URI given in the request line. The URI header improves upon the Location header field. For backward compatibility with older clients, servers are encouraged to include both header fields in 301 and 302 responses.

The URI header can also be used by a client performing a `POST` request to suggest a URI for the new entity. The server's response must include the actual URI(s) of the new resource if one is successfully created (status 201).

If a URI refers to a set of variants, then the dimensions of that variance must be given with a *vary* parameter. One example is this:

```
URI: <http://info.cern.ch/hypertext/WWW/
TheProject.multi>; vary="type,language"
```

This indicates that the URI covers a group of entities that vary in media type and natural language. A request for that URI will result in a response that depends upon the client's request headers for Accept and Accept-Language. Similar dimensions exist for the Accept-Encoding, Accept-Charset, Version, and User-Agent header fields, as demonstrated in the following example:

```
URI: <TheProject.ps>;vary="encoding,version",
<TheProject.html>; vary="user agent,charset,version"
```

Version (Entity's Version)

The Version field defines the version tag associated with a rendition of an evolving entity. Together with

IPR2025-00714
DATABRICKS EX1008 Page 161

the Derived-From field, it enables a group of people to work simultaneously on the creation of a work as an iterative process. The field should be used to allow evolution of a particular work along a single path. Examples of the Version field include:

```
Version: 3.1.2

Version: "R5 19950404-07:39:26"

Version: 1.4a3-gamma6
```

The version tag should be considered opaque to all parties except the origin server. A user agent can request a particular version of an entity by including its tag in a Version header as part of the request. Similarly, a user agent can suggest a value for the version of an entity transferred via a PUT or POST request. However, only the origin server can reliably assign or increment the version tag of an entity.

Entity Body

The entity body (if any) sent with an HTTP 1.0 request or response is in a format and encoding defined by the entity header fields.

An entity body is included with a request message only when the request method calls for one. This specification defines two request methods, POST and PUT, that allow an entity body. In general, the presence of an entity body in a request is signaled by the inclusion of a Content-Length and/or Content-Transfer-Encoding header field in the request message headers.



Most current implementations of the POST and PUT methods require a valid Content-Length header field. This can cause problems for some systems that do not know the size of the entity body before transmission. Experimental implementations (and future versions of HTTP) use a packetized Content-Transfer-Encoding to obviate the need for a Content-Length.

For response messages, whether an entity body is included with a message is dependent on both the request method and the response code. All responses to the HEAD request method must not include a body, even though the presence of Content header fields might lead one to believe they should. Similarly, the responses `204 No Content`, `304 Not Modified`, and `406 None Acceptable` must not include a body.

Type

When an entity body is included with a message, the data type of that body is determined by the header fields Content-Type, Content-Encoding, and Content-Transfer-Encoding. These define a three-layer, ordered encoding model, which follows:

```
entity-body←Content-Transfer-Encoding( Content-
Encoding( Content-Type ) )
```

A Content-Type specifies the media type of the underlying data. A Content-Encoding can be used to indicate an additional encoding mechanism applied to the type (usually for the purpose of data compression) that is a property of the resource requested. A Content-Transfer-Encoding can be applied by a transport agent to ensure safe and proper transfer of the message. Note that the Content-Transfer-Encoding is a property of the message, not of the resource.

IPR2025-00714
DATABRICKS EX1008 Page 162

The Content-Type header field has no default value. If—and only if—the media type is not given by a Content-Type header (as is always the case for simple response messages), the receiver might attempt to guess the media type through inspection of its content or the name extension(s) of the URL used to access the resource. If the media type remains unknown, the receiver should treat it as type application/octet-stream.

Length

When an entity body is included with a message, the length of that body can be determined in many ways. If a Content-Length header field is present, its value in bytes (number of octets) represents the length of the entity body. Otherwise, the body length is determined by the Content-Type (for types with an explicit end-of-body delimiter), the Content-Transfer-Encoding (for packetized encodings), or the closing of the connection by the server. Note that the latter cannot be used to indicate the end of a request body because it leaves no possibility for the server to send back a response.



Some older servers supply an invalid *Content-Length* when sending a document that contains additional bytes (for example, preprocessor supplied data) dynamically inserted into the data stream. Therefore, unless the client knows that it is receiving a response from a compliant server, it should not depend on the *Content-Length* value being correct.

Protocol Parameters

The protocol parameters specify the HTTP version, URIs, and date/time formats used in HTTP.

HTTP Version

The protocol version indicates the format of a message and the sender's capacity for understanding further HTTP communication.

The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message. If the protocol version is not specified, it defaults to the simple HTTP 0.9 format.

A proxy must never send a message with a version number greater than its native version; if a higher version request is received, the proxy must either downgrade the request version or respond with an error. Requests with a version lower than that of the proxy's native format can be upgraded by the proxy before being forwarded.

Universal Resource Identifiers

For details on the URI, the reader is referred to RFC 1630 (BL 1994), which provides a brief description of the allowed characters and the hex encoding used in the escaping scheme. Examples of URI follow:

```
telnet://debra.dgbt.doc.ca:3000

http://www.mcom.com/

ftp://prep.ai.mit.edu/pub/gnu/
```

Date/Time Formats

HTTP 1.0 applications have historically allowed three different formats for the representation of date/time stamps:

IPR2025-00714
DATABRICKS EX1008 Page 163

```
Tue, 04 Apr 1995 07:39:26 GMT   ; RFC 822, updated
by RFC 1123

Tuesday, 04-Apr-95 07:39:26 GMT ; RFC 850, obso-
leted by RFC 1036

Tue Apr  4 07:39:26 1995        ; ANSI C's
asctime() format
```

The first format is preferred as an Internet standard and represents a fixed-length subset of that defined by RFC 1123 (Braden 1989), which is an update to RFC 822 (Crocker 1982). The second format is in common use, but is obsolete and lacks a four-digit year. HTTP 1.0 clients and servers must accept all three formats, but should never generate the third (asctime) format.

Content Parameters

The content parameters specify the media types, character sets, encoding mechanisms, transfer encodings, and language tags used in HTTP.

Media Types

HTTP uses Internet Media Types (Postel 1994), formerly referred to as MIME Content-Types (BF 1993), to provide open and extensible data typing and type negotiation. Examples of registered Internet media types include:

```
audio/basic

video/mpeg

image/gif

text/plain

application/postscript
```

With HTTP, user agents can identify acceptable media types as part of the connection. They are thus also allowed to use non-registered types, but their usage must not conflict with the IANA registry. All media types registered by IANA must be preferred over extension tokens.



HTTP does not encourage the use of an *x-* prefix for unofficial types except for short experimental use between consenting applications.

Character Sets

Character sets are identified by case-insensitive tokens. The complete set of allowed charset values are defined by the IANA Character Set registry (RP 1994). The following are the names for those character sets most likely to be used with HTTP entities.

- US-ASCII
- ISO-8859-1
- ISO-8859-2
- ISO-8859-3
- ISO-8859-4
- ISO-8859-5
- ISO-8859-6
- ISO-8859-7

IPR2025-00714
DATABRICKS EX1008 Page 164

- ISO-8859-8
- ISO-8859-9
- ISO-2022-JP
- ISO-2022-JP-2
- ISO-2022-KR
- UNICODE-1-1
- UNICODE-1-1-UTF-7
- UNICODE-1-1-UTF-8

This set of charset values includes those registered by RFC 1521 (BF 1993)—the US-ASCII (ANSI 1986) and ISO8859 (ISO 1990) character sets—and other character set names specifically recommended for use within MIME charset parameters.

Encoding Mechanisms

Encoding mechanism values indicate an encoding transformation that has been or can be applied to a resource. Encoding mechanisms allow a document to be compressed or encrypted without losing the identity of its underlying media type.

Typically, the resource is stored with this encoding and is only decoded before rendering. Two values are defined by this specification: gzip and compress.



HTTP 1.0 applications should consider x-gzip and x-compress to be equivalent to gzip and compress, respectively.

All encoding-mechanism values are case-insensitive. HTTP 1.0 uses encoding-mechanism values in the Accept-Encoding and Content-Encoding header fields. Although the value describes the encoding-mechanism, it also indicates which decoding mechanism is required to remove the encoding.

Transfer Encodings

Transfer encoding values are used to indicate an encoding transformation that has been, can be, or might need to be applied to an entity body to ensure safe transport through the network.



Transfer encodings are only used with entities destined for or retrieved from MIME-conformant systems. They rarely occur in an HTTP 1.0 message. This differs from an encoding-mechanism in that the transfer encoding is a property of the message, not of the original resource.

HTTP defines the following transfer-encoding values:

- **Binary.** No encoding and body can contain any set of octets.
- **8bit.** Same as binary but with added restrictions that carriage return and linefeed characters only occur as part of CR/LF line separators, all lines are short (less than 1000 octets), and no NULs (octet 0) are present.
- **7bit.** Same as 8bit but with added restriction that all octets are 7-bit US-ASCII characters.
- **Quoted-printable.** Encoding consisting of 7-bit US-ASCII characters applied to body.
- **Base64.** Encoding consisting of 7-bit US-ASCII characters applied to body.

IPR2025-00714
DATABRICKS EX1008 Page 165

All transfer-encoding values are case-insensitive. HTTP 1.0 uses transfer-encoding values in the Accept-Encoding and Content-Transfer-Encoding header fields.

Language Tags

A language tag identifies a natural language used by human beings for communication of information to other human beings. Computer languages are explicitly excluded. The HTTP 1.0 protocol uses language tags within the Accept-Language and Content-Language header fields.

The syntax and registry of HTTP language tags is the same as that defined by RFC 1766 (Alvestrand 1995). Sample tags include the following:

```
en, en-US, en-cockney, i-cherokee, x-pig-latin
```

All tags are to be treated as case-insensitive. The namespace of language tags is administered by the IANA.

Content Negotiation

Content negotiation is an optional feature of the HTTP protocol. It allows a preferred content representation to be pre-selected within a single HTTP request-response round-trip.

During content negotiation, the server first determines whether there are any content variants for the requested resource. Content variants can be multiple copies of the same image or text in different file formats. They can also be implemented by means of a set of dynamic conversion filters.

If there are no variant forms of the resource, the negotiation is limited to whether that single media type is acceptable under the constraints given by the Accept request header field (if any).

If variants are available, those variants that are completely unacceptable should be removed from consideration first. Unacceptable variants include:

- Those with a Content-Encoding not listed in an Accept-Encoding field
- Those with a character subset (other than the default ISO-8859-1) not listed in an Accept-Charset field
- Those with a media type not within any of the media ranges of an explicitly constrained Accept field (or listed with a zero quality parameter)

If no acceptable variants remain at this point, the server should respond with a `406 None Acceptable` response message.

If more than one variant remains, and at least one has a Content-Language within those listed by an Accept-Language field, any variants that do not match the language constraint are removed from further consideration.

If multiple choices still remain, the selection is further narrowed by calculating and comparing the relative quality of the available media types. If multiple representations exist for a single media type, then the one with the lowest byte count is preferred.

Finally, there might still be multiple choices available to the user. If so, the server can either choose

IPR2025-00714
DATABRICKS EX1008 Page 166

one from those available and respond with `200 OK`, or respond with `300 Multiple Choices` and include an entity describing the choices.

Access Authentication

In HTTP, a server can challenge a user agent request, and a user agent can provide authentication information in response to that challenge. HTTP provides a simple challenge-response authorization mechanism to do this.

The basic authentication scheme is based on the model that the user agent must authenticate itself with a user-ID and a password for each realm of the resource being requested. The server will service the request only if it can validate the user-ID and password for the domain of the requested resource.

The server issues the `401 Unauthorized` response message (in response to a user agent request) to challenge the authorization of a user agent. This response must include a WWW-Authenticate header field containing the challenge applicable to the requested resource.

The user agent can authenticate itself with a server (after receiving a 401 response) by including an Authorization header field with the next request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

If the user agent wants to send the user-ID *Aladdin* and password *open sesame*, for example, it would use the following header field:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

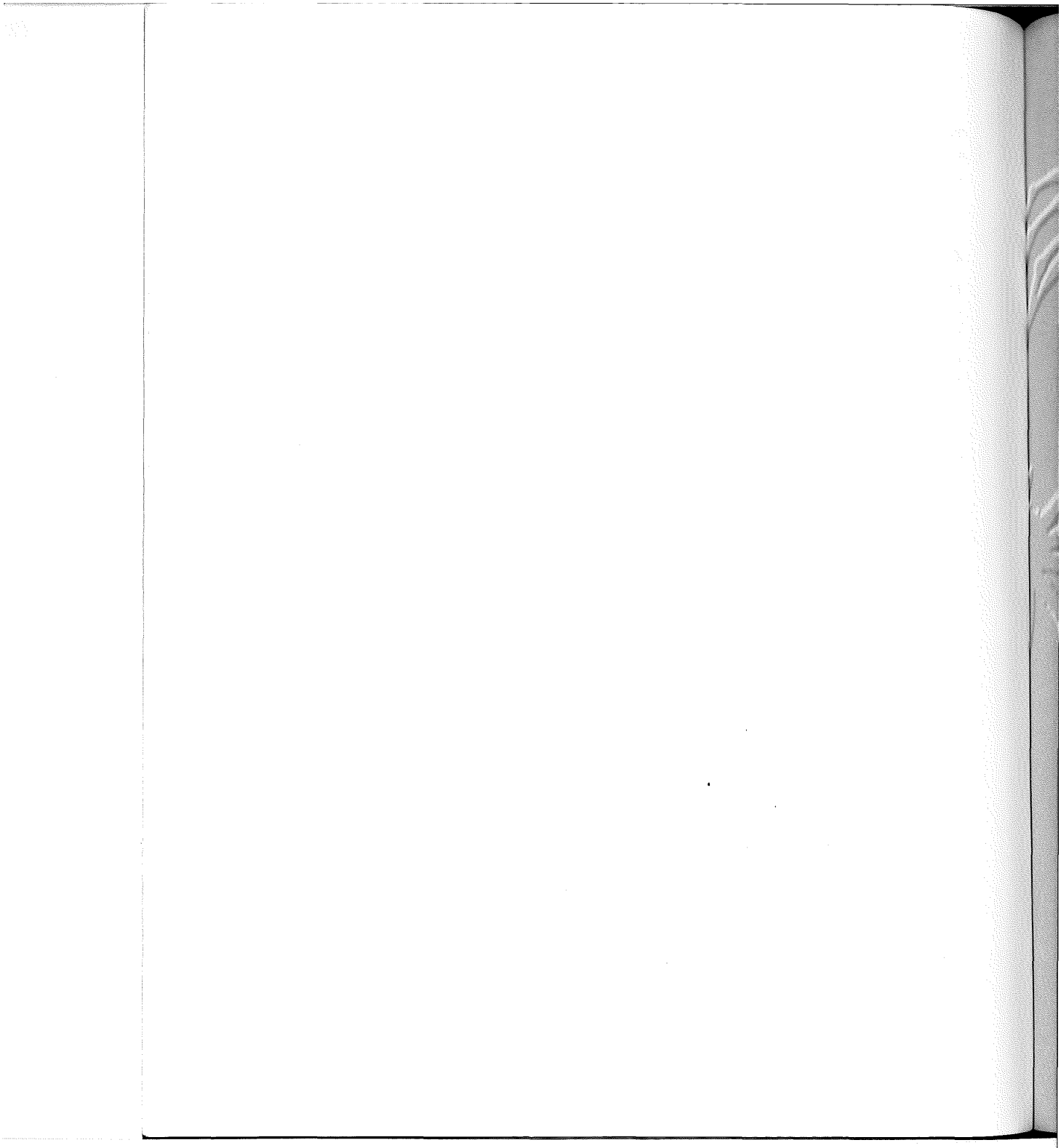
The user-ID and password (separated by a single colon (:)) in the above example are encoded using the base64 method (BF 1993).



The basic authentication scheme is a non-secure method of filtering unauthorized access to resources on an HTTP server. It does not prevent the entity body from being transmitted in clear text across the physical network.

Proxies are completely transparent regarding user agent access authentication. That is, they forward the WWW-Authenticate and Authorization headers intact.

IPR2025-00714
DATABRICKS EX1008 Page 167



chapter 7

WebWalker: Your Web Maintenance Robot

As was discussed in Chapter 5, "Web Robots: Operational Guidelines," one of the major applications of Web robots and spiders is in automated maintenance of Web information structure. As E. B. White writes in the popular children's book *Charlotte's Web*:

> A spider's web is stronger than it looks. Although it is made of thin, delicate strands, the web is not easily broken. However, a web gets torn every day by the insects that kick around in it, and a spider must rebuild it when it gets full of holes.

IPR2025-00714
DATABRICKS EX1008 Page 169

In a similar fashion, some hypertext links can become out-of-date as the Web information structure changes over time. Hypertext links that have become obsolete are called *dead links*. This happens when referenced information has changed or when referenced Web pages have been moved or deleted.

The former case presents a problem that is purely semantic, and requires an understanding of how the text has changed to mean something different. The latter case is purely syntactic and structural, and can be readily identified by a Web robot. The job of a *Web maintenance spider* is thus to detect dead links and to "rebuild [the Web] when it gets full of holes."

This chapter explores the Web maintenance problem and examines the basic operational principles behind spiders that perform automated Web maintenance. This chapter also describes the design and operation of the WebWalker spider, which has been developed for purpose of illustration and experimentation.

The Web Maintenance Problem

The terms *Web information structure*, *Web information space*, and *hypertext information structure*, have all been used interchangeably throughout this book. The term *infostructure* is perhaps a more concise term for describing the same thing.

Web Infostructure

An *infostructure* is a layout of information in a manner such that it can be navigated (Tilton 1993). Infostructure can be any resource database with a specifically designed structure that gives it body and shape. For example, a table of contents is an infostructure, as is a bibliography. A collection of World Wide Web documents hyperlinked together is also an infostructure. In fact, the World Wide Web as a whole can be considered the ultimate infostructure. Figure 7.1 shows the prototypical Web infostructure published by a research center.

An infostructure builds its contents from multiple information sources, in the form of hyperlinks to Web documents residing at distributed sites. These collections of Web documents often are maintained by different document owners. Individual Web documents also can be shared by more than one infostructure.

Past Approaches

An infostructure is rarely static. It changes over time as the contents of individual Web pages are updated. Reference information might be added, deleted, or changed. Web documents also might be moved or deleted. As a result, hyperlinks can become broken and the infostructure corrupted. In many cases, unfortunately the ensuing flurry of complaints from users and the information in the error logs of each server seldom are seen by the actual document owners.

Server Log Analysis

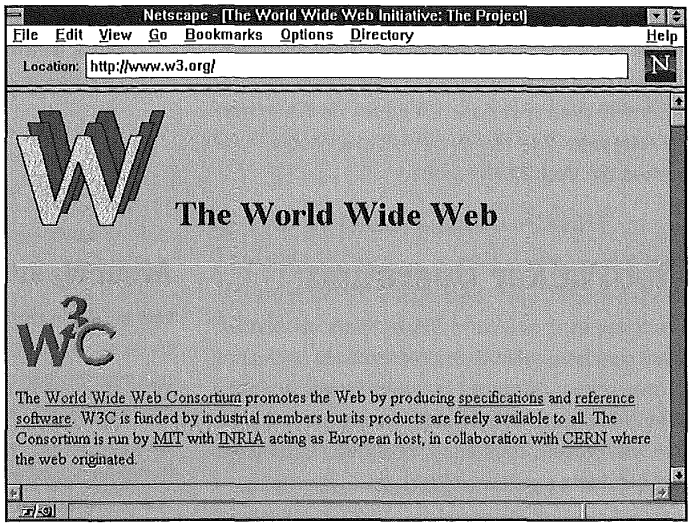
Webmasters, however, can generate a partial list of Web pages that need updating. The required information sometimes can be extracted from servers logs:

- URL of dead links (identified by failed HTTP transactions that generates the `301 Moved Permanently` response code)

IPR2025-00714
DATABRICKS EX1008 Page 170

Figure 7.1

Web infostructure of a Research Center.



➜ URL of Web pages that refer these links (specified in the Referer field of HTTP request headers).

However, this ad-hoc post-mortem approach is hardly a solution; portions of the Web that are out of date but remained unexplored might have undiscovered dead links. They remain undiscovered until someone on the Web actually needs to link through them, only to find out belatedly that the links are dead.

Manual Traversal

To detect dead links and other inconsistencies early on, individual document owners resort to manual traversal of the portion of the Web for which they are responsible. This job is both boring and time-consuming. As these infostructures evolve over time, they grow to become more complex and harder to maintain.

What is needed is an automated means of traversing a Web of documents and checking for changes that might require the attention of the human document owners. Web robots and spiders are automated client programs that can traverse the Web infostructure in a systematic fashion.

Web Maintenance Spiders

There has to be a better way of systematically exploring the Web information structure in order to uncover all dead links, and this is where Web maintenance spiders or robots become useful. They assist Web document owners and Webmasters maintain Web information structures by automatically traversing the Web space checking for dead links. These spiders can then compile a complete list of problem Web pages that contains dead links.

IPR2025-00714
DATABRICKS EX1008 Page 171

Roy Fielding's MOMspider (Multi-Owner Maintenance Spider) is one of the better known examples of Web maintenance robots (1994). WebWalker is a simple Web maintenance robot derived from MOMspider. The remainder of this chapter discusses the WebWalker robot.

WebWalker Operation

WebWalker is a World Wide Web robot that traverses designated Web infostructures on the net to perform automated hyperlinks verification. WebWalker can identify individual hypertext links that are broken, redirected, changed, or expired, and provide a summary of results.

During traversal, WebWalker sets the User-Agent field in the HTTP request header to WebWalker/1.00 and also includes additional information in the headers for identifying the robot operator and the Web page referrer. The following is a sample fragment in the HTTP protocol stream:

```
User-Agent: WebWalker/1.00
From: webmaster@www-cis.stanford.edu
Referer: http://www-cis.stanford.edu/NanoNet/
```

Processing Task Descriptions

WebWalker can be run from the command line by the user or as a batch program. WebWalker can also be run as a CGI script by the Web server (in response to user submission of task descriptions through a Web browser online). As a result, WebWalker can learn about what infostructures it is supposed to visit by one of the following means:

- WebWalker can look up the task descriptions from a *task file* when invoked from the command line or run as a batch program.
- WebWalker can get the task descriptions using the Common Gateway Interface when WebWalker is run as a CGI script.

Avoiding and Excluding URLs

Not all URLs on the Web are safe for a spider to traverse; some infinite virtual spaces generated by program scripts on the server can trap an unsuspecting spider. There are also many URLs, such as gateway program scripts and image files, for which it makes no sense to collect maintenance information. Furthermore, some sites on the Web are simply not intended for robots.

WebWalker complies with the robot exclusion standard (described previously in Chapter 5) by respecting all restrictions set up for it by the Webmaster. These restrictions can be viewed as roadblocks on the Web, beyond which WebWalker would not venture. The roadblock information is communicated to WebWalker by means of the robot exclusion file at the target Web site, and which WebWalker must first read prior to traversing the Web site. WebWalker avoids all URLs that are disallowed to it (that is, it will perform no HTTP requests on those URLs).

In addition to the robot exclusion file, WebWalker can also depend on *Exclude directives* supplied to it in a *task file* by the robot operator to help it navigate around such spider traps. URLs that are excluded can only be tested with HTTP HEAD

IPR2025-00714
DATABRICKS EX1008 Page 172

requests, and not traversed with HTTP GET requests. Excluded URLs are essentially the leaf nodes in a Web infostructure.

Keeping History

WebWalker maintains a traversal history and remembers where it has been on the Web. In this way, WebWalker knows how to avoid being lured into chasing cycles of repeating URLs. The history information also allows WebWalker to reuse results of previous visits. Status updates to the history occurs throughout the traversal process and WebWalker tracks whether specific nodes in the infostructure were seen but not yet tested, avoided, to be excluded, to be tested, to be traversed, or already traversed.

Traversing the Web

WebWalker follows a simple breadth-first traversal strategy, implemented by keeping an internal queue of URLs that WebWalker needs to visit. WebWalker knows how to crawl slowly on the Web so as not to overload any one server with a series of rapid-fire requests. This is implemented by keeping track of the number of consecutive requests to the last visited site, and remember the time when the previous request was last made.

Generating Reports

WebWalker reports its findings in the form of a collective summary table of statistics, one for each infostructure examined. Regularly scheduled visits by WebWalker ensure the correctness and consistency of a large and growing collection of distributed WWW infostructures, and make the task of maintaining complex infostructures much easier for the already overworked Webmasters.

Is WebWalker a Good Robot?

WebWalker is a stripped-down implementation of MOMspider for automated Web maintenance. WebWalker would be considered a *good* robot depending if it complies with the four laws of Web robotics, listed here:

1. *A Web Robot Must Show Identifications.*

   WebWalker supplies all three HTTP request header fields: User-Agent, From, and Referer, as required.

2. *A Web Robot Must Obey Exclusion Standard.*

   WebWalker understands the robot exclusion standard which it implements by means of an avoidance strategy.

3. *A Web Robot Must Not Hog Resources.*

   WebWalker knows how to crawl slowly on the Web so as not to overload any one server with a series of rapid-fire requests. WebWalker also remembers where it has been on the Web so as not to chase infinite cycles of URLs.

4. *A Web Robot Must Report Errors.*

   WebWalker generates a statistics summary report at the end of each infostructure traversal and which highlights a list of URLs that are broken, redirected, changed, or expired.

IPR2025-00714
DATABRICKS EX1008 Page 173

WebWalker Limitations

The design and implementation of WebWalker is based on the architecture and Perl source code of Roy Fielding's MOMspider robot. WebWalker is a simplified version of MOMspider in that it does not have all of MOMspider's user input features and report generation capabilities. But then WebWalker is only 1,800 lines of Perl code, as versus 4,000 lines for MOMspider. WebWalker, however, retains the internal Web traversal engine of MOMspider and performs the mechanics of Web traversal in exactly the same fashion as MOMspider.

WebWalker does not support the sharing of Web maintenance work load across multiple users, nor does it save the results into files for sharing across multiple runs at different times, as does MOMspider. WebWalker is not designed for heavy-duty production use. WebWalker is built for illustrating the basic operational principles of Web robots, and for experimentation. WebWalker can, however, come in quite handy for light-duty use by Webmasters to traverse Web infostructures within the local network.



WebWalker should not be used to traverse remote Web sites across the Internet (although it is perfectly capable of doing so), as this is wasteful of network resources. A better solution is to run WebWalker on a machine local to where the bulk of the Web infostructure resides.

WebWalker Program Installation

The WebWalker program is a 1,800-line Perl script that is built on top of the Roy Fielding's libwww-perl library package for accessing the World Wide Web. Currently, WebWalker can only be run from a Unix machine. Before the WebWalker can be used to help maintain your local Web sites, you need to install and set up the following:

- The Perl interpreter, written by Larry Wall
- The Perl WWW library, written by Roy Fielding
- The WebWalker program, a single file written in Perl

The following are step-by-step instructions for installing the WebWalker robot on a Unix machine:

1. If you don't already have it, get and install the Perl software package from one of its many distribution sites. A list of Perl archive sites can be found in the following Web pages:

```
http://www.cis.ufl.edu/perl/
http://web.nexor.co.uk/perl/perl.html
```

   Be sure to install the user and system libraries along with Perl. Specifically, the execution of WebWalker requires that the getopts.pl Perl library package be installed.

2. If you don't already have it, get and install Roy Fielding's libwww-perl package from any of its distribution sites at the following addresses:

```
http://www.ics.uci.edu/WebSoft/libwww-perl/

ftp://liege.ics.uci.edu/pub/arcadia/libwww-perl/
```


IPR2025-00714
DATABRICKS EX1008 Page 174

If you have not included the libwww-perl directory on the standard include path for Perl, be sure to set the $LIBWWW_PERL environment variable so that client programs, for example, WebWalker, can find it.

3. Get the WebWalker source program from any one of the following addresses:

```
http://deluge.stanford.edu:8000/book/WebWalker

http://www.mcp.com/softlib/Internet/WebWalker

ftp://www.mcp.com/softlib/Internet/WebWalker
```

4. Examine the WebWalker program script and follow its configuration instructions to properly set up WebWalker for operation. You must follow the instructions there to configure the locations of the Perl WWW library and WebWalker task file, and to specify the domain name of the local network.

5. Make sure the WebWalker program is executable (on most Unix systems) by typing the following command:

```
chmod 755 WebWalker
```

Now that WebWalker is properly installed and set up, you can turn your attention to specifying what infostructures you want WebWalker to traverse. The specification for each infostructure is called a *task* and are collected in a WebWalker *task file*, to be discussed next.

WebWalker Task File

The task file usually resides in the robot operator's home directory and specifies the infostructures that WebWalker needs to traverse. At the begining of processing, WebWalker reads all the task specifications from the task file and loads them into internal tables.



The task file can be named by the -f command-line option or by the default name ".webwalk" set in the configuration section of the source program.

A WebWalker task file consists of a series of optional global directives followed by a series of traversal tasks, each traversal task is specified with a set of task directives. Both global and task directives are case-sensitive.

WebWalker sets the configuration options associated with the global directives, then proceeds to perform each of the tasks in the given order as listed in the task file. After completing the last task, WebWalker prints out a summary of the overall process results and then exits.

Global Directives

The following are the recognized global directives:

➜ ReplyTo *email_address*

Specifies the real e-mail address of the robot operator running the WebWalker program, which usually is the local Webmaster. This e-mail address *must* correspond to the human being that should be notified in case someone is having problems with how WebWalker is operated. The e-mail address information is communicated to the Web server by means of

IPR2025-00714
DATABRICKS EX1008 Page 175

the HTTP From request header. The default address is normally set by libwww-perl to be `user@hostname`.

➜ MaxDepth *number*

Specifies the maximum allowed depth of any WebWalker traversal. Its purpose is to prevent the spider from falling into an infinite virtual space. The default value (usually 20) should be larger than any of the infostructures that WebWalker will ever want to traverse.

Task Directives

Each WebWalker task consists of a set of task directives surrounded by angle brackets. For each task, WebWalker traverses the Web infostructure, in breadth-first order, from the specified top document (TopURL directive) down to each leaf node. A *leaf node* is defined to be any Web resource which is either not of content-type HTML (and thus cannot contain any further links), or which is outside the boundary of the given infostructure boundary (specified with the BoundURL directive).

➜ Name *infostructure_name*

Specifies the name of the Web infostructure to be traversed. It is used to identify the infostructure in a WebWalker generated report. The name is required for all tasks and must be a single word containing no whitespace.

➜ TopURL *URL*

Specifies the URL of the top of the infostructure to be traversed. If the given URL is relative, then it is resolved as a local URL URL (that is, with the prefix file://localhost/) relative to the current working directory where WebWalker is started. The top URL is required for all tasks and must be a single word containing no whitespace. Any fragment identifier will be ignored.

➜ BoundURL *URLprefix*

Specifies that only encountered URLs that contain the given prefix will be traversed. This sets the boundary for the intended infostructure and prevents WebWalker from trespassing onto other remote Web sites where it will be unwelcome.

➜ ChangeWindow *number*

Specifies the window in *number* days prior to the current date within which a tested URL's last-modified date is considered "interesting" and should be reported by WebWalker. If *number* is zero (0), no last-modification dates are considered interesting. This directive is optional and defaults to seven (7) days.

➜ ExpireWindow *number*

Specifies the window in *number* days after the current date within which a traversed URL's expires date is considered "interesting" and should be reported by WebWalker. If *number* is zero (0), no expiration dates are considered interesting. This directive is optional and defaults to zero (0). Because expire dates are rarely used in the Web, this directive is rarely useful.

➜ Exclude *URLprefix*

Specifies that all encountered URLs that contain the given URL prefix will only be tested and not traversed. It is always useful to exclude the cgi-bin directory, as well as other

IPR2025-00714
DATABRICKS EX1008 Page 176

directories that contain image files, from Web. Multiple Exclude directives can be specified for any task.

Task File Format

The WebWalker task file format is fairly rigid but quite simple. Blank lines and any lines beginning with '#' are ignored. All task directives should be on a single line regardless of length and there is no facility for line-continuation.

The specification of each task begins with a left angle bracket character (<) and ends with a right angle bracket character (>), both of which must be on a line by itself. For example, the following is a sample task file that can be used by the Webmaster at Yahoo to maintain the Yahoo directory:

```
ReplyTo         webmaster@yahoo.com
MaxDepth        1
<
  Name          Yahoo
  TopURL        http://www.yahoo.com/
  BoundURL      http://www.yahoo.com/
  ChangeWindow  1
  ExpireWindow  1
  Exclude       http://www.yahoo.com/
>
```

Another example is the following sample task file that can be used by Webmasters at Stanford's Center of Integrated Systems for maintaining their Nanofab project infostructure:

```
ReplyTo         webmaster@www-cis.stanford.edu
MaxDepth        10
<
  Name          Stanford CIS NanoNet Home Page
  TopURL        http://www-cis.stanford.edu/
                NanoNet/
  BoundURL      http://www-cis.stanford.edu/
                NanoNet/
  ChangeWindow  2
  ExpireWindow  1
  Exclude       http://www-cis.stanford.edu/
                NanoNet/cgi-bin/
>
```



Do *not* use these sample task files! They are targeted at other people's Web sites, not yours, so don't bother using them as they are strictly for illustrative purposes only. Lots of people will be upset if you do. You should build your own task file that is customized for your local Web site.

WebWalker Usage Examples

Before you start up WebWalker, you should double check the contents of the task file to make sure that it covers exactly the infostructure you have intended. It would upset many users and Webmasters if you were to unleash WebWalker on the net but failed to target the correct Web sites, and in the process wasted valuable network resources.

To start up WebWalker, simply type the name of the WebWalker program on the command line (after checking the contents of the task file for correctness) as in the following:

```
WebWalker
```


IPR2025-00714
DATABRICKS EX1008 Page 177

WebWalker can display the following usage information in response to an invalid option:

```
usage: Webwalker [-h] [-f taskfile] [-d maxdepth]
WebWalker/1.00
WWW Robot for maintenance of distributed hypertext
infostructures.
Options:
[DEFAULT]
     -h  Help — just display this message and
quit.
     -f  Get your task instructions from the
following file.  [$TaskFile]
     -d  Maximum traversal depth.
[$MaxDepth]
```



Do not attempt to run WebWalker from a remote machine that is outside of your local network! Valuable network bandwidth will be wasted if you do. WebWalker is strictly for Webmasters to run on their *local network* targeted at their *own Web sites.*

Sample WebWalker Output

Here is what you would see as the output from WebWalker using the task file for traversing the infostructure at the Yahoo Web site (intended strictly for illustrative purposes only):

```
WebWalker/1.00 starting at Tue, 12 Sep 1995 10:00:54
Reading task specifications from /home/fcc/.webwalk

Starting Infostructure [Yahoo] at Tue, 12 Sep 1995 10:00:54
Checking for http://www.yahoo.com:80/robots.txt ... 200 OK
Traversing http://www.yahoo.com/ ... 200 OK
Testing http://www.yahoo.com/bin/top1 ... 200 OK
Testing http://www.yahoo.com/images/main.gif ... 200 OK
Testing http://www.yahoo.com/headlines/ ... 200 OK
Testing http://www.yahoo.com/weblaunch.html ... 200 OK
Testing http://www.yahoo.com/text/ ... 200 OK
Testing http://www.yahoo.com/search.html ... 200 OK
Testing http://www.yahoo.com/Arts/ ... 200 OK
Testing http://www.yahoo.com/Arts/Literature/ ... 200 OK
Testing http://www.yahoo.com/Arts/Photography/ ... 200 OK
Testing http://www.yahoo.com/Arts/Architecture/ ... 200 OK
Reusing test of http://www.yahoo.com/Arts/ ...
Testing http://www.yahoo.com/Business_and_Economy/ ... 200 OK
Testing http://www.yahoo.com/headlines/current/business/ ... 200 OK
Testing http://www.yahoo.com/Business_and_Economy/Business_Directory/ ... 200 OK
Testing http://www.yahoo.com/Business_and_Economy/Markets_and_Investments/ ... 200 OK
```


IPR2025-00714
DATABRICKS EX1008 Page 178

```
Testing http://www.yahoo.com/Business_and_Economy/Classifieds/ ... 200 OK
Reusing test of http://www.yahoo.com/Business_and_Economy/ ...
Testing http://www.yahoo.com/Computers_and_Internet/ ... 200 OK
Testing http://www.yahoo.com/Computers_and_Internet/Internet/ ... 200 OK
Testing http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/ ... 200 OK
Testing http://www.yahoo.com/Computers_and_Internet/Software/ ... 200 OK
Testing http://www.yahoo.com/Computers_and_Internet/Multimedia/ ... 200 OK
Reusing test of http://www.yahoo.com/Computers_and_Internet/ ...
Testing http://www.yahoo.com/Education/ ... 200 OK
Testing http://www.yahoo.com/Education/Universities/ ... 200 OK
Testing http://www.yahoo.com/Education/K_12/ ... 200 OK
Testing http://www.yahoo.com/Education/Courses/ ... 200 OK
Reusing test of http://www.yahoo.com/Education/ ...
Testing http://www.yahoo.com/Entertainment/ ... 200 OK
Testing http://www.yahoo.com/headlines/current/entertainment/ ... 200 OK
Testing http://www.yahoo.com/Entertainment/Television/ ... 200 OK
Testing http://www.yahoo.com/Entertainment/Movies_and_Films/ ... 200 OK
Testing http://www.yahoo.com/Entertainment/Music/ ... 200 OK
Testing http://www.yahoo.com/Entertainment/Magazines/ ... 200 OK
Testing http://www.yahoo.com/Entertainment/Books/ ... 200 OK
Reusing test of http://www.yahoo.com/Entertainment/ ...
Testing http://www.yahoo.com/Government/ ... 200 OK
Testing http://www.yahoo.com/Government/Politics/ ... 200 OK
Testing http://www.yahoo.com/headlines/current/politics/ ... 200 OK
Testing http://www.yahoo.com/Government/Agencies/ ... 200 OK
Testing http://www.yahoo.com/Government/Law/ ... 200 OK
Testing http://www.yahoo.com/Government/Military/ ... 200 OK
Reusing test of http://www.yahoo.com/Government/ ...
Testing http://www.yahoo.com/Health/ ... 200 OK
Testing http://www.yahoo.com/Health/Medicine/ ... 200 OK
Testing http://www.yahoo.com/Health/Pharmacology/Drugs/ ... 200 OK
Testing http://www.yahoo.com/Health/Diseases_and_Conditions/ ... 200 OK
Testing http://www.yahoo.com/Health/Fitness/ ... 200 OK
Reusing test of http://www.yahoo.com/Health/ ...
Testing http://www.yahoo.com/News/ ... 200 OK
Testing http://www.yahoo.com/headlines/current/news/ ... 200 OK
Testing http://www.yahoo.com/News/International/ ... 200 OK
```


IPR2025-00714
DATABRICKS EX1008 Page 179

```
Testing http://www.yahoo.com/headlines/current/international/ ... 200 OK
Testing http://www.yahoo.com/News/Daily/ ... 200 OK
Testing http://www.yahoo.com/News/Current_Events ... 200 OK
Reusing test of http://www.yahoo.com/News/ ...
Testing http://www.yahoo.com/Recreation/ ... 200 OK
Testing http://www.yahoo.com/Recreation/Sports/ ... 200 OK
Testing http://www.yahoo.com/headlines/current/sports/ ... 200 OK
Testing http://www.yahoo.com/Recreation/Games/ ... 200 OK
Testing http://www.yahoo.com/Recreation/Travel/ ... 200 OK
Testing http://www.yahoo.com/Recreation/Automobiles/ ... 200 OK
Reusing test of http://www.yahoo.com/Recreation/ ...
Testing http://www.yahoo.com/Reference/ ... 200 OK
Testing http://www.yahoo.com/Reference/Libraries/ ... 200 OK
Testing http://www.yahoo.com/Reference/Dictionaries/ ... 200 OK
Testing http://www.yahoo.com/Reference/Phone_Numbers/ ... 200 OK
Reusing test of http://www.yahoo.com/Reference/ ...
Testing http://www.yahoo.com/Regional/ ... 200 OK
Testing http://www.yahoo.com/Regional/Countries/ ... 200 OK
Testing http://www.yahoo.com/Regional/Regions/ ... 200 OK
Testing http://www.yahoo.com/Regional/U_S__States/ ... 200 OK
Reusing test of http://www.yahoo.com/Regional/ ...
Testing http://www.yahoo.com/Science/ ... 200 OK
Testing http://www.yahoo.com/Science/Computer_Science/ ... 200 OK
Testing http://www.yahoo.com/Science/Biology/ ... 200 OK
Testing http://www.yahoo.com/Science/Astronomy/ ... 200 OK
Testing http://www.yahoo.com/Science/Engineering/ ... 200 OK
Reusing test of http://www.yahoo.com/Science/ ...
Testing http://www.yahoo.com/Social_Science/ ... 200 OK
Testing http://www.yahoo.com/Social_Science/History/ ... 200 OK
Testing http://www.yahoo.com/Social_Science/Philosophy/ ... 200 OK
Testing http://www.yahoo.com/Social_Science/Linguistics_and_Human_Languages/ ... 200 OK
Reusing test of http://www.yahoo.com/Social_Science/ ...
Testing http://www.yahoo.com/Society_and_Culture/ ... 200 OK
Testing http://www.yahoo.com/Society_and_Culture/People/ ... 603 Timed Out
Testing http://www.yahoo.com/Society_and_Culture/Environment_and_Nature/ ... 200 OK
Testing http://www.yahoo.com/Society_and_Culture/Religion/ ... 200 OK
Reusing test of http://www.yahoo.com/Society_and_Culture/ ...
Reusing test of http://www.yahoo.com/ ...
```


IPR2025-00714
DATABRICKS EX1008 Page 180

```
Testing http://www.yahoo.com/images/netscape4.gif ... 200 OK
Reusing test of http://www.yahoo.com/ ...
Testing http://www.yahoo.com/docs/pr/credits.html ... 200 OK
Done Traversing http://www.yahoo.com/ ...
... at Tue, 12 Sep 1995 10:10:34 — 0 remaining on queue

Broken Links:
    http://www.yahoo.com/Society_and_Culture/People/  (603 Timed Out)
Changed Links:
    http://www.yahoo.com/Regional/Countries/  (200 OK)
    Last-modified:
    http://www.yahoo.com/weblaunch.html  (200 OK)
    Last-modified:
    http://www.yahoo.com/Government/Agencies/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Regional/Regions/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Computers_and_Internet/Internet/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Entertainment/Books/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Government/Law/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Recreation/Sports/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Business_and_Economy/Business_Directory/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Computers_and_Internet/Multimedia/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Regional/U_S__States/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Government/  (200 OK)
    Last-modified:
    http://www.yahoo.com/Government/Military/  (200 OK)
    Last-modified:
```


IPR2025-00714
DATABRICKS EX1008 Page 181

```
http://www.yahoo.com/Business_and_Economy/Markets_and_Investments/  (200 OK)
Last-modified:
http://www.yahoo.com/Government/Politics/  (200 OK)
Last-modified:
http://www.yahoo.com/Recreation/Games/  (200 OK)
Last-modified:
http://www.yahoo.com/Regional/  (200 OK)
Last-modified:
http://www.yahoo.com/Business_and_Economy/Classifieds/  (200 OK)
Last-modified:

Summary of Results:
```

	References		Unique URLs		Local URLs	
	number	pct	number	pct	number	pct
Traversed	2	2.15	1	1.28	0	0.00
Tested	77	82.80	78	100.00	0	0.00
Reused	16	17.20	0	0.00	0	0.00
Avoided	0	0.00	0	0.00	0	0.00
Untestable	0	0.00	0	0.00	0	0.00
Broken	1	1.08	1	1.28	0	0.00
Redirected	0	0.00	0	0.00	0	0.00
Changed 1	21	22.58	19	24.36	0	0.00
Expired 1	0	0.00	0	0.00	0	0.00
Local	0	0.00	0	0.00	0	100.00
Remote	93	100.00	78	100.00	0	0.00
Totals	93	100.00	78	83.87	0	0.00

```
Finished Infostructure [Yahoo] at Tue, 12 Sep 1995 12:10:36

WebWalker/1.00 finished at Tue, 12 Sep 1995 12:10:36
```


IPR2025-00714
DATABRICKS EX1008 Page 182

The summary above indicates that only the Yahoo home page was traversed. A total of 93 URLs were encountered in the Yahoo home page, and they were pointing to 77 different Web pages. Obviously, some URLs were sharing the same Web page. The actual contents of these 77 Web pages were not retrieved, they were merely tested (using HTTP HEAD request). Of the 78 Web pages tested (including the home page), 19 of them were found to be new—that is, they changed within the last 24 hours.

There was only one broken link at the top-level Yahoo infostructure during WebWalker's traversal. Closer examination of the output shows that the link was only temporarily broken, as indicated by the `603 Timed Out` response code. This is probably due to Yahoo server overload and not because of a dead link.

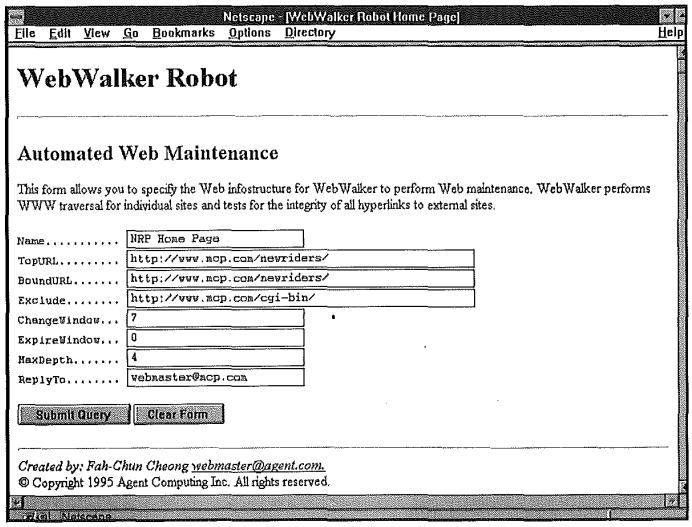
WebWalker Forms Interface

For Webmasters who prefer to work with a Web interface than to type on a command line, the WebWalker program can be configured to run as a Common Gateway Interface (CGI) script. All that needs to be done is to put the WebWalker program under the cgi-bin directory at your local Web site, and to prepare a Web page containing an HTML form that can be used to submit task description to your WebWalker. A sample home page for WebWalker is shown in figure 7.2.

The following is the output from WebWalker using the task file for traversing Nanofab project infostructure located at Stanford's Center of Integrated Systems Web site:

Figure 7.2

WebWalker's Web User Interface.



IPR2025-00714
DATABRICKS EX1008 Page 183

```
WebWalker/1.00 starting at Tue, 12 Sep 1995 10:53:20
Reading task specifications from /home/fcc/.webwalk

Starting Infostructure [Stanford CIS Nanofab Home Page] at Tue, 12 Sep 1995 10:53:21

  <...text ommitted...>

Broken Links:
    http://www-cis.stanford.edu/NanoNet/communications/lead/submission/three.html  (404 Not Found)
    http://www-cis.stanford.edu/NanoNet/communications/lead/completed.html  (404 Not Found)
    http://www-cis.stanford.edu/NanoNet/communications/lead/submission/one.html  (404 Not Found)
    http://www-cis.stanford.edu/NanoNet/communications/lead/vote.html  (404 Not Found)
    http://www.nnf.cornell.edu/NanoLine/NNF/Staff/HaroldCraighead.html  (602 Connection Failed)
    http://www.nnf.cornell.edu/  (602 Connection Failed)
    http://www-cis.stanford.edu/NanoNet/communications/lead/submission/two.html  (404 Not Found)
    http://www-cis.stanford.edu/NanoNet/communications/lead/started.html  (404 Not Found)
    http://www.nnf.cornell.edu/NanoLine/NNFPubs/nm/nm.html  (602 Connection Failed)
    http://www-cis.stanford.edu/NanoNet/communications/lead/modify.html  (404 Not Found)
Redirected Links:
    http://www.commerce.digital.com/palo-alto/chamber-of-commerce/home.html  (302 Found)
Changed Links:
    http://www.city.palo-alto.ca.us/home.html  (200 OK)
    Last-modified:

Summary of Results:
```

	References		Unique URLs		Local URLs	
	number	pct	number	pct	number	pct
Traversed	35	19.23	33	52.38	33	66.00
Tested	61	33.52	62	98.41	50	100.00
Reused	91	50.00	0	0.00	0	0.00
Avoided	0	0.00	0	0.00	0	0.00
Untestable	30	16.48	1	1.59	0	0.00
Broken	10	5.49	10	15.87	7	14.00
Redirected	1	0.55	1	1.59	0	0.00
Changed 2	1	0.55	1	1.59	0	0.00
Expired 1	0	0.00	0	0.00	0	0.00

IPR2025-00714
DATABRICKS EX1008 Page 184

```
Local      |   140  76.92 |    50  79.37 |    50 100.00 |
Remote     |    42  23.08 |    13  20.63 |     0   0.00 |
           |--------------+--------------+--------------|
Totals     |   182 100.00 |    63  34.62 |    50  27.47 |
           '---------------------------------------------'

Finished Infostructure [Stanford CIS Nanofab Home Page] at Tue, 12 Sep 1995 10:56:58

WebWalker/1.00 finished at Tue, 12 Sep 1995 12:56:58
```

WebWalker tested a total of 62 different Web pages, of which 33 Web pages (including the home page) were retrieved with full HTML contents for further traversal. There were ten broken links, one redirected link, and one changed link (within the past two days, or 48 hours), in the Nanofab project infostructure.

Closer examination reveals that of the ten broken links, seven of them were actually dead links (that is, 404 Not Found) while the remaining three were inaccessible due to problems connecting with Cornell's Web server at the www.nnf.cornell.edu address (that is, 602 Connection Failed).

WebWalker Program Organization

The WebWalker/1.00 program is written in about 1,800 lines of Perl code and consists of 40 subroutines, plus a main body. The full source code of the WebWalker can be found in Appendix C.

For purpose of exposition and clarity, the WebWalker program functions and variables are logically grouped together by purpose and function into packages, which are shown in the following table:

Packages	*Purpose*
Configuration	Setting configurable options and parameters
Instruction	Receiving input tasks from task file
Avoidance	Respecting the robot exclusion standard
History	Keeping track of Web traversal history
Traversal	Traversing and testing an infostructure
Summary	Collecting and displaying summarizing statistics

External Library Calls

In addition, selected subroutines from the following collection of packages belonging to the original Perl library, as well as Roy Fielding's WWW library, have been used in WebWalker 1.00:

IPR2025-00714
DATABRICKS EX1008 Page 185

Perl Package	*Function Called by WebWalker*
getopts.pl	Getopts
www.pl	request and set_def_header
wwwurl.pl	parse, absolute and get_site
wwwdates.pl	wtime and get_gmtime
wwwhtml.pl	extract_links
wwwurl.pl	set_content

WebWalker Program Call-Graph

The WebWalker program organization can be visualized with the aid of a subroutine call-graph depicted in the following figure. In addition to showing how the subroutines are related to each other (for example, via the caller/callee relationship), the alignment of the subroutines into columns also indicates how the subroutines are grouped into packages.

```
   Traversal          Avoidance        Summary          History     Instruction
   ---------          ---------        -------          -------     -----------
main
+- ..................................................................... usage
+- ..................................................................... read_tasks
+- ..................... add_leaf
+- traverse_web
   +- ................................. begin_summary
                                        +- init_summary
   +- ..................................................... remember
   +- should_avoid
      +- ............... check_url
                         +- check_site
                            +- add_avoid
                         +- add_site
   +- ................................... traversed
                                        +- ................ get_url
                                        +- ................ recall
                                        +- ................ was_avoided
                                        +- ................ is_untestable
                                        +- save_broken
                                           +- ............. is_local
                                        +- save_redirect
                                           +- ............. is_local
                                        +- save_changed
                                           +- ............. is_local
                                        +- save_expired
                                           +- ............. is_local
```


IPR2025-00714
DATABRICKS EX1008 Page 186

```
+- traverse_link
   +- ..................................................... get_url
   +- slow_down
   +- extract_links
      +- is_html
      +- decode
   +- ..................................................... store
      +- ..................................................... set_status
+- ..................................................... was_tested
+- ..................................... tested
                                          +- ................ get_url
                                          +- ................ recall
                                          +- ................ is_local
                                          +- ................ was_avoided
                                          +- ................ is_untestable
                                          +- save_broken
                                             +- ............. is_local
                                          +- save_redirect
                                             +- ............. is_local
                                          +- save_changed
                                             +- ............. is_local
                                          +- save_expired
                                             +- ............. is_local
+- test_link
   +- ..................................................... get_url
   +- slow_down
   +- ..................................................... store
      +- ..................................................... set_status
+- should_traverse
   +- should_avoid
      +- ............ check_url
                     +- check_site
                        +- add_avoid
                     +- add_site
   +- ..................................................... is_known
   +- ..................................................... recall
   +- ..................................................... get_url
   +- is_html
+- ..................................................... set_status
+- ..................................................... get_url
+- ..................................................... reset_status
```


IPR2025-00714
DATABRICKS EX1008 Page 187

```
+- ..................................... end_summary
                                        +- update_summary
                                           +- ............. was_avoided
                                           +- ............. was_tested
                                           +- ............. was_traversed
                                           +- ............. is_untestable
                                           +- ............. is_local
                                        +- get_summary
```

As can be seen from the code listing, the Traversal package is used by the main program via the "traverse_web" subroutine (which is invoked exactly once for each infostructure to be examined). The bulk of the work and processing logic resides under the "traverse_web" subtree, which includes both the Summary and History packages in their entirety, plus almost all of the Avoidance package (with the exception of the "add_leaf" subroutine called directly from the main program).

It also can be noted that the Avoidance package is self-contained and is used mainly by invoking the "check_url" subroutine from the "should_avoid" subroutine. The Summary package is used at several places throughout the body of the "traverse_web" subroutine for marking the different points in time during the traversal of the infostructure (for example, before starting and after ending the traversal, as well as after having finished traversing or testing a link). Unlike other packages, which are better organized as hierarchical trees of subroutines, the History package is all flat and actually is a loose collection of self-contained subroutines that do not call out to other subroutines.

Configuration Section

The *configuration section* allows users to configure options for setting up WebWalker according to the local operating environment. The more important options are the following:

- The $Version parameter identifies to the destination Web servers the specific version of WebWalker that is being targeted at them.
- The $LibWWW parameter tells WebWalker where to locate the Perl WWW library on the client machine. WebWalker needs the library to handle Web-related format and protocol processings.
- $LocalNetwork should be the network domain that you consider to be local to your organization. In other words, a network request to sites in this domain does not create any external network costs to your organization. Any periods in the network domain name need to be escaped with a backslash (for example, `stanford\.com`).
- The $Taskfile parameter tells WebWalker where to locate the task file. WebWalker needs to examine the task file to find out what infostructures to visit.

IPR2025-00714
DATABRICKS EX1008 Page 188

For most purposes, configuring the above options would be adequate to prepare WebWalker for operation. However, there are other configurable parameters that WebWalker uses, and they are described next.

Setting up WebWalker

There must be some ways to control the traversal behavior of WebWalker. This can be accomplished by means of three configurable parameters, which together dictate that there must be at least a minimum of $BetweenTime seconds of elapsed time in between cosecutive HTTP requests, and that a long pause of $PauseTime seconds is required after making a stream of $MaxConsec consecutive requests to the same Web site.

The complete list of WebWalker configurable parameters is shown in the following table.

Parameters	*Description*
$Version	User-Agent identification for the WebWalker WWW robot
$LibWWW	Directory path that holds the WWW library written in Perl
$LocalNetwork	Network domain that is considered local
$TaskFile	Default pathname of task instruction file
$RobotsURL	Standard URL that defines access control for WWW robots, defaults to "/robots.txt"
$BaseURL	The initial base URL to use if TopURL is relative
$MaxDepth	Default maximum traversal depth
$Timeout	Maximum number of seconds to wait for a HTTP response
$MaxConsec	Maximum consecutive requests to any site before a long pause
$PauseTime	Duration of a long pause (in seconds)
$BetweenTime	Time required between any two requests to the same site (in seconds)Instruction Package

There must be a way for WebWalker to find out what infostrutures it is supposed to visit. There are are two ways of doing so:

- WebWalker looks up the task descriptions from a task file when it is invoked from the command line by the user or run as a batch program.
- WebWalker gets the task descriptions using the Common Gateway Interface when WebWalker is invoked as a CGI script by the Web server, in response to a user submitting the task directives through an online HTML form using a Web browser.

The *instruction package* is made up of variables and functions that handle the processing of input task descriptions, either described in the task file or communicated through the Common Gateway Interface. The variables in this package, as listed in the following table, are used to hold the values of task directives.

IPR2025-00714
DATABRICKS EX1008 Page 189

Parameters	*Description*
@TaskName	Value of Name task directive; specifies a name with which to identify the infostructure
@TaskTopURL	Value of TopURL directive; specifies the starting URL of the infostructure to be traversed
@TaskBoundURL	Value of BoundURL directive; specifies the prefix URL for bounding the infostructure
@TaskChange Window	Value of ChangeWindow directive; specifies the past number of days within which a change would be of interest
@TaskExpire Window	Value of ExpireWindow directive; specifies the future number of days within which a scheduled expiration would be of interest
@TaskExclude	Value of Exclude directive; specifies the URL to exclude (leaf) from this task

Processing Tasks

The functions in this package print out proper usage information on the command line, handle task file processing, and implement the Common Gateway Interface. These functions are listed in the following table.

Function	*Description*
usage	Print usage information.
read_task	Handle GET and POST methods if WebWalker is used as a CGI script.
read_tasks	Read task descriptions from task file.

Avoidance Package

There must be some ways to guide or restrict WebWalker's scope of activity. Specifically, robot operators might want WebWalker to exclude certain URLs from its traversal path and not visit there. In addition, there might be certain infinite virtual spaces that Webmasters at the target site would want WebWalker to avoid.

The *avoidance package* consists of variables and functions that implements various means of restricting WebWalker's scope of activity. The robot exclusion standard is implemented here, and WebWalker avoids all URLs disallowed to it. WebWalker also does not retrieve the content of any Web page that it has been told to exclude (by means of the Exclude directive); it merely tests for the document's existence. The variables of this package are listed below.

Variables	*Descriptions*
$SitesNum	Number of sites visited
@SitesAddr	Sites table containing Web sites visited
%Sites	Reverse sites table for duplicates detection
$AvoidNum	Number of URLs to be avoided

IPR2025-00714
DATABRICKS EX1008 Page 190

@AvoidURL	Avoids table containing URL's that are not to be tested
$LeafNum	Number of nodes in leaf table
@LeafURL	Leaf table of nodes to exclude (leaf)

Avoiding Blackholes

The functions of this package implement various means of restricting the scope of WebWalker's traversal (including the robot exclusion standard), and keeps track of which sites it has visited (so that it does not retrieve the robot exclusion file more than once per site). The functions are listed in the following table:

Function	*Description*
check_url	Check the given URL for any restrictions on its access.
check_site	Has this site already been checked for restrictions? If not, perform a check using the robot exclusion protocol and update both the sites table and the avoids table accordingly.
add_site	Add the given site to the sites table while detecting duplication.
add_avoid	Add the given URL to the avoids table while checking for duplication and overlap.
add_leaf	Add the given URL to the leaf table for the duration of the current infostructure traversal while checking for duplication and overlap.

History Package

A Web robot has to keep track of all the places it has visited in the past so it doesn't revisit the same URL repeatedly and thus waste valuable resources. More importantly, a robot's capability to keep a history of where it's been on the Web enables it to extricate itself when trapped in an infinite loop embedded deep inside the Web.

The *history package* consists of variables and functions that allow WebWalker to record and recall where it has been on the Web. The following table lists the variables used in this package, most of which are actually arrays that hold the results of past visitations.

Variables	*Description*
$VisNumber	Number of URL's visited since process start
%Visited	Associative array of URL's visited mapped to @Vis* index
@VisURL	URL of node (maps @Vis* index to URL visited)
@VisStatus	Status of a seen node
@VisRespCode	Server response code from last access
@VisConType	MIME Content-type of response
@VisRedirect	Redirected URL (from a 302 Moved response)
@VisTitle	Title text from headers or last traversal

continues

IPR2025-00714
DATABRICKS EX1008 Page 191

Variables	*Description*
@VisOwner	Owner name from headers or last traversal
@VisReplyTo	Reply-To address from headers or last traversal
@VisLastMod	Last-modified date from headers
@VisExpires	Expires date from headers
@VisInTask	Has the URL been seen during the current task?
@VisLocal	URL considered to be local to this network?

Remembering Past Visits

The functions listed in the following table implement WebWalker's memory of past history. The remember function is used to write into history the results of making a HTTP GET request. The store function is used to write into history the results of a HTTP HEAD request. The recall function retrieves from history results of past visitations.

Status updates to history are handled with set_status and reset_status functions. The remaining functions handle history-related status queries to various parts of the infostructure, such as whether specific nodes in the infostructure were seen but not yet tested, avoided, to be excluded, to be tested, to be traversed, or already traversed.

Function	*Description*
set_status	Sets or updates the status of the given node in history
reset_status	Resets the status of the given node in history so that it is no longer considered traversed
remember	Remembers the URL in history by either creating a history record or update the node status as appropriate
store	Stores node history from meta information held in headers, along with status and response code from recent WWW request
recall	Recalls meta information held in history for the given node
was_avoided	Indicates if the given node was previously avoided
was_tested	Indicates if the given node was previously tested
is_untestable	Indicates if the given node is untestable
is_known	Indicates if the given node will be, or has already been, checked for traversal status
is_traversing	Indicates if the given node will be, or has been, traversed
was_traversed	Indicates if the given node was traversed
is_local	Indicates if the given node is considered local
get_url	Retrieves the stored URL of the given node

IPR2025-00714
DATABRICKS EX1008 Page 192

Traversal Package

The *traversal package* consists of variables and functions that WebWalker needs for actually traversing and testing the Web infostructure. The variables are listed in the table below. Some of the variables are actually arrays that implement a queue data structure needed for WebWalker's breadth-first traversal strategy.

Variables	*Description*
$CurConsec	Current number of consecutive requests to a site
$PrevSite	Site of the last network request
$PrevTime	Time of the last network request
@TravNodes	Nodes that we have yet to traverse for this task
@TravDepth	Nodes' traversal depth
@TravParent	Nodes' parent's URL
@TestLinks	Absolute URL's (without query or tag)
@TestType	Anchor type (for example, Link, Image, Query, Redirect)

WebWalker keeps track of the number of consecutive requests ($CurConsec) to the latest site ($PrevSite) and records the time ($PrevTime) when the previous request was last made. With such information, WebWalker would know when and how to crawl slowly and not overload any one site with a series of rapid-fire HTTP requests.

Roaming the Web

The functions listed in the following table implement the actual breadth-first strategy and mechanism that WebWalker uses to crawl on the Web. The top-level function is traverse_web, which in turn calls the traverse_link to perform a HTTP GET reuest, or calls the test_link functions to perform a HTTP HEAD request. HTML documents retrieved by WebWalker are processed by the extract_links function to find all hyperlinks needed for future traversal and testing.

Functions	*Description*
traverse_web	Traverses entire infostructure in breadth-first order, bounded by a URL-based task bound prefix and maximum traversal depth.
should_avoid	Indicates if the node should be avoided or has already been avoided.
should_traverse	Indicates if the node should be traversed for the current infostructure
test_link	Tests the URL via HTTP HEAD request. Stores meta information in history and update node status.
traverse_link	Traverses URL via HTTP GET request. Stores meta information in history and update node status. Extracts links from headers and document HTML content to be queued for further traversal.

continues

IPR2025-00714
DATABRICKS EX1008 Page 193

Functions	*Description*
extract_links	Extracts links and document meta information from headers and HTML body content, and deposits it in queue for further traversal.
slow_down	Makes sure the robot is not making too many consecutive requests and/or making too many rapid-fire requests to a single site.
is_html	Determines if the Web document is in HTML, based upon its URL suffix and header content-type.
decode	Translates encoded content into its decoded form, usually to decompress a compressed Web document.

Summary Package

The *summary package* consists of a set of variables that are used as counters for keeping track of statistical data related to the current Web infostructure under investigation. These counters are classified into three categories, $Hrefs*, $Nodes*, and $Local*. They are used to keep track of statistical information related to occurrences of HTTP references (there could be multiple such occurrences with the same URL), unique URLs, and local URLs.

A set of associative arrays (%BrokenNodes, %RedirectNodes, %ChangedNodes, and %ExpiredNodes) is used for the purpose of collecting and displaying information related to broken or redirected links, and for keeping track of URLs that have recently been changed or have expired. In this way, WebWalker can easily generate useful reports on problem areas that have been identified in the infostructure. The next table describes the variables used in the summary package.

Variables	*Description*
%BrokenNodes	URL information on broken links indexed by node
%RedirectNodes	URL information on redirected links indexed by node
%ChangedNodes	URL information on changed nodes indexed by node
%ExpiredNodes	URL information on expired nodes indexed by node
$HrefsTrav	Traversed URL reference count
$HrefsTest	Tested URL reference count

IPR2025-00714
DATABRICKS EX1008 Page 194

$HrefsReus	Reused URL reference count
$HrefsAvd	Avoided URL reference count
$HrefsUnt	Untestable URL reference count
$HrefsBroke	Broken URL reference count
$HrefsRedir	Redirected URL reference count
$HrefsChg	Changed URL reference count
$HrefsExp	Expired URL reference count
$HrefsLoc	Local URL reference count
$HrefsRmt	Remote URL reference count
$NodesTrav	Unique traversed node count
$NodesTest	Unique tested node count
$NodesReus	Unique reused node count
$NodesAvd	Unique avoided node count
$NodesUnt	Unique untestable node count
$NodesBroke	Unique broken node count
$NodesRedir	Unique redirected node count
$NodesChg	Unique changed node count
$NodesExp	Unique expired node count
$NodesRmt	Unique remote node count
$LocalTrav	Local traversed node count
$LocalTest	Local tested node count
$LocalReus	Local reused node count
$LocalAvd	Local avoided node count
$LocalUnt	Local untestable node count
$LocalBroke	Local broken node count
$LocalRedir	Local redirected node count
$LocalChg	Local changed node count
$LocalExp	Local expired node count
$TotalHrefs	Total URL reference count
$TotalNodes	Total unique node count
$TotalLocal	Total local node count

Statistics Table

The following code template indicates how the statistical counters are used for displaying the fitness of the infostructure in a summary of results table.

IPR2025-00714
DATABRICKS EX1008 Page 195

```
Summary of Results:

           .-------------------------------------------.
           |  References  |  Unique URLs  |  Local URLs  |
           |  number      |  number       |  number      |
           |--------------+---------------+--------------|
Traversed  |  $HrefsTrav  |  $NodesTrav   |  $LocalTrav  |
Tested     |  $HrefsTest  |  $NodesTest   |  $LocalTest  |
Reused     |  $HrefsReus  |  $NodesReus   |  $LocalReus  |
Avoided    |  $HrefsAvd   |  $NodesAvd    |  $LocalAvd   |
Untestable |  $HrefsUnt   |  $NodesUnt    |  $LocalUnt   |
           |--------------+---------------+--------------|
Broken     |  $HrefsBroke |  $NodesBroke  |  $LocalBroke |
Redirected |  $HrefsRedir |  $NodesRedir  |  $LocalRedir |
Changed    |  $HrefsChg   |  $NodesChg    |  $LocalChg   |
Expired    |  $HrefsExp   |  $NodesExp    |  $LocalExp   |
           |--------------+---------------+--------------|
Local      |  $HrefsLoc   |  $TotalLocal  |  $TotalLocal |
Remote     |  $HrefsRmt   |  $NodesRmt    |      0       |
           |--------------+---------------+--------------|
Totals     |  $TotalHrefs |  $TotalNodes  |  $TotalLocal |
           '-------------------------------------------'
```

Reporting Statistics

The summary package also includes a set of functions to initialize, update, manipulate, generate, and print the corresponding statistical results derived from statistics counters in the form of a summary table. This set of functions is listed in the following table.

Functions	*Description*
begin_summary	Initializes counters for statistical summary and data structures for diagnostic information about the infostructure.
init_summary	Initializes all counters for statistical results summary.
tested	Updates all reference counters for statistical summary and collects diagnostic information about the tested link as appropriate and prints the http response message.
traversed	Collects diagnostic information about the traversed link as appropriate and prints the http response message.
save_broken	Saves the URL and related information about the broken link. Updates the relevant node-broken counters as appropriate.

IPR2025-00714
DATABRICKS EX1008 Page 196

save_redirect	Saves URL and related information about the redirected link. Updates relevant node-redirected counters as appropriate.
save_changed	Saves URL and related information about the changed link. Updates relevant node-changed counters as appropriate.
save_expired	Saves URL and related information about the expired link. Updates relevant node-expired counters as appropriate.
end_summary	Prints diagnostic results and statistical summary table.
update_summary	Updates counters for statistical summary table.
get_summary	Generates statistical summary of results in a table.

Growing into the Future

The World Wide Web is currently experiencing phenomenal growth with no sign of abatement. Not only are people authoring more HTML pages today than yesterday, there will be many more Web sites coming up tomorrow. At this rate of growth, the problem of managing and maintaining complex Web infostructures is increasingly a difficult one.

If this problem is not satisfactorily resolved soon enough, large portions of the global Web infostructure can become seriously corrupted and the entire Web edifice can easily collapse under the weight of tons of dead links. Such misfortunes can seriously reduce the usefulness of the Web.

Fortunately, there are many good spiders that can perform automated Web maintenance quite competently. In addition to MOMspider and WebWalker, there are also other Web maintenance spiders like the HTML Analyzer, EIT Link Verifier, ChURL, Weblayers, and WebWatch robots, many of which are freely available to the public. For now, it appears that the problem has at least been contained.

All of these spiders are not very much different at the core. They all do one thing well: automated traversal of the Web. As such, they are also not much different from resource discovery spiders that can handle keyword-based searches of the Web (which we have studied previously in chapter 4).

As the Web continues to grow, we can expect to see many more new Web-wandering spiders that can perform a variety of innovative and interesting new services for its users. It is hoped that WebWalker's simple design will better illustrate how the core traversal engine of new Web robots can be constructed.

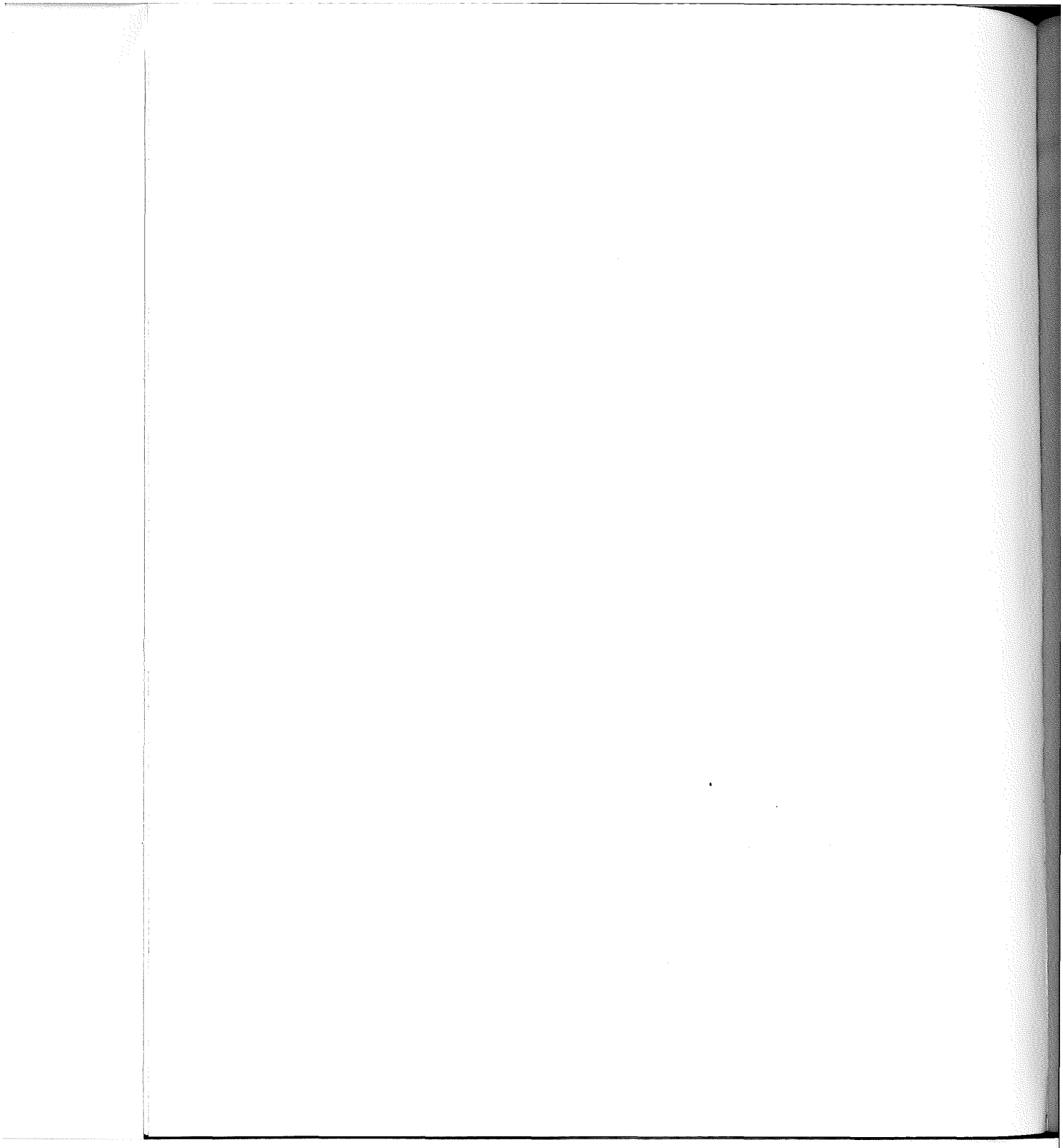


For the advanced readers who are interested in a Multi-Owner Maintenance spider, Roy Fielding's MOMspider program source code is freely available from the following distribution sites:

`http://www.ics.uci.edu/WebSoft/MOMspider/`

`ftp://liege.ics.uci.edu/pub/arcadia/MOMspider/`

IPR2025-00714
DATABRICKS EX1008 Page 197



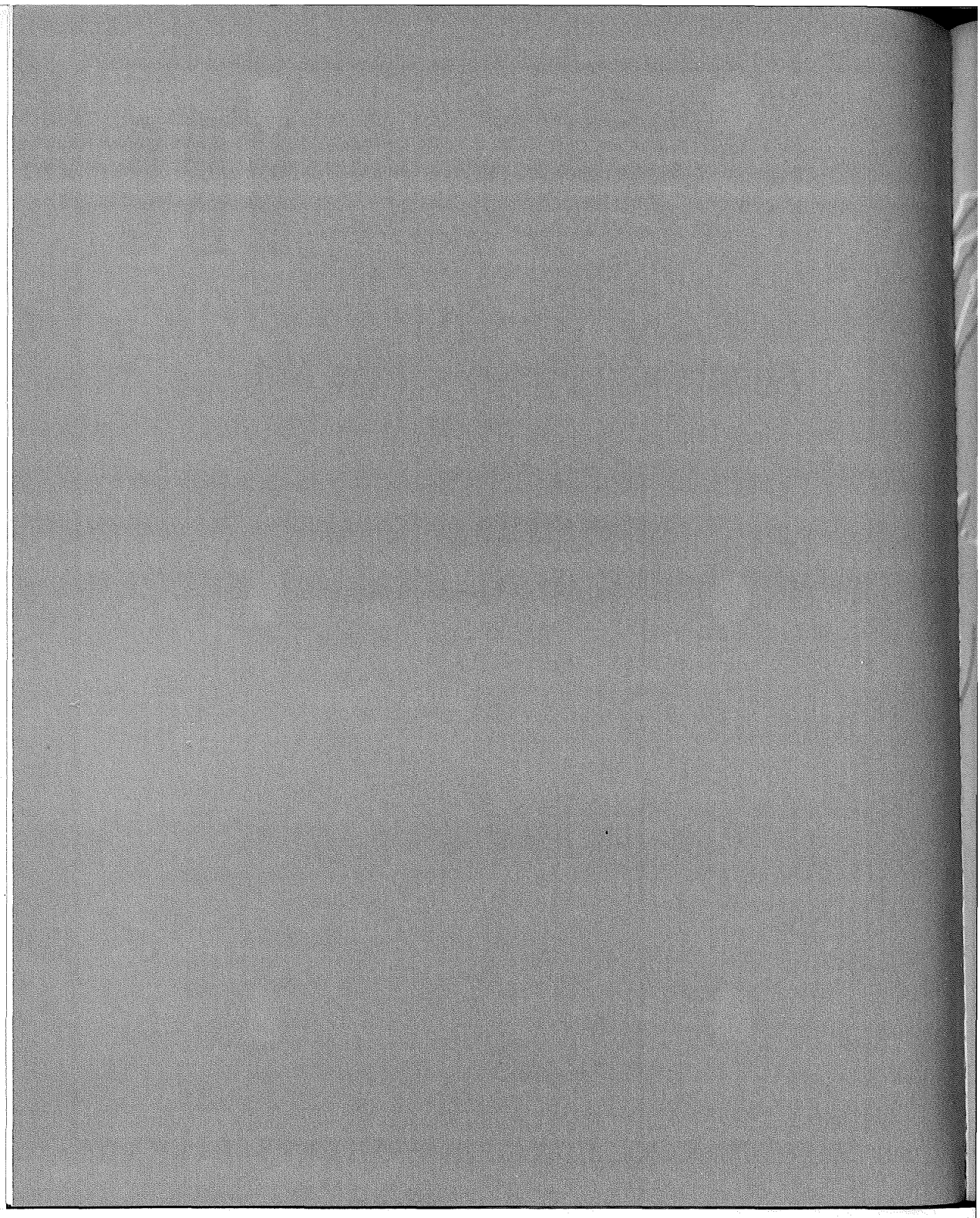
part III

Agents and Money on the Net

8 Web Transaction Security 185

9 Electronic Cash and Payment Services 205

IPR2025-00714
DATABRICKS EX1008 Page 199



chapter 8

Web Transaction Security

The advent of electronic commerce on the Internet is to a large extent facilitated by the launch of the World Wide Web. In the realm of agents, there are currently hordes of spiders, wanderers, brokers and bots on the Web performing various tasks for their human clients—for example, searching for information, maintaining Web infostructure, brokering for buyers and sellers, as well as finding the best bargain for books and CDs online.

IPR2025-00714
DATABRICKS EX1008 Page 201

For the class of agents that are designed for electronic commerce in a digital economy, for example, the brokers and bargain hunters, there must be some measures of security built into the basic transaction-based communications infrastructure for them to function reliably. In particular, assurance of secure transactions is required for online shopping through the World Wide Web, the sale of information over the Internet, as well as the execution of certain business operations like online ticket reservations. Despite the growing interest in the Internet and World Wide Web, the commercial potential has been held back by competing and incompatible security approaches.

This chapter examines the various notions of security. It also discusses the use of cryptography and digital signatures as solutions for achieving specific security goals, briefly exploring their colorful history in the process. Finally, this chapter explains two specific approaches that have been developed for secure transaction on the World Wide Web: SSL and Secure HTTP. It is anticipated that a future generation of agents on the Internet, and especially on the Web, shall incorporate these fundamental techologies and be able to interoperate across various economic domains.

Concepts of Security

Internet Security consists of the following two distinct areas:

- **Access security.** This refers to the capability of an organization to protect its computers, memory, disk, printers, and other computing equipment from unauthorized use. Standard practice is usually a combination of techniques that include the use of authentication software (for example, MIT's Kerberos (SNS 1988)), installation of proxies on Internet "firewalls," stricter access control with passwords, and diligent enforcement of security policies.
- **Transaction security.** This refers to the capability of two entities on the Internet to conduct a transaction privately with the help of cryptographic systems while being authenticated with properly certified digital signatures as needed. SSL and Secure HTTP are mechanisms for transaction security on the World Wide Web.



Access security is already well covered in a number of other books. For example, *Firewalls and Internet Security*, by William Cheswick and Steven Bellovin, or *Internet Firewalls and Network Security*, by Karanjit Siyan and Chris Hare, are recommended books on the subject.

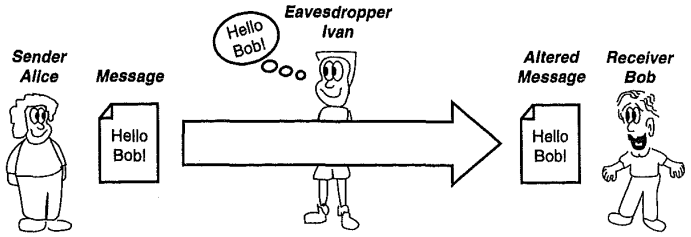
A number of transaction security issues arise between Web clients and servers, several of which are addressed in the context of SSL and Secure HTTP later in this chapter. In general, transaction security on the Internet is concerned with the following fundamental goals:

- Privacy
- Authentication
- Integrity

IPR2025-00714
DATABRICKS EX1008 Page 202

Figure 8.1

Ivan eavesdrops on a message sent from Alice to Bob over an insecure channel.



Privacy: Keeping Private Messages Private

The purpose of privacy is to ensure that information is kept hidden from anyone for whom it is not intended. Privacy is particularly important on the Internet and the World Wide Web when transmission of sensitive data, such as credit card numbers, is involved. In addition, privacy is particularly important on the Internet due to the insecure nature of the communications channel—a loose confederation of machines and networks under different authorities with no trusted, centralized administration. A data packet or an e-mail message sent over the Internet usually is routed through multiple hosts before arriving at the final destination. During this journey, the data packet or e-mail's unprotected content is copied from host to host and can be easily eavesdropped by a third party. This is illustrated in figure 8.1, where a message sent from Alice to Bob is eavesdropped by Ivan as it makes its way across the network.

The need for sensitive data to be protected from prying eyes is further amplified when you consider the types of data that could get onto the wire in the not-too-distant future: personal income tax returns, employee records, stock transactions, bank statements, and so on. *Encryption*, or the transformation of data into a form unreadable by anyone without a secret key, can be used to ensure private communication over an insecure channel. The original message, or *plaintext*, is first encrypted with a secret password, called a *secret key*, by the sender prior to transmission.

As illustrated in figure 8.2, privacy is ensured by allowing only the encrypted form of the message, or *ciphertext*, to be sent to the receiver. The eavesdropper is not able to make sense out of the ciphertext because it is unintelligible and bears no resemblance to the original plaintext. In a secure cryptosystem, the original plaintext message cannot be recovered except by using the secret key. The receiver with the secret key decrypts the ciphertext to recover the original plaintext message, which he then can read. This is called *secret-key cryptosystem*, or *symmetric cryptosystem*, because a single secret key is used for both encryption and decryption.

IPR2025-00714
DATABRICKS EX1008 Page 203

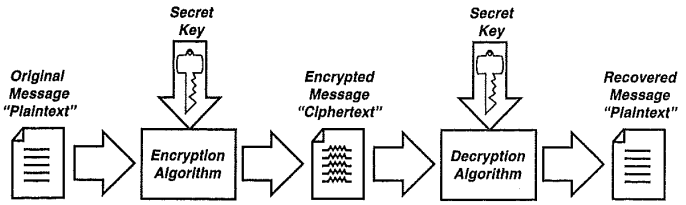


Figure 8.2

The message sent over an insecure channel is encrypted so that eavesdroppers cannot decipher the message contents.

Authentication: Proving You Are Who You Claim to Be

In networked digital communications, the receiver of a message needs to be confident of the identity of the sender. Given the insecure nature of communications over the Internet, the perils of unauthenticated messages are not to be underestimated. As illustrated in figure 8.3, an imposter on the Internet can easily impersonate another person without her knowledge and send fake messages in her name to an unsuspecting recipient, sometimes with grave consequences.

The World Wide Web provides limited capabilities for user identification and client/server authentication. For commercial use of the Web, client and server need to verify and validate each other's identity in order to ensure that information that flows across the Internet is authentic. When press releases and official announcements are distributed over the World Wide Web, for example, the client needs to be sure of their place of origin. Similarly, in the case of home banking or stock transaction over the World Wide Web, the Web server needs to ensure that the clients with whom it is transacting are who they claim to be. It is equally important that a form of authentication be used that cannot be faked. Digital signatures are a recent development answering to the need for authentication in the realm of networked digital communications.

Digital signatures play a role for digital documents similar to that played by handwritten signatures for printed documents. The signature is an unforgeable piece of data asserting that the named person wrote or otherwise agreed to the document to which the signature is attached. The recipient, as well as a third party, can verify that the document did indeed originate from the person whose signature is

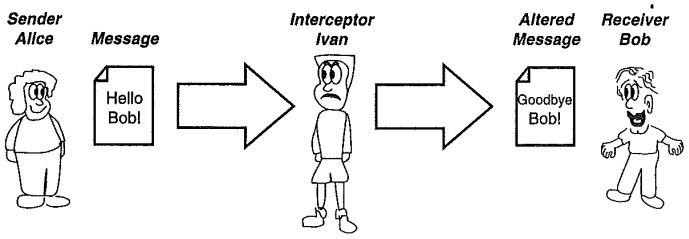


Figure 8.3

Ivan impersonates as Alice and sends an ill-intended message to Bob.

IPR2025-00714
DATABRICKS EX1008 Page 204

attached. A secure digital signature system consists of these parts:

- A method of signing a document such that forgery is unfeasible.
- A method of verifying that a signature actually was generated by whomever it represents.

Digital signatures will be discussed in a later section.

Integrity: Ensuring Message Content Remains Unaltered

With electronic commerce on the Internet, data integrity is critical. When product catalogs are distributed over the World Wide Web, for example, the recipient needs to be sure that the listed prices are authentic and have not been secretly altered by potentially unscrupulous competitors. Data integrity is critical for many other things as well. The message contents of official academic documents in electronic form from universities and colleges, for example, must not be modified. As figure 8.4 illustrates, however, there is real danger of an unprotected message being intercepted as it travels on the Internet. Furthermore, the message contents could be tampered with maliciously, with potentially grave consequences.

A valid digital signature on a message ensures that the message has not been altered since it was signed. Furthermore, secure digital signatures cannot be repudiated; the signer of a message cannot later disown it by claiming the signature or the message was forged. In this way, a digital signature acts like a tamper-proof seal testifying to the integrity of the message.

Before I discuss how encryption and digital signatures are used to implement secure transactions in a digital economy populated by agents, a brief tour of classical cryptography is in order.

Brief Tour of Classical Cryptography

Cryptography—the science of secret-writing to hide the meaning of messages—has been around for millennia. Cryptography is an ancient art first carried out in the form of hieroglyphic inscriptions on Egyptian tombs of noble men because it was believed that cryptic epitaphs induce an aura of mystic powers. Throughout the ages, cryptography was fulfilling its more important role of protecting vital communications through hostile environments, for both military and political purposes. During Roman

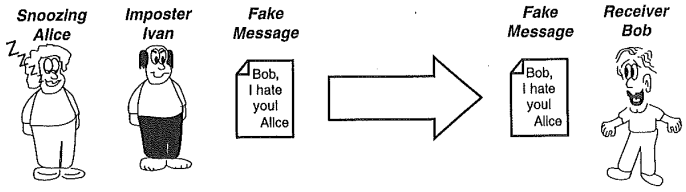


Figure 8.4

Ivan intercepts a message sent from Alice to Bob and alters it with less than noble intention.

IPR2025-00714
DATABRICKS EX1008 Page 205

times, Julius Caesar was known to have used the famous Caesar cipher for protecting military communications from Gaul to Rome. In the sixteenth century, Mary, Queen of Scots, lost her life after lending her support to a failed political coup, hastened in part by insecure cryptography. She was convicted for high treason and was decapitated after an incriminating letter sent from her prison was intercepted and deciphered.

In the modern era, the invention of the telegraph and radio has brought instantaneous communications to the army. Without the use of cryptography, communications using these newly invented technologies would have been easily compromised and rendered worse than useless. After all, telegraph lines can be wiretapped (as occurred during the Civil War on both the Confederate and Union troops), and radio waves can be intercepted simply by tuning in with the right antenna.



An excellent history of cryptography can be found in the book *The Codebreakers*, by David Kahn (1967). An introduction to modern cryptography can be found in Ron Rivest's 1990 article, "Cryptography," as well as in "Modern Cryptography," by G. Brassard (1988). A highly readable account of various developments in cryptography up to the present day can be found in Simson Garfinkels book entitled *PGP: Pretty Good Privacy* (1995).

The Role of NSA

During World War II, the first digital computers were developed by the Allies to crack the Germans' Enigma code under the brilliant leadership of Alan Turing (Hodges 1983; Kahn 1991). After the war, the world's cryptographic activities became concentrated in the National Security Agency (NSA), a highly secretive branch of the U.S. Department of Defense that was created by order of President Harry Truman in 1952. Located a half-hour drive from Washington D.C. at Fort Meade, Maryland, the agency's existence was kept secret for many years. In fact, it was rumored that NSA actually stood for "No Such Agency" or "Never Say Anything."

It is widely believed that NSA's classified charter is to intercept and decode all foreign communications of interest to the security of the U.S. The agency operates a global intelligence network, employs a host of top-notch cryptographers, and is always eager to have the world's fastest computer for breaking codes (Bamford 1982). To prevent potential national enemies from employing encryption methods too strong for the NSA to crack, the NSA has an interest in slowing the spread of publicly available cryptography. As a result, the NSA is widely believed to have followed policies with the practical effect of weakening and limiting publicly available crytographic tools. As a premier cryptographic government agency, the NSA has huge resources to exert a profound influence on the development and use of cryptography in the U.S., with potentially world-wide repurcussions.

Development of Data Encryption Standard (DES)

The proliferation of digital computing equipment in the decades after World War II led private firms and individuals to demand security for stored computer files and electronically transmitted messages.

IPR2025-00714
DATABRICKS EX1008 Page 206

To meet this demand, private researchers began to invade the highly technical realms of cryptography that had long been a government monopoly (Kahn 1983). In the late 1960s, IBM set up a cryptographic research group at its Yorktown Heights research laboratory to develop a cipher code-named Lucifer, which it promptly sold to Lloyd's of London for use in a cash-dispensing system. Spurred by its initial success, IBM set about to transform Lucifer into a highly marketable commodity. By 1974, the cipher was ready for market. At the time, there also were several other companies developing and selling cryptographic products, and none of them could interoperate.

At around the same time, the National Bureau of Standards (NBS), later to be renamed National Institute of Standards and Technology (NIST), began to study government and civilian need for computer security. NBS concluded that the nation could benefit from using a single data encryption standard for the purpose of storing and transmitting unclassified information. In response to a request from NBS for a proposal, a version of the Lucifer algorithm, which was weakened in some ways and strengthened in other ways by the NSA, was submitted as a candidate. NBS accepted the resulting algorithm in 1975 and formally adopted it as the Data Encryption Standard (DES) in 1976 for use in all classified government communications. The details of DES can be found in the official FIPS publication (1988).

Problems with DES were widely acknowledged as soon as the standard was first proposed. DES was made just strong enough to withstand commercial attempts to break it, yet weak enough to yield to government crypt analysis. In keeping with rapid advances in computing speed over the years, however, DES has been strengthened with longer keys, larger block sizes, and more rounds of encryption. Variations such as the triple-DES now are in common use. More recent algorithms such as IDEA (International Data Encryption Algorithm) (LM 1991), RC2, and RC4 (RC for Rivest Code) also have been popular.

Development of Public-Key Cryptography

In the early 1970s, a growing awareness of the need for data encryption in digital communications coupled with a sense of urgency brought on by the imminent deployment of DES (which many computer scientists abhorred) led to a series of surprising breakthroughs in cryptographic research.

Problems with Secret Keys

The traditional approach to cryptography, also called *secret-key cryptography*, is based upon the sender and receiver of a message sharing common knowledge of the same secret key. As illustrated previously in figure 8.2, this secret key is used to both encrypt and decrypt the message. Secret-key cryptography, however, has a fundamental problem: how to get both the sender and the receiver to agree on a secret key without a third party finding out.

If the sender and the receiver are at separate physical locations, they must trust a courier, the phone system, the computer network, or some other means of transmission not to disclose the secret key being communicated. Anyone who overhears

IPR2025-00714
DATABRICKS EX1008 Page 207

or intercepts the key in transit can later decipher all messages encrypted using that key, and future communications between the two parties are compromised.

Key Management

The generation, distribution, and storage of keys is called *key management*. Secret-key cryptography often has difficulty providing secure key management. For many years, the U.S. government used a key distribution center to generate and distribute keys to any pair of individuals who wanted to communicate. The key transmission method was crude but simple: Cryptographic keys were placed in locked briefcases that were handcuffed to couriers who physically transported them from Washington to embassies and consulates around the world.

In 1976, two researchers at Stanford University, Whitfield Diffie and Martin Hellman, devised a devilishly clever technique that enables two communicating parties to derive a cryptographic session key in such a way that a snooping third party cannot deduce the key's value. The session key then can be used in a secret-key algorithm such as DES. The Diffie-Hellman algorithm requires that the two communicating parties actively participate in carrying out the key exchange protocol at the same time. This technique works great for two parties talking over the telephone but is otherwise not practicable for asynchronous modes of communication, such as electronic mail.

The RSA Alternative

In 1977, three scientists at MIT's Laboratory for Computer Science, Ron Rivest, Adi Shamir, and Len Adleman, refined Diffie and Hellman's idea of secure key exchange and invented what came to be known as the RSA public-key cryptosystem. The name RSA stands for its developers, Rivest, Shamir, and Adleman. As an improvement over the Diffie-Hellman key exchange system, RSA requires no active participation between the sender performing the encryption and the receiver performing the decryption. Each person gets a pair of keys, called the *public key* and the *private key*. Each person's public key is published while his private key is kept secret. All communications involve only public keys; no private keys are transmitted or shared. Suppose, for example, that Alice wants to send a message to Bob. She looks up Bob's public key in a directory, uses it to encrypt the message, and sends the message. Bob then uses his private key to decrypt the message and read it. An eavesdropper without the private key cannot decipher the message. This is illustrated in figure 8.5.

Because there is no need for the sender and the receiver to share secret information, it is no longer necessary to trust a communications channel to be secure against eavesdropping. Furthermore, the two communicating parties do not have to know each other or have any type of previous communication. Anyone, for example, can send a confidential message to Bob using only Bob's public key, without requiring any prior arrangement with Bob.

The RSA cryptosystem is based on an amazingly simple number-theoretic idea that has been able to resist all cryptanalytic attacks. The idea is that although it is easy to multiply two large prime numbers, it is extremely difficult to factorize their product. Thus the product can be publicized and used as the public encryption key. The primes

IPR2025-00714
DATABRICKS EX1008 Page 208