UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SAMSUNG ELECTRONICS CO. LTD. and SAMSUNG ELECTRONICS AMERICA, INC.,
Petitioners,

v.

VB ASSETS, LLC
Patent Owner

IPR2025-00871

U.S. Patent 10,510,341

DECLARATION OF JUNE ANN MUNFORD

1. My name is June Ann Munford. I am over the age of 18, have personal knowledge of the facts set forth herein, and am competent to testify to the same.

2. I earned a Master of Library and Information Science (MLIS) from the University of Wisconsin-Milwaukee in 2009. I have over ten years of experience in the library/information science field. Beginning in 2004, I have served in various positions in the public library sector including Assistant Librarian, Youth Services Librarian and Library Director. I have attached my Curriculum Vitae as Appendix CV.

3. During my career in the library profession, I have been responsible for materials acquisition for multiple libraries. In that position, I have cataloged, purchased and processed incoming library works. That includes purchasing materials directly from vendors, recording publishing data from the material in question, creating detailed material records for library catalogs and physically preparing that material for circulation. In addition to my experience in acquisitions, I was also responsible for analyzing large collections of library materials, tailoring library records for optimal catalog

search performance and creating lending agreements between libraries during my time as a Library Director.

4. I am fully familiar with the catalog record creation process in the library sector. In preparing a material for public availability, a library catalog record describing that material would be created. These records are typically written in Machine Readable Catalog (herein referred to as "MARC") code and contain information such as a physical description of the material, metadata from the material's publisher, and date of library acquisition. In particular, the 008 field of the MARC record is reserved for denoting the date of creation of the library record itself. As this typically occurs during the process of preparing materials for public access, it is my experience that an item's MARC record indicates the date of an item's public availability.

5. Typically, in creating a MARC record, a librarian would gather various bits of metadata such as book title, publisher and subject headings among others and assign each value to a relevant numerical field. For example, a book's physical description is tracked in field 300 while title/attribution is tracked in field 245. The 008 field of the MARC record is reserved for denoting the creation of the library record itself. As this is the only date reflecting the inclusion of said materials within the library's collection, it is my experience

that an item's 008 field accurately indicates the date of an item's public availability.

6. I have reviewed Exhibit 1005, SmartKom: Foundations of Multimodal Dialogue Systems by Wolfgang Wahlster.

7. Attached hereto as Appendix WAHLSTER01 is a true and correct copy of the MARC record for SmartKom: Foundations of Multimodal Dialogue Systems as held by the University of Pittsburgh Library. I secured this record myself from the library's public catalog. The MARC record contained within Appendix WAHLSTER01 accurately describes the title, author, publisher, and ISBN number of SmartKom: Foundations of Multimodal Dialogue Systems.

8. Attached hereto as Appendix WAHLSTER02 is a true and correct copy of SmartKom: Foundations of Multimodal Dialogue Systems as held by the University of Pittsburgh Library. These scans were secured from the library's collection. In comparing Exhibit 1005 to Appendix WAHLSTER02, it is my determination that Exhibit 1005 is a true and correct copy of SmartKom: Foundations of Multimodal Dialogue Systems by Wolfgang Wahlster.

9. The 008 field of the MARC record in Appendix WAHLSTER01 indicates the date of record creation. The 008 field of Appendix WAHLSTER01 indicates the University of Pittsburgh Library first acquired SmartKom: Foundations of Multimodal Dialogue Systems as of August 4, 2006. Considering this information, it is my determination that SmartKom: Foundations of Multimodal Dialogue Systems was made available to the public shortly after its initial acquisition as of August 4, 2006.

10. I have reviewed Exhibit 1006, User Models in Dialog Systems by Alfred Kobsa.

11. Attached hereto as Appendix KOBSA01 is a true and correct copy of the MARC record for User Models in Dialog Systems as held by the Penn State University Library. I secured this record myself from the library's public catalog. The MARC record contained within Appendix KOBSA01 accurately describes the title, author, publisher, and ISBN number of User Models in Dialog Systems.

12. Attached hereto as Appendix KOBSA02 is a true and correct copy of User Models in Dialog Systems as held by the Penn State University Library. These scans were secured from the library's collection. In comparing Exhibit

1006 to Appendix KOBSA02, it is my determination that Exhibit 1006 is a true and correct copy of User Models in Dialog Systems by Alfred Kobsa.

13. The 008 field of the MARC record in Appendix KOBSA01 indicates the date of record creation. The 008 field of Appendix KOBSA01 indicates the Penn State University Library first acquired User Models in Dialog Systems as of December 5, 1989. Considering this information, it is my determination that User Models in Dialog Systems was made available to the public shortly after its initial acquisition as of December 5, 1989.

14. I have been retained on behalf of the Petitioner to provide assistance in the above-illustrated matter in establishing the authenticity and public availability of the documents discussed in this declaration. I am being compensated for my services in this matter at the rate of $200.00 per hour plus reasonable expenses. My statements are objective, and my compensation does not depend on the outcome of this matter.

15. I declare under penalty of perjury that the foregoing is true and correct. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made the knowledge that

willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: 5/1/2025

June Ann Munford

APPENDIX CV

June A. Munford
Curriculum Vitae

Education

University of Wisconsin-Milwaukee - MS, Library & Information Science, 2009
Milwaukee, WI

- Coursework included cataloging, metadata, data analysis, library systems, management strategies and collection development.
- Specialized in library advocacy, cataloging and public administration.

Grand Valley State University - BA, English Language & Literature, 2008
Allendale, MI

- Coursework included linguistics, documentation and literary analysis.
- Minor in political science with a focus in local-level economics and government.

Professional Experience

Researcher / Expert Witness, October 2017 – present
Freelance ● Pittsburgh, Pennsylvania & Grand Rapids, Michigan

- Material authentication and public accessibility determination. Declarations of authenticity and/or public accessibility provided upon research completion. Experienced with appeals and deposition process.

- Research provided on topics of public library operations, material publication history, digital database services and legacy web resources.

- Past clients include Alston & Bird, Arnold & Porter, Baker Botts, Fish & Richardson, Erise IP, Irell & Manella, O'Melveny & Myers, Perkins-Coie, Pillsbury Winthrop Shaw Pittman and Slayden Grubert Beard.

Library Director, February 2013 - March 2015
Dowagiac District Library ● Dowagiac, Michigan

- Executive administrator of the Dowagiac District Library. Located in Southwest Michigan, this library has a service area of 13,000, an annual operating budget of over $400,000 and total assets of approximately $1,300,000.

● Developed careful budgeting guidelines to produce a 15% surplus during the 2013-2014 & 2014-2015 fiscal years while being audited.

● Using this budget surplus, oversaw significant library investments including the purchase of property for a future building site, demolition of existing buildings and building renovation projects on the current facility.

● Led the organization and digitization of the library's archival records.

● Served as the public representative for the library, developing business relationships with local school, museum and tribal government entities.

● Developed an objective-based analysis system for measuring library services - including a full collection analysis of the library's 50,000+ circulating items and their records.

November 2010 - January 2013
Librarian & Branch Manager, Anchorage Public Library ● Anchorage, Alaska

● Headed the 2013 Anchorage Reads community reading campaign including event planning, staging public performances and creating marketing materials for mass distribution.

● Co-led the social media department of the library's marketing team, drafting social media guidelines, creating original content and instituting long-term planning via content calendars.

● Developed business relationships with The Boys & Girls Club, Anchorage School District and the US Army to establish summer reading programs for children.

June 2004 - September 2005, September 2006 - October 2013
Library Assistant, Hart Area Public Library
Hart, MI

● Responsible for verifying imported MARC records and original MARC cataloging for the local-level collection as well as the Michigan Electronic Library.

● Handled OCLC Worldcat interlibrary loan requests & fulfillment via ongoing communication with lending libraries.

Professional Involvement

Alaska Library Association - Anchorage Chapter

● Treasurer, 2012

Library Of Michigan
● Level VII Certification, 2008
● Level II Certification, 2013

Michigan Library Association Annual Conference 2014
● New Directors Conference Panel Member

Southwest Michigan Library Cooperative
● Represented the Dowagiac District Library, 2013-2015

Professional Development

Library Of Michigan Beginning Workshop, May 2008
Petoskey, MI
● Received training in cataloging, local history, collection management, children's literacy and reference service.

Public Library Association Intensive Library Management Training, October 2011
Nashville, TN
● Attended a five-day workshop focused on strategic planning, staff management, statistical analysis, collections and cataloging theory.

Alaska Library Association Annual Conference 2012 - Fairbanks, February 2012
Fairbanks, AK
● Attended seminars on EBSCO advanced search methods, budgeting, cataloging, database usage and marketing.

Depositions

2019 ● Fish & Richardson

Apple v. Qualcomm (Case No. IPR2018-001281, 39521-00421IP, IPR2018-01282 and 39521-00421IP2)

2019 ● Erise IP

Implicit, LLC v. Netscout Systems, Inc (Case No. 2:18-cv-53-JRG)

2019 ● Perkins-Coie

Adobe Inc. v. RAH Color Technologies LLC (Case No. IPR2019-00627, IPR2019-00628, IPR2019-00629 and IPR2019-00646)

2020 ● O'Melveny & Myers

Maxell, Ltd. v. Apple Inc. (Case No. 5:19-cv-00036-RWS)

2021 ● Pillsbury Winthrop Shaw Pittman LLP

Intel v. SRC (Case No. IPR2020-1449)

2022 ● Perkins-Coie

Realtek v. Future Link (Case No. IPR2021-01182)

2023 ● Fish & Richardson

Neuroderm Ltd. v. Abbvie, Inc (Case No. PGR2022-00040)

2023 ● Fish & Richardson

Nearmap US Inc. v. Pictometry International Corp. (Case No. IPR2022-00735)

2023 ● Fish & Richardson

Samsung Electronics v. MemoryWeb LLC (Case No. 39843-0136PS1)

2023 ● Pillsbury Winthrop Shaw Whitman LLP

Gravel Rating Systems v. Costco Wholesale Corp. (Case No. 4:21-cv-149-ALM)

2024 ● Willkie-Farr

Netflix, Inc. v. VideoLabs. Inc. (Case No. IPR2023-00628)

2024 ● Quinn Emanuel Urquhart & Sullivan

Purdue University v. Wolfspeed (Case No. 1:21-CV-840)

2024 ● Fish & Richardson

Dish Network v. Entropic Communications (Case No. IPR2024-00393)

2025 ● Perkins Coie

Amazon.com, Inc. v. Nokia Technologies Oy (Case No. IPR2024-00572)

2025 ● Fish & Richardson

Dish Network v. Entropic Communications (Case No. IPR2024-00373)

Limited Case and Clientele History

Alston & Bird

● Ericsson

v. Collision Communications (Case No. IPR2022-01233)

● Nokia

v. Neptune Subsea, Xtera (Case No. 1:17-cv-01876)

● Universal Electronics Inc

v. Roku Inc (Case No. IPR2022-00818)

Alavi & Anaipakos PLLC

● Stingray Group Inc. and Stingray Music USA, Inc.

v. Edwin A. Hernandez-Mondragon and Egla Corp. (Case No. 1:24-cv-21226-RAR)

Arnold & Porter

● Ivantis

v. Glaukos (Case No. 8:18-cv-00620)

● Samsung

v. Jawbone (Case No. 2:21-cv-00186)

Benesch Friedlander Coplan & Aronoff

- Voyis

  v. Cathx (Case No. 5:21-cv-00077-RWS)

Buchanan, Ingersoll & Rooney PC

- Google LLC, AT&T Service Inc., T Mobile USA Inc., Cellco Partnership, Ericsson Inc. & Nokia of America Corp.

  v. KT Corp & Pegasus Wireless Innovation LLC (Case No. IPR2025-00293)

Deschert LLP

- Smaxtec Animal Care

  v. ST Reproductive Technologies, LLC (Case No. IPR2024-00885)

Duane Morris

- Kangxi Communication Technologies

  v. Skyworks Solutions Inc. (Case No. IPR2025-00373)

- Microsoft

  v. Edge Networking Systems LLC (Case No. 1:24-cv-00215)

Erise I.P.

- Apple

  v. Ericsson Inc. (Case No. IPR2022-00715)

  v. Future Link Systems (Case No. IPRs 6317804, 6622108, 6807505, and 7917680)

  v. INVT (Case No. 20-1881)

  v. Navblazer LLC (Case No. IPR2020-01253)

  v. Qualcomm (Case No. IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

v. Quest Nettech Corp (Case No. 2:19-cv-00118-JRG)

v. Telefonaktiebolaget LM Ericsson (Case No. IPR2022-00275)

v. Theta IP, LLC (Case No. IPR2024-00818)

v. THL Holding Company LLC (Case No. 1:23-cv-00548)

- Fanduel

  v. CGT (Case No. 19-1393)

- Garmin

  v. Phillips North America LLC (Case No. 2:19-cv-6301-AB-KS)

- Netscout

  v. Longhorn HD LLC (Case No. 2:20-cv-00349)

  v. Implicit, LLC (Case No. 2:18-cv-53-JRG)

- Sony Interactive Entertainment LLC

  v. Bot M8 LLC (Case No. IPR2020-01288)

  v. Infernal Technology LLC (Case No. 2:19-CV-00248-JRG)

- Tesla

  v. Charge Fusion Technologies LLC (Case No. IPR2025-00032)

- Unified Patents

  v. GE Video Compression (Case No. 2:19-cv-248)

Fish & Richardson

- Apple

  v. AliveCor (Case No. 3:21-cv-03958)

  v. LBS Innovations (Case No. 2:19-cv-00119-JRG-RSP)

  v. Koss Corporation (Case No. IPR2021-00305)

v. Masimo (Case No. IPR 50095-0012IP1, 50095-0012IP2, 50095-0013IP1, 50095-0013IP2, 50095-0006IP1, 50095-0135IP1)

v. Neonode (Case No. 21-cv-08872-EMC)

v. Qualcomm (Case No. IPR2018-001281, 39521-00421IP, IPR2018-01282, 39521-00421IP2)

v. Resonant Sys. (Case No. 7:23-cv-00077-ADA)

- AsusTek Computer Inc.
  v. Videolabs, Inc. (Case No. 22-CV-00720-ADA)

- Dell
  v. Neo Wireless (Case No. IPR2022-00616)

- Dish Network

  v. Entropic Communications, LLC (Case No. 2:2023-CV-01043)

  v. Entropic Communications LLC (Case No. IPR2024-00393)

  v. Realtime Adaptive Streaming (Case No. 1:17-CV-02097-RBJ)

  v. TQ Delta LLC (Case No. 18-1798)

- Evapco Dry Cooling

  v. SPG Dry Cooling (Case No. IPR2021-00688)

- Genetec

  v. Sensormatic Electronics (Case No. 1:20-CV-00760)

- Huawei

  v. Bell Northern Research LLC (Case No. IPR2019-01174)

- Kianxis

  v. Blue Yonder (Case No. 3:20-cv-03636)

- LG Electronics

  v. Bell Northern Research LLC (Case No. 3:18-cv-2864-CAB-BLM)

- MediaTek

  v. MOSAID (Case No. IPR2024-00718)

- Metaswitch

  v. Sonus Networks (Case No. IPR2018-01719)

- Microsoft

  v. Throughputer Inc (Case No. IPR2022-00757)

- Mom Enterprises

  v. Ddrops Company (Case No. 1:22-cv-00332)

- MLC Intellectual Property

  v. MicronTech (Case No. 3:14-cv-03657-SI)

- Nearmap Inc

  v. EagleView Technologies (Case No. IPR2022-01009)

- Neuroderm Ltd.

  v. Abbvie, Inc (Case No. PGR2022-00040)

- Posco Co Ltd.

  v. Pascal Drillet et al. (Case No. IPR2025-00370)

- Realtek Semiconductor

  v. Future Link (Case No. IPR2021-01182)

- Quectel
  - v. Koninklijke Philips (Case No. 1:20-cv-01710)

- Samsung
  - v. Aire Technology (Case No. IPR2022-00877)
  - v. Bell Northern Research (Case No. 2:19-cv-00286-JRG)
  - v. Communication Technologies Inc (Case No. IPR2022-01221)
  - v. Jawbone Innovations (Case No. IPR2022-00865)
  - v. MemoryWeb LLC (Case No. IPR2022-00885)
  - v. Dodots Licensing Solutions (Case No. IPR2023-00701)
  - v. Telefonaktiebolaget LM Ericsson (Case No. IPR2021-00615)

- Texas Instruments
  - v. Vantage Micro LLC (Case No. IPR2020-01261)

- Xilinx
  - v. Sentient Sensors LCC (Case No. 1:22-cv-00173)

Goodwin Proctor

- Intel
  - v. Proxense, LLC (Case No. 6:24-cv-00283)

- Samsung Electronics
  - v. Mobile Data Technologies (Case No. 2:24-cv-00435)

- Taiwan Semiconductor Manufacturing Company Ltd.
  - v. Advanced Integrated Circuit Process LLC (Case No. 2:24-cv-00623)

Groombridge Wu

- Nearmap

  v. Eagle View Technologies Inc. and Pictometry International Corp. (Case No. 2:21-cv-00283-TS-DAO)

Irell & Manella

- Curium

Latham & Watkins LLP

- Apple

  v. Advanced Coding Research LLC

Leydig, Voit & Mayer, Ltd.

- Public Availability Consultancy

O'Melveny & Myers

- Apple

  v. Maxell (Case No. 5:19-cv-00036-RWS)

- Disney Media & Entertainment Distribution LLC

  v. Digital Media Technology Holdings, LLC (Case No. 2:22-cv-01642)

- Micron Technology Inc.

  v. Besang Inc (Case No. IPR2023-00900)

- Samsung

  v. Daedalus Prime LLC (1335 Investigation)

Perkins-Coie

- Amazon

  v. Nokia (Case No. IPR2024-00847 and IPR2024-00848)

  v. Nokia Technologies Oy (Case No. IPR2024-00572)

- Heru Industries

  v. The UAB Research Foundation (Case No. IPR2022-01148)

- Intel Corporation

  v. BeSang Inc. (Case No. IPR2023-00991)

  v. BitMICRO (Case No. 5:23-cv-00625)

  v. Telefonaktiebolaget L M Ericsson (Case No. IPR2024-00610)

- Realtek Semiconductor

  v. Future Link (Case No. IPR2021-01182)

- r-pac

  v. Adasa, Inc. (Case No. 1:2024cv06102)

- Twitter Inc

  v. VOIP-Pal.com (Case No. 3:20-cv-02397-JD)

- TCL Industries

  v. Koninklijke Philips NV (Case No. IPR2021-00495, IPR2021-00496 and IPR2021-00497)

- VusionGroup

  v. Hanshow (Case No. IPR2024-00857)

Pillsbury Winthrop Shaw Pittman

- Intel

  v. FG SRC LLC (Case No. 6:20-cv-00315)

- Gravel Rating Systems

  v. Costco (Case No. 4:21-cv-149)
  v. Lowe's Home Centers (Case No. 4:21-cv-150)
  v. T-Mobile USA (Case No. 4:21-cv-152)
  v. Kohl's Inc. (Case No. 4:21-cv-258)
  v. Under Armor (Case No. 4:21-cv-356)

Quinn Emanuel

- Exact Sciences Corporation

  v. Geneoscopy, Inc. (Case No. IPR2024-00459 and IPR2024-1330)

- ServiceNow, Inc.

  v. InQuisient, Inc. (Case No. 22-900-CJB)

- Wolfspeed

  v. The Trustees of Purdue University (Case No. 1:2021-cv-00840)

Sheppard, Mullin, Richter & Hampton LLP

- Advanced Micro Devices, Inc and Pensando Systems, Inc.

  v. Concurrent Ventures LLC and Xtreamedge, Inc. (Case No. IPR2025-00478)

- Cadence Design Systems Inc.

  v. Semiconductor Design Technologies (Case No. 3:23-cv-01001)

Shook, Hardy & Bacon LLP

- HP Inc.

  v. Universal Connectivity Technologies (Case No. 1:23-cv-01177)

Stern, Kessler, Goldstein & Fox

- Preliminary Research

Wilmer, Cutler, Pickering, Hale and Dorr

- Apple

  v. Shunock (Case No. 1:23-cv-08598)

- Quest Diagnostics Inc., Haystack Oncology Inc. and Haystack Oncology GmbH

  v. Natera, Inc. (Case No. 1:23-cv-08598)

- Roche Diabetes, Inc.

  v. Trividia Health, Inc. (Case No. 1:24-cv-00668)

Willkie, Farr & Gallagher LLP

- Lenovo, Dell, HP

  v. Universal Connectivity Technologies Inc. (Case No. 2:2023cv00449)

- Neurent

  v. The Foundry, LLC (Case No. IPR2024-00669)

- Netflix, Inc.

  v. VideoLabs, Inc. (Case No. IPR2023-00628)

APPENDIX KOBSA01

PennState University Libraries | Catalog

Keyword ▾ Search... Search Advanced Search Start Over

Share Bookmark Report an Issue

MARC View

LEADER 01214nam a22003018a 4500

001 1006186

003 SIRSI

008 891205s1989 xx b 000 0 eng d

035 a| LIAS1457450

010 a| 88028156

019 a| MARS

020 a| 0387183809 q| (U.S. : alk. paper)

040 c| PSt d| WaOLN

050 0 a| QA76.9.H85 b| U74 1989

245 0 0 a| User models in dialog systems / c| Alfred Kobsa, Wolfgang Wahlster (eds.) ; with contributions by S. Carberry ... [et al.].

260 a| Berlin ; a| New York : b| Springer-Verlag, c| c1989.

300 a| xi, 471 p. : b| ill. ; c| 25 cm.

490 1 a| Symbolic computation. Artificial intelligence

500 a| Based on UM86, the first international workshop on user modeling, held in Maria Laach, Germany.

500 a| Includes index.

504 a| Includes bibliographical references.

650 0 a| Human-computer interaction.

650 0 a| Artificial intelligence

650 0 a| Question-answering systems

700 1 a| Kobsa, Alfred.

700 1 a| Wahlster, Wolfgang.

830 0 a| Symbolic computation p| Artificial intelligence

949 a| QA76.9.H85U74 1989 w| LC c| 1 i| 24460203017 d| 6/28/2000 e| 5/4/2000 l| CATO-PARK m| UP-ANNEX n| 4 r| Y s| Y t| BOOK u| 4/13/2001 x| B2 z| 2446 0| 0203 1| 017

ASK A LIBRARIAN

PennState

Copyright © 2019 The Pennsylvania State University. All rights reserved. Except where otherwise noted, this work is subject to a Creative Commons Attribution 4.0 License. Details and exceptions.

Legal Statements | Accessibility

PENN STATE UNIVERSITY LIBRARIES

Libraries Home
Give Feedback or Ask Questions
Help
Report Problems (IRS)
Website Feedback
Hours and Locations
Jobs/Careers

CONNECT WITH PENN STATE UNIVERSITY LIBRARIES

Facebook
Twitter
Instagram

APPENDIX KOBSA02

PennState University Libraries | Catalog

Bookmarks Course Reserves My Account

Keyword Search... Search Advanced search Start Over

Share Bookmark Report an issue

User models in dialog systems / Alfred Kobsa, Wolfgang Wahlster (eds.) ; with contributions by S. Carberry ... [et al.].

Published	Berlin ; New York : Springer-Verlag, c1989.
Physical Description	xi, 471 p. : ill. ; 25 cm.
Additional Creators	Kobsa, Alfred and Wahlster, Wolfgang

ASK A LIBRARIAN

Availability

Browse Nearby on Shelf I Want It

Library Storage (1 item)

Call number	Material	Location
QA76.9.H85U74 1989	Book	Being transferred between libraries

Series

Symbolic computation Artificial intelligence

Subject(s)

Human-computer interaction
Artificial intelligence
Question-answering systems

ISBN

0387183809 (U.S. : alk. paper)

Note

Based on UM86, the first international workshop on user modeling, held in Maria Laach, Germany.
Includes index.

Bibliography Note

Includes bibliographical references.

View MARC record | catkey: 1006186

PennState

Copyright ©2019 The Pennsylvania State University. All rights reserved. Except where otherwise noted, this work is subject to a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 license. Details and provisions.

Legal Statements | PCI Compliance

PENN STATE UNIVERSITY LIBRARIES

CONNECT WITH PENN STATE UNIVERSITY LIBRARIES

PSU LIBRARIES ANNEX
2446 02 03 017

SYMBOLIC COMPUTATION

A. Kobsa W. Wahlster (Eds.)

User Models in Dialog Systems

With Contributions by

S. Carberry J. G. Carbonell D. N. Chin R. Cohen
J. Fain Lehman T. W. Finin A. Jameson M. Jones
R. Kass A. Kobsa K. F. McCoy K. Morik C. L. Paris
A. E. Quilici E. Rich K. Sparck Jones W. Wahlster

Springer-Verlag

A. Kobsa, W. Wahlster (Eds.)

User Models in Dialog Systems

Math
QA76
.9
.H85
U74
1989

Kobsa · Wahlster (Eds.) User Models in Dialog Systems

User models have recently attracted much research interest in the field of artificial intelligence dialog systems. It has become evident that flexible user-oriented dialog behavior of such systems can be realized only if the system has a model of the user containing assumptions about background knowledge as well as the user's goals and plans in consulting the system. Research in the field of user models investigates how such assumptions can be automatically created, represented and exploited by the system in the course of an interaction with the user.

This volume is the first comprehensive survey of the field of user modeling. It is divided into four sections. The introductory section contains a general view of the field as a whole, and a number of surveys of specific problems and techniques in user modeling. Sections two and three describe eight user modeling systems, with the focus on the automatic creation and exploitation of assumptions about the user. The final section discusses several limits of current systems, and proposes solutions as to how some of the shortcomings might be overcome.

All sections start with an editorial introduction. A subject index and an extensive bibliography supplement the book.

A0000015697721

ISBN 3-540-18380-9
ISBN 0-387-18380-9

PENNSTATE

UNIVERSITY
LIBRARIES

MATHEMATICS LIBRARY

PENNSTATE

UNIVERSITY
LIBRARIES

MATHEMATICS LIBRARY

SYMBOLIC COMPUTATION

Artificial Intelligence

Managing Editor: D. W. Loveland

Editors: S. Amarel A. Biermann L. Bolc A. Bundy
H. Gallaire P. Hayes A. Joshi D. Lenat A. Mackworth
E. Sandewall J. Siekmann W. Wahlster

Alfred Kobsa Wolfgang Wahlster (Eds.)

User Models in Dialog Systems

With Contributions by

S. Carberry J.G. Carbonell D.N. Chin R. Cohen
J. Fain Lehman T.W. Finin A. Jameson M. Jones
R. Kass A. Kobsa K.F. McCoy K. Morik
C.L. Paris A.E. Quilici E. Rich K. Sparck Jones
W. Wahlster

With 113 Figures

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong

Alfred Kobsa
Wolfgang Wahlster
Department of Computer Science
University of Saarbrücken
D-6600 Saarbrücken 11
Fed. Rep. of Germany

ISBN 3-540-18380-9 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-18380-9 Springer-Verlag New York Berlin Heidelberg

Library of Congress Cataloging-in-Publication Data.
User models in dialog systems / Alfred Kobsa, Wolfgang Wahlster (eds.); with contributions by S. Carberry ... (et al.)
p. cm. — (Symbolic computation. Artificial intelligence)
Based on UM86, the first international workshop on user modeling held in Maria Laach, Germany.
Bibliography: p. Includes index.
ISBN 0-387-18380-9 (U.S.: alk. paper)
1. Human-computer interaction. 2. Artificial intelligence. 3. Question-answering systems. I. Kobsa, Alfred. II. Wahlster, Wolfgang. III. Series. QA76.9.H85U74 1989 004'.01'9—dc19 88-28156 CIP

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. Duplication of this publication or parts thereof is only permitted under the provisions of the German Copyright Law of September 9, 1965, in its version of June 24, 1985, and a copyright fee must always be paid. Violations fall under the prosecution act of the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1989
Printed in the United States of America

The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Media conversion: RL-Sort, Robert Laudensack, Cham, Fed. Rep. of Germany
2145/3140-543210 Printed on acid-free paper

Preface

User models have recently attracted much research interest in the field of artificial intelligence dialog systems. It has become evident that flexible user-oriented dialog behavior of such systems can be achieved only if the system has access to a model of the user containing assumptions about his/her background knowledge as well as his/her goals and plans in consulting the system. Research in the field of user models investigates how such assumptions can be automatically created, represented and exploited by the system in the course of an "on-line" interaction with the user. The communication medium in this interaction need not necessarily be a natural language, such as English or German. Formal interaction languages are also permitted. The emphasis is placed on systems with natural language input and output, however.

A dozen major and several more minor user modeling systems have been designed and implemented in the last decade, mostly in the context of natural-language dialog systems. The goal of UM86, the first international workshop on user modeling, was to bring together the researchers working on these projects so that results could be discussed and analyzed, and hopefully general insights be found, that could prove useful for future research. The meeting took place in Maria Laach, a small village some 40 miles south of Bonn, West Germany. 25 prominent researchers were invited to participate. The pleasant setting of the conference site close to the medieval abbey of Maria Laach and the volcanic Lake Laach fostered a nice atmosphere for intensive discussions and the exchange of ideas until the early hours of the morning.

This book is one of the documented results of the workshop (the other is a special issue of the *Computational Linguistics* journal containing topics which are more closely related to natural language processing). The volume consists of four parts: Part 1 contains three survey chapters, the first introducing the field of user modeling as a whole, and the two others analyzing an important modeling technique and the contents of user models, respectively. Part 2 deals with the construction, and Part 3 with the exploitation, of user models by a dialog system in an ongoing dialog. Part 4 analyzes shortcomings of user models as developed to date and outlines promising directions for future research.

Although this volume is based on a workshop, it was designed right from the beginning to form not only a collection of unintegrated conference papers, but to provide a rather coherent survey of the field of user modeling. Each part is prefaced by a separate editorial introduction which classifies the individual contributions within the topic of the respective subfield, and summarizes their central

issues. Both in the prefaces and in the individual chapters, cross-references to other contributions have been included whenever appropriate. The references of all contributions have been compiled into a single bibliography and were supplemented by a number of papers which have not been cited by the contributors, but certainly form part of a representative bibliography of the field. All authors were able to contribute to this extended bibliography. A subject index facilitates the comparison of how individual issues are treated by different authors. A citation index integrated into the bibliography allows for easy back-reference from the titles to the citing contributors and the page numbers where cited.

An unusual amount of discussion and reviewing has been necessary to achieve this coherency: speakers of the workshop had to distribute tentative versions of their papers long before the workshop so that their talks could be discussed more deeply at the meeting. Tape recordings of these discussions were made and distributed to all speakers, and written reviews of each paper were prepared by all participants. Selected speakers were then invited to submit their revised and extended papers for publication in the book. The editors made additional comments which authors had to take into account. Final problems were discussed at an editorial meeting at the IJCAI-87 conference in Milan. Altogether, the written correspondence *about* this book would certainly fill another volume.

The publication of the book would not have been possible without the assistance and support of a number of individuals and institutions, to whom we would like to express our gratitude:

- The Collaborative Research Programme on Artificial Intelligence and Knowledge-Based Systems (SFB 314) of the German Science Foundation (DFG), which funded the workshop;
- Mark Line and Bernd Schäfer who prepared transcripts and rendered the editors' English intelligible even to native speakers;
- Bernd Nessen and Sokrates Evangelidis for their work as sound engineers;
- Susi Schild and Christa Weinen who carefully compiled the bibliography; and
- Doris Borchers, Gabriele Jacquinot, and Parinaz Mohammadzadeh for their administrative assistance.

The international user modeling community owes a lot to them.

Saarbrücken, April 1988 *Alfred Kobsa* *Wolfgang Wahlster*

Table of Contents

Part I. A Survey of User Modeling

Introduction 2

1 **User Models in Dialog Systems** 4
Wolfgang Wahlster and Alfred Kobsa

1. Introduction 4
2. Constructing User Models 10
3. Representing User Models 14
4. Exploiting User Models 18
5. Open Questions and Future Research in User Modeling 29
6. References 33

2 **Stereotypes and User Modeling** 35
Elaine Rich

1. Introduction 35
2. The Definition of a Stereotype 36
3. The Space of User Models 38
4. Stereotypes and User Modeling 39
5. Stereotypes and Contradiction Resolution 41
6. Adaptation of Stereotype Systems 48
7. Parallel Models of System Knowledge and of Users 50
8. Summary 51
9. References 51

3 **A Taxonomy of Beliefs and Goals for User Models in Dialog Systems** 52
Alfred Kobsa

1. Introduction 52
2. Beliefs, Goals and Plans 53
3. Basic Beliefs 56
4. Basic Goals 59
5. Beliefs with Respect to Other Agents' Beliefs and Goals 60
6. Goals with Respect to Other Agents' Beliefs and Goals 63
7. A Classification of Existing User Models 65

8. Discussion 67
9. References 68

Part II. Building User Models

Introduction 70

4 **KNOME: Modeling What the User Knows in UC** 74
David N. Chin

1. Introduction 74
2. Internal Representation of Users 78
3. Deducing the User's Level of Expertise 83
4. Modeling UC's Knowledge 97
5. Exploiting KNOME 103
6. Conclusion 104
7. References 107

5 **Detecting and Responding to Plan-Oriented Misconceptions** 108
Alexander Quilici

1. Introduction 108
2. An Explanation-Based Approach 110
3. Representing User and Advisor Beliefs 111
4. Explanation-Based Misconception Recognition and Response 114
5. A Taxonomy of Potential Explanations 116
6. A Detailed Process Model 120
7. Accessing Advisor Planning Knowledge 122
8. Related Work 125
9. Implementation Details 128
10. Limitations and Future Work 128
11. Conclusions 131
12. References 132

6 **Plan Recognition and Its Use in Understanding Dialog** 133
Sandra Carberry

1. Introduction 133
2. Plan Recognition in Dialog Systems 135
3. Inferring and Modeling the Task-Related Plan 136
4. Application of User Models 144
5. Improving Plan Recognition 150
6. Constructing and Exploiting Other Components of a User Model 160
7. Conclusions and Current Research 161
8. References 162

7 **Learning the User's Language: A Step Towards Automated Creation of User Models** 163
Jill Fain Lehman and Jaime G. Carbonell

1. Introduction: Adaptable Interfaces 163
2. Foundations: Least-Deviant-First Parsing and MULTIPAR 165
3. CHAMP: Design for an Adaptive Parser 170
4. Hidden Operator Experiments with Professional Secretaries 182
5. Concluding Remarks 192
6. References 194

Part III. Exploiting User Models

Introduction 196

8 **The Use of Explicit User Models in a Generation System for Tailoring Answers to the User's Level of Expertise** 200
Cécile L. Paris

1. Introduction 200
2. Identifying What Needs to Be in the User Model 203
3. Two Descriptions Strategies Found in Texts: Constituency Schema and Process Trace 205
4. Mixing the Strategies 212
5. TAILOR 221
6. Further Work and Related Issues 226
7. Conclusions 231
8. References 232

9 **Highlighting a User Model to Respond to Misconceptions** 233
Kathleen F. McCoy

1. Introduction 233
2. Knowledge Available 235
3. Related Work on Correcting Misconceptions 237
4. Misclassifications 238
5. Misattributions 240
6. A Rule for Choosing a Strategy 241
7. Highlighting and Object Similarity 242
8. Object Perspective 245
9. Using Perspective to Set *f* 247
10. Modeling a Domain with Perspectives 248
11. Choosing the Active Perspective 251
12. Perspective's Influence on Responses 252
13. Conclusions 253
14. References 254

10 **But What Will the Listener Think? Belief Ascription and Image Maintenance in Dialog** 255
Anthony Jameson

1. Introduction 255
2. Situation 1: Generating an Informative Monolog 260
3. Situation 2: Positive or Negative Bias 269
4. Situation 3: Anticipating the Pragmatic Interpretation of Comments 271
5. Situation 4: Discrepancy Between Actual and Projected Biases 274
6. Situation 5: Responding to Specific Questions 278
7. Situation 6: Establishing a Desired Projected Bias 286
8. Situation 7: Discrepancy Between Actual and Projected Ascriptions 293
9. Situation 8: Uncertainty in the Listener About the Speaker's Ascriptions 299
10. Situation 9: Uncertainty in the Speaker About Her Projected Ascriptions 303
11. Conclusions 307
12. References 311

11 **Incorporating User Models into Expert Systems for Educational Diagnosis** 313
Robin Cohen and Marlene Jones

1. Introduction 313
2. Application Domain 314
3. Designing the System 316
4. The User Model Structure 320
5. Related Work 329
6. A General Proposal for User Modeling in Expert Systems 331
7. Current Status and Future Work 331
8. Conclusions 332
9. References 333

Part IV. Shortcomings of Current Models, Prospects for the Future

Introduction 336

12 **Realism About User Modeling** 341
Karen Sparck Jones

1. Introduction 341
2. What Is Being Modeled 342
3. What Modeling Is For 346
4. What Modeling Is From 352

5. Rational Principles for Modeling 357
6. Conclusion 360
7. References 363

13 User Models and Conversational Settings: Modeling the User's Wants 364
Katharina Morik

1. Introduction 364
2. Terminology 365
3. Conversational Settings 367
4. User Modeling in HAM-ANS 376
5. Conclusion 383
6. References 385

14 Student Modeling in Intelligent Tutoring Systems – Implications for User Modeling 386
Robert Kass

1. Introduction 386
2. Intelligent Tutoring Systems 387
3. Student Modeling 389
4. Four Intelligent Tutoring Systems That Model Students 393
5. Further Work in Student Modeling 403
6. Summary: Comparing User Modeling and Student Modeling 406
7. Conclusion 410
8. References 410

15 GUMS – A General User Modeling Shell 411
Timothy W. Finin

1. Introduction – The Need for User Modeling 411
2. What Kind of User Model? 412
3. A General User Modeling System 417
4. The Current GUMS System 428
5. The GUMS Command Language 429
6. Conclusions 430
7. References 430

Appendices

Bibliography 432

Subject Index 459

List of Contributors 468

Part I

A Survey of User Modeling

Introduction

The aim of Part I is to give a survey of current research on user modeling in artificial intelligence dialog systems. The first chapter introduces the field as a whole, and the two other contributions analyze an important modeling technique and the contents of user models, respectively. The chapters describe the aims and the scope of the field, and discuss influential systems that have been implemented, as well as important general results that have been obtained. Part I thus provides a good basis for the more technical chapters in Parts II and III, though all chapters are self-contained and do not presume that other contributions have already been read before.

An analysis of the field of user modeling from different points of view can also be found in Part IV, which is devoted to the discussion of shortcomings of current systems, and proposals for future research. Readers who do not wish to proceed too deeply into descriptions of particular systems are therefore encouraged to have a look at the final part after reading the introductory chapters.

The survey chapter by **Wolfgang Wahlster** and **Alfred Kobsa** is divided into five subsections. First, reasons are identified as to why user modeling has now become so important for artificial intelligence dialog systems. Definitions of the central notions in this field are given, and the relationship to related areas of research within and outside of artificial intelligence is outlined. Section 2 describes by what means and from what information sources user modeling components can construct a user model in the course of an ongoing dialog. Section 3 surveys the representation schemes that have been proposed to date for the representation of the system's assumptions about the user. Possible applications of user models are then extensively reviewed, and some social problems associated with practical applications are discussed. Finally, a number of open questions in user modeling are pointed out, and prospects for application-oriented research are considered. Throughout the chapter, references to the other contributions are made when appropriate in order to facilitate their classification within the field as a whole.

The chapter by **Elaine Rich** analyzes the stereotype approach, an important technique for user modeling that has been widely employed (e.g. [CHIN*, FINI*, MORI*]). A stereotype is a collection of user properties that often co-occur. In the stereotype approach, these properties are attributed to the user by a modeling component as soon as a limited number of preconditions are fulfilled. The great advantage of this technique stems from the fact that predictions concerning the user

can be acquired from an amount of evidence that is small enough to allow for easy acquisition *before* these predictions are needed.

As a matter of fact, Rich is certainly a suitable person to review the current position of this technique in user modeling, since she introduced it into the field a decade ago. In her present paper, she first defines the components of stereotypes and characterizes the types of user models that can profit the most from stereotype techniques for the acquisition of assumptions. The central issues of her chapter (and a strong contribution to the theoretical underpinnings of the stereotype approach) is a general discussion of *contradiction resolution* and of *stereotype adaptation*: Since the use of stereotypes constitutes a form of default reasoning, conflicts may arise among stereotypical assumptions from different sources, or between stereotypes and direct observations. Moreover, stereotypical predictions which were originally correct may become outdated. Rich discusses both a number of heuristics for resolving such conflicts, and more elaborate strategies which are related to the general methods of default and probabilistic reasoning as developed in AI research.

With stereotype adaptation techniques, a dialog system can exploit the experiences gained from the application of stereotypes to a number of users, and modify the original stereotypes. In practice, the integration of such techniques into any user modeling component which uses stereotypes is highly desirable, since the reason for choosing particular values for stereotype properties is often just the intuition of the designer.

Alfred Kobsa's contribution is concerned with an analysis of the *contents of user models*. He investigates the types of beliefs and goals that have been modeled by user modeling components developed to date (and adds some more which have not yet been, but ought to be modeled), and puts them into a taxonomical order. Apart from its theoretical contribution, such a taxonomy can be practically exploited for supporting designers of application systems in investigating what types of user beliefs and goals are relevant – and should therefore be modeled – in their respective domains, and for judging the expressive power of representation schemes for beliefs and goals. Using this taxonomy, Kobsa analyzes several major existing user modeling components with respect to what is being modeled by them. A striking difference is found between the number of modeled belief and goal types in those systems that employ the user model for explicit speech act recognition and dialog planning, and those which do not. This observation might have implications for the development of user modeling shell systems (see [FINI*]), which can presumably be much less complex if not developed for the former purpose.

1 User Models in Dialog Systems [1]

Wolfgang Wahlster and Alfred Kobsa

Abstract

This chapter surveys the field of user modeling in artificial intelligence dialog systems. First, reasons why user modeling has become so important in the last few years are pointed out, and definitions are proposed for the terms 'user model' and 'user modeling component'. Research within and outside of artificial intelligence which is related to user modeling in dialog systems is discussed. In Section 2, techniques for constructing user models in the course of a dialog are presented and, in Section 3, recent proposals for representing a wide range of assumptions about a user's beliefs and goals in a system's knowledge base are surveyed. Examples for the application of user models in systems developed to date are then given, and some social implications discussed. Finally, unsolved problems like coping with collective beliefs or resource-limited processes are investigated, and prospects for application-oriented research are outlined. Although the survey is restricted to user models in natural-language dialog systems, most of the concepts and methods discussed can be extended to AI dialog systems in general.

1. Introduction

1.1. Why User Models?

During the 1970s, many special-purpose natural-language (NL) *interfaces* were developed for various domains of discourse, e.g. moon rock samples, airline fares, computer installations, payroll data, aircraft maintenance data, or university courses. These systems had no interests beyond providing the information-seeking user with relevant data by just *responding* in a mechanically cooperative way to the user's questions. The conversational setting for such dialog systems was somewhat unnatural compared to human dialogs. The user of these systems had to find an appropriate

[1] This research was supported by the Special Collaborative Research Programme on Artificial Intelligence and Knowledge-Based Systems (SFB 314) of the German Science Foundation. We would like to thank G. Ferrari, Ch. Habel, M. Hecking, A. Jameson and an anonymous referee for their comments on earlier versions of this chapter. This is an extended and revised version of [WAHL86b].

question strategy for getting the information s/he thought might be relevant for solving his/her problem, and could hardly expect any assistance from the system in this respect.

In the 1980s, a new class of task-oriented dialog systems has emerged, all of which are mainly characterized by the *conversational roles* which the system and the user are expected to take on. Here, more realistic conversational settings are modeled in which the system may play the part of, for example,

- a clerk in an information booth at a train station (cf. [ALLE79]),
- a hotel manager who tries to rent all his rooms (HAM-ANS, cf. [JAME80]),
- a librarian who recommends novels that the user might like to read (GRUNDY, cf. [RICH79a, RICH79b]),
- a nosey guest at a cocktail party who wants to find out as much as possible about the user (ULLY, cf. [HAYE76]),
- an expert who assists an apprentice in repairing an air compressor (KAMP, cf. [APPE82b, APPE85]),
- a UNIX and a SINIX consultant for beginners (UC, cf. [WILE84, WILE86, CHIN*]; SC, cf. [KEMK86, NESS87, HECK88]),
- a tax advisor who assists the user in filling in his/her income tax form (XTRA, cf. [KOBS86b, ALLG89]).

These systems are designed as *active* dialog partners who engage in a mixed-initiative dialog [WAHL84]. In contrast to the NL interfaces mentioned above, the user of these systems needs no prepared question strategy since the system tries to recognize the user's *intention* with respect to the domain of discourse in order to exhibit more cooperative dialog behavior (e.g. in order to provide better advice). Although even in NL interfaces such a characteristic was necessary to a limited extent, it is particularly in the above conversational settings that the construction and use of an *explicit model of the user's beliefs, goals and plans* becomes a central problem.

Thus, one reason for the recent emphasis on user modeling is the fact that such models are necessary prerequisites in order for a system to be capable of exhibiting a wide range of cooperative dialog behavior (cf. also [CARN83c, RICH83, SPAR84, BOGU85, KASS88c]). A cooperative system must certainly take into account the user's goals and plans, his/her prior knowledge about a domain, as well as false conceptions a user may possibly have concerning the domain (empirical evidence for this in the case of human expert consultation can be found in [POLL82a]). Thus it is no longer only the user's task to construct a mental model of the technical functioning of the system. Instead, it should also be up to the system to form assumptions about what the user believes, wants and plans, i.e. to develop a model of the user. Ideally, neither the user's nor the system's modeling task should differ from that which is required in person-to-person communication.

A simple example for such cooperative dialog behavior might look as follows:

(1) User: Where can I find the nearest gas station?
System: The nearest one which is still open is located ...

In order to respond in this cooperative way, a system must discover the presumable plans underlying the user's question, represent them in its knowledge base, examine them for hidden obstacles, and provide information to enable the user to overcome these obstacles. A user model is an indispensable prerequisite for these complex inference processes.

A second reason for the recent emphasis on user modeling is that it has become evident in the last few years that a model of the user is also an important basis for intelligent dialog behavior in general, regardless of whether the dialog is cooperative or not. Such models are required for identifying the objects which the dialog partner is talking about, for analyzing a non-literal meaning and/or indirect speech acts in his/her dialog contributions, and for determining what effects a planned dialog contribution will have on the dialog partner, etc. (cf. [ALLE86, ALLE87]). Thus a user model does not just sweeten-up a dialog system to render it more cooperative. Rather, user models constitute an indispensable prerequisite for any flexible dialog in a wider domain. They interact closely with all other components of the system and often cannot easily be separated from them.

1.2. Some Basic Definitions

Although in this chapter we will restrict the discussion to user models in NL dialog systems, most of the concepts and methods discussed can be extended to AI dialog systems in general. Furthermore, many of the user modeling components developed for NL dialog systems accept or generate expressions of some formal representation language, since they belong to the language-independent part of the NL system. In this context, we will use the following basic definitions:

> A *user model* is a knowledge source in a natural-language dialog system which contains explicit assumptions on all aspects of the user that may be relevant to the dialog behavior of the system. These assumptions must be separable *by the system* from the rest of the system's knowledge.
>
> A *user modeling component* is that part of a dialog system whose function is to incrementally construct a user model; to store, update and delete entries; to maintain the consistency of the model; and to supply other components of the system with assumptions about the user.

We stress that user models cannot be simply defined as information that the system has about its users. Consider an NL interface to a database, which contains the relation EMPLOYEE (EMP#, NAME, AGE, BONUS) with (26, Jones, 32, 40) as one of its tuples. When Mr. Jones happens to be the user of such a system and asks 'What is my bonus?', his query is translated e.g. into (Select BONUS from EMPLOYEE where NAME = Jones), and the system responds '40'. In this case, the system has information about the user, but we would not like to say that its response was based on a user model, since the system does not view the database tuple as an explicit assumption about its current user (cf. [WAHL88]).

In many conversational situations, the system has to construct and exploit models of more than one agent. Imagine an input to a hypothetical 'marriage guidance

system' such as 'This is Bob Jones. My son has problems with his wife. She always .... But I think that my son ...'. In this case, the system's advice should be based on models of Bob Jones, his son, his daughter-in-law and the relationships between these agents. With the definition given above we propose to restrict the term 'user model' to assumptions about aspects of the human directly interacting with the system. In addition, however, we introduce the concept of an *agent model* (or 'actor' model in [KOBS85a]) as a superconcept for 'user model'. Thus, in the example discussed, there are three agent models, only one of which is a user model. For user modeling the system can evaluate the observed behavior of its dialog partner, whereas in the general case of agent modeling this is not necessarily possible. It follows that not all of the techniques developed for user modeling can be generalized to agent modeling.

1.3. User Modeling in Related Areas

1.3.1. User Modeling Within Artificial Intelligence Research

Within the area of artificial intelligence (AI), research on user modeling in NL dialog systems is related to several other fields, namely to the work on *intelligent computer-aided instruction (ICAI), multiple-agent planning systems, text comprehension and generation, intelligent help systems, game playing*, and *expert systems* (see Figure 1).

ICAI systems [SLEE82c, DEDE86] typically incorporate a "student model" which represents the student's understanding of the material to be taught. A simple modeling technique in this field is the overlay approach (e.g. [CARR77, CLAN79]), in which the student's knowledge is represented as a subset of the system's *knowledge* about the domain (cf. Section 3 and [KASS*]). Compared to NL dialog systems, however, the possible user's input to ICAI systems is usually very restricted. Sometimes only expressions in a formal language are accepted, such as mathematica formulas or statements in a programming language.

Research in the field of multiple-agent planning (e.g. [POWE79, KONO80, ROSE82, GEOR83, ROSE85, WERN88]) investigates the problems that a group of agents have in collectively constructing and/or executing a plan for a given task. To coordinate their actions, each agent has to represent the beliefs and goals of the others. Much of this work is grounded on the belief representations developed in the possible-world approach (see Section 3). Problems of natural-language communication between agents are not studied in this field of research. Moreover, for representational purposes, a number of idealizations are typically made with respect to the kinds of beliefs and goals which agents can hold. Sometimes agents cannot possess beliefs which are contradictory to other agents' beliefs (e.g. to those of the system), or cannot hold various forms of mutual and infinite-reflexive beliefs (see Section 2.3). Contradictory beliefs on the part of different agents, however, are quite common in man-computer interaction,[2] and agents which hold various forms of mutual or infinite-reflexive beliefs are practically ubiquitous in this field.

[2] A lot of work in the field of user modeling is devoted to recognizing user misconceptions, i.e. conflicts between system and user beliefs (e.g. [SCHU83b, MCCO*, QUIL*]).

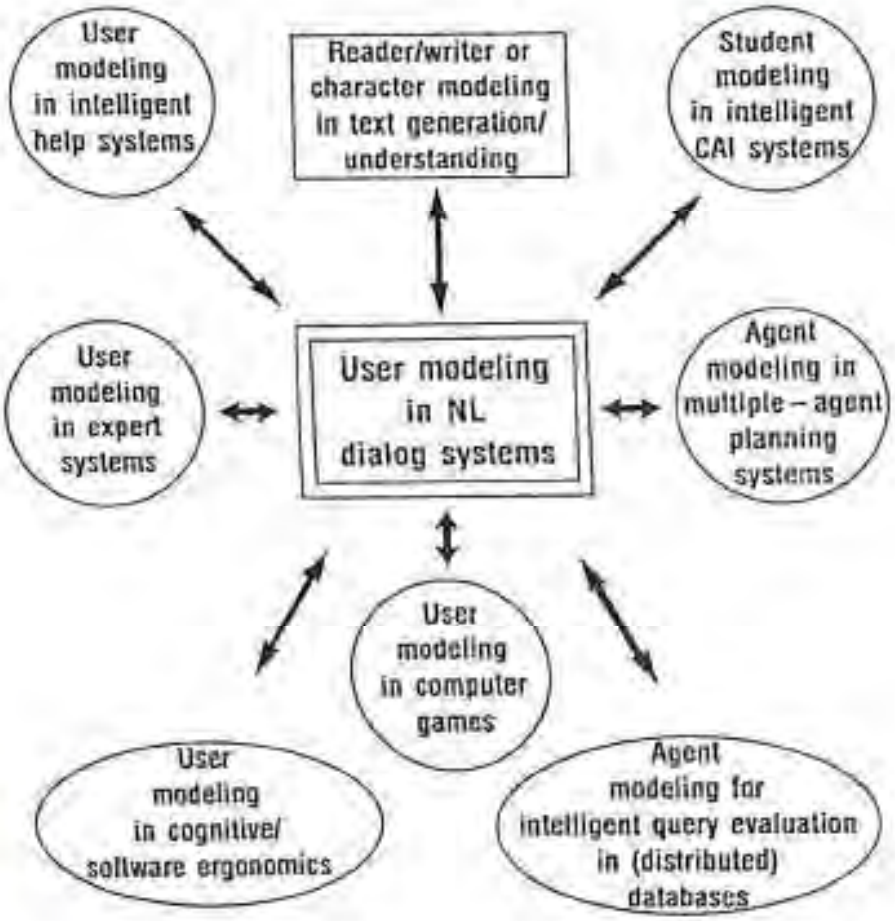

Figure 1. Relationships to other research areas

The adversary nature of two-person games like chess, checkers or backgammon makes game-playing another challenging domain for user modeling research, since in contrast to most other dialog situations, the best systems here behave in a most uncooperative way. It is clear that for brilliant play a model of the opponent's strengths and weaknesses as well as an understanding of his/her playing style is needed. Such a model of the opponent forms the basis for imagining the likely consequences of a planned move. Suppose that the system, being an end game expert in chess, believes that its opponent is a very weak end game player. In such a situation the system will not accept a proposed draw, even when at a disadvantage. Today, the best game-playing systems use an asymmetric evaluation function (see e.g. [HORA87]) and analyze the opponent's previous moves in order to construct a model of his playing strategy. Unfortunately, the user modeling techniques developed so far in this subfield of AI are extremely domain-dependent (they are mostly based on various numeric parameters and scoring functions), so that they cannot be transferred directly to other communicative situations.

1.3.2. User Modeling Outside of Artificial Intelligence Research

Outside of artificial intelligence, a growing interest in user models can be observed in the field of *Human-Computer-Interaction*[3] (for surveys see [CLOW85, MURR87a, MURR87b]). In HCI, however, the notion of 'user model' is employed in a much wider sense than in artificial intelligence research. On the most general level, user models are regarded as forming some kind of mental model [GENT83] which, according to current cognitive psychology, persons usually form of the things and people with whom they interact. Such mental models provide predictive and explanatory power for understanding the interaction. Entries in AI user models, by contrast, need not necessarily be psychologically relevant, i.e. correspond to "mental entities" of the user. They must only enable the system to reconstruct the user's beliefs and goals in sufficient detail.

The glossary in [NORM86b] discusses three different types of models that are called 'user models' in HCI research: "[...] the individual's own personal idiosyncratic model; the generalized 'typical user' model that the designer develops to help in the formulation of the design model; or the model that an intelligent program constructs of the person with which it is interacting." In artificial intelligence, the notion of 'user model' is only used in the last sense.

The aim of research in HCI is not necessarily to develop computer systems that construct a model of the user. Instead, knowledge about users' mental models should enable designers to build systems that can be more easily understood by users, or at least to predict learnability and performance. In order to make such models explicit, formal representation methods have recently been proposed for describing the user's and the system's knowledge about the task structure. These representations include state transition networks, formal grammars, and some sort of production systems. Perhaps closer convergence of research pertaining to user modeling in HCI and in AI dialog systems (which are currently nearly independent lines of research) can be achieved on a representational basis as soon as the HCI formalisms have been further developed.

It is worth noting that dialog-based user models can help to improve the performance of systems even for standard tasks like database retrieval. Botman, Kok and Siklossy [BOTM87] describe a so-called *active* database (ADB) system which extends the literal response to a user question by including extra information the user might be interested in, so that less interaction with the system will be necessary. In many retrieval situations a user is not able to give a precise and complete explication of his/her preferences at the beginning of the dialog. In the ADB system the user can indicate which items mentioned in the response to his/her original query s/he finds interesting. Using knowledge about similarities between database values or attributes, the system tries to extract the characteristics of the items upon which the user's choice was based. These assumptions about the user's interests are stored as user profiles together with a certainty factor in the user model. The profiles are used to compute individual interest values for all relevant tuples in the database, which are then used to select and rank information before presentation to the user.

[3] Related notions are 'Cognitive Ergonomics', 'Software Ergonomics', 'Cognitive Engineering', etc.

For example, when the user wants to see information about second-hand cars under $ 2,000.00 and then marks several of the displayed BMWs and Audis as interesting, the system may conclude that this choice was based on the 'make' and 'price' attributes, but not on 'color'. When updating the display, the system removes the 'color' attribute and shows more BMWs and Audis, but also includes Alfa-Romeos, because this make was often considered similar to BMWs in previous dialogs.

2. Constructing User Models

In this section, a number of knowledge sources will be identified from which information about the user can be obtained, and a number of techniques for acquiring a user model in the course of a dialog will be discussed.

2.1. Default Assumptions

In systems with a limited area of application, various beliefs and goals can be attributed to any user of the system, as long as there is no evidence to the contrary. Such default assumptions made by the system may concern, for example,

- *The user's general knowledge*: A management information system (MIS) with a user modeling component, for instance, can assume that all of its users possess basic knowledge in business administration.
- *The user's beliefs about individual facts*: An MIS installed in a firm can also assume that the user is familiar with the principal facts concerning the firm.
- *The user's goals*: All users of a hotel reservation system (cf. [JAME80, HOEP83b]) can be assumed to be searching for rooms which meet their requirements.

Other possible default assumptions concern the user's beliefs about the system's goals (e.g. that the system wants to rent its rooms), or assumptions about the user's beliefs about the system's beliefs, etc.

2.2. Initial Individual User Models

One criterion which helps to differentiate between the various dialog systems that include a user modeling component is whether

(A) The system can, at the beginning of a dialog, access an already existing model of a particular user which it had constructed during previous interactions with him/her.

(B) The system possesses no previously acquired information about individual users at the beginning of the dialog.

In the former case, the recorded user model will probably prove to be a valuable source of information for the current dialog as well. An example of a system with

such an initial user model is GRUNDY [RICH79a, RICH79b] which, at the end of a dialog session, records all information about the user inferred from his/her dialog behavior in a corresponding file. Whenever that user returns to the system and types in his/her name, this file will be retrieved and used as the initial user model.

Today, however, the latter type of system is developed and implemented much more often than the former. This is due not only to the legal, social and ethical problems of security and privacy in such systems, but also to the fact that for many prospective applications in the current era of distributed systems the probability that a user will consult the same system more than once is quite low.

2.3. Assumptions Based on the User's Input

Assumptions formed on the basis of the user's input into the system are the most direct and, hence, also the most reliable in problem-solving domains.

There are various techniques for transforming a user's input into assumptions about him which differ in complexity and in the reliability of the inferred assumptions. The simplest type occurs if the user directly expresses a belief or goal, as in

(2) User: I want to buy your new 500 XGL.
Assumption of the system: The user wants to buy one 500 XGL.

Unfortunately, such direct statements of the user about his/her beliefs or goals are usually very rare. Much more frequent are cases in which assumptions about the user have to be indirectly inferred from the user's input. One type of such inferences is purely *structure-based*. In this case, only the form of a user's input is used for drawing inferences about him/her, and not its meaning. The VIE-DPM system, for instance, forms the following assumptions based on a user's wh-question, and enters them into its knowledge base [KOBS85d, KOBS86a].

(3) User: To whom does Peter give the book?
Assumptions of the system:
(a) The user believes that Peter and the book exist and that Peter gives the book to somebody whose identity is unknown to the user.
(b) The user believes that the system knows to whom Peter gives the book.
(c) The user wants to be in a situation where
(*) both the user and the system have the same opinion about whom Peter gives the book to, and believe that (*)
(d) The user believes that the system believes that (a) – (d).

The infinite loops in the above definition describe specific belief constellations which are fundamental in dialog settings, namely *mutual beliefs* (or more precisely, *infinite-reflexive beliefs* – cf. [KOBS84]). All of the above assumptions can be formed simply because the user has entered a wh-question, without regard to the content of his/her question.

Other purely structural rules of inference (which were used, for example, in [ALLE80, CARB83, CARB*, SIDN83b, MCKE85c]) include

(4) If the user wants to know how to achieve an effect, then his/her goal may be to achieve that effect.

(5) If the user wants to know whether p, then his/her goal may be that p, or not p.

Other forms of inference depend on the *meaning* of a user's input to the system, and on a large amount of universal knowledge about the domain of discourse. From (3), for instance, the system could also infer the following:

(3)(e) Assumption of the system: The user believes that the book is now in the possession of the person whose identity is unknown to him/her.

This inference is based upon the content of the user's question, upon world knowledge about the meaning of 'to give' (namely that it denotes an action, the result of which being the possession of an object by the recipient), and the general inference rule

(6) If the user believes that an action occurred, then s/he also believes that the effects of the action occurred.

[KASS87a] proposes a variety of such general rules for acquiring additional assumptions about the user. Examples include:

Action rule: If the user model includes the belief that a user is familiar with an action, then the user modeling module can attribute to the user knowledge of the preconditions and effects of that action.

Concept generalization rule: If the user model indicates that the user knows several concepts that are specializations of a common, more general concept in the domain model, the user modeling module may conclude that the user knows the more general concept, and the subsumption relationship between the more general concept and the more specialized concept as well.

Inheritance rule: If the user believes a concept A has property P, and further believes that A subsumes concept B, then the user believes B has property P.

Assumptions from the user's input can also be used for inferring his/her possible plans. The systems of Allen & Perrault [ALLE80], Carberry [CARB83, CARB*] and McKeown et al. [MCKE85c] have a large number of domain-dependent precondition-effect sequences (i.e., operators) stored in their knowledge bases. They can then use inference rules such as

(7) If the user's goal is to achieve certain effects, then s/he will probably use a sequence of operators which yield this effect and whose preconditions are fulfilled

for inferring probable user plans. In all these models, however, the system and the user hold identical beliefs about these action operators (e.g. about their preconditions). This restriction is relaxed in the work of Pollack [POLL86a, POLL86b].

In contrast to the other systems, however, Pollack's model does not allow for the chaining of action operators.

Another type of domain-dependent inference is that by which, on the basis of "objective" facts about the user, assumptions are formed about his/her probable traits, in particular his/her preferences, requirements and criteria of assessment in a certain domain. Rich's [RICH79a, RICH79b] GRUNDY system, for instance, forms assumptions about the user's selection criteria for books to read from information about personality characteristics supplied by him/her. Morik & Rollinger [MORI85b] form assumptions about the user's assessment criteria for apartments from information about his/her personal situation, such as the number of children, the acceptable price range, etc. If a great number of inferences are drawn from an antecedent (as is in the purest form the case in GRUNDY), then the set of their consequents may be regarded as a *stereotype* (see Section 4.2 and [RICH*]).

Furthermore, the interpretation of *modal verbs* like 'must, should, need, may' in the user's input makes it possible to infer some of his/her wants and beliefs. Unfortunately, modals are ambiguous in most languages. For example, in German most modals can be used for expressing a want as well as a belief. Sprenger [SPRE88] has developed a computational model for disambiguating modals, so that they can be mapped onto modal operators in the semantic representation language of a user model which is integrated in the consultation system WISBER [SPRE87]. There are many technical problems relating to how a modal verb expresses a belief or goal. Even the correct interpretation of a modal like 'want' in (2) can be quite complex in other contexts. In (8),

(8) User: I want to buy a Porsche, because I would like to have a very fast car.

the user expresses the *belief* that buying a Porsche results in having a very fast car and the *goal* of having a very fast car. In (9), the clause

(9) User: I wanted to buy a BMW, but then I heard that a Porsche is even faster.

beginning with 'but' should cause the user modeling component to assume that the user now wants to buy a Porsche.

Additional assumptions about the user can be formed from *linguistic particles* in his/her input, as in

(10) User (to a text editor system): *Why* is this line *not* centered?
Assumption of the system: The user wants the line to be centered.

(11) User: I do *not* have *enough* disk space.
Assumption of the system: The user wants more disk space.

Up to now, the analysis of linguistic particles in the user's input has mostly been investigated on a theoretical basis [KOBS85a]. Such an analysis would offer a simple way of obtaining assumptions about the user. It seems, however, that interesting particles are not very frequently used in goal-oriented dialogs. Moreover, in contrast to the techniques discussed above, this method is also only applicable in natural-language dialog systems.

2.4. Assumptions Based on Dialog Contributions Made by the System

Dialog contributions of the system (i.e. answers, questions and commands) also lead to new entries in the user model. If the system had previously planned the dialog contribution by trying to anticipate its probable effect on the user, it can now hypothetically enter these effects into the user model (e.g. after answering a user's question the system can assume that the user now knows the content of the answer). These entries may give rise to the planning of additional unsolicited comments (cf. Section 4.3.3 and [JAME*]) or follow-up information and explanation.

2.5. Assumptions Based on Non-Linguistic Input of the User

In Section 2.3, a number of techniques were discussed for forming assumptions about the user based on his/her input into the system. In contrast to a face-to-face conversation between humans, present dialog systems are restricted to deriving such assumptions solely from the strings typed in by the user; they cannot access any visual, acoustic or other external clues.

By expanding Norman's futuristic idea of a personal user interface on a pocket-sized "smart" card, which, when plugged into any terminal, makes it emulate the one at the user's home or office (cf. [JOYC83]), one could imagine a slight deviation from that restriction, namely, that a system's user model is initialized by a kind of electronic identity card storing some of its owner's personality characteristics. The information on this card may be regarded as an initial individual user model in the sense of Section 2.2. The information thereon would be more general, however, and not restricted to the specific domain of a particular system.

At the beginning of an interaction with a dialog system, the use of such a device containing a user-provided individual profile seems no more unnatural than responding to a system's request like

(12) I'd like to know what sort of person you think you are. Please type in a few single words that you think characterize the most important aspects of yourself

as is required in Rich's [RICH79a, RICH79b] GRUNDY system. Of course, the problem remains as to whether the conveniences offered by such personality cards are not far outweighed by the dangers they give rise to in respect to loss of privacy and easier surveillance.

3. Representing User Models

In the field of user modeling, a wide range of formal means are employed for representing the assumptions of the dialog system regarding the user. What representation one opts for is usually determined by the application purpose of the system and by the demands imposed on the expressive power of the user model.

Rich's GRUNDY system, for instance, makes extensive use of simple *linear parameters*. For every personality trait in her system, one parameter indicates the assumed degree to which the personality trait characterizes the user, and another indicates the degree to which the system believes that its assumption about this specific personality trait is correct. An example of such personality traits is given in Figure 2.

Personality trait	Value	Certainty of assumption
Education	5	900
Seriousness	5	800
Piety	-3	423
Tolerate-sex	5	700
Tolerate-violence	-5	597
Tolerate-suffering	-5	597

Figure 2. Representation in the GRUNDY system

Sleeman's UMFE system [SLEE85] also uses very simple methods for representing its assumptions about the expertise of a user in a given domain. UMFE is a front end system for the explanation component of expert systems. It takes the inference chains yielded by such components and adapts them to the specific level of expertise of the current user. This is done by making sure that only those concepts are used in an explanation which the user has explicitly indicated familiarity with, or which the system has assumed him to know.

For representing the concepts that a user is probably familiar with, a sort o *overlay technique* is employed in Sleeman's model (also see [KASS*]). A simple parameter associated with the representation of each individual concept in the system's knowledge base indicates whether, according to the system's assumptions, the concept is KNOWN or NOT-KNOWN by the user, or whether the system has NO-INFORMATION in that respect. Another factor associated with each concept indicates the strength of belief with which the system holds its assumptions about the user's familiarity with these concepts. Currently, however, no use is yet made of these factors. An example of a simple overlay model (which is inspired by the UMFE system) is given in Figure 3.

As soon as the user model is to give a slightly more detailed account of what the user knows or does not know, of his/her misconceptions, or of his/her goals and plans, etc., the linear scale values must be replaced by representational structures with greater expressive power. In natural-language dialog systems, for representing knowledge about the *world*, representational schemes developed in the field of knowledge representation are employed. Such schemes are, for instance, formulas of first-order predicate calculus, network-like representational schemes, frames, etc. It thus seems most natural to extend these schemes so that the system's assumptions about the beliefs, goals and plans of the user can be represented.

Unfortunately, a simple extension is not possible. When using predicate calculus, one has to decide by what epistemological primitives notions like 'believe',

Concept hierarchy of the system	User model	
	user's knowledge state	certainty of assumption
INFECTIOUS-PROCESS	KNOWN	100
HEAD-INFECTION	KNOWN	100
SUBDURAL-INFECTION	NOT-KNOWN	100
OTITIS-MEDIA	NO-INFORMATION	100
SINUSITIS	NO-INFORMATION	100
MENINGITIS	KNOWN	100
BACTERIAL-MENINGITIS	KNOWN	90
MYCOBACTERIUM-TB	NOT-KNOWN	70
VIRUS	NOT-KNOWN	90
FUNGAL-MENINGITIS	NOT-KNOWN	100
MYCOTIC-INFECTION	NOT-KNOWN	100
ENCEPHALITIS	NOT-KNOWN	90
SKIN-INFECTION	KNOWN	100

Figure 3. An example of an overlay model

'want', etc. should be represented. Opting to regard them as sentential operators (as are, for example $\vee$, $\wedge$, $\sim$ or $\supset$) causes serious problems, since these operators would not be extensional (an operator is extensional [or truth-functional] if the truth value of a complex formula constructed by it depends solely on the truth value, and not on the "meaning", of its constituent formulas). The lack of this property causes serious problems because fundamental principles of predicate logic (as, for example, substitution of terms with same extension, existential generalization) are no longer applicable.

Another possibility is to regard these notions as meta-language predicates which are applied to meta-language constants and variables that denote object-language formulas. These formulas, in return, express the content of the beliefs and goals. The system's assumption that the user believes that Peter loves Mary thus would be represented in this approach by something like 'BEL(USER,P)', where 'BEL' is a meta-language predicate, 'USER' a meta-language constant that denotes the user, and 'P' a meta-language constant that denotes an object-language formula like 'loves(peter,mary)'. Konolige [KONO81] developed a logic for belief representation that is based on this idea. For more complex belief constellations, however, such representations become computationally untenable. Moreover, as has been shown in [KOBS85a], a number of beliefs that are fundamental for dialog planning based on a user model cannot be represented in Konolige's scheme.

Another approach is to regard notions like 'believe' and 'want' as modal operators, and to use one of the many modal calculi for their axiomatization. This idea was first developed in [HINT62]. Moore [MOOR80] and Appelt [APPE85] investigated a belief and goal representation which formalizes the standard semantics of modal logic, namely *possible-world semantics*. In their representation, modal formulas are immediately translated into possible-world sentences, to which the

standard deduction techniques can be applied. The modal operators 'believe' and 'want' are thereby translated into accessibility relations between possible worlds. An agent (e.g. the user) knows some fact p if p is true in all worlds which are compatible with what s/he knows.

The system's assumption that Peter loves Mary thus would be represented in this approach by a formula like

(13) $\forall \mathcal{W}\ [K(USER, W0, \mathcal{W}) \supset T(\mathcal{W}, P)]$

'$\mathcal{W}$' thereby is a meta-language variable ranging over possible worlds, 'W0' a meta-language constant denoting the world in which the user currently finds himself, and 'P' a meta-language constant denoting some object-language formula like 'loves (peter, mary)'.

'K (USER, W0, $\mathcal{W}$)' is true iff $\mathcal{W}$ is accessible from W0 according to what the user knows. 'T($\mathcal{W}$, P)' is true iff the formula denoted by P is true in $\mathcal{W}$. Such a situation is depicted in Figure 4. Here the user knows the fact expressed by 'loves(peter, mary)' but is uncertain with respect to the fact expressed by 'loves(mary, john)', since there is a world which is accessible to him according to what he knows in which the negation of the latter formula is true.

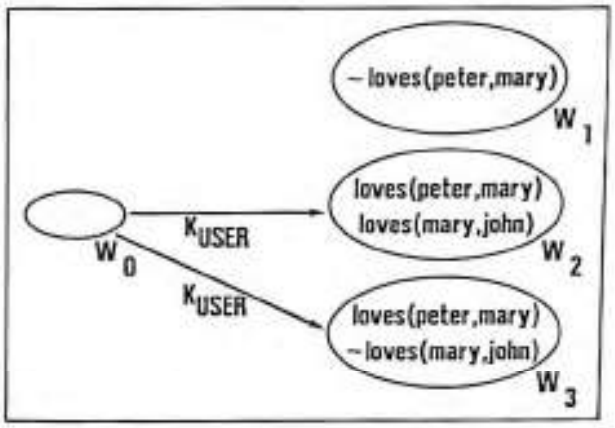

Figure 4. A possible-world representation of 'The user knows that Peter loves Mary and is uncertain whether Mary loves John'

Possible-world semantics has recently been intensively investigated as a means for belief representation [HALP85a, FAGI85, VARD85]. However, as was pointed out in [KOBS85a], as far as the representational needs in the field of user modeling are concerned, the expressive power of this approach is somewhat limited.

Another approach to belief and goal representation is the "*partition approach*", which is applicable to all forms of concept-based knowledge representations. It was first used by Cohen [COHE78] for his special network-oriented representation. The idea behind this approach is to maintain a number of separate partitions to store the system's beliefs about the domain, the system's goals, the system's assumptions about the user's beliefs about the domain, the system's assumptions about the user's goals, the system's assumptions about the user's assumptions about the system's beliefs about the domain, etc. For representing the contents of these beliefs and goals, the standard concept-based knowledge representation schemes (e.g. network-like

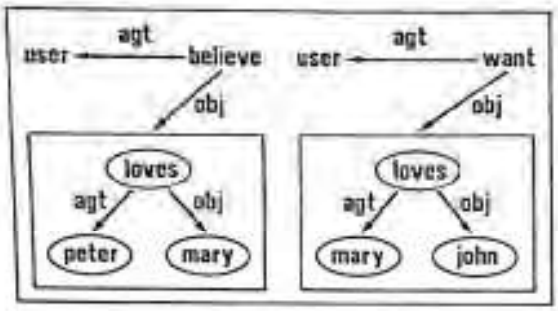

Figure 5. A partition representation of 'The user believes that Peter loves Mary and wants Mary to love John'

representations, predicate calculus) may then be employed. An example of such a representation with a partition for the system's beliefs and another for the system's assumptions about the user's goals is given in Figure 5.

Unfortunately, in the form described here, this approach raises some problems which were spelled out, for example, in [MOOR80, SOUL84, KOBS85a]. In [KOBS85d], a new interpretation is therefore given for this sort of belief and want representation, and new epistemological primitives (the so-called '*acceptance attitudes*') are added to increase its expressive power for more subtle belief and goal types. In the BGP-MS system [KOBS88, ALLG89], this representation scheme is currently enhanced by *partition inheritance* which, for instance, allows the user's and system's mutual beliefs to be stored in a separate partition, the contents of which are inherited by the partitions containing the system's beliefs about the domain and the system's assumptions about the user's beliefs about the domain. Kass [KASS87a] plans to extend this representation in such a way that acceptance attitudes can also be applied to rules, and that beliefs also carry justifications for why the belief is held or not held.

4. Exploiting User Models

4.1. Exploitation Outside of the System

In most systems, user models are applied as aids which help the system in pursuing a particular set of dialog goals or in behaving cooperatively when performing a task together with the user. However, there are conversational settings where the construction of a user model is itself the *central* dialog goal (as is the case with, for example, exams or personnel interviews) and not just a means to an end (in these cases, the exploitation of the user model is left to human interpreters).

An early proposal for such a distinguished function of a user model in an AI system was the NL dialog system ULLY (cf. [HAYE76]). The conversational setting for ULLY was a fictitious cocktail party at which both the user and the program were "guests". The main goal of ULLY was to find out as much as possible about the user. ULLY was designed "to be a friendly and reasonably co-operative conversational partner, who is politely nosey" (cf. [HAYE76], p.138).

The authors assumed that at the beginning of the conversation, ULLY has a certain amount of knowledge about the other party guests, such as their names, what they do, how they are related to each other (e.g. X detests Y), but does not know anything about the user. At the end of the session, ULLY should have a more or less refined idea of what the user is like.

ULLY has never been implemented, since – we think – some of the project's goals were overambitious considering the primitive state of the art of user modeling at that time. Although we have seen some advances in user modeling during the last ten years, even today some of ULLY's maxims point to areas open to further research, for example

- try to increase the level of intimacy without being impolite.
- try to keep the conversation at a level which is understandable.
- treat assertions by the user with "care".

4.2. The Relevance of User Modeling in Various Applications of Natural-Language Dialog Systems

If a system is designed as a cooperative consultant who actively recommends X (e.g. a book in GRUNDY or a hotel room in HAM-ANS) or advises the user against X, a model of the user is a necessary prerequisite for performing the corresponding speech acts (cf. [MORI84, MORI*]). A dialog system which generates only fixed and biased recommendations, on the other hand (e.g. a version of GRUNDY which always suggests the Bible first) has hardly any need for a user model. The following types of conversational behavior relevant to AI applications are listed according to an increasing need for a user model:

(A) question-answering and biased consultation
(B) cooperative question-answering
(C) cooperative consultation
(D) biased consultation pretending objectivity.

A system that responds to the literal meaning of simple questions about a flight schedule or a hotel reservation system which always offers the most expensive room first are examples of category (A). Such systems must possess limited user models only, e.g. for handling discourse phenomena like anaphoric reference and ellipsis appropriately. While all systems of this category implemented to date are simple question-answering systems, it is highly probable that biased systems will be installed as soon as the technology of NL processing becomes further commercialized. The focus of current research is shifting from systems of category (B) (e.g. CO-OP, cf. [KAPS79, KAPS82, KAPS83]) to category (C) (e.g. ARGOT, cf. [ALLE82a, ALLE83b]; UC, cf. [WILE84, WILE86]; XTRA, cf. [ALLG89, COHE*]). Systems of both of these categories include user modeling techniques for the generation of various kinds of over-answering (cf. [WAHL83]) and for the generation and recognition of (indirect) speech acts (cf. [ALLE79, PERR80, APPE82b, APPE85]). Systems

of category (D), which mimic for example professional salesmanship, require a very sophisticated form of user modeling. The experimental dialog system IMP, which will be discussed below, includes some aspects of that kind of dialog behavior.

As was pointed out by Rich [RICH83], user modeling is especially important if the dialog system is used by a heterogeneous group of users and if the system shows some flexibility in what it tells the user. A typical example of this is her GRUNDY system which, in playing the part of a librarian, can select from a large number of books one that it assumes the user might like to read. The selection is made on the basis of a model of the particular user which the system constructs in the course of the dialog.

Another example of a dialog system for a heterogeneous user group is HAM-ANS which, in its hotel reservation application, plays the part of a hotel manager who tries to rent all rooms available in his/her hotel. On the basis of an explicit model of the individual user, the system can select one of four room categories and suggest a vacant room in the selected category. In this conversational setting the user is assumed to have the overall goal of getting a hotel room which meets his/her requirements.

From a greeting like that in Figure 6 (which includes the name and optionally the title, the affiliation and the home or office address of the user) and from further booking information like that provided by the user in Figure 6, the system can form some assumptions about the prospective hotel guest, e.g. his profession, the purpose of his trip (business trip, vacation, etc.) and his solvency. Several stereotypes are associated with each of these dimensions.

As in GRUNDY, stereotypes are used as collections of frequently recurring characteristics of users. In the example above, HAM-ANS may assume that Mayer is Mueller's secretary and that Mueller is a manager in a high-tech company ("Sunshine Computers"), since he holds a Ph.D. and has a secretary. Furthermore, the system assumes that Mueller is on a business trip and, because his expenses are reimbursed, he is able to spend a lot of money for a room. The user-provided information relevant for the consulting procedure is stored as factual knowledge in the user model, whereas the stereotypes triggered by these facts are stored as hypotheses, which can be withdrawn if they are disconfirmed by the subsequent dialog. Note incidentally that if the caller is booking a room for someone else, the model discussed here does not contain information about the actual user of the system, but about the prospective hotel guest instead. Hence, according to the terminology introduced, this is a case of agent modeling in general, and not specifically user modeling.[4] One can imagine future dialog systems which also model the intermediary persons (e.g. for also taking into account private interests of secretaries), but this is beyond the scope of our discussion.

HAM-ANS can characterize a particular hotel guest by several distinct stereotypes. For example, its assumptions about a user named Schmidt may be that he is a university professor who wants to spend his vacation with his family, and that he is not able to spend much money because he has five children. As soon as the system

[4] Sparck Jones [SPAR84, SPAR*] calls the person being talked about in the discourse the *patient*, and the person interacting with the system (i.e. the user in our terminology) the *agent*.

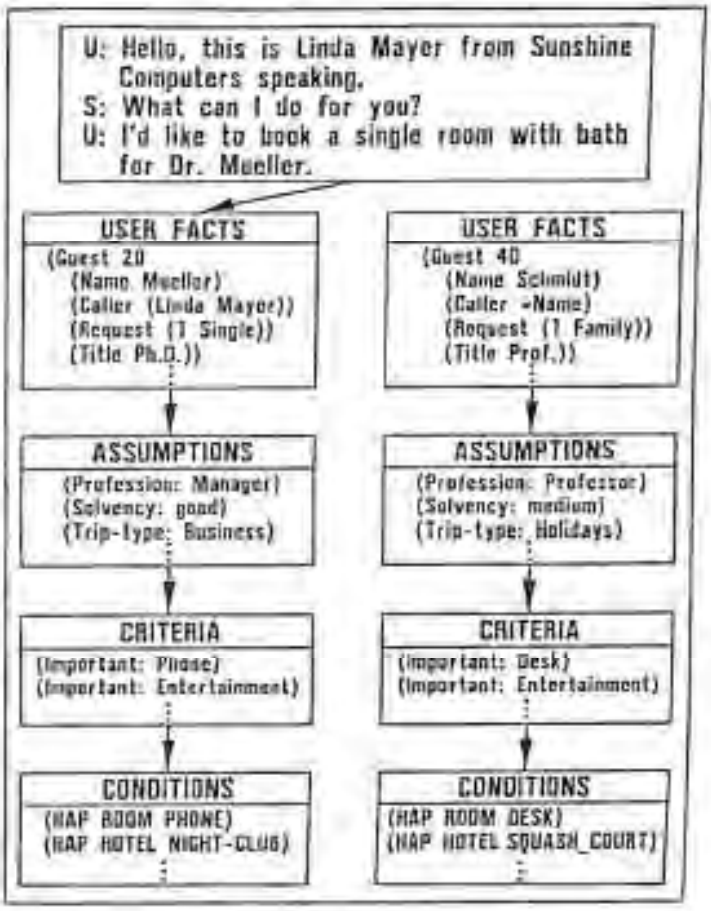

Figure 6. The use of stereotypes in HAM-ANS

has all necessary booking facts at its disposal, it checks whether there is a vacan room which meets the requirements explicitly stated by the user (e.g. period of stay, number of beds). If several rooms belonging to different categories meet these explicit requirements, the system uses the activated stereotypes to predict further demands of the user. For example, HAM-ANS may assume that for Mueller, as a manager on a business trip, a room telephone is very important. Prof. Schmidt on the other hand, being a scientist, actually needs a large desk even on his vacation, but in this particular situation a room telephone is not so important for him. These stereotypical inferences generate a list of criteria rated by different importance values. The set of derived criteria is matched against the available room categories in order to generate a strong or weak recommendation and to indicate which criteria are fulfilled and which are not. However, since individuals differ in the conditions they impose for the fulfillment of a criterion, this matching process again depends on the user profile (cf. [MORI85a,MORI85b]). For example, if the guest plans to stay for a longer period in the hotel, attractive entertainment facilities become very important. This criterion, of course, is evaluated differently by a businessman who travels by himself compared with a professor who spends his vacation together with his family.

4.3. Anticipation Feedback

A special case of exploitation of a user model (which is realized in dialog systems like HAM-ANS, NAOS and IMP) is its use in various forms of *anticipation feedback loops*. Anticipation feedback loops involve the use of the system's language analysis and interpretation components to simulate the user's interpretation of an utterance which the system plans to verbalize. The application of anticipation feedback loops is based on the implicit assumption that the system's procedures for language analysis and interpretation (but not necessarily the contents of the sources of knowledge used by these procedures) are similar to those of the user. Such a similarity assumption seems to be quite common among participants in everyday conversations. Since most people (except, of course, some computational linguists and AI researchers) cannot be assumed to have an exact model of the comprehension capacity of their dialog partners (except for some naive assumptions concerning small children and non-native speakers) it is highly plausible that they use the similarity assumption as a default rule. Although the current state of the art in NL dialog systems in no way provides sufficient evidence for assuming a far-reaching similarity, user modeling based on anticipation feedback loops can considerably improve the quality of present dialog systems.

4.3.1. Local Anticipation Feedback for Ellipsis Generation

The ellipsis generation component of the dialog system HAM-ANS incorporates a local anticipation feedback loop which helps to ensure that the system's utterances are not so brief as to be ambiguous or misleading (cf. [JAME82]). Consider for example the user's question (14).

(14) User: Do five colleagues have more than three weeks vacation?

If five colleagues have in fact not more than three, but more than two weeks vacation or if only two colleagues have more than three weeks vacation, a cooperative system might respond with (15), instead of simply answering 'No'.

(15) System: No, two.

By using its ellipsis resolution component in an anticipation feedback loop, the system can recognize that (15) would be a confusing response since it is so elliptical that, without further context, no unambiguous interpretation can be assigned to it. Acceptable responses like (16) and (17) include more overlap with question (14).

(16) System: No, two weeks.

(17) System: No, two colleagues.

Having rejected (15), the system internally generates the less elliptical responses (16) or (17), depending on the intended meaning. This time, the anticipation of the user's understanding causes a positive feedback, so that the elliptical structure can now be mapped onto a well-formed NL utterance by the surface transformation component of the system (for a more complete account of HAM-ANS's user modeling in ellipsis generation cf. [JAME82]).

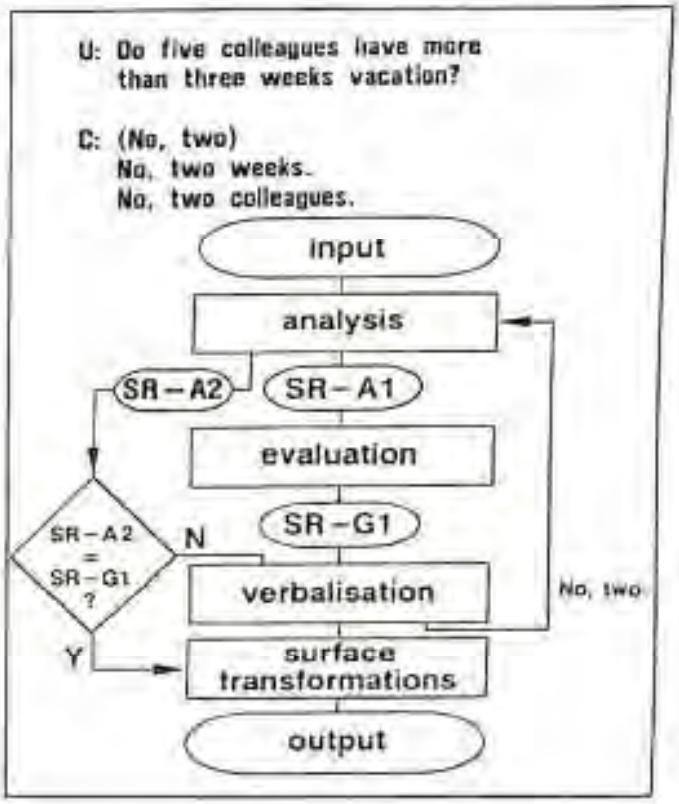

Figure 7. A simple anticipation feedback loop

Figure 7 shows an extremely simplified version of the general structure of a dialog system with such an anticipation feedback loop for user modeling. The parsing of an input results in the semantic representation construction SR-A1, whose evaluation yields SR-G1 as the semantic representation of the intended response. The result of a preliminary verbalization process for SR-G1 (which includes the above-mentioned ellipsis generation) is fed back into the system's analysis component, which is equipped with an ellipsis resolution component. The semantic representation construction SR-A2, which forms the result of the ellipsis resolution process (and which may consist of a disjunction if the input is ambiguous) is then compared with the intended meaning of the response, namely SR-G1. If SR-A2 matches SR-G1, the structure fed into the analysis component is transformed and finally outputted, otherwise an alternative verbalization of SR-G1 is checked in the next iteration of the feedback process.

The anticipation process discussed above is based on a local feedback loop, i.e. one in which the generation procedures of one particular aspect of an utterance are linked with the recognition procedures for the same aspect.

A variety of phenomena, such as the generation of definite descriptions, pronouns, deictic expressions and noun-noun modifications can be handled in a dialog system either by a corresponding multiplicity of local feedback loops (cf. [WAHL83, MARB83]) or by a larger one within which the whole generated utterance would be tested once, just before applying the final surface transformations (see Figure 8).

Such global anticipation feedback loops require dependency-directed backtracking, so that the system returns to the adequate decision point in the generation process whenever the result of the feedback process is negative.

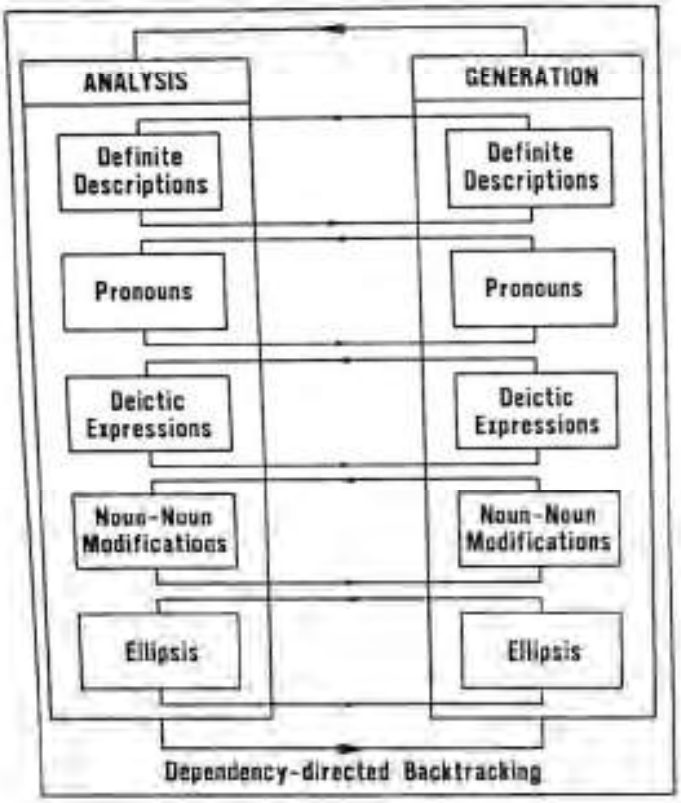

Figure 8. Local and global anticipation feedback loops

4.3.2. A Global Anticipation Feedback Loop

An example of a more global anticipation feedback loop is the imagination component of the system NAOS, which generates NL descriptions of real-world image sequences. [NOVA84] describes the design of a user model for NAOS, which contains a representation of the imagined scene resulting from an anticipation of the user's understanding of a planned utterance. Such a system would take into account the present state of the dialog and the assumed knowledge and dialog goals of the user. On this basis, the representation of the imagined scene (which always contains some uncertainty) and the actual scene observed by the system must be matched in order to check whether the planned utterance will be interpreted by the user in a consistent manner and whether it will generate the intended perceptual impression in adequate detail. For instance, it is quite clear that the image sequence of a car accident in which one was involved is described differently to the police, the lawyer of the opposing party, or a good friend.

This application shows that a user model cannot completely rely on representation schemes only suited for the representation of discrete domains. Instead, they

must also provide means for the representation of continuous domains, e.g. of the fuzzy regions of an object's trajectory which might form part of the image anticipated by the speaker for a planned utterance like (18).

(18) A red BMW passed a small VW.

4.3.3. Anticipation Feedback in Evaluation-Oriented Discourse

Jameson [JAME83, JAME*] has studied a very advanced form of anticipation feedback for user modeling in his IMP system. IMP was designed to answer the questions of a user whose sole dialog goal was assumed to be the assessment of some object. On the basis of explicit assumptions about the user's standards and prior expectations, IMP goes beyond the literal answering of questions by adding unsolicited comments that influence the evaluation of the object under discussion. In evaluation-oriented dialogs (e.g. personnel selection interviews, dialogs with travel agents, hotel managers or other salespeople) the selection of such comments is influenced by the nature of the speaker's bias (e.g. a desire to present something in a favorable light). Since listeners take these facts into account when interpreting such comments, IMP anticipates the interpretation of the user while taking into account the bias which the user possibly ascribes to it. IMP's actual bias can be positive, negative or objective, and it can attempt to maintain a positive, negative or objective image. The actual and projected bias may differ (e.g. usually a salesman's actual bias is positive though he attempts to maintain a neutral image).

On the representational level, IMP's user model includes an explicit record of the user's expectations concerning a particular room offered for rent. This record is updated according to the user's anticipated interpretation of the system's answer. The most interesting procedural aspect of IMP's user modeling is an anticipation feedback loop which takes into account the effect of a planned utterance on the user's impression of the room and helps to filter out comments which would betray the system's bias.

IMP's user model is also used to select expressions like 'but', 'by the way' and 'unfortunately', which precede the additional comments generated by the system. An example is given in (19):

(19) User: What about facilities outside the room?
System: There's a kitchen and there's a bathtub and by the way, the room is very large.
User: Is the room on a quiet street?
System: No, in fact unfortunately there's a lot of noise from the street.
([JAME83], p.618)

For selecting such connectives and sentential adverbs the system compares the shift in the expected value produced by the present comment with that produced by the preceding statement. IMP shows that for an explanation of the complex semantics and pragmatics of such expressions a user model is necessary.

As we have shown, the use of an anticipation feedback loop is an important part of the process of designing a successful utterance. In essence, anticipation on the part of the speaker means answering question (20) (cf. [ROSN81]).

(20) If *I* heard this utterance relative to a set of beliefs B and goals G, where B and G are what *I* assume the hearer believes and wants, respectively, then what would be the effect on me?

It is obvious that a dual question like (21) is useful for the hearer's analysis of utterances.

(21) If *I* had made this statement, on the basis of what beliefs and goals would I have said it?

4.4. Correcting User Misconceptions

In the last few years, a number of techniques have been developed for recognizing misconceptions underlying unsuitable dialog contributions of the user, and for responding to these dialog contributions in an appropriate way (for a brief survey, see [WEBB83]). Each of these methods is restricted to a very particular type of user misconception. The work of Kaplan [KAPS79, KAPS82, KAPS83], for instance, is concerned with the detection of false presuppositions in users' database queries. These include merely "syntactic" presuppositions regarding the existence of the objects mentioned in the discourse, such as

(22) How many female employees in division 3 are managers?
(23) Presupposition: there are female employees in division 3.

Kaplan also considers "indirect" existential presuppositions which can only be inferred using domain knowledge, such as in

(24) User: Which students got a grade of F in CS105 in Spring 1980?
(25) System: CS105 was not given in Spring 1980.

[MAYS80b] is concerned with the recognition of violations of argument restrictions presupposed in users' questions, such as in

(26) User: Which undergraduates teach courses?
(27) System: Undergraduates don't teach courses. Only the faculty teaches courses.

A similar mechanism for detecting "intensional" errors has been implemented in HAM-ANS in its application as an access system to a relational database containing information about research voyages to the antarctic sea [MARB87, MARB88]. The user is informed if relations whose existence is implied by the user's query are not modeled in the database.

All these techniques for detecting and responding to user misconceptions are *not* based on an explicitly represented user model (though an "implicit" user model is hidden in the operation of the mechanisms which implement these techniques). Such an approach is fully justified in limited domains. In more flexible systems,

however, it will become necessary that the assumptions about the user inferred by such techniques be explicitly represented in the user model (e.g. for using them in the determination of the user's level of expertise) and that the handling of user misconceptions be performed on the basis of such an explicit model. Moreover, it can be expected that most types of user misconceptions not described above can only be adequately dealt with on the basis of an explicit user model.

A good example of work along these lines is McCoy's ROMPER system [MCCO85a, MCCO85b, MCCO*] which corrects users' object misclassification (i.e. their subsumption under a wrong superconcept in a concept hierarchy) and the attribution to objects of properties which they do not actually have. Response patterns exist for each type of misconception which enable the system to correct the user. An example of a response to an object misclassification looks as follows:

(28) User: I thought that whales were fish.

(29) System: No, they are mammals. You may have thought they were fish because they are fin-bearing and live in the water. However, they are mammals since, (while fish have gills) whales breathe through lungs and feed their young with milk.

Another example of misconception handling on the basis of a user model is Quilici's [QUIL*] AQUA system. AQUA recognizes user misconceptions about UNIX inferable from his/her reports on problems which s/he has had in interaction with UNIX, and selects appropriate responses. A misconception is discovered by checking each recognized user belief about UNIX against the system's expert knowledge. If the incorrect belief is detected in a catalogue of frequent misconceptions held by the system, the user can be immediately corrected. Otherwise, the type of misconception is determined and associated explanation patterns tested as to whether their preconditions match the recognized user and system belief situation. When an explanation matches completely, the user can be informed of the recognized misconception such as in

(30) User: So to remove a file, I have to own the directory that contains it.

(31) System: No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

4.5. Some Social Problems in Exploiting User Models

If the user is *aware* that the system with which s/he is interacting is equipped with a user modeling component, then s/he has the chance to apply all the techniques for hiding his/her beliefs, goals and plans which s/he constantly uses in every-day person-to-person interaction. This includes, for example, pretending that one knows less or more about a domain, hiding one's goals and plans, beating about the bush, etc. In some applications (e.g. tutoring or unsolicited advice) the user is necessarily aware of the existence of a modeling component; in other applications its existence might not be so apparent to him/her.

Control of the user over the user modeling component means, for example, that the user can inspect the model which the system has made of him/her in the course of a dialog. However, since the number of assumptions made by the system in a normal interaction can be expected to be enormous (cf. Section 2.3.), is it not possible for the user to inspect them all. Another problem is rendering these assumptions comprehensible to him/her. The representational categories employed for user modeling often do not correspond to the usual "folk-psychological" belief and goal categories. Thus the contents of user models are often very difficult to translate into ordinary English.

Control of the user over the user modeling component may also mean that the user is entitled to alter the assumptions that s/he has inspected. In this case, as well, the problem arises of how to explain the consequences of such alterations to the user. One might argue that such an option might seduce him/her to change actually correct assumptions in order to make them correspond to his/her (more positive) self concept. In the case of cooperative systems and computer-based tutoring, however, it is the user's problem if the system's cooperativeness declines as a result of these manipulations of the user model.

Another interpretation of a user's control over a system's user modeling component is that the user has the possibility to "switch off" this component and to communicate with the remaining "unintelligent" system alone, if s/he does not consent to being modeled by the system. In some cases (e.g. tutoring), this is not possible due to the nature of the application. In other cases (e.g. cooperative information), it is certainly not advisable to do so. However, there are also applications (e.g. unsolicited advice) where such a demand on the part of the user may be justified. The technical question remains, however, whether it is possible to separate the user modeling component from the remaining system in such a way that it is possible to "switch it off".

User modeling therefore involves the risk of *misunderstandings* (as is also the case in normal human communication). It can be expected, however, that the gain in cooperativity will greatly exceed that risk. Often it is only this cooperativity which enables the casual user to gain access to the system at all. In traditional dialog systems, by the way, there is also the problem of misconception, however only on the part of the user (e.g. about the meaning of some command or some technical term, which, when undetected, may lead to a wrong interpretation of the results.) In any case, a necessary characteristic of a user model must be that the assumptions stored in it be revisable if counterevidence is observed (cf. [DOYL80]). As we mentioned already in our introductory definitions (see Section 1.2) it is one of the tasks of a user modeling component to maintain the consistency of the model. Therefore a truth maintenance system (TMS, also known as belief revision or reason maintenance system, cf. [DOYL79, DOYL83]) is an important subpart of a user modeling component. Actually, the TMS is a crucial factor in the effectiveness of recent user modeling shells like GUMS (cf. [FINI86, FINI*]) and TRUMP (a TMS- and Rule-based User Modeling Prototype, cf. [BONA87]).

5. Open Questions and Future Research in User Modeling

5.1. Coping with Collective Beliefs

In the research discussed so far, user modeling was restricted to models of individuals. However, as Wilks & Bien [WILK83] have pointed out, it is necessary in some dialog situations to model the beliefs of organizations (e.g. NSF's view of AI research), of states (e.g. Germany's view of the European Community), or of classes of individuals (e.g. the sales division's view of AI). Sometimes in a conversation, an individual (e.g. a sales representative, a district attorney, a diplomat) has to advocate the beliefs and goals of such a collective third party without referring to his/her personal beliefs or goals. Thus it may be reasonable for a cooperative NL interface to a commercial DBMS to restrict the dialog-independent user model, for example to the point of view of the sales division, the research division, or the production division of a company (the view mechanism in database technology may be seen as a primitive precursor of such a group-oriented user modeling scheme). Based on the assumption that an individual user inherits all beliefs and goals from the division s/he is working for, a simple user model may in such a professional environment ignore the "private" beliefs of the dialog partner, and model only the collective ones.

However, for most conversational settings, this is an oversimplified approach since often in the course of the dialog "private" beliefs and "inherited" beliefs are mixed (see Figure 9). For example, a sales representative may begin a conversation with a client by referring to a "private" belief like (32), but later on express the beliefs and goals of his employer, as in (33).

(32) Last time we met we had an excellent dinner together.

(33) This product is going to be a big seller.

Here the open question is how the system can decide which beliefs are to be regarded as "inherited", and therefore to be passed on to that part of the knowledge base which is reserved for the representation of collective beliefs, and which ones are to be stored in the individual user model. Sometimes even a human listener has difficulty coming to a decision and therefore must ask a meta-question like (34).

(34) Is this your personal opinion or an official statement of your company?

Although our current techniques of plan recognition and speech act interpretation must be greatly refined before we can cope with this problem, a future solution may have great practical importance in certain domains, e.g., in consultative and argumentative dialogs and in knowledge acquisition dialogs with expert systems.

Another open question related to the previous one is: How can the system construct, maintain, and update consistent permanent models of collective beliefs on the basis of dialog sessions with individual users? At least for the sake of storage economy, it is necessary for a dialog system with heterogeneous user groups to identify consistent sets of beliefs and construct inheritance networks of beliefs. A solution to this problem must rely on techniques for machine learning, especially inductive inference and stereotype formation.

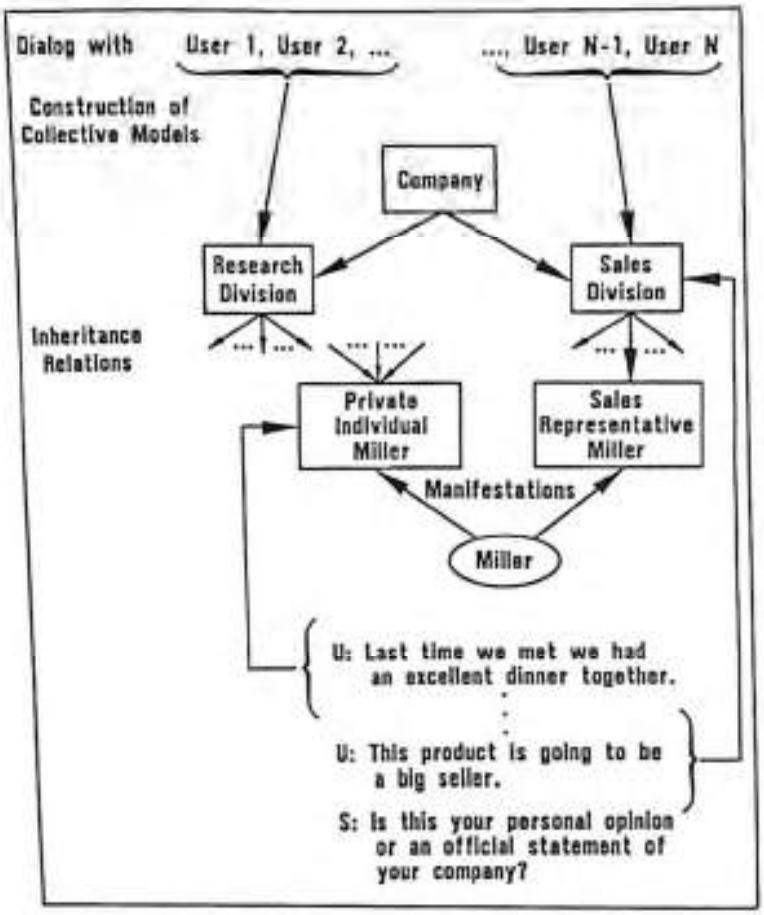

Figure 9. Copying with individual and collective beliefs in a user model

5.2. Adaptability to Diverse Conversational Settings

Today, an important trend in the design of NL dialog systems is the development of *transportable* systems. HAM-ANS (cf. [HOEP83b]), TEAM (cf. [MARP83]) and LDC [BALL84] are examples of transportable systems that can be adapted to different domains by the experts in these fields (rather than by AI or linguistics specialists).

In [HOEP83b] we proposed to go beyond domain-independence by creating dialog systems adaptable to applications that differ not only with respect to the domain of discourse, but also to dialog type, user type and intended system behavior: *transmutable systems* as we call them (see Figure 10).

Such are transportable and adaptable to diverse conversational settings and dialog games (cf. [LEVI77]). As a fictitious scenario for the application of a transmutable system let's suppose that the system's main task is to play the part of a sales representative for a particular company. Sometimes the sales manager interviews the system concerning its job performance. If s/he is dissatisfied with the system's performance, s/he asks an expert in sales promotion to instruct the system.

If you recall the DEC system family XCALIBUR, XSEL and XCON, you will notice that such a scenario is not too unrealistic: XCALIBUR [CARN83c]

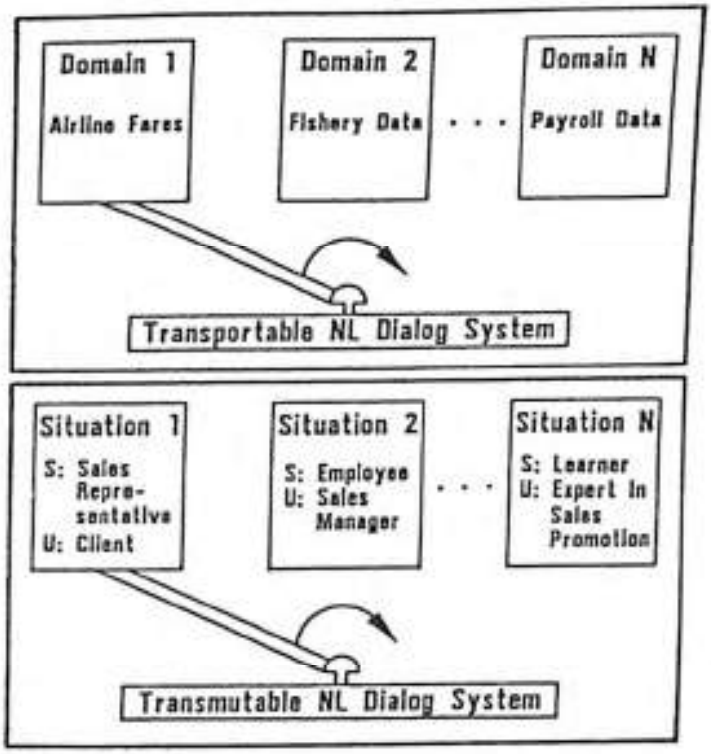

Figure 10. Transportable vs. transmutable systems

is a natural-language interface to XSEL, an automated salesman assistant which advises a customer on the selection of appropriate VAX components. XSEL then produces a sales order and passes it on to the XCON expert system for an automatic configuration of the ordered components.

Nilsson [NILS83] proposed the development of a class of computer systems that are never to be turned off. A dialog system with continuing existence seems to provide a good framework for basic research on transmutable systems. Such a system should have a constantly changing model of its various users and should be able to adapt to diverse conversational settings and to take on various communicative roles.

The major problem that developers of transmutable systems are confronted with is the lack of a representational vocabulary for the declarative description of the relationship between the system and the user, the system's intended dialog behavior, and the associated conversational tactics. Although some of the research discussed in this volume tackles aspects of this problem, a general solution requires lengthy future research on user modeling.

5.3. User Modeling as a Resource-Limited Process

Before asking one's supervisor for a raise in salary, one usually prepares oneself for this dialog by analyzing mutual beliefs and by anticipating the various possible plans

and goals of the dialog partner. But in most everyday conversations, the evaluation of complicated embedded, reflexive, or mutual beliefs and goals as well as a detailed plan analysis is a luxury which one may often not be able to afford. Thus, a computationally plausible outline of a user modeling component must provide some built-in processing restrictions that result from resource-limited real-time understanding of real-life discourse. Rosner [ROSN81] has pointed out that if one carries the user modeling approach to its extreme, there is a very good chance that nothing will get said at all (or if said, not understood) because the speaker and hearer will spend all of their time simulating each other's roles without actually doing anything else. Obviously this process must stop somewhere, but the question is where. Allen & Perrault [ALLE80], for instance, have recognized that their plan-based approach to user modeling may have to be supplemented with a rule-based one. We believe that a promising direction is the compilation of the results of a detailed plan-based analysis of speech act sequences into more efficient "shallow" conversational rules (for a first step in this direction, see the "dialog act plans" in [KOBS85a]).

A related question which so far has not been settled is at what time nested beliefs should be constructed. In the approach proposed by Perrault, Cohen and Allen, most of the possible perspectives on beliefs are already considered as computed (e.g. what one believes one's supervisor believes about one's beliefs about the supervisor's beliefs about a salary increase). Wilks [WILK83, WILK87] adopts the opposite point of view and proposes in his multiple environment approach as a least-effort hypothesis that nested beliefs should only be constructed when needed.

5.4. Prospects for Application-Oriented Research

As can be seen, research on user modeling has been very active in the last few years. A number of interesting proposals have been made concerning representation systems for user models, methods for generating assumptions about the user from his/her natural-language input, and strategies for the planning of the system's dialog contributions on the basis of its model of the user. Also, several prototype systems have been implemented in order to exemplify and further investigate these proposals.

However, a great number of fundamental problems still remain unsolved, some of which have been mentioned in the last few sections. As far as possible applications are concerned, an additional problem arises in that the development of flexible user modeling components is rather costly. Therefore, application-oriented research will probably lead in two directions:

– *Development of user modeling components in help systems for widely used existing systems*

Such a component would observe the user's interaction with the host system, form assumptions about the user's plans and experience, and provide user-specific help on request. Initial efforts along these lines are help systems for UNIX and UNIX-like operating systems [WILE84, WILE86, KEMK86], a help system for the SCRIBE text editor [RICH82], and a system which provides unsolicited advice for users of the VMS operating system [FINI83].

Although work on user modeling to date has concentrated on interfaces to software systems, it should be pointed out that there are possible applications of the methods described above to many electronics-based consumer products, even photocopiers or microwave ovens. In order not to discourage the user by the system's complexity and to help him/her to take full advantage of the capabilities of such multifunctional products, future systems must include an adaptive interface based on an explicit user model. Quinn and Russell [QUIN86] describe a photocopier with an intelligent interface, which analyzes the user's task description (e.g. 'make multiple sets of double-sided copies of an article from a book'), generates a plan to attain the expressed goal, and tells the user what s/he should do next. In testing the system, the critical role of user modeling for plan execution monitoring and the interpretation of the user's behavior (e.g. his replanning after clearing an unexpected misfeed) became clear. The system's individual user model must include information about frequency and recency of particular tasks, frequency and type of successes and failures, points of confusion, and type of help requested and received. The experiments described in [QUIN86] indicate that an intelligent interface must model more than just the operations of the machine and the actions of the user. It must also model the extent of the user's knowledge and plan when to inform the user how to handle problems.

– *Development of general user modeling components*

Another possible line of research seems to be the development of general user modeling components, comparable to the development of expert system shells in the last few years. Such components might provide, for example, a representation scheme for assumptions about a user's beliefs and goals, an inference mechanism for this representation, a mechanism for detecting inconsistencies in the assumptions and for belief revision, etc. Such a shell could then be filled with domain-specific inference rules as well as domain-specific default assumptions about any user, and then be integrated into a dialog system. Beginning research in this direction is the GUMS system [FINI86, FINI*], the TRUMP system [BONA87], the BGP-MS system [KOBS88, ALLG88] and the work of [KASS87a].

6. References

[ALLE79], [ALLE80], [ALLE82a], [ALLE83b], [ALLE86], [ALLE87], [ALLG88], [ALLG89], [APPE82b], [APPE85], [BALL84], [BOGU85], [BONA87], [BOTM87], [CARB83], [CARB*], [CARN83c], [CARR77], [CHIN*], [CLAN79], [CLOW85], [COHE78], [COHE*], [DEDE86], [DOYL79], [DOYL80], [DOYL83], [FAGI85], [FINI83], [FINI86], [FINI*], [GENT83], [GEOR83], [HALP85a], [HAYE76], [HECK88], [HINT62], [HOEP83b], [HORA87], [JAME80], [JAME82], [JAME83], [JAME*], [JOYC83], [KAPS79], [KAPS82], [KAPS83], [KASS87a], [KASS88c], [KASS*], [KEMK86], [KOBS84], [KOBS85a], [KOBS85d], [KOBS86a], [KOBS86b], [KOBS88], [KONO80], [KONO81], [LEVI77], [MARB83], [MARB87], [MARB88], [MARP83], [MAYS80b], [MCCO85a], [MCCO85b], [MCCO*], [MCKE85c], [MOOR80], [MORI84], [MORI85a], [MORI85b], [MORI*], [MURR87a], [MURR87b], [NESS87], [NILS83], [NORM86b], [NOVA84], [PERR80], [POLL82a], [POLL86a],

[POLL86b], [POWE79], [QUIL*], [QUIN86], [RICH79a], [RICH79b], [RICH82], [RICH83], [RICH*], [ROSE82], [ROSE85], [ROSN81], [SCHU83b], [SIDN83b], [SLEE82c], [SLEE85], [SOUL84], [SPAR84], [SPAR*], [SPRE87], [SPRE88], [VARD85], [WAHL83], [WAHL84], [WAHL86b], [WAHL88], [WEBB83], [WERN88], [WILE84], [WILE86], [WILK83], [WILK87]

2 Stereotypes and User Modeling

Elaine Rich

Abstract

A stereotype represents a collection of attributes that often co-occur in people. Stereotypes can play an important role in a user modeling system because they enable the system to make a large number of plausible inferences on the basis of a substantially smaller number of observations. These inferences must, however, be treated as defaults, which can be overridden by specific observations. Thus any stereotype-based user-modeling system must include techniques for nonmonotonic reasoning. This chapter will discuss the role that stereotypes can play in a user modeling system and it will outline specific techniques that can be used to implement stereotype-based reasoning in such systems.

1. Introduction

If one is given the task of constructing a model of another person, one has two choices:

- Construct the model one piece at a time as information is gleaned from experience with the person. In this approach, individual facts about people are viewed as independent events.
- Observe that, empirically, facts about people are not statistically independent events. This suggests that facts can be clustered into groups that frequently co-occur. Now the way to build a user model is by adding to it a whole cluster of facts at a time, as soon as some evidence that is known to be a predictor of the cluster is observed.

This second approach is based on the idea of a stereotype.

The original use of stereotypes was by people forming models of other people. In this chapter, we will extend the idea to include use by programs forming models of people (namely their users). The underlying idea in the two cases is the same: predictive power can be obtained by capturing naturally occurring regularities in human characteristics. Stereotypes are important to people because they permit us to make *predictions* about other people on the basis of an amount of evidence that is sufficiently small that it can be acquired before action is required [MCCA80]. For these predictions to be useful, they do not need to be completely accurate; they

only need to be more accurate than a generic guess, based on no information about the individual, would be. The role of these predictions is not to take the place of specific knowledge about the individual. Instead, their role is to provide a basis for action until such specific knowledge becomes available. As that happens, the model of the individual must be updated, and stereotype-based predictions that conflict with specific observations must be abandoned in favor of the specific facts. This same description is true of the use of stereotypes in programs. Stereotype-based inferences must be treated as default assertions that are defeasible and that may be contradicted by additional specific information as the interaction between the program and the user proceeds.

The use of stereotypes by people can produce negative results. This happens for two reasons. The first is that the stereotypes that people use may be biased so that they make exaggeratedly negative predictions about some groups of people. The second is that subsequent direct observation of an individual may not be used effectively to revise the initial predictions made by the stereotypes. There are many reasons for these problems in people. One of the main ones is that people have cognitive limitations that prevent them both from forming accurate stereotypes and from noticing characteristics that are at variance with the predictions made by stereotypes [TAYL78, HAMI76, CANT76]. In describing stereotype-based user-modeling systems, we will show that it is not necessary for such systems to experience these problems.

The idea of applying stereotypes in a user-modeling system is not new. It was used, for example, in Grundy [RICH79b], which recommends novels to people to read. It was used in UC [CHIN*], which answers questions about UNIX. It was used in HAM-ANS [MORI*], which finds an appropriate hotel room for a user. A very similar idea, that of a *perspective*, was used in the RABBIT [WILL84] system, which allowed users to formulate data base queries by critiquing examples of retrieved objects. Another similar idea, the use of sets of inference rules, each associated with a class of users, was used in UMFE [SLEE85], a general user-modeling front end system.

In the rest of this chapter, we will define what stereotypes are and show how they can be used as components of user-modeling systems.

2. The Definition of a Stereotype

A stereotype is a knowledge structure that can be used by a user-modeling system. A user modeling system, in turn, is part of the user interface component of a complete system, which includes a performance program, which we will call the *back-end system*. Within this context, we will define a *stereotype* to be:

- a *body*, which contains information that is typically true of users to whom the stereotype applies. The form of this information may vary, but it must be compatible with the inference mechanisms that are used in the back-end system.

- a set of *triggers*, which are observable values of attributes. The triggers may be a subset of the values in the body of the stereotype or they may be specific inputs or actions that do not themselves belong in the stereotype.

There may also be a third component to a stereotype:

- a set of relations between it and some number of other stereotypes in the system. The most commonly used relations describe paths in a generalization hierarchy.

SPORTS-PERSON

Triggers (Used-description "athletic")

Facets

Name	Value	Rating
Motivations	*excitement*	600
Character-Strengths	*physical-strength*	900
	perseverance	600
Interests	*sports*	800
Thrill	5	700
Tolerate-violence	4	600

Generalizations
ANY-PERSON [a canonical user model]

Figure 1. A stereotype from Grundy

Figure 1 shows an example of a stereotype called SPORTS-PERSON. It is adapted from a stereotype used by Grundy in its modeling of users as potential readers of novels. The trigger for this stereotype is that, when asked to give a self description, the user used the term, "athletic". The bodies of all Grundy stereotypes are frames (in the sense of [MINS75]), each of which consists of a set of attributes (sometimes called facets) and their associated values. There are two kinds of facets. Some, such as Motivations and Character-Strengths, can contain multiple values. Also, it turns out that those same facets contain symbolic values, such as *excitement*. Other facets, called *scalar facets*, contain single numeric values, on a scale of −5 to 5. All values have associated confidence measures, called *ratings*. These measures are on a scale of 0 to 1000. This stereotype is related to another one, ANY-PERSON, by the generalization relation. This means that the class of people described by SPORTS-PERSON is a subclass of the class of people described by ANY-PERSON.

Figure 2 shows another example of a stereotype. This one is called BUSINESS-EXECUTIVE and it might be used by a system that did travel planning. The structure of this stereotype is not a frame. It is a set of if-then rules that establish likely user subgoals, given a stated top-level goal. Although the actual structure of the knowledge that is contained in a set of stereotypes may vary, as shown in these two examples, we will need a single way to describe the knowledge about users that

```
BUSINESS-EXECUTIVE

if      goal = plan(business-trip)
then    subgoal = get-flight(first-class)
        subgoal = get-hotel(first-class, quiet, close-to(meeting-place))
        subgoal = get-car(high-quality,small)
        subgoal = get-restaurant-list(expensive)

if      goal = plan(pleasure-trip)
then    subgoal = get-flight(coach)
        subgoal = get-hotel(moderate,adjoining-rooms, pool)
        subgoal = get-car(intermediate,family-size)
subgoal = get-restaurant-list(moderate,take-children)
```

Figure 2. A stereotype for a travel system

is contained in these various structures. For simplicity, we will view all stereotypes as composed of a set of *assertions*, regardless of the actual form in which those assertions are represented.

From our definition and from these examples, it should be clear that the main structural property of a stereotype is that it groups together a set of assertions, specifically a set that typically co-occur in people. That set is made accessible by the triggers, which must correspond to readily observable events or traits. A stereotype may be *activated* whenever one of its triggers is observed (unless there is conflicting information that blocks activation. See Section 5 for a discussion of conflict resolution.) When a stereotype is activated, the assertions that it contains are integrated into the system's model of the current user. Thus, a stereotype-based user-modeling system has the ability to make a first guess at a model for a particular user on the basis of relatively little information about that specific user.

3. The Space of User Models

The term *user model* describes a large class of structures, not all of which can effectively exploit the idea of a stereotype. So before we can examine the role that stereotypes play in user modeling, we need to survey briefly the space of user models. Then we will be in a position to pinpoint the specific kinds of systems to which stereotypes can contribute.

There are several important dimensions in the space of user models. These include the following, whose descriptions are adapted from [RICH83]:

- Canonical vs. individual user models. The simplest model is one of a canonical user of a computer system. Such a model describes the characteristics of a "typical" system user. When such a model is the only one used by a system, it is not usually stored explicitly during execution, since it is the same for all users and does not change during an interaction. Instead, its only purpose is to guide initial

system design. The primary weakness of canonical user models is that they do not account for individual differences among users. Specific traits of specific users are not represented, even when they are highly salient and relevant to the task being performed. To add the ability to model these individual characteristics requires the use of individual user models. But now it becomes necessary to do two new things: user models must be stored explicitly at run time; and some mechanism for inferring the individual models from observed events must be provided.

- Long-term vs. short-term user models. Long-term models describe relatively static characteristics of users, while short-term models describe specific topics and goals in the current discourse. This view of a short-term model probably overlaps with what is usually meant by the term discourse model, but see [SCHU88, CHIN88a, COHE88, KOBS88, MORI88, SPAR88, WAHL88] for a discussion of this issue. This same distinction has also been called static vs. dynamic in [SPAR86].

In this chapter, we will focus primarily on long-term models of individual users, because it is there that stereotypes can play the most important role. Canonical user models do not need the explicit mechanisms that we will present for manipulating models during user-system interactions. Short-term models cannot generally benefit directly from the use of stereotypes because short-term models must contain information not just about general characteristics of the user but rather about details of the current interaction. It is usually necessary to get these details directly from the user or to infer them using techniques for plan and goal detection. (See, for example, [GENE82, CARB*, QUIL*].) There are, however, interesting ways in which a stereotype-based long-term model can interact with a specialized, short-term one. For example, the BUSINESS-EXECUTIVE stereotype of Figure 2 can be used to form a long-term model of the travel preferences of a business executive. That model, once formed, may be stable for a long time. But, during that time, the business executive will present a series of different short-term goals to the system. Each time that happens, the system will need to build an appropriate short-term model that can be used to guide it in helping the user satisfy the current goal. One source of information that the system can use in building these various short-term models is the set of inference rules contained in its long-term model of the same user. So, for example, the assertions that are contained in the BUSINESS-EXECUTIVE stereotype (and thus may also be contained in the long-term model of a particular business executive) may be used to augment the current short-term model with a set of appropriate subgoals that the system should attempt to achieve. Thus, although the major role of the stereotype-manipulating techniques that are discussed in this chapter is in the construction and use of long-term user models, these techniques may also be of use in systems where a major emphasis is placed on the manipulation of accurate short-term models.

4. Stereotypes and User Modeling

Stereotypes play a sort of dual role with respect to user modeling. On the one hand, a stereotype represents a kind of canonical user model since it describes a collection

of attribute values that are typically exhibited by users who belong to the class that the stereotype describes. But, on the other hand, stereotypes provide an excellent mechanism for constructing individual user models. Their contribution then is that they enable one, or a small number of, observable events (corresponding to the stereotype's triggers) to provide the basis for inferring probable values for a much larger set of attribute values. The key to an integrated understanding of these two views of stereotypes lies in the notion of a generalization hierarchy.

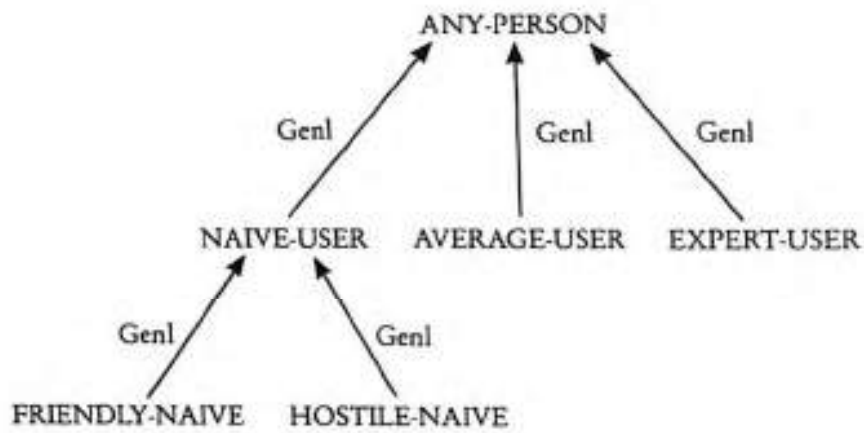

Figure 3. Stereotypes arranged in a generalization structure

As we said in Section 2, a stereotype may have associated with it a set of relations between itself and other stereotypes in the system. The most important and useful of such relations are those that define a generalization hierarchy among a set of stereotypes in a system. The most general stereotype in such a hierarchy then describes a model of a canonical system user. More specific stereotypes describe models of subclasses of system users. See Figure 3 for an example of a set of stereotypes arranged in a generalization hierarchy. In this example, we have used the property Genl as the name for the generalization relation. Other systems have used other names, such as ISA. Such a hierarchy of stereotypes functions in the same way that generalization hierarchies of other sorts function. Specifically, properties are inherited from more general stereotypes to more specific ones so that an efficient representation of shared properties is possible.

All of the stereotypes in a generalization hierarchy must have associated triggers (except the most general one, which can always be assumed to apply if more general information is not available, as it often is not at the beginning of a user's first interaction with the system.) If the only triggers that occur are those that correspond to fairly general stereotypes, then only the corresponding general information will be available for incorporation into the user model. But, if the triggers of more specific stereotypes occur, then a more specific user model will be able to be constructed.

5. Stereotypes and Contradiction Resolution

The use of stereotypes constitutes a form of default reasoning [REIT80]. It generates plausible but defeasible inferences and thus any system that exploits stereotypes must also exploit methods for contradiction detection and resolution. As in any default reasoning system, conflicts may arise for either or both of the following reasons:

- An incorrect inference was made on the basis of incomplete information and now, with additional information, that inference is shown to be false.
- An inference described something that was true once (presumably when it was made), but the situation has changed and the inference should be retracted. Notice that this second source of conflict can also occur in systems that do not use default reasoning.

We will be concerned here with only the first of these sources of contradiction because of our emphasis on the use of stereotypes in the construction of long-term user models.

Facet	Value	Rating	Justification
Gender	Male	900	Mail name
Thrill	5	900	MAN SPORTS-PERSON
Tolerate-violence	5	866	MAN SPORTS-PERSON
Motivations	Exitement	760	MAN SPORTS-PERSON
C-Strengths	Perseverance Courage Phys-Strength	600 700 950	SPORTS-PERSON MAN MAN SPORTS-PERSON
Interests		800	SPORTS-PERSON

Figure 4. A Grundy model of an individual user

Because conflicts may appear at any time in a stereotype-based system, it is necessary that such a system record not just assertions about users, but also justifications for each of those assertions. Without a record that states clearly why each assertion is currently believed, explicit reasoning about the validity of that belief will not be possible. The method that should be used for storing these justifications depends on the form of the assertions themselves. For example, in Grundy, assertions were represented as values of attributes. In such a system, a list of justifications can be associated with each such value. Figure 4 shows a simplified version of a Grundy user model in which this is done. In this example, two stereotypes, MAN, and SPORTS-PERSON have been activated for the user. Thus either of these stereotypes may

justify a value. In addition, direct observations (such as the fact that the user's name is one that is usually male), may serve as a justification. In other kinds of user modeling systems, other techniques for recording justifications may be used. For example, if a system of logical statements is used, it may be appropriate to use a truth maintenance system (TMS) of some sort [DOYL79] to record dependencies among the statements.

There are two kinds of contradiction that can occur in a stereotype-based system:

1. Contradiction between/among two or more stereotypes. This may occur whenever the system's observations about a user result in the activation of multiple stereotypes that make predictions about a given attribute. This category can be further divided on the basis of the resolution strategy that is appropriate:
   a. Contradiction between one stereotype and another that is either above or below it in the stereotype generalization hierarchy. For example, a COMPUTER-SCIENTIST stereotype might predict a value of HIGH for the attribute COMPUTER-COMFORT. But the more specialized stereotype, THEORETICAL-CS, might predict the value LOW.
   b. Contradiction between any other group of stereotypes. For example, an OFFICE-WORKER stereotype might predict a value of LOW for the attribute PHYSICAL-CONDITION, while, for the same attribute, the MARATHON-RUNNER stereotype would predict a value of HIGH. But both stereotypes may be activated for a single user, and, if they are, a conflict will result.
2. Contradiction between stereotype-based inferences and assertions that are based directly on observations of the user or statements by the user.

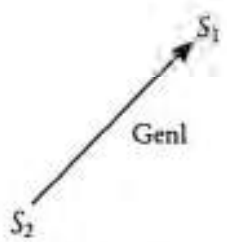

Figure 5. A fragment of the stereotype generalization structure

In case 1a, the conflict resolution procedure is straightforward. The value predicted by the more general stereotype should be ignored and the value predicted by the more specific one should be used. More precisely, let S_1 and S_2 be two stereotypes that predict contradictory values for attribute A. (If more than two stereotypes are involved and they all occur in a single generalization chain, then this procedure can be applied to successive pairs, in any order, to produce the correct result.) If S_1 occurs above S_2 in the generalization hierarchy (see Figure 5) then the

set of users contained in the class described by S_1 is a superset of the set of users contained in the class described by S_2. In this case, use the value for A contained in S_2. Otherwise, use the value contained in S_1. The justification for this procedure is that, since S_2 describes a more restricted set than does S_1, it is interpreted as describing a special case of the class described by S_1. This procedure is analogous to the standard definition of property inheritance [TOUR86], in which values associated with subclasses specify exceptions to general values associated with superclasses.

In the other two cases, the situation is somewhat more complicated. To define a procedure to handle them, it is necessary to notice that a stereotype-based inference is the result of a two-step reasoning process:

- Step 1: On the basis of the trigger, infer that the user is a member of the group described by the stereotype.
- Step 2: On the basis of the stereotype, infer specific values of the attributes mentioned in the stereotype.

When a contradiction arises, either or both of these reasoning steps may be the culprit. If a trigger signals the applicability of its associated stereotype with less than complete certainty, then even when the trigger has been observed, it may be that the corresponding stereotype should not have been activated. Likewise, if a stereotype predicts a particular attribute value with less than complete certainty, then it is possible that, even though a user is a member of the class described by the stereotype (i.e., the stereotype was correctly activated), it is nevertheless the case that this particular prediction is inappropriate for this user.

When a contradiction is detected, two kinds of remedial action may be necessary. The first is to resolve the contradiction within the user model for the current user. When there is a contradiction between two stereotypes (case 1b, above), it may not be clear how the conflict should be resolved. Some possible methods will be discussed below. But when there is a contradiction between a stereotype and a direct observation (case 2, above), the prediction of the stereotype must be ignored in favor of the direct observation. Doing this guarantees that stereotype-based predictions can drive a model only until more direct evidence is available; they can never prevent the use of such evidence once it appears. The second kind of remedial action that is required is to modify the stereotype knowledge base in such a way as to decrease the likelihood of a similar contradiction arising in the future. This step is required in both kinds (1b and 2) of conflict situations. We will describe a system that performs this step in the following section.

Resolving conflicts within a user model is an instance of the more general problem of reasoning in a default system. There are two approaches that can be taken:

- Use some sort of numeric estimates of values and of the system's confidence in those values. Define some combining functions that can be used when more than one piece of evidence relates to a particular value.
- Avoid numeric certainty measures. Instead, select, either arbitrarily or on the basis of some knowledge, one of the competing values and ignore the others.

In many systems, a combination of the two approaches may be useful. As we discuss these approaches in the rest of this section, we will have to consider three parts of the process of resolving a conflict within a user model:

- Resolve the current conflict by deciding what assertions should be contained in the user model.
- If certainty measures are being used, decide what certainty to assign to the assertions chosen above.
- If, as a result of resolving the conflict, any assertion has been retracted or modified, find all other assertions that depended on it. Retract or modify them appropriately and propagate any changes to any other dependent assertions. If certainty measures are used and any of their values change, they too must be propagated.

One issue that must be dealt with is how to terminate this third step and how to keep the amount of work spent on the process to a manageable level. Note that this propagation step means that substantial reasoning may be necessary even if the process of resolving the original conflict is trivial (as, for example, in the case (2) of a conflict between a stereotype-based prediction and a direct observation).

5.1. Resolving the Current Conflict

Any reasoning that the user modeling system is capable of performing could, in principle, be used to resolve conflicts between the predictions of different stereotypes. Any such reasoning method has available to it two kinds of information that it can use in making its decision:

- Information about the specific stereotypes that made the contradictory predictions.
- Information about the specific assertions that are in conflict.

Either, or both of these kinds of information may be useful.

Information about the stereotypes that made the conflicting suggestions corresponds to world knowledge about characteristics of people who belong to several groups simultaneously. The easiest-to-use form of this knowledge has already been presented as case 1a above. When the two stereotypes exist in a subclass-superclass relationship to each other, then it is clear that the more specific information should override the more general. Even when this simple situation does not exist, there may still be a way to use information from the generalization hierarchy to help resolve a conflict. Suppose that stereotypes A and B predict conflicting assertions and suppose that the least upper bound of A and B in the hierarchy is C. Then one strategy is to ignore the conflicting predictions of A and B and to use the corresponding prediction of C instead. This corresponds to the notion that perhaps our information about exactly which sublass of C the user belongs in is not accurate so the best we can do is to treat the user just like any member of the more general class C.

A second way that information about stereotypes themselves can be used to resolve conflicts among stereotypes is to recognize that a stereotype contains two kinds of information about people who are members of the class that the stereotype

describes. The first kind is necessary information, i.e., definitional information or information that is prerequisite to definitional information. The second kind is default information. Whenever there is a conflict between these two types, the first will win over the second. As an example, consider the two stereotypes shown in Figure 6, each of which describes a class of computer scientists. The attributes shown in boldface are necessary attributes of the class described by the stereotype.

HACKER	
hacking skill	high
math skill	med
work hours	night

THEORETICIAN	
hacking skill	low
math skill	high
work hours	day

Figure 6. Two stereotypes that may conflict

You cannot be a hacker if you cannot code. You cannot be a theoretician if you cannot do mathematics. Suppose that both of these stereotypes have been activated for a particular user and the system needs to know what the user's level of skill at hacking is. Then the value from the HACKER stereotype should be selected and the value from the THEORETICIAN stereotype should be ignored. Another way of saying this same thing applies if stereotypes are viewed as collections of logical expressions rather than as frames. Then the necessary information will take the form of standard, first-order formulae such as

$$\forall x\ hacker(x) \rightarrow hacking\text{-}skill(x,high)$$

Default information will be stated as such explicitly, using a higher-order predicate such as Consistent-to-Assume. Thus we might write, for example:

$$\forall x\ theoretician(x) \wedge Consistent\text{-}to\text{-}Assume(hacking\text{-}skill(x,low)) \rightarrow hacking\text{-}skill(x,low)$$

Then the inference rules that are used to form extensions (i.e., consistent theories that contain all ordinary assertions and maximal consistent sets of assertions derivable from the defaults) will prevent the default rules from applying when they would lead to an inconsistency.

So far, we have presented domain-independent ways of resolving conflicts between stereotypes by using information about the stereotypes. Yet another possibility is to augment this general system with a set of domain-specific facts that describe what to do when predictions of specific stereotypes conflict with each other. For example, consider again a travel system that might use a stereotype like the BUSINESS-EXECUTIVE one of Figure 2. Such a system might also possess a stereotype for a young person. This stereotype would differ from BUSINESS-EXECUTIVE in several ways, among them that the user is predicted, when going

on a pleasure trip, to be interested in the cheapest possible fare and probably not to be interested in a hotel room at all since he or she may be planning to stay with friends. A separate piece of knowledge that a user-modeling system in this particular context might have is that BUSINESS-EXECUTIVE should override YOUNG-PERSON. The rationale for this (which does not need to be encoded explicitly in the system at all) is that what really matters in this context is the amount of money that the user can afford to spend and that there are some young people, including those who have managed to become business executives, who have money, even though most do not. But notice that this knowledge is highly domain-specific. If one were building a system that chose the music that should be played to a user who has been put on hold on the telephone, young person should probably override business executive.

In addition to using information about the stereotypes that produced the conflicting predictions, it is possible to resolve contradictions by using knowledge about the predictions themselves. Just as in the case of reasoning about stereotypes, there are general, domain-independent strategies and there may also be more specific, domain-dependent strategies. Domain-independent strategies are those that depend only on the form of the predictions and not their actual content. The following simple, domain-independent strategies are available: (1) If the thing that is being predicted is a numeric value, then predict the average of the competing values. (2) Compute the least upper bound (in a generalization hierarchy) of the predicted values and predict it. This is the analogue of (1) when symbolic values are being predicted. (3) Prefer the older assertion since it is better established. (4) Prefer the newer assertion since it is more likely to be current. (5) Prefer not to believe any assertion that has previously been involved in a conflict (even if it won). (6) Prefer the assertion whose consequences are more attractive. Notice that these general strategies may compete with each other so they cannot all be used simultaneously without some other conflict-resolution strategy. Whenever these general strategies are not adequate, of course, any number of specific inference rules (analogous to the stereotype inference rule that business person should win over young person) can be used to specify that certain predictions should be believed over others.

5.2. Assigning a Certainty Value

It is not necessary for a stereotype-based user-modeling system to exploit explicit certainty measures. But it is often useful to do so in order that the back-end system will know how much to rely on the information in the user model. If certainty measures are used, then the procedure for activating stereotypes and combining them into a consistent user model must compute appropriate certainty values. When only one piece of evidence bears on a particular conclusion, this poses no problem, assuming that some estimate of predictive certainty is associated with each trigger/stereotype pair and with each stereotype/prediction pair. The difficulty comes when there are multiple sources of a prediction about the same thing. If the sources agree, then the resulting certainty should be higher than it would be after only one vote. If the sources disagree, then the degree of uncertainty arising from the disagreement must be computed.

The problem of representing degrees of certainty in a stereotype-based system is not any different from the problem of representing degrees of certainty in any other default reasoning system, and several frameworks for solving the problem exist. The simplest one mathematically, although not usually computationally, is to use Bayes' theorem [CHAR83], which describes the relationships that hold among various conditional probabilities. Another possibility is to use the Dempster-Shafer [SHAF76] model, in which certainty estimates are viewed not as points but as ranges of likelihood. A third possibility is to use a somewhat more *ad hoc* set of combination rules, such as the ones that were defined for the Mycin [SHOR76] system. This last approach worked well, for example, in Grundy.

5.3. Propagating the Change

Default reasoning of all sorts, including the kind introduced through the use of stereotypes, is nonmonotonic in the sense that when a new assertion is added to the system, some other assertions may need to be removed or modified. Thus it is critical to the success of a stereotype-based system that it be able to propagate the necessary modifications throughout a user model whenever any change is made.

It is useful to distinguish two kinds of propagation that may need to occur. The first is of symbolic objects. These objects may correspond either to assertions about the current user or they may correspond to justifications of those assertions. The most straightforward way to propagate symbolic objects is to build the user modeling system on top of a reasoning system, such as a TMS [DOYL79] or an ATMS [DEKL86], that provides the mechanism for recording justifications and for restoring consistency when necessary. Here again, the needs of a stereotype system are not fundamentally different from the needs of any other system that does default reasoning.

The second kind of propagation that may need to occur is in numeric objects. For example, suppose that trigger T1 suggests stereotype S1 with some rating R1. Then S1 suggests assertion A1 with some rating R2. Thus (in the absence of any conflicting information), A1 will be asserted with some certainty, which is a function of R1 and R2 (perhaps just their product). Now suppose trigger T2 is observed and it also suggests stereotype S1, this time with the rating R2. As a result, the combined rating of stereotype S1 must be changed. But if this rating is changed, then the rating of assertion A1 must also be changed, since it was a function of the rating of S1. Now, if some other assertions were made as a result of A1, their ratings must also be changed and so forth. An issue arises here because of the way that numeric values are usually combined. Specifically, if two values are combined to form a third, then the influence of either of the originals on the third is usually such that, if one of the originals changes, then the change in the third will be smaller than the change to the original one was. This is the case, for example, if the two values are combined by multiplying them together and if they are both less than 1 (which they often are if they are viewed as certainty measures). This means that, as a change is propagated through the system, it becomes smaller and smaller. But, of course, the amount of work required to propagate it does not decrease correspondingly. Thus there often comes a point at which substantial effort is being expended on the propagation of

insubstantial changes. It is often useful to define some threshold, δ, such that any change with absolute value less than δ will not be propagated further.

6. Adaptation of Stereotype Systems

Adaptation is an important part of an effective stereotype-based system because of the difficulty of defining accurate stereotypes *a priori*. A variety of learning and adaptation techniques can be applied to stereotypes, depending on exactly the form in which the stereotypes and their triggers are represented. In the last section, we presented two broad classes of techniques for dealing with the issue of conflict resolution within a particular user model: general, domain-independent techniques that rely only on the structure of the objects being examined and not on their content, and specific techniques in which domain-specific information (such as a preference for using the BUSINESS-PERSON stereotype over the YOUNG-PERSON one) was exploited. The same two kinds of approaches are also possible for dealing with the problem of stereotype adaptation. In this section we will deal only with the first of these strategies, since it is the only one that can be described in a general way.

Because of the inherent uncertainty in stereotype-based inferences, one natural adaptation technique applies to stereotypes (such as the one shown in Figure 1) that contain information to which explicit measures of certainty have been attached. In this case, adaptation consists of modifying the certainty measures on the basis of each experience with a user. In this section, we will describe a modification system (that was used in Grundy) that takes this approach.

If we are going to use concrete interactions with users as a basis for adapting the certainty factors associated with a stereotype system, we must look for two kinds of feedback from the interactions:

- Evidence that the predictions made by a stereotype are correct and should be made again in the future in similar circumstances.
- Evidence that the predictions made by the stereotype are incorrect and should not be repeated in similar circumstances in the future.

The adaptation algorithm must then combine these two kinds of evidence into a single, coherent strategy for updating the certainties associated with stereotypes and their triggers. This strategy must deal adequately with both of the following issues:

- How should credit and blame be assigned among all the factors that made a given prediction?
- How much weight should be given to the initial values in the stereotypes and to each additional piece of evidence?

In defining Grundy's adaptation strategy, the following answers were used. For the first question, the approach is not to try to apportion credit or blame intelligently. Instead, all contributing factors are modified. This means that some inappropriate modifications will occur. But on the average, the factors that are actu-

ally responsible for the prediction will have their values changed in the appropriate direction. The others will fluctuate some, but will not show any noticeable drift. The answer to the second question is somewhat *ad hoc*, and is shown in the formulae given below. When values are being modified, the old value is weighted by a system-defined constant (which was set to 100) and then combined with the new value. When ratings are being modified, the weight given to the new value depends on the magnitude of the old rating (so low ratings can easily be changed but higher ones change only slowly). More sophisticated answers to this question are possible. For example, one might record, with each value, some measure of the total amount of evidence that has thus far contributed to it. Then the relative influence of each new piece of evidence could decrease over time. The most interesting thing about Grundy from this point of view is not that it uses a sophisticated method but that it uses an unsophisticated one that does, nevertheless, work.

In Grundy, adaptation of the stereotype system could occur every time a new fact or inference (i.e., something other than a stereotype-based prediction) was added to some facet of an individual user model. Every stereotype that had made a prediction about the value of that facet was modified. If the new fact supported the stereotype's prediction, then the stereotype was reinforced. If the new fact contradicted the stereotype's prediction, then the stereotype was weakened.

Three kinds of changes are necessary to cause this to happen. The first is to modify the actual value that the stereotype predicts for the facet. This was only done for facets with numeric values (such as Thrill and Tolerate-violence in Figure 1). For these facets, the following formula describes the change that was made:

$$NEW\text{-}STEREOTYPE\text{-}VALUE = \frac{OLD\text{-}STEREOTYPE\text{-}VALUE \bullet CONSTANT + NEW\text{-}INFO\text{-}VALUE}{CONSTANT + 1}$$

The second kind of change that must be made is to the rating attached to the prediction in the stereotype. When a confirmation of a stereotype's prediction occurred, the rating attached to that prediction was modified according to the following formula:

$$NEW\text{-}RATING = OLD\text{-}RATING + \frac{NEW\text{-}INFO\text{-}RATING}{OLD\text{-}RATING}$$

When a contradiction to a prediction occurred, the rating attached to that prediction was modified similarly except that the change was subtracted from rather than added to the rating:

$$NEW\text{-}RATING = OLD\text{-}RATING - \frac{NEW\text{-}INFO\text{-}RATING}{OLD\text{-}RATING}$$

The third kind of change that must be made is to the rating of all of the triggers of the stereotype that made the prediction. Recall that this is necessary because it is not possible to tell whether the problem is that the stereotype was appropriate for the user but it made an inaccurate prediction or whether the problem has arisen

	Number of users interacted with so far					
	0		10		20	
Facet	Value	Rating	Value	Rating	Value	Rating
Thrill	4	700	.9	699	.1	702
Personalities	−2	500	−1	503	.9	504
Suspense	4	700	2.6	700	1.2	700

Figure 7. Three snapshots of the Grundy MAN stereotype

because the stereotype should not have been activated for this user. The trigger ratings were modified using the same formulas that were used to modify prediction ratings.

Grundy was never run as a large-scale performance program, so there was no opportunity to monitor the evolution of its stereotypes over a very large number of users. But some interesting adaptation did occur. Recall that Grundy's task was to recommend novels to people to read. So the information contained in its stereotypes corresponded to preferences for certain kinds of books. Figure 7 shows three snapshots of part of the MAN stereotype that Grundy used. The first value and rating are the ones that were initially provided to Grundy (based on a best guess about the characteristics of the population that the stereotype would describe.) The second value and rating correspond to the computed numbers after complete interactions with ten users to whom the stereotype applied. The last value and rating correspond to the computed numbers after complete interactions with twenty such users. The Thrill facet was intended to describe the user's level of interest in a thrilling plot (on a scale of −5 to 5). The Personalities facet was intended to describe the user's level of interest in well-developed characters. The Suspense facet was intended to describe the user's level of interest in a suspenseful plot. Roughly, the original stereotype described a user who wanted action as opposed to character development. The modified stereotype described a more balanced user.

Other approaches to the adaptation problem are also possible. For example, stereotypes can be viewed as concepts, and then they can be learned with statistical concept-learning methods. Another approach that could work is to train a neural network [RUME86] to recognize instances of stereotypes.

7. Parallel Models of System Knowledge and of Users

Stereotypes provide a mechanism for organizing information about classes of users in a way that makes possible rapid inference about users. One of the important things to know about a user of a knowledge-based system is how the user's knowledge compares to the system's model of knowledge in the relevant domain. One way to make inferences of this sort possible is to organize system knowledge in a parallel

structure to that in which user knowledge is organized. In other words, something like a set of stereotypes can be used to group together pieces of system knowledge in much the same way they are used to group together knowledge about users. The user modeling system in UC [CHIN*] does exactly this in order to be able to guess quickly whether a particular user knows about a particular feature of UNIX.

8. Summary

This chapter has two main points. The first is that the idea of a stereotype, viewed as a cluster of assertions about possible users, can be useful in a user-modeling system. The second is that stereotype-based inference is a form of default reasoning. It thus possesses all of the difficulties that such systems normally possess. And some of those difficulties may be overcome by using various of the techniques that have been proposed for reasoning in default systems.

Acknowledgements. I am indebted to Jim Miller and Alfred Kobsa for substantial comments on an earlier version of this chapter.

9. References

[CANT76], [CARB*], [CHAR83], [CHIN88a], [CHIN*], [COHE88], [DEKL86], [DOYL79], [GENE82], [HAMI76], [KOBS88], [MCCA80], [MINS75], [MORI88], [MORI*], [QUIL*], [REIT80], [RICH79b], [RICH83], [RUME86], [SCHU88], [SHAF76], [SHOR76], [SLEE85], [SPAR86], [SPAR88], [TAYL78], [TOUR86], [WAHL88], [WILL84]

3 A Taxonomy of Beliefs and Goals for User Models in Dialog Systems [1]

Alfred Kobsa

Abstract

In this chapter, a taxonomy of beliefs and goals is proposed, the purpose of which is threefold: it should allow for a classification of current user models with regard to *what is being modeled* by them; it should support the designer of a user model in investigating what types of beliefs and goals are relevant to his/her particular application; and it should allow the evaluation of the expressive power of representation schemes for beliefs and goals. For each belief and goal type, examples are presented for their occurrence in application domains of dialog systems. In addition, a number of existing systems are compared with respect to what beliefs and goals are being modeled by their user modeling components, and the consequences of an observed dichotomy of user models with respect to complexity is discussed.

1. Introduction

In [WAHL*], a user model in an artificial intelligence dialog system has been defined as a knowledge source which "contains explicit assumptions on all aspects of the user that may be relevant for the dialog behavior of the system". This chapter concentrates on that part of the user model which contains the system's assumptions about the user's *beliefs, goals* and *plans*. For a dialog system, assumptions about the user's beliefs are necessary, for example, to discover misconceptions on the part of the user or to adapt its answers to the level of his/her expertise in the domain of discourse. Assumptions about the user's goals and plans can be used, for example, for providing better recommendations or for supplying additional information not explicitly requested by the user.

There are several application domains in which it is necessary that the system also take into account "objective facts", about the user. Morik & Rollinger's

[1] This research was supported by the German Science Foundation in its Special Collaborative Research Program No. 314 on AI and Knowledge-Based Systems. The development of the VIE-DPM system reported herein was partly funded by the Austrian Ministry of Science. I am indebted to David Chin, Mark Line, Gudula Retz-Schmidt and several anonymous reviewers (mostly participants of the User Modeling Workshop in Maria Laach) for their comments on earlier versions of this chapter.

[MORI85b] system, for instance, stores the information which the user provides about the number of occupants of the apartment s/he wants to rent, about the accepted price range, the age of his/her children, etc. HAM-ANS, when run in the hotel reservation domain, makes assumptions about the customer's profession, solvency, and the type of his/her trip [MORI85a, MORI*]. In most cases, however, such "objective" entries in the user model are not directly used for planning the system's dialog contributions, but for inferring probable *goals* of the user.

A look at proposed or implemented user models demonstrates that rather different types of user's beliefs and goals are being modeled in this field. In most systems, however, only a few of these belief and goal types are typically modeled, and the expressive power of the representation scheme is restricted to these particular beliefs and goals alone. In Rich's [RICH79a, RICH79b] GRUNDY system, for instance, the user model consists only of feature-value pairs which are weighted by certainty factors. In Sleeman's [SLEE85] UMFE system, concepts are labeled with markers that indicate whether the system believes / does not believe / is uncertain whether the user knows the concept. In the user model of HAM-ANS, only those beliefs of the form 'the user believes p' and those goals of the form 'the user wants p' can be represented in which p is a fully instantiated proposition about the situation. Only in systems specifically devoted to speech act recognition or dialog planning (ARGOT [COHE79, ALLE83b]; KAMP [APPE85]; VIE-DPM [KOBS85a, KOBS85d]) is a much greater number of belief and goal types modeled.

In this chapter a taxonomy of beliefs and goals is proposed, the purpose of which is threefold: it should allow the classification of current user models with regard to *what is being modeled* by them (and possibly help to avoid some confusion which seems to have recently come up); it should support the designer of a user model in investigating what types of beliefs and goals are relevant in his/her particular application; and it should allow the evaluation of the expressive power of representation schemes for beliefs and goals, since it seems that a great number of belief and goal types should be representable in user modeling components of more general natural-language dialog systems – particularly, of course, in user modeling shells designed for NL dialog systems. For each belief and goal type, examples are presented for their occurrence in application domains of dialog systems. Moreover, a number of existing systems are compared with respect to what beliefs and goals are being modeled by their user modeling components.

2. Beliefs, Goals and Plans

This is certainly not the right place to investigate what beliefs, goals and plans "really are" or should be considered to be. As a matter of fact, the ascription of these and similar "intentional" states to people, small children, animals, computers, thermostats, etc. is abundant in our commonsense explanation of behavior. We explain Prince Andrew's marrying Sarah Ferguson by attributing to him the *goal* of marrying her. And we explain the stampede of gerbils upon sighting an airplane by ascribing to them the *belief* that there is a bird of prey in the air.

In the scientific psychology of cognition, these notions have had a rather checkered fate. While fully recognized in German 19th century psychology, they were banned together with all other mental terms by behaviorism, which regarded them as unscientific since not observable. Now, in the era of cognitive science, their good reputation has been restored, though they have recently been attacked anew, this time from the side of the philosophy of psychology (cf. [STIC83]).

In the field of user modeling, the assumption that users have beliefs, goals and plans which must be modeled in dialog systems is doubtlessly a very fruitful one. Also in other fields of artificial intelligence research or computer science in general, the modeling of beliefs has been the object of increasing interest in the last few years. This is particularly the case in the fields of *multiple-agent planning systems, distributed databases* and *intelligent computer-aided instruction*. In the first field (cf. [POWE79, KONO80, ROSE82, GEOR83, ROSE85, WERN88]), a group of robots (possibly equipped with divergent perceptual capabilities) is to construct and execute a plar for the achievement of some goals. In the field of distributed databases [LEHD84, HALP85b], independent processors (each working with one database) interact with each other in order to answer queries. In both fields it seems desirable for a robot/processor to be able to represent the fact that some other robot/processor knows who knows some fact, whether there exists one who knows some specific fact, etc. In the field of intelligent computer-aided instruction [SLEE82c, DEDE86], the system makes assumptions about what a student already knows of the material to be taught, about what false beliefs s/he has, etc.

For modeling beliefs, plans and goals it thus makes sense to investigate what *types* of beliefs and goals should be discerned, how complex beliefs and goals are *composed*, and how the discerned beliefs and goals can be put into a *systematic order*. Quite in the spirit of artificial intelligence planning, we regard plans as sequences of intermediate goals, and action operators for achieving these goals, and will therefore disregard them for the remainder of this chapter. Since, in the field of user modeling, beliefs and goals can be found both on the side of the system and on the side of the user, the term *'agent'* will be employed to refer to both of them. This term was chosen deliberately since our commonsense attribution of beliefs and goals to individuals essentially depends on whether they are judged to play an active role in events of our world. The agent who models a belief or goal will be referred to as the *modeling agent*, and the agent whose beliefs or goals are being modeled will be called the *modeled agent*.

When a classification of beliefs and goals is to be defined, means are necessary for characterizing what beliefs and goals should be grouped together. In this chapter, *linguistic descriptions* which are commonly used for attributing these beliefs and goals to other agents will be employed for this purpose. However, we must be somewhat careful with such an approach since, as was pointed out by many authors in the fields of epistemic logic and linguistics, a number of problems and ambiguities are hidden in certain kinds of linguistic belief and goal descriptions. Moreover, recent research in several disciplines has revealed that there are types of beliefs that should be discerned for which no everyday description exists (for these, the technical terms "infinite-reflexive" and "mutual beliefs" have been introduced).

One of these ambiguities, which has long puzzled philosophers and which has been most prominently exposed by Quine [QUIN56], is that many linguistic descriptions of beliefs and goals can be interpreted in two different senses, namely a so-called *de dicto* and a *de re* reading. For example, when John watches a man with a pipe stealing into Buckingham Palace, a speaker might describe John's beliefs from John's perspective (i.e., *de dicto*) as follows:

(1) John believes that the man with the pipe is a spy.

Assume that the speaker, unlike John, believes that the suspicious character is no less a person than the famous Captain Maigret solving his latest case. Seen from the speaker's perspective, John's belief then can also be described *de re* by

(2) John believes that Maigret is a spy.

The problem is that, in this particular situation, (2) is true only when interpreted in a *de re* sense, and false when interpreted in a *de dicto* sense. The general problem is that belief and goal descriptions can be interpreted both from the speaker's perspective, and from the perspective ("belief context") of the individual whose beliefs or goals are being described.[2] In this chapter, all linguistic belief descriptions (such as 'agent a *does not believe that* p' or 'agent a *does not want* p') should be read in a *de dicto* sense, i.e. from the point of view of the individual a. Belief descriptions de dicto "essentially involve the believer's conception of the issue at hand" [BURG78: 341]. For checking whether the beliefs or goals attributed to agents by the use of such a description have been correctly identified, readers should put themselves into the position of this agent and test whether they could exhaustively *report* these beliefs or goals by replacing the variable 'a' in the description with the pronoun 'I' (which yields 'I *do not believe that* p' and 'I *do not want* p' in our example).[3]

Another problem with linguistic belief descriptions concerns the meaning of the concept 'know'. Philosophers have argued that a belief description such as 'a knows that p' implies that p is the case. In the field of user modeling, however, the point of reference for judging the correctness of a belief is not the "world", but the beliefs of the system. It makes no sense to represent *within* the system beliefs of the system describable by 'The system believes that ... But actually ...' or 'The system believes that ... But the user knows correctly that ...'. Constellations like these are only observable from *outside* the system. Thus the concept 'know that p' is used in a somewhat different sense in this chapter: an agent a1 believes that an agent a2 knows p if a1 believes a2 believes p and a1 also believes p. Definitions for 'know whether p', 'know the x such that p(x)', 'want to know the x such that p(x)', etc. will be given below.

[2] For related views, see [MOOR82, KRAU83, RAPA84, BARN86]. As was pointed out by Stich [STIC83], however, this "ambiguity" of belief or goal descriptions is mostly due to the fact that they are seen in isolation. When investigated in a situational context, there is only one possible reading.

[3] Here I ignore the problem of the substitution of so-called 'quasi-indicators' such as personal pronouns and demonstratives. See [RAPA84] for a discussion.

3. Basic Beliefs

Basic beliefs of an agent are beliefs which are *not* about goals or beliefs of the same or some other agent, but about facts in a certain domain (the so-called *situation*). Beliefs which *are* about other beliefs and goals will be called *complex*. (The system's beliefs about the situation, for instance, form basic beliefs, whereas the system's beliefs about the user's beliefs or goals are already complex.) We will see in the next few sections, however, that complex beliefs and goals can be recursively defined and contain one basic belief or goal of a given agent (e.g. the user). It therefore makes sense to investigate basic beliefs and goals from the user's perspective as well, as we do below.

Situational facts may be individual or general. Individual facts comprise single events in the past, present or future. General facts concern general properties of objects in the situation, general relations between objects, general rules, etc. Examples of both will be given below. The symbol 'p' will be used to refer to arbitrary individual or general facts.

3.1. Basic Saturated Beliefs

Saturated beliefs are beliefs about a fact p, with p being a proposition the formalization of which in predicate calculus contains no free variables or existential quantifiers. There exist 4 types of basic saturated beliefs (the three periods at the beginning of each belief description have to be replaced with the name of an agent):

(1.1.1.) ... *believes that* p.
(1.1.2.) ... *does not believe that* p.
(i.e. believes that not p.)
(1.1.3.) ... *is uncertain whether* p.
(1.1.4.) ... *knows whether* p.
(i.e. has a definite attitude as to whether p.)

Type (1.1.1.) forms the most frequent kind of belief that is represented in current user models. At the level of general facts, such beliefs are modeled for the purpose of representing what concepts, rules or procedures a user knows about. At the individual level, they represent the objects and events that occurred explicitly or implicitly in the preceding dialog.

Examples include:

(3) The user knows the meaning of the concept 'DIPLOCOCCUS-PNEUMONIAE' (general fact; [SLEE85]).
(4) The user knows the rule 'Elevated cyclic AMP leads to increased osteoclast activity' (general fact; [WALL84]).
(5) The user believes that the function LIST combines two lists into a single one (general fact; [REIS85])
(6) The user believes that Tony has a tennis game in the afternoon (individual fact; [MARS83])

(7) The user believes that John loves Mary (e.g., since the system told him/her so; individual fact; [KOBS85a])

All basic beliefs and goals to be discussed here should be regarded as being embedded in a complex belief or goal of the system (a full description of the first belief would read, for instance, that the system believes that the user knows the concept 'DIPLOCOCCUS-PNEUMONIAE').

Beliefs of type (1.1.2.) express that an agent definitely does not believe that a fact p holds. An example would be

(8) ... the user *does not believe* that the system is printing file F1.

This belief would be a pragmatic presupposition of the user's command 'Print F1' and is inferred by default in the VIE-DPM system when such a command is entered. Remember that all belief descriptions are to be interpreted *de dicto* here, and that p describes a fact in the situation. Descriptions of the form 'a does not believe that p' and 'a believes that not p' are then equivalent.[4] Seen from a's perspective, the belief reports 'I do not believe that p' and 'I believe that not p' are equivalent, apart from possible pragmatic differences regarding the emphasis of assertion and the strength of the negative force (cf. [HORN78]). (8) is thus equivalent to

(9) ... the user believes that the system is not printing file F1.

Moreover, it should be stressed that entries in the user model expressing the assumption that a user does not know a concept [SLEE85, FINI86, FINI*, CHIN86, CHIN*] do not form beliefs of type (1.1.2.), but are "not-beliefs" *of the system* about beliefs of the user (more precisely, they are beliefs of the type S (3.1.2.) U (1.1.1.),[5] which will be dealt with later on).

Beliefs of type (1.1.3.) occur when the assumption that the user is uncertain about some fact is to be represented in the user model. In VIE-DPM, such an assumption is made, for example, based on the user's question to the system *whether p is the case*. This explicit representation of a *user's uncertainty* must be discerned from the case that there is no explicit information in the user model about whether the user believes or does not believe p. In the latter case, it can at best be *inferred* that the *system* is uncertain whether the user believes, does not believe, or is uncertain whether p (these are beliefs of type S (3.1.3.) U (1.1.1.)–(1.1.3.),[6] which will be discussed below). In most representation systems, neither kind of uncertainty can be explicitly represented.

Beliefs of type (1.1.4.) are most important in the case of Y/N-questions of both the user and the system. In ARGOT [ALLE80, ALLE83b] and VIE-DPM [KOBS85a], a precondition for the applicability of a Y/N-question is the belief

[4] This equivalence does not hold if p, instead of being a description of a fact in the situation, is itself a belief or goal description (which must also be interpreted *de dicto*). In this case, the general rule applies that 'a does not believe that p' can be inferred from 'a believes that not p', but not vice versa (cf. [PART70, KOBS85a]).

[5] 'a1 (3.1.2.) a2 (1.1.1.)' is a concatenated belief description which characterizes a *subtype* of type (3.1.2.) beliefs.

[6] '(1.1.1.)–(1.1.3.)' signifies that '(1.1.1.)', '(1.1.2.)' or '(1.1.3.)' can be selected at this point.

that the addressee knows whether p. Thus, when the user enters such a question whether p, the system assumes s/he believes that the system knows whether p. On the other hand, before posing such a question to him/her, the system checks whether it believes the user to know whether p.

3.2. Basic Unsaturated Beliefs

Basic unsaturated beliefs are similar to saturated ones except for the fact that existential quantification occurs in their formalization in predicate logic. In other representational schemes, e.g. that of VIE-DPM [KOBS85c], no specific additional construct need be introduced for their representation. Basic unsaturated beliefs can be divided into 5 types:

(1.2.1.) ... *believes that* $\exists$ x p(x).
(1.2.2.) ... *does not believe that* $\exists$ x p(x).
(i.e. believes that not $\exists$ x p(x).)
(1.2.3.) ... *is uncertain whether* $\exists$ x p(x).
(1.2.4.) ... *knows whether* $\exists$ x p(x).
(i.e. has a definite attitude as to whether $\exists$ x p(x).)
(1.2.5.) ... *knows the x such that* p(x).
(i.e. knows more about the x such that p(x).)

Beliefs of type (1.2.1.) are among the presuppositions of wh-questions. For example, when the user asks the VIE-DPM system for the x such that p(x), the system infers, among other things, that the user *believes that* $\exists$ x p(x). This presupposition in the user's question could then be checked against the system's knowledge (cf. [KAPS79, KAPS83]). Beliefs of types (1.2.2.) – (1.2.4.) are rather similar to their saturated counterparts (1.1.2.) – (1.1.4.). Beliefs (1.2.2.) and (1.2.3.) have not been dealt with in user modeling components developed to date, though both also occur in domains which are currently being investigated. A belief of type (1.2.3.), for instance, is among the presuppositions of the Y/N-question whether there is an x such that p(x) (e.g., "Is there a train to Toronto tomorrow morning?"). Cooperativity requires that somebody who poses this question must be *uncertain whether* $\exists$ x p(x).

Beliefs of type (1.2.5.) are also among the presuppositions of wh-questions. When the user asks the VIE-DPM system for the x such that p(x), the assumption is made that the user *believes* the system to "*know* this x". Conversely, before asking the user such a question, the system checks whether it *believes* the user to know this x. There has been a long debate in the fields of epistemic logic and the philosophy of language as to what it means for a modeling agent to attribute to a modeled agent the knowledge of the x such that p(x). Kaplan [KAPD69], Konolige [KONO81] and Appelt [APPE85] argue that this means that the modeling agent assumes the modeled agent to possess a "*standard name*" or "*rigid designator*" for x, i.e. a term denoting x in all possible worlds. Allen [ALLE83a] and Haas [HAAS86] convincingly refute such a view and demonstrate that the information an agent must possess about an x in order to "*know*" it (e.g. its name, its address, ways to find it, etc.) is essentially context-dependent. The only necessary (ergo context-independent)

requirement for the information which the modeled agent is assumed to have is that it would allow the modeling agent to uniquely identify x in the situation. Thus the belief description (1.2.5.) should be regarded as being synonymous with "...believes more facts about the x such that p(x), which would allow the modeling agent to identify x".

4. Basic Goals

Basic goals are similar to basic beliefs in that they are about a situation and not about beliefs and goals of other agents. Unlike beliefs, however, they are not about the current situation but about a desired situation. Thus, for an agent to have a goal p means to desire a situation in which p is the case. As is the case for the above belief descriptions, goal descriptions are also to be read in a *de dicto* sense.

4.1. Basic Saturated Goals

Basic saturated goals are about a fact p in a desired situation. As is the case for basic saturated beliefs, p's formalization in predicate logic must not contain free variables or existential quantifiers. Four types of basic saturated goals can be discerned:

(2.1.1.) ... *wants* p.
(2.1.2.) ... *does not want* p.
(i.e. wants p not to be the case.)
(2.1.3.) ... *has no preference with respect to* p.
(2.1.4.) ... *wants to know whether* p.
(i.e. wants to have a definite attitude as to whether p.)

Goals of type (2.1.1.) – (2.1.3.) are rather similar to the corresponding beliefs (1.1.1.) – (1.1.3.). In HAM-ANS [MORI85a] and [MORI*], goals of type (2.1.1.) are attributed to the user in order to predict (and possibly influence) his/her decisions with respect to hotel rooms to be rented. In ARGOT [COHE79, ALLE83b], KAMP [APPE85] and TRACK [CARB83, CARB85b, CARB*], assumptions about such basic goals are compiled into hypothesized user plans. Similar to the belief description (1.1.2.), the goal description (2.1.2.) is equivalent to 'wants not p', when read in the *de dicto* interpretation. Such goals occur in Martins' [MARS83] MBR system which represents, for example, that an agent does not want the faculty meeting to be held in the morning. (Actually, Martins only talks about "belief spaces"; however, some of the formulas contained therein may be more profitably regarded as descriptions of an agent's goals.) It seems that goals of type (2.1.3.) have not been employed up till now in any user modeling components. Assumptions relating to this type might be important in a consultation process in which the system tries to determine whether the user will agree with the side-effects of a plan proposed to him/her for the achievement of his/her principal goal.

Goals of type (2.1.4.) are fundamental in the field of speech act recognition and planning dialog (ARGOT, VIE-DPM). System beliefs about such user goals

are generated, for example, from Y/N questions of the user; they then form the starting point for appropriate search and answer generation processes of the system. In ARGOT and TRACK they are also used for inferring further goals of the user, for example that the user wants p or does not want p (beliefs (2.1.1.) or (2.1.2.), respectively). In UC [WILE86, CHIN*], the assumption that the user does not know p (i.e., a belief of type (3.1.2.)) is inferred.

4.1.1. Basic Unsaturated Goals

Besides basic saturated goals, there exist 5 types of basic unsaturated goals:

(2.2.1.) ... *wants* $\exists$ x p(x).

(2.2.2.) ... *does not want* $\exists$ x p(x).
(i.e. wants it not to be the case that $\exists$ x p(x).)

(2.2.3.) ... *has no preference with respect to* $\exists$ x p(x).

(2.2.4.) ... *wants to know whether* $\exists$ x p(x).
(i.e. wants to have a definite attitude as to whether $\exists$ x p(x).)

(2.2.5.) ... *wants to know the x such that* p(x).
(i.e. wants to know more about the x such that p(x).)

Goals (2.2.1.) – (2.2.4.) are rather similar to goals (2.1.1.) – (2.1.4.), the desired situation merely being "vaguer" in the unsaturated goals. System assumptions about user goals of type (2.2.5.) are important for user modeling since they form the basis for dealing with wh-questions of the user (cf. [ALLE80, ALLE83b, CARB83, CARB85b, APPE85]). In VIE-DPM, the conceptual representation of a user's NL wh-question is transformed into a representation of a user's belief of type (2.2.5.) [KOBS85a, KOBS86a]. Such a belief, wherever it stems from, then gives rise to a search process in the system's knowledge base, and eventually to the generation of an appropriate response. As was already discussed for beliefs of type (1.2.5.), however, no further context-independent interpretation can be given to goal description (2.2.5.) other than that the respective agent wants to believe more facts about x, so that x can be discerned from all other objects the agent knows about. What exactly an agent wants to know about such an x (its name, its address, ways to find it) can only be inferred from the context (if at all), and can only then be explicitly represented.

5. Beliefs with Respect to Other Agents' Beliefs and Goals

The beliefs which an agent holds may refer not only to the current situation, but also to beliefs and goals of other agents. The beliefs of a dialog system about the user's beliefs and goals are of that kind. Beliefs about beliefs can, in principle, be arbitrarily embedded, i.e. an agent a1 may hold beliefs about the beliefs/goals of an agent a2 about the beliefs/goals of an agent a3, etc. One may suppose, however, that arbitrarily complex beliefs do not occur in dialog situations, due to resource limitations. Current work in user modeling and dialog planning also provides practical

evidence for the existence of such a limitation: in the planning of dialog contributions, the depth of nesting hardly ever goes beyond the levels of 4-5 (cf. [SEAR69, COHE79, ALLE83b, AIRE83, AIRE84, KOBS85a]).

The symbol '(1.1.1.) – (4.2.5.)' at the ends of the belief descriptions to follow may be replaced by any belief or goal description. Thus, arbitrarily complex descriptions can be generated. Every description of such a complex belief, of course, must include the description of a *basic* belief or goal.

The general case of belief and goal embedding will probably not be so frequent in the field of user modeling. For some examples of beliefs about "third-party beliefs", i.e. agent beliefs in general, see [WAHL*, SPAR84, SPAR*]. Much more important is the special case of reflexive beliefs, in which a1 and a3 are identical.[7] Their most frequent subtypes include the so-called *infinite-reflexive* and, more specifically, *mutual beliefs,* which will be dealt with later on.

5.1. Beliefs with Respect to Beliefs and Goals of Known Agents

There exist 5 types of beliefs with respect to beliefs and goals of known agents:

(3.1.1.) ... *believes that* a (1.1.1.) – (4.2.5.)
(3.1.2.) ... *does not believe that* a (1.1.1.) – (4.2.5.)
(i.e. believes that not a (1.1.1.) – (4.2.5.))
(3.1.3.) ... *is uncertain whether* a (1.1.1.) – (4.2.5)
(3.1.4.) ... *knows whether* a (1.1.1.) – (4.2.5.)
(i.e. has a definite attitude as to whether a (1.1.1.) – (4.2.5.))
(3.1.5.) ... *believes that* a (1.1.1.) – (4.2.5.) *and believes that* a *believes that* ... (3.1.5.)

Beliefs (3.1.1.) – (3.1.4.) correspond to beliefs (1.1.1.) – (1.1.4.) except for the fact that they are about beliefs or goals, not about situations. It should be noted that a belief of the form a1 (3.1.1.) a2 (1.1.2.) is different from a1 (3.1.2.) a2 (1.1.1.), and a1 (3.1.1.) a2 (1.1.3.) is different from a1 (3.1.3.) a2 (3.1.1.). Thus, for example, the following pairs of belief descriptions express different facts when interpreted in our standard *de dicto* sense:

(10) S *believes that* U *does not believe that* Johnsons Inc. have ordered three F34 engines.
(11) S *does not believe that* U *believes that* Johnsons Inc. have ordered three F34 engines.
(12) S *believes that* U *is uncertain whether* Johnsons Inc. have ordered three F34 engines.
(13) S *is uncertain whether* U *believes that* Johnsons Inc. have ordered three F34 engines.belief types

[7] I will not deal with so-called "meta-beliefs" here (i.e. beliefs which can be described, for example, by "a1 believes that a1 believes ...") since their role in user modeling is still rather unclear (but see [CHIN*]).

In cases (10) and (12), S believes that U's beliefs could be described from U's point of view by the descriptions "I do not believe (I am uncertain whether) Johnsons Inc. have ordered three F34 engines". In cases (11) and (13), the system does not believe (is uncertain whether) this could be done. The inferences that can be drawn in each case are different. In possible-world representations (e.g. [HALP85a, MOOR80, APPE85]), which are mostly concerned with "knowledge" rather than beliefs, beliefs such as (10) can be represented, whereas beliefs such as (11) cannot.

A specific constellation of reflexive beliefs is type (3.1.5.), the so-called *infinite-reflexive* belief [KOBS84], which seems to be quite fundamental in belief and goal recognition as well as dialog planning. An implicit requirement in the belief description of (3.1.5.) is that the agent who instantiates 'a' must be different from that entered at '...'. An example for an infinite-reflexive belief is

(14)(a) S *believes that* U *wants to know whether* John loves Mary, and
(b) S *believes that* U *believes that* S *believes that* U *wants to know whether* John loves Mary,
(c) etc.

(14) is part of the inferences which the VIE-DPM system draws about the user when s/he enters the Y/N-question "Does John love Mary?". Beliefs of type (3.1.5.) should not be seen as an actual infinite set of reflexive beliefs but rather as *primitive beliefs* from which each member of this set can be inferred when needed.

A specific belief constellation composed of infinite-reflexive beliefs and a basic belief is the so-called *mutual belief*. In our notation, such a belief can be defined as follows:

(15)(a) a1 (1.1.1.)
(b) a1 (3.1.5.) [a2 (1.1.1.)]
(c) a1 (3.1.1.) a2 (3.1.5.) [a1 (1.1.1.)]

The elements in square brackets characterize the type of belief description that must be entered in (3.1.5.) at the place denoted by '(1.1.1.) – (4.2.5.)'. An example for such a mutual belief would be:

(16)(a) S *believes that* Peter loves Mary, (from 15a)
(b) S *believes that* U *believes that* Peter loves Mary, (from 15b)
(c) S *believes that* U *believes that* S *believes that* Peter loves Mary, (from 15c)
(d) S *believes that* U *believes that* S *believes that* U *believes that* Peter loves Mary, (from 15b).
(e) etc.

It can be shown easily that definition (15) is equivalent to the shorter classical one which reads as follows:

(17)(a) a1 *believes that* p.
(b) a1 *believes that* a2 *believes that* p, (a) and (b).

Mutual beliefs have been intensively investigated in the fields of philosophy of language [LEWI69, SCHI72], [CLHH81, SMIB82], psychology [CLHH83], artificial intelligence [PERR81, JOSH82, NADA83], and distributed databases [LEHD84,

HALP85b]. In the field of user modeling, however, the concept of mutual beliefs seems to be somewhat too narrow. For example, the inferences which can be drawn about a user's goals and beliefs from his/her dialog contributions do not form mutual, but infinite-reflexive beliefs. The same holds true for the assumptions which the user makes about the system's beliefs and goals as a result of its dialog contributions (these assumptions must be anticipated by the system in the process of dialog planning).

Since infinite-reflexive beliefs are so fundamental in dialog systems equipped with a user modeling component, a user model should provide simple means for representing them. Currently, infinite-reflexive beliefs can be modeled in ARGOT and VIE-DPM only. Appelt's [APPE85] KAMP system and, to a limited extent, Konolige's [KONO81] representation allow mutual beliefs to be expressed. Also, the beliefs that are modeled in HAM-ANS are interpreted by Hoeppner & Morik [HOEP83a] as being mutual beliefs.

5.2. Beliefs with Respect to Beliefs and Goals of Unknown Agents

There exist 5 types of beliefs with respect to beliefs and goals of unknown agents:

(3.2.1.) ... *believes that there exists an a1 such that* a1 (1.1.1.) - (4.2.5.)

(3.2.2.) ... *does not believe that there exists an a1 such that* a1 (1.1.1.) – (4.2.5.)
(i.e. believes that no a1 exists such that a1 (1.1.1.) – (4.2.5.))

(3.2.3.) ... *is uncertain whether there exists an a1 such that* a1 (1.1.1.) – (4.2.5.)

(3.2.4.) ... *knows whether there exists an a1 such that* a1 (1.1.1.) - (4.2.5.)
(i.e. has a definite attitude as to whether there exists an a1 so that a1 (1.1.1.) – (4.2.5.))

(3.2.5.) ... *knows the a1 such that* a1 (1.1.1.) – (4.2.5.)
(i.e. knows more about the a1 such that a1 (1.1.1.) – (4.2.5.))

In the field of user modeling, these types of beliefs are probably not so important. They might be relevant in the fields of multiple-agent planning systems and distributed databases since it seems desirable for a robot/processor to be able to represent the fact that some other robot/processor knows who knows some fact, whether there exists some agent who knows some fact, etc. Currently, such assumptions can be represented only in VIE-DPM, but are not put to use.

6. Goals with Respect to Other Agents' Beliefs and Goals

6.1. Goals with Respect to Beliefs and Goals of Known Agents

There exist 4 types of goals with respect to beliefs and goals of known agents:

(4.1.1.) ... *wants* a (1.1.1.) – (4.2.5.)

(4.1.2.) ... *does not want* a (1.1.1.) – (4.2.5.)
(i.e. wants it not to be the case that a (1.1.1.) – (4.2.5.))

(4.1.3.) ... *has no preference with respect to whether it should be the case that* a (1.1.1.) – (4.2.5.)

(4.1.4.) ... *wants to know whether* a (1.1.1.) – (4.2.5.)
(i.e. wants to have a definite attitude as to whether a (1.1.1.) – (4.2.5.))

Except for (4.1.3.), all these goals are very important in user modeling. They frequently occur in the recognition process for user intentions and in the generation of dialog contributions of the system. In ARGOT, for instance, the user's assertion *that p* leads to the mutual belief that the user wants p to be believed by the system (i.e. U (4.1.1.) S (1.1.1.)). In VIE-DPM, a user's question *whether p* leads to the assumption that the user wants it to be mutually believed whether or not the system believes p (i.e. ... U (4.1.4.) S (1.1.1.)). On the other hand, in the process of dialog planning, VIE-DPM produces an assertion that p upon discovering the following beliefs in its knowledge base:

(17)(a) The system *believes that* the user *wants to know whether* the system *believes* p (i.e. S (3.1.1.) U (4.1.4.) S (1.1.1.)).
(b) The system *believes* p (i.e. S (1.1.1.)).

Goals of type (4.1.3.) currently do not seem to be relevant in user modeling. However, as soon as one takes aspects of privacy into account, it might become interesting to model the assumption that the user doesn't care if some third party knows some fact about him/her. At the moment, such beliefs can be represented in KAMP and VIE-DPM only.

6.2. Goals with Respect to Beliefs and Goals of Unknown Agents

5 types of goals with respect to beliefs and goals of unknown agents can be discerned:

(4.2.1.) ... *wants* ∃ a1 *such that* a1 (1.1.1.) – (4.2.5.)

(4.2.2.) ... *does not want* ∃ a1 *such that* a1 (1.1.1.) – (4.2.5.)
(i.e. wants no a1 to exist such that a1 (1.1.1.) – (4.2.5.))

(4.2.3.) ... *has no preferences with respect to whether there should be an a1 such that* a1 (1.1.1.) – (4.2.5.)

(4.2.4.) ... *wants to know whether there exists an a1 such that* a1 (1.1.1.) – (4.2.5.)
(i.e. wants to have a definite attitude as to whether there exists an a1 such that a1 (1.1.1.) – (4.2.5.))

(4.2.5.) ... *wants to know the a1 such that* a1 (1.1.1.) – (4.2.5.)
(i.e. wants to know more about the a1 such that a1 (1.1.1.) – (4.2.5.))

The goals of this group do not seem to be relevant for user modeling since beliefs or goals of unknown third parties currently do not belong to the domain of discourse of artificial intelligence dialog systems. This might change, however, as soon as privacy considerations are taken into account. Goals of type (4.2.4.) and

(4.2.5.) are probably interesting for the fields of multiple-agent planning systems and distributed databases when a robot or processor is searching for some other who shares its assumptions. Currently, such goals can be represented only in VIE-DPM, but are not put to use.

7. A Classification of Existing User Models

In this section, a tentative classification of several existing user models is presented with respect to the proposed belief and goal taxonomy (cf. Table 1). Its purpose is to render transparent the sorts of beliefs and goals being modeled in the respective systems, so that designers of user models can compare their application with that of existing systems. Emphasis was placed on systems in which a greater number of beliefs and goals have been modeled.

This classification should not be regarded as the last word in the matter, since it is somewhat difficult to judge whether a certain belief or goal type is modeled in a given system. A necessary condition was that the representational scheme underlying the user model provide formal means for an *explicit* representation of the particular belief or goal type. This, of course, does not automatically mean that the representation is complete or fully adequate. For example, some systems (GUMS, HAM-ANS, UMFE) provide means for representing the system's beliefs about a user's beliefs or goals (i.e. beliefs of the form S (3.1.1.) U (1.1.1.) or U (2.1.1.)); their representation, however, does not extend to beliefs of the user about third parties' beliefs or goals. In other systems (e.g. KAMP, VIE-DPM), a number of belief and goal types that can be represented in the user model do not actually occur in the current operation of the system. Therefore, a third category has been introduced in Table 1 to mark those belief and goal types that are representable in the knowledge base of the user modeling component, but not yet put to use.

The most striking result of the comparison in Table 1 is that systems which exploit the user model for *explicit* speech act recognition and dialog planning (i.e., ARGOT, KAMP, VIE-DPM) have to model far more beliefs and goals than systems which only employ the user model for interacting with the user in a more personalized way (e.g. by taking into account his/her presumed goals or background knowledge). Systems in the latter application domain use an explicit representation only for beliefs of the form S (3.1.1.) – (3.1.3.) U (1.1.1.) & (2.1.1.). An intermediate case is the Unix Consultant (UC), which analyzes and generates a restricted number of speech acts only [WILE86, CHIN86, CHIN*].

This observation might have implications for the design of general user modeling systems, i.e. shell systems that are to be employed in a large number of different dialog systems (cf. [FINI*]). It seems that designers of such tools must decide whether or not their system should support the explicit recognition and planning of dialog contributions as well. In this case, the system must possess a considerably greater expressive power for belief and goal representation than is required otherwise.

Table 1. A comparison of existing user models with respect to the belief and goal taxonomy

	ARGOT Speech Act Recognition and Planning in Train Domain	GUMS Inferences about Users Beliefs Shell System	HAM-ANS Cooperative Dialog in Hotel Reservation	KAMP Planning Referential Expressions in Assembly Domain	MBR Conflict Resolution in Time Scheduling	TRACK User Plan Recognition and Cooperative Response in Course Scheduling	UC Help System for Unix	UMFE Personalized Explanation of Medical Reasoning	VIE-DPM Inferring Users Beliefs and Goals, Speech Act Planning, Shell System
(1.1.1.)	×	×	×	×	×	×	×	×	×
(1.1.2.)	×	—	—	×	×	×	×	—	×
(1.1.3.)	—	×[1]	—	×	—	—	—	—	×
(1.1.4.)	×	—	—	×	—	—	—	—	×
(1.2.1.)	×	—	—	•	—	×	—	—	×
(1.2.2.)	×	—	—	•	—	—	—	—	×
(1.2.3.)	—	—	—	•	—	—	—	—	•
(1.2.4.)	×	—	—	•	—	—	—	—	×
(1.2.5.)	×	—	—	×	—	—	—	—	×
(2.1.1.)	×	—	×	×	×[4]	×	×	—	×
(2.1.2.)	×	—	—	×	×[4]	•	×	—	×
(2.1.3.)	—	—	—	×	—	—	—	—	•
(2.1.4.)	×	—	—	•	—	×	•	—	•
(2.2.1.)	×	—	•	•	—	—	—	—	×
(2.2.2.)	×	—	—	•	—	—	—	—	×
(2.2.3.)	—	—	—	•	—	—	—	—	•
(2.2.4.)	×	—	—	•	—	—	—	—	•
(2.2.5.)	×	—	—	×	—	×	×	—	×
(3.1.1.)	×	×	—	×	×	×	×	×	×
(3.1.2.)	×	×	—	—	—	×	×	×	×
(3.1.3.)	—	×	—	×	—	—	×	×	×
(3.1.4.)	—	—	—	—	—	—	—	—	•
(3.1.5.)	×	—	×[2]	×[2]	—	—	—	—	×
(3.2.x.)	—	—	—	—	—	—	—	—	•
(4.1.1.)	×	—	—	×	—	—	×	—	×
(4.1.2.)	×	—	—	—	—	—	×	—	×
(4.1.3.)	—	—	—	•	—	—	—	—	•
(4.1.4.)	—	—	—	•	—	—	—	—	•
(4.2.x.)	—	—	—	—	—	—	—	—	•

× modeled by the user modeling component

• representable in the user model but not put to use

— not modeled

[1] Envisaged.

[2] According to Hoeppner & Morik (1983), all beliefs of the system about the user are to be regarded as mutual beliefs.

[3] Mutual beliefs only.

[4] Martins only talks about "belief spaces", some of the formulas contained therein may be more profitably regarded as descriptions of an agent's goals.

8. Discussion

The belief and goal classification presented in this chapter should be primarily regarded as having a more *pre-theoretical* character. A theoretically satisfying classification would have to prove that the belief and goal types discerned are both *necessary* and *sufficient* for user modeling in dialog systems. Evidence for their necessity has been given by demonstrating where they occur in current user modeling components, or could occur in future systems. A proof for the sufficiency of the proposed taxonomy (i.e. a proof that there is no other belief or goal type apart from (1.1.1.) – (4.2.5.)), however, would presuppose a deeper understanding of belief and goal formation processes in person-to-person communication than we currently have.

Of course, the above taxonomy can be refined by also taking into account *additional classification criteria.* In the above belief and goal descriptions, for example, the *content* of the beliefs and goals has been totally ignored (i.e. the structure of p has not been further analyzed). However, there is certainly a difference between believing, say, general vs. individual facts (cf. (3) – (7)). Rich [RICH79a, RICH79b], for instance, has proposed that one discern whether an entry in the user model represents a highly specific piece of information or longer-term, more general information. Other possible dimensions in terms of which p can be classified might be suggested by classifications of the underlying representation of the facts described by p. Different representational instruments are employed when p contains quantifiers, disjunction, conjunction, etc. More on this can be found in the literature on belief and goal representation (cf. [HALP85a, KOBS85a, HALP86, WAHL*] for surveys).

The proposed taxonomy imposes demands on representational schemes for beliefs and goals in that all of the above belief and goal types should be expressible in more general representational systems. As a matter of fact, formal representation requires rendering the structure of the above belief and goal types with even more precision. For example, the above-mentioned analysis of what it means to know the x such that p(x) was initiated when inconsistencies in superficial *ad hoc* formalizations of this belief type were detected. One may suppose that, in the long run, we may knock over the ladder after having climbed it and replace the familiar and cherished belief and goal talk by more precise representational notions. Also, an intermediate "epistemological" level as, for example, possible-world theory or that proposed in [KOBS85c] might be an interesting alternative. At the moment, however, there is little consensus as to "the" correct belief and goal representation, and the explanatory power and compactness of linguistic belief descriptions is still so great that this step does not yet seem advisable.

9. References

[AIRE83], [AIRE84], [ALLE80], [ALLE83a], [ALLE83b], [APPE85], [BARN86], [BURG78], [CARB83], [CARB85b], [CARB*], [CHIN86], [CHIN*], [CLHH81], [CLHH83], [COHE79], [DEDE86], [FINI86], [FINI*], [GEOR83], [HAAS86], [HALP85a], [HALP85b], [HALP86],

[HOEP83a], [HORN78], [JOSH82], [KAPD69], [KAPS79], [KAPS83], [KOBS84], [KOBS85a], [KOBS85c], [KOBS85d], [KOBS86a], [KONO80], [KONO81], [KRAU83], [LEHD84], [LEWI69], [MARS83], [MOOR80], [MOOR82], [MORI85a], [MORI85b], [MORI*], [NADA83], [PART70], [PERR81], [POWE79], [QUIN56], [RAPA84], [REIS85], [RICH79a], [RICH79b], [ROSE82], [ROSE85], [SCHI72], [SEAR69], [SLEE82c], [SLEE85], [SMIB82], [SPAR84], [SPAR*], [STIC83], [WALL84], [WAHL*], [WERN88], [WILE86]

Part II

Building User Models

Introduction

The recognition of relevant facts about the user, including most prominently his/her beliefs, background knowledge, goals and plans, is a crucial research issue in the field of user modeling. The effectiveness of a user modeling component in enhancing a system's cooperativity essentially hinges on its ability to draw correct assumptions about the user in the course of the dialog, and to decide which of them are relevant to the dialog purpose at hand.

As can be seen from [WAHL*], a number of construction techniques have been developed in the field of user modeling (possible limitations of which are discussed in [SPAR*] and [KASS*]). The chapters in this part give a detailed account of how such techniques have been put to use in a number of user modeling components developed for various application domains. In addition, an outline is presented as to how the acquired assumptionsassumptions, about the user are exploited by the system. The focus, however, lies on recognition aspects.

David Chin is concerned with the development of KNOME, the user modeling component of the UNIX Consultant (UC) system. UC is an online natural-language advisory system for the UNIX operating system (a related application domain is presented in the work of [QUIL*] which is summarized below). Users can ask UC how to do things in UNIX, request definitions of UNIX terminology, and obtain help debugging problems with command usage. In its interaction with the user, UC should take his/her level of expertise into account.

KNOME's user model contains two sorts of assumptions about the user's knowledge concerning UNIX, namely the expertise stereotype probably applying to the user, and additional individual assumptions about him/her. In the KNOME approach, four expertise stereotypes (novice, beginner, intermediate, expert) are discerned along which users are classified, and four difficulty classes exist for the classification of UNIX commands, command formats, terminology and other relevant information. Assumptions are associated with each expertise stereotype concerning the plausibility of a user belonging to a certain stereotype knowing UNIX-related information of a certain difficulty level. Chin thus uses a *double* stereotype approach (cf. [RICH*]).

The individual part of the user model contains assumptions about the user's UNIX knowledge that were directly inferred from his/her natural language input. Chin discerns six deduction types for inferring what the user probably knows or does not know (including, for instance, the rules 'user mentions the concept ?x → user knows the concept ?x' and 'user wants to know ?x → user does not

know ?x'). In the early stages of the dialog, the inferred assumptions are used for determining the plausibility that a given stereotype applies to the user, and, after enough evidence has been collected (typically after three interchanges), they are used for making the ultimate decision on the most plausible stereotype. As soon as this decision has been made, new individual assumptions may only override the stereotype, but not cause its replacement by some other. Also, individual assumptions are then explicitly represented only when different from the stereotype.

The assumptions which are represented in KNOME's user model are exploited by various UC components. They are used, for example, in determining the correct interpretation of a (possibly indirect) speech act, for expressing information in terms of only such concepts with which the user is already familiar, or for pruning away those parts of an answer that the user already knows. "Meta-knowledge" about whether the system knows all options of a UNIX command (i.e., local closed world assumptions) can be used for correcting users' inquiries regarding options unknown to the system.

Alex Quilici is also involved with the development of a help system for UNIX. His envisaged system should allow the user to report problems which s/he has had in interaction with UNIX, and provide cooperative advice. Quilici concentrates on the recognition of misconceptions about UNIX inferable from these reports, and on the selection of appropriate responses. In particular, he considers misconceptions involving plan applicability conditions (whether a particular plan should be used to achieve a goal), enablements (whether a particular state must exist before a plan can achieve a goal), and effects (whether a state will exist as a result of a plan execution).

The basic knowledge representation scheme in Quilici's model is a plan representation containing action operators and associated applicability conditions, enablements, and effects. Various forms of knowledge are expressed within this scheme:

1. the system's knowledge about preconditions and effects of UNIX commands, including knowledge about the effect of command usage in the case of unfulfilled preconditions;
2. the user's beliefs and goals as inferred from his/her input;
3. a number of beliefs about UNIX commands which the system does not hold, particularly frequent misconceptions of the user (i.e., some sort of "bug catalog" as used in the field of computer-aided instruction – cf. [KASS*]); this knowledge is stored along with (1);
4. an uninstantiated user belief pattern and a number of so-called 'explanations' for each type of user misconception discerned (namely misconceptions concerning command applicability, enablements and effects); each explanation consists of a number of uninstantiated system belief patterns which together "contradict" the user belief pattern (e.g. the system belief patterns 'Ap causes So' and 'Ao causes Sp' may explain why the system does not share a user belief of the form 'action Ap causes state Sp').

Quilici's model assumes that the user's natural-language report about his/her problems in the use of UNIX has already been analyzed, and that a number of user beliefs and goals have been recognized and entered into the user model. For each

user belief, the system first determines whether it shares it by searching its knowledge base for the belief, or whether it is a common misconception by searching the misconception catalog for its negation. If neither search is successful, the system assumes that the user's belief is wrong (see [CHIN*], however, for a critique of this so-called 'closed world assumption').

When an incorrect belief is detected in the misconception catalog, the system can immediately correct the user. Otherwise, the misconception type is determined by matching the user belief patterns with the incorrect belief. The explanations associated with the determined misconception type are then tested as to whether all their system belief patterns (which became partly instantiated in the first step) have a match in the system's knowledge base. When an explanation matches completely, a divergence between system knowledge and user beliefs has been recognized about which the user can be informed.

The contribution by **Sandra Carberry** is concerned with the recognition of users' plans in an ongoing dialog. In her model, a number of domain-independent heuristics are used for inferring candidate goals and plans from user statements (these heuristics are extensions of inference rules proposed by [ALLE80] and [ALLE83]). Focusing strategies are then employed for selecting the candidate goals and plans most related to the current dialog context and for integrating them into a model of the user's actual plan. These strategies are based on recent research on dialog organization and focusing done, for example, by [MCKE85a, MCKE85b, GROS86].

Even before all its details have been determined, the assumed user plan can be exploited for generating expectations about dialog contributions of the user. Carberry's model uses this potential for handling two forms of linguistically problematic dialog contributions, namely pragmatically ill-formed sentences and intersentential ellipsis (only the former is described in this contribution, however). Pragmatically ill-formed user sentences, in her definition, are syntactically and semantically correct, but do not fit the system's model about the domain. Carberry discusses pragmatic ill-formedness resulting from attribute confusions and missing specifications in a user's question, and proposes repair strategies based on the expectations from the user model.

Up till now, nearly all plan recognition models have assumed that the system's model of the user's plan is perhaps incomplete, but never wrong. In a final section, Carberry discusses how her analysis could be extended to be able to detect clues indicating disparities between the system's model of the user and the user's actual goals and plans, reasoning on the system's model and the system's domain knowledge to form hypotheses as to the source of these disparities, responding to the user and negotiating with him/her to isolate the errors, and appropriately repairing the user model.

The work of **Jill Fain Lehman** and **Jaime Carbonell** demonstrates that the field of user modeling cannot confine itself to models which only include assumptions about "objective facts" concerning the user, or reconstruct his/her beliefs, goals and plans (of which the user is probably aware). Instead, other kinds of cognitive competence (which are possibly unconscious to the user) must sometimes also be reconstructed by the system to enable it to adapt to the user in a cooperative way.

An example of such a system is the CHAMP parser for natural language that adapts to the idiosyncratic linguistic usage patterns of a user and stores them in a user model. Its central knowledge base consists of a so-called "kernel grammar", which comprises a lexicon, case-frame concept definitions, default semantic values and syntactic forms that are present prior to interaction with any user. The latter constitute a sort of ANY-PERSON stereotype (cf. [RICH*]) which contains the grammatical regularities in the input of all users. If a user's input cannot be parsed correctly because it is ungrammatical according to the syntactic forms currently known, recovery strategies are applied in a least-deviant-first manner both to letters in input words (this amounts to spelling correction) and to sentence constituents. The strategies used consist of a combination of four basic operations, namely insertion, deletion, substitution, and transposition. The parsing process is split for each recovery strategy that applies. If the application of a small number (typically not more than two) of such operations results in a successful parse, the user is asked to confirm the system's interpretation. If the interpretation is correct, the new syntactic form is subjected to a conservative generalization process, and then added to the user model.

The practicality of the CHAMP approach essentially hinges on the assumption that users' idiosyncratic language patterns vary a great deal (otherwise the kernel grammar could simply be extended to cover all users) and that the patterns exhibited by an individual user are bounded, given a specific task at hand. Both hypotheses were confirmed in a number of hidden-operator experiments carried out by Fain Lehman. In addition, counter-evidence was obtained through control experiments against the assumption that users would naturally adapt to the system's limitations faster than the system can usefully adapt to his/her idiosyncrasies.

4 KNOME: Modeling What the User Knows in UC[1]

David N. Chin

Abstract

KNOME is the user modeling component of UC, a natural language consultation system for the UNIX operating system. During the course of an interactive session with a user, KNOME infers the user's level of expertise from the dialog and maintains a model of the user's knowledge of the UNIX domain. KNOME's model of the user makes use of a *double-stereotype* system in which one set of stereotypes represents the user's expertise and another represents the difficulty level of the information. KNOME is used in UC to help disambiguate the user's statements, avoid telling the user something that the user already knows, take advantage of prior user knowledge in presenting new information, and detect situations where the user lacks pertinent facts or where the user has a misconception. UC also models its own knowledge of UNIX with *meta-knowledge* (explicit facts about the limitations of the system's own knowledge base), which is used to help in correcting user misconceptions.

1. Introduction

UC, the UNIX Consultant [WILE84, WILE86], is a natural language computer consultant system under development at the University of California at Berkeley. UC advises users on using the UNIX[2] operating system. Users can ask UC how to do things in UNIX, get definitions of UNIX terminology, and get help debugging problems in using commands.

To better accommodate a greater range of potential users, UC contains an explicit user modeling component called KNOME (KNOwledge Model of Expertise). Some of the early results of this research were reported in [CHIN86]. KNOME contains information about what users know. Using such information, UC can:

[1] This research was sponsored in part by the Defense Advanced Research Projects Agency (DoD), ARPA order No. 4871, monitored by the Space and Naval Warfare Systems Command under contract N00039-84-C-0089, and by the Office of Naval Research under contract N00014-80-C-0732.

[2] UNIX is a trademark of Bell Laboratories

1. avoid telling the user something that the user already knows
2. tailor explanations to the user's level of understanding
3. utilize the user's background knowledge in interpreting what the user says

KNOME is also helpful in detecting cases where the user's knowledge is incorrect (misconceptions) and cases where the user lacks needed information. In such cases, *UCEgo* [CHIN87], the component of UC that implements an intelligent agent for UC, can take the initiative and try to correct the user's misconception, or try to inform the user of the needed information.

1.1. Some UC Examples

To see how KNOME can infer a user's level of expertise from a conversation with the user, and how this information can be used advantageously by UC in helping the user, consider the two UC sessions shown in Figure 1 and Figure 2.

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type ''D' or '(exit)'.

Hi.
How can I help you?

# How can I delete a file?

Use rm.
For example, to delete the file named foo, type 'rm foo'.

# What does ruptime do?

ruptime is used to list the uptime of all machines on the network,
list the number of all users on them, and list their load average.
```

Figure 1. UC session with User1

The user of the first session, User1, is apparently a novice who knows very little about UNIX. After all, User1 does not even know something as simple as how to delete a file (the subject of User1's first question). On the other hand, the user in the second session, User2, knows what an inode is, a relatively complex UNIX concept. So User2 is probably a much more advanced user of UNIX than User1. KNOME uses this type of evidence about particular facts that users know or do not know in order to deduce their level of expertise.

Based on the different levels of expertise of the two users as deduced by KNOME, UC is able to provide qualitatively different answers to the two users.

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type "D' or '(exit)'.

Hi.
How can I help you?

# How can I find out the inode of a file?

Use ls -i.

# What does ruptime do?

ruptime is like uptime, except ruptime is for all machines on the
network.
```

Figure 2. UC session with User2

In the first answer to User1, KNOME gives an example of the format of the rm command (simply "rm" followed by the name of the file to be deleted). UC gives an example of the format to User1, since a user as unsophisticated as User1 probably would not know the format of rm. On the other hand, User2 is a more advanced user who undoubtedly already knows the format of ls. Hence UC is able to give a very concise answer to User2's first query, "Use ls -i," that does not include an example of the format of ls. This illustrates how KNOME is used to avoid telling the user something that the user already knows.

The second query of both users is the same. For the more advanced user, UC was able to explain the ruptime command in terms of a similar command, uptime, that the more advanced user presumably already knows. Such a simile would not work for User1, since a novice user probably would not know the uptime command, and so could not exploit the simile to understand ruptime. KNOME's modeling of the user's knowledge allows UC to choose whether or not to use different forms of presentation like similes based on whether or not the user will understand them.

1.2. Problems in Modeling

There are many difficulties in building and maintaining a model of what the user knows. The system must first determine what the user knows, and then represent this information internally. To determine the user's knowledge of the domain, the system could quiz the user exhaustively. However, for most domains, such a quiz would be extremely time consuming and most users would not consent to such extensive quizzing. Fortunately, users tend to learn domain knowledge in a predictable order, so the system can make additional inferences about what a user is likely to know or not know based on a partial user model. That is, the user modeling system should be able to *predict* a user's knowledge based on partial information about

what the user knows or does not know. Exactly which predictions/inferences are allowable is problematic. This is the key problem for any user modeling system that wishes to take advantage of the order in which users typically acquire knowledge in a domain.

Besides the basic question of which inferences are allowable, there is also the problem of uncertainty. Making inferences about the user based on limited prior knowledge about the user is a form of default reasoning. Such default reasoning is not always valid, so the system needs to be able to handle the fact that such inferences are uncertain. Usually this requires that the user modeler be able to judge the certainty of its default inferences. Also, as the system obtains more concrete information, it must be able to update its user model. The new information may contradict previous predictions, so maintenance is not simple.

Another problem concerns the organization of the inferences. Many inferences tend to be grouped together, because people tend to go through distinct stages of learning (see [KAYD85]). People at a particular stage of learning tend to have similar knowledge. Once a system has determined its user's level, it can assume that this user knows about as much as other users at that level. A good user modeling system needs to group such inferences together in a simple and efficient manner.

Another problem in user modeling is how to represent the model internally. The simplest method is the *overlay* model [CARR77] wherein the user's knowledge is represented as a subset of the system's knowledge. However, a simple overlay model cannot predict what a user might know based on partial information. An overlay model does not represent the order in which users typically learn information in a domain. What is needed is a model that represents such ordering information in a manner useful for prediction of what a user might know based on partial information.

1.3. Other User Models

User models have appeared frequently in Intelligent Tutoring Systems (e.g. [BURT76, SLEE82c]). [KASS*] provides a survey of user models in ITS. Typically, ITS use some variant of the simple overlay model [CARR77]. Overlay models encode the user's knowledge as a subset of the system's expert knowledge base. One variant is to encode users as a collection of differences from the system's built-in expert. Differences may include a lack of user knowledge relative to an expert. This is frequently augmented by a library of buggy procedures/rules that are common among users (e.g. [BROW78, STEV79, SLEE81, JOHN84, REIS85, ANDJ85b]). These types of user models are not designed to *predict* a user's knowledge based on partial information about what the user knows or does not know. This sort of default inference is not so appropriate for ITS where the system needs to keep an accurate user model. On the other hand, systems usually do not interact with a single user for as much time as tutoring systems, so they do not have the time to build a complete user model. In such cases, default reasoning is needed. Therefore, consultation systems need a different type of user model than is commonly found in ITS.

An exception is the hierarchical overlay model of USCSH [MATT86] which does some limited prediction. USCSH predicts the user's proficiency for parent nodes in the hierarchy by taking a weighted sum of the proficiency ratings of the children nodes. This approach allows prediction of highly related topics, but does not extend to prediction of the user's knowledge of less related topics. For example, such a system could predict that a user has high proficiency in process monitoring when the system knows that the user has high proficiency in using the ps, who and kill commands. However such a system could not predict the proficiency of the user in an unrelated topic, such as using the cat command, based on the system's knowledge of the user's proficiency in process monitoring commands.

Other user models have been built to model a number of aspects of the user (see [WAHL*] for a survey). For example, Grundy [RICH79b, RICH83, RICH*] modeled user personality traits. Grundy used user *stereotypes* to group default inferences about user traits, then used these traits for suggesting books to the user for reading. To determine which stereotypes were applicable to a particular user, Grundy asked the user for a self description at the start of a session. Individual phrases (e.g. self descriptions such as athletic or feminist) would *trigger* stereotypes that were likely to apply to that user. Such an approach worked well for modeling personality traits, but does not carry over to modeling domain knowledge. A user's self description as a "beginner" often will not coincide with the system's idea of a "beginner." Also, stereotype triggers cannot be used to make inferences such as inferring that the user does not know something when the user asks about it. Stereotype triggers are not sufficient for inferring what users know.

Another use of user models is for representing user preferences. The hotel reservation system of HAM-ANS [HOEP84b, MORI*] and the Real Estate Agent [MORI85b] chose different hotel rooms or apartments based on users' needs and preferences. These systems expect users to mention at the start of their session their requirements and preferences for a room/apartment. Such an approach works well in domains where users tend to specify their needs and preferences at the start of a session. However this is not the case in consultation domains where users are unlikely to mention their level of expertise in the domain at the start of a session.

Many systems including UC have modeled the user's plans and goals (e.g. [ALLE80, CARB83, CARB*, LITM84, JOHN84, WILE86]) Such systems infer the user's plans and goals by matching the user's actions to steps of prestored plans. This type of inference works well for detecting the user's plans, but does not extend to determining what the user does or does not know.

2. Internal Representation of Users

KNOME represents what UC believes users know about UNIX. In this sense, KNOME's model of the user is not necessarily meant to accurately model actual users, but is meant to conform to the way human UNIX consultants model their clients. So in developing KNOME's user models, no attempt was made to psychologically profile what different users know. Instead, former UNIX consultants (UC's implementors) were informally surveyed to determine how they viewed users.

2.1. Double-Stereotypes

Humans use categorization to organize inferences (see [ROSC77, ROSC78, ROSC83]). These categories serve as reference points or prototypes for judging objects. Once an object has been identified as belonging to a particular category, default inferences about the object can be made based on the object's membership in the category.

An approach similar to human categorization is used in KNOME to organize inferences concerning users. In KNOME, users are grouped according to their level of expertise in using UNIX. Each group is represented by a prototype-style category. There are four such categories in UC: *novice, beginner, intermediate,* and *expert.* Each of the categories represents an increasing mastery of UNIX information.

Individual users are represented as members of one of these categories and inherit properties from their category. Specific information about individual users overrides inherited information. These user categories are also termed stereotypes after the stereotypes of Grundy [RICH79b, RICH83, RICH*], which pioneered categorization of users in computer programs. However, unlike the stereotypes in Grundy, which consist of collections of attributes, the stereotypes in KNOME are implemented as categories within the KODIAK representation system [WILE87]. As such, KNOME's stereotypes are integrated into KODIAK's multiple inheritance hierarchy and its default inheritance mechanisms.

Besides categorizing users, KNOME also categorizes information into levels of difficulty. Commands, command-formats, terminology, and other relevant information are categorized according to their level of difficulty. These stereotype categories include *simple, mundane,* and *complex.* Information is grouped into categories based on their typical location on the learning curve, in other words, based on when the typical user would learn the information. For example, some simple concepts include the rm, ls, and cat commands, the technical term "file," and the simple file command-format (the name of the command followed by the name of the file to be operated upon). These simple concepts are learned early in the experience of a typical user. Somewhat more advanced are mundane level concepts. Some examples are the vi, diff and spell commands, the technical term "working directory," and the -l option of ls. Examples of complex concepts are the grep, chmod, and tset commands, the term "inode," and the fact that write permission on the containing directory is a precondition for using the rm command for deleting a file. Complex concepts are learned relatively late by the typical user.

In addition to the three previously mentioned levels of difficulty, KNOME has the category *esoteric* that includes information that is not typically learned at any particular stage of experience. Such concepts are learned only when users have special needs. Some may be familiar to beginners, yet unfamiliar to many experts. An example of an esoteric command is spice, a program used for electronic circuit simulation. Only people who need to perform circuit simulations would know how to use spice. This group would likely include beginning users as well as intermediate and expert users. On the other hand, there are many expert UNIX users who do not know about spice, because they have never needed to perform semiconductor circuit simulations.

Once information has been classified into levels of difficulty, inferences based on user stereotypes can easily be represented as relations between the various user

stereotypes and difficulty levels. For example, Figure 3 shows a graphical KODIAK representation of one such relation. This Figure shows the KODIAK representation of the fact that intermediate users know most facts that have the difficulty level of mundane. The predicate of MOST is used, because there is uncertainty as to whether any particular intermediate user will know any particular mundane level fact. However, any intermediate level user will know most mundane level facts. Table 1 summarizes the relation between different user stereotypes and different knowledge difficulty levels.

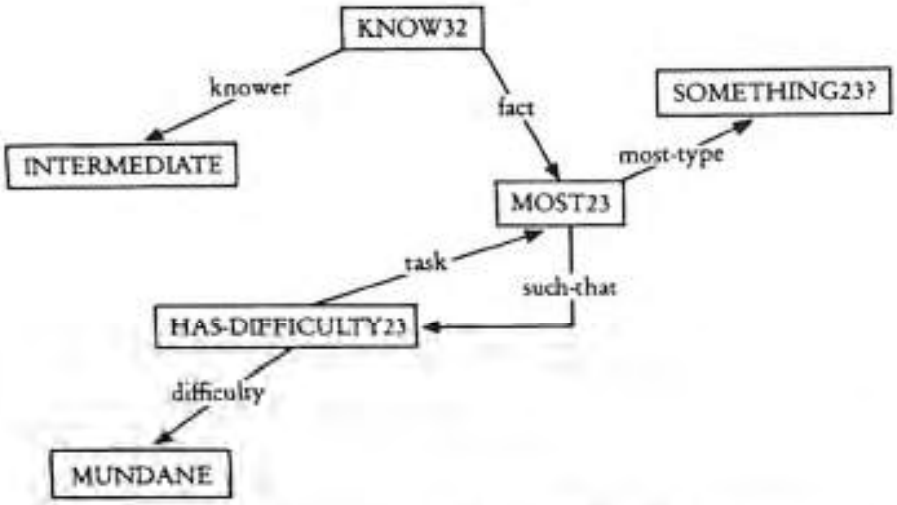

Figure 3. KODIAK representation of: intermediates know most mundane facts

Table 1. Relation between user stereotypes and knowledge difficulty levels

User stereotype	Knowledge difficulty level			
	simple	mundane	complex	esoteric
expert	ALL	ALL	MOST	—
intermediate	ALL	MOST	AFEW	—
beginner	MOST	AFEW	NONE	—
novice	AFEW	NONE	NONE	NONE

The use of stereotype categories for both users and information is an example of a *double-stereotype* system. Single stereotype systems like Grundy used stereotypes for users (e.g., FEMINIST, INTELLECTUAL, SPORTS-PERSON) but not for information (i.e., Grundy did not have stereotypes for books such as GOTHIC-ROMANCE, DETECTIVE-STORY, or SCIENCE-FICTION). KNOME is the first user modeling system to utilize double-stereotypes.

The use of stereotypes allows KNOME to *predict* what users are likely to know or not know based on a partial user model. Once KNOME has accumulated a small

number of facts (typically 3)[3] about what users do or do not know during the course of a UC session, it can infer the user's level of expertise (see Section 3). Then based on the user's level of expertise, KNOME can predict whether or not the user is likely to know other facts. The small set of facts that allows KNOME to infer the user's level of expertise constitutes a partial user model, and the inference that the user belongs to a particular stereotype represents a set of default inferences that are made on the basis of the partial user model.

The use of stereotypes allows KNOME to efficiently group these default inferences together. A system that did not use stereotypes would need to represent individually every partial user model and all the default inferences from that partial user model. Since a partial user model is simply a set of facts about a user, there is a combinatorically large number of partial user models. So, representing all such partial user models and their default inferences would be extremely inefficient. However many partial user models have similar default inferences. These can be grouped together efficiently using user stereotypes. A double-stereotype system is even more efficient, since it allows the system to encode for each fact only its level of difficulty rather than having to encode its relationship to every user stereotype. The rest of a double-stereotype system can then be encoded in a small number of statements relating the two levels of stereotypes. This relationship in KNOME between user stereotypes and knowledge difficulty levels is summarized in Table 1.

KNOME currently has only one double-stereotype range for users. This range represents users' knowledge of UNIX file manipulation commands, UNIX information gathering commands, and UNIX concepts and terminology. This corresponds to UC's domain of expertise within UNIX. If UC were extended to cover more of UNIX or even other operating systems, then KNOME would need more ranges of double-stereotypes. For example, if UC also covered usage of the vi and emacs editors, then KNOME would need separate expertise ranges for each of these domains. KNOME would also need to encode how the user's level of expertise in one domain might be used to predict the user's level of expertise in another domain. So, an expert in using vi would undoubtedly have a high level of expertise in manipulating the UNIX file system. On the other hand, a high level of expertise in using emacs does not necessarily indicate a high level of expertise in using UNIX, since the emacs editor is commonly found in other operating systems.

2.2. Modeling Individual Users

Individual users usually do not completely fit any one stereotype. As a result, a user model that uses stereotypes must also encode how individual users differ from the stereotypes. KNOME stores specific information about what individual users know and do not know. Such information is stored as a collection of propositions about what a user knows or does not know. These propositions are represented

[3] The actual number varies, because facts may mutually reinforce KNOME's evaluation of the user's level of expertise, in which case fewer facts will be needed; or facts may provide contradictory evidence, in which case more facts will be needed to evaluate the user's level of expertise.

using KODIAK. In reasoning about what a user knows, this collection of individual facts is checked before resorting to reasoning from the user's stereotype category.

To avoid storing too many individual facts about what users know, only those facts that cannot be inferred with high likelihood from a user's category are stored. For example, the fact that a particular intermediate user knows the rm command (a simple command) does not need to be stored explicitly, since it is directly inferable from the fact that intermediate users know *all* simple facts. Also, the fact that a particular beginner knows the rm command does not need to be stored. This is because beginners know *most* simple facts and hence it is likely that any particular beginner would know rm. On the other hand, the fact that a particular novice knows the rm command does need to be stored explicitly, since that is not inferable with high likelihood. Because novice users know just *a few* simple facts, it is unlikely that any particular novice would know rm.

2.3. Types of Knowing

In UC, two senses of "know" are used. The first is the classical sense of know: the knower believes that some proposition P is true, and P is true. An example of this sense of knowing is: " the user knows that the cd command is used to change the current working directory." There is also another sense of know that is found in UC. This sense of know is closer to the colloquial usage wherein knowing an object means being familiar with that object. However, in UC this vague usage is made precise. For UC, knowing a command such as the UNIX-RM-COMMAND implies knowing that there exists a command with the name rm, knowing the main purposes of this command, knowing the format of the command, knowing the main effects of the command, and knowing the main preconditions of the command.

The main purposes of a command are represented in UC using the *planfor* [CHIN83, WILE86] relation, which relates a goal and the plan that is typically used to satisfy that goal. The planfors of a plan are different from the effects of the plan, since not all of a plan's effects can be considered goals that the plan is typically used to satisfy. All goals in the planfors of a command are also effects of the command, but some side effects of the command may not be part of its planfors. For instance, calling the game program rogue with a file argument has a side effect that is not found in the planfors of rogue. When rogue is given a file argument that is not a rogue-format save file, then rogue arbitrarily removes the file. This is a side effect of the rogue program, but it is not in any planfors, since users do not typically run rogue in order to remove files.

2.4. Dealing with Uncertainty

In any user model, the inferences that the model makes about the user contain some degree of uncertainty. In KNOME, an attempt was made to avoid the use of numeric representations of uncertainty, because their use tends to lead systems to overestimate the accuracy of their ratings of uncertainty. So, UC uses a fixed number of simple rating levels instead: LIKELY, UNLIKELY, VERY-LIKELY, VERY-UNLIKELY, SOMEWHAT-LIKELY, SOMEWHAT-UNLIKELY, TRUE, FALSE,

and UNCERTAIN (cf. fuzzy logic [ZADE65]). KNOME also uses predicates such as AFEW, MOST, ALL, and NONE (cf. [ZADE82]). Such predicates and ratings are used by KNOME to express the certainty of its inferences. For example, when KNOME is asked whether a particular user knows some particular fact, it will return either TRUE, LIKELY, UNLIKELY, FALSE, or UNCERTAIN. TRUE and FALSE indicate that KNOME has complete confidence. LIKELY and UNLIKELY indicate some uncertainty. An UNCERTAIN result indicates that KNOME does not have enough information to guess whether the user does or does not know that fact.

To deduce whether a particular user knows some particular fact, KNOME uses a multi-step inference process. First, KNOME checks the individual user model for that user. If the individual user model records that this user does or does not know this fact, then the answer is, respectively, TRUE or FALSE. If there is no specific information in the user's individual user model, KNOME next checks the stereotype category of the user. If users of the user's stereotype do or do not know that fact, then the answer is again TRUE or FALSE, respectively. If these checks fail, KNOME resorts to inference based on the difficulty level of the fact. If the user's stereotype knows ALL facts of that difficulty level, then the answer is TRUE; if the stereotype knows MOST facts of that difficulty, then the answer is LIKELY; and if the stereotype knows AFEW facts of that difficulty, then the answer is UNLIKELY. Finally, if this process fails due to lack of information, then the answer is UNCERTAIN. This algorithm is summarized in the flow chart of Figure 4.

3. Deducing the User's Level of Expertise

One of the main problems in user modeling is how to build up a model of a particular user. In a user modeling system based on stereotypes, this means determining which stereotype(s) best fit the current user and how that user differs from the stereotypes. In UC, this means that KNOME must determine the user's level of expertise in using UNIX.

3.1. Other Model Acquisition Systems

Various approaches to building up models of users have been used. Some systems such as the Real Estate Agent [MORI85b] expect users to mention enough self-characteristics at the start of a session so that the system can build up the user model. This works for situations such as a real estate agent interchange with a client where the client is likely to mention what he/she needs/wants at the beginning of a dialog. However, this does not extend to other consultation situations where the user usually will not mention what he/she knows at the start of a session. In such consultation situations the user typically begins by describing his/her problem to the consultant.

Another approach is to ask the user for a self-description at the start of a session as in Grundy [RICH79b] This is a reasonable approach for judging the user's

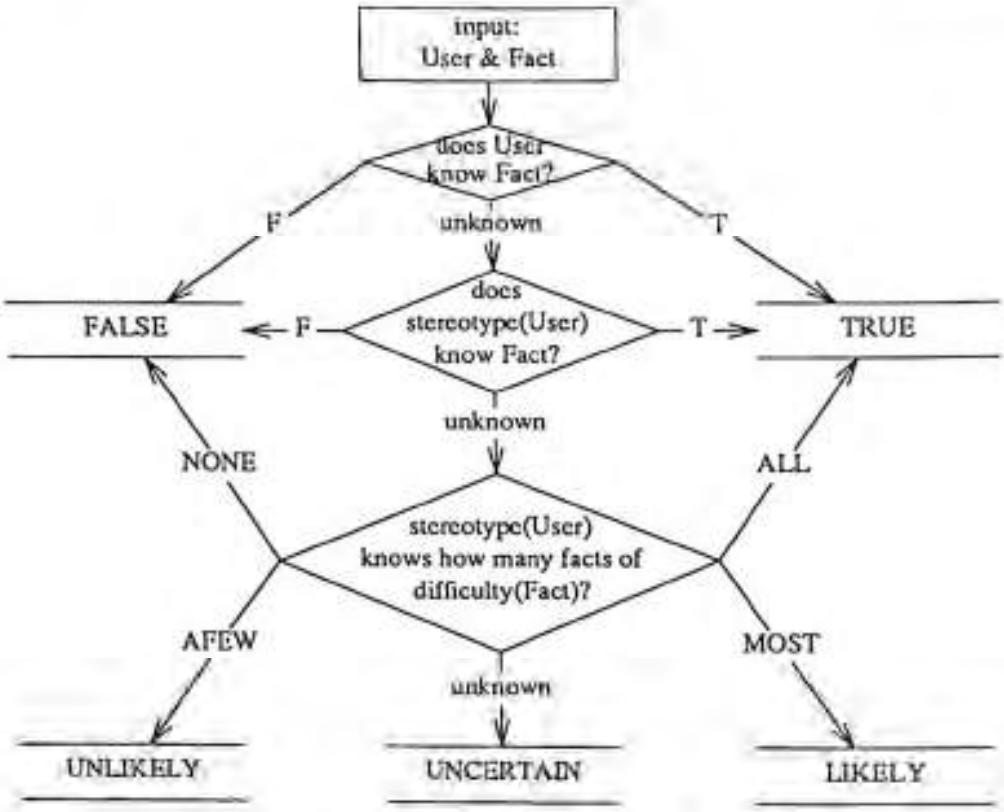

Figure 4. Algorithm for determining whether User knows Fact

personality traits, but does not work well for determining what the user knows. For judging the user's level of expertise, this approach is inaccurate, since the user's concept of different levels of expertise like "intermediate" may differ considerably from the system's concept or from other users' concept of an "intermediate." Also, this requirement is not very practical for occasional users, since this increases the overhead of using the system.

A third approach is to quiz users at the start of a session so that the system may estimate the user's expertise. For example, the MAP system [MACM83] helps a user to specify a correct user model of how that user interacts with an operating system. Such an approach is extremely time-consuming. For tasks like intelligent operating system interfaces for which MAP was designed, such a protracted process is somewhat more acceptable, since the user will realize its benefits over a long period of time. However for interacting with a consultation system like UC where typical sessions are short, such a process would be too time-consuming. The casual user who only needed the answer to a single question would be better off without a user model rather than having to put up with such a lengthy process.

The most common approach in ITS (Intelligent Tutoring Systems) is to compare the user's performance with what a built-in expert would do. Differences in performance can then be attributed to either a lack of user knowledge or differences

in user knowledge such as improper procedures or rules. However such an approach is only applicable if the system can "look over the shoulder" of the user. Consultation systems often cannot observe the user at work. Even when this is possible, consultation systems should not be expected to constantly observe the user, since human consultants do not do so.

To properly emulate a human consultant, the system should be able to assess the expertise level of the user while talking to the user. None of the previously mentioned approaches model this ability, which is a difficult problem in natural language understanding. One system that has addressed this problem is the VIE-DPM user modeling system [KOBS85d]. However, VIE-DPM uses only a syntactic interpretation of the user's statements to derive the user's beliefs. VIE-DPM makes assumptions about the user's beliefs based on the form of the user's input (a yes/no question, a wh-question, or a command) [KOBS85d, KOBS86a] or the presence of linguistic particles (e.g. "not") in the user's input [KOBS85b]. Such a syntax-only system would not be able to properly interpret phenomena such as indirect speech acts [SEAR69]. For example, "Can you pass the salt?" is a yes/no question, but one should not assume from this question that the speaker does not know whether the listener can pass the salt. Despite such limitations, VIE-DPM is a step in the right direction.

3.2. KNOME's Approach

KNOME uses the approach of inferring the user's level during the course of a session. At the beginning of a session, UC has no information about the user. In such cases, KNOME remains open to the possibility that the user may belong to any stereotype. KNOME maintains a list of candidate stereotypes to which the user may belong. KNOME also rates the likelihood that the user may be a member of any particular stereotype. At the start, the likelihood ratings for membership in any stereotype is neutral, except for the beginner stereotype. The likelihood rating for beginner is initially slightly higher than neutral, since more of UC's users tend to be beginners rather than novices, intermediates, or experts. As more information is gathered about the user during the session, the stereotypes may rise or fall in likelihood.

KNOME deduces the user's level of expertise using a two phase process. First, during the course of the conversation, KNOME deduces individual facts about what the user does or does not know. As such facts are gathered, KNOME combines the evidence to adjust the likelihood ratings for the different levels of expertise. Eventually KNOME identifies the stereotype level of expertise to which the user belongs.

3.3. Collecting Evidence

The first step in the process of deducing the user's level of expertise is to collect evidence on that subject. Typically, such evidence takes the form of individual facts about what the user knows or does not know. Such facts and other forms of evidence can be deduced from what the user says (or does not say) and from an analysis

of the user's goals. KNOME distinguishes six classes of inferences that were found to be useful in UC for determining the user's level of expertise. These inference classes are summarized in Table 2, which lists the inference class and the kinds of inferences that are commonly found in each inference class.

Table 2. Taxonomy of deduction types for inferring what users know

Inference class	Kinds of inference
Claim	user states that user does (not) know ?x → user does (not) know ?x
Goal	user wants to know ?x → user does not know ?x
Usage	user uses ?x → user knows ?x
Background	user mentions his/her background → user knows as much as the stereotype indicated by the background
Query-Reply	system asks if user knows, user replies → user's reply
No-Clarify	system uses new terminology, user does not ask for clarification → user understands terminology

The first class of inferences, Claim inferences, applies when the user claims to know or not know something. The second class, Goal inferences, applies when the system can infer that the user wants to know something. In such cases, the system can also infer that the user must not know whatever is wanted, since if the user already knew it, then the user would not have the goal of knowing it. The third class, Usage inferences, are made when the user properly uses some command or terminology. KNOME can then deduce that the user must know the command or terminology. The fourth class, Background inferences, are made when the user mentions some relevant information about his/her background. The fifth class of inference, Query-Reply, applies when the system queries the user about the user's knowledge and the user replies. The last type of deduction, No-Clarify, is made when the system uses some new terminology that the user may or may not know. If the user does not ask for a clarification of the terminology, then the system can assume that the user probably already knew the terminology. Each of these classes of inference is described in greater detail below.

Claim inferences are concerned with direct statements by the user about what the user does or does not know. Examples of such statements and the inferences that KNOME makes based on the statements include:

I already know about mkdir.
→ the user knows the mkdir command

I don't know how to move a file.
→ the user does not know how to move a file

→ the user does not know that mv is a plan for moving files

→ the user does not know the mv command

In the first example, the user claims to already know the mkdir command. This statement is not as straightforward as it might seem, since the user and the system may not agree on what it means to know a command. A reasonable interpretation of what a user might mean is that the user knows how to use mkdir, knows the main purpose of mkdir and knows the main effects and preconditions of mkdir. This interpretation is the same as what UC means when it encodes that a user knows a command (see Section 2.3).

Of course, it may be that the system should not believe the user's statement about knowing mkdir. It is possible that the user may be confused and does not actually know mkdir. But, there is always doubt about what someone might really know, no matter how strong the evidence. For example, even if the system were to watch the user use the mkdir command successfully, it is possible that the user may have actually wanted to create a file instead of a directory. However, in none of these cases is there any reason to doubt the user. Unless KNOME obtains evidence to the contrary, it assumes that the user really does (or does not) know something when the user makes such a claim.

The second example requires somewhat more complex processing than the first example. In the second example, the user claims not to know how to move a file. Since UC knows that mv is used to move files, KNOME is able to deduce that the user does not know that mv is a plan for moving files. Finally, because the user does not know one of the main purposes of the mv command, KNOME deduces that the user must not know mv.

KNOME only stores information about levels of difficulty for objects and not for descriptions, so only facts about a user knowing (or not knowing) specific objects (e.g. commands, concepts, etc.) can be used in deducing the user's level of expertise. This approach eliminates redundancy, since any object will have many possible descriptions. For example, the user could have referred to mv by "how to rename a file" rather than "how to move a file." So, the fact that the user does not know mv is useful for deducing the user's level of expertise, but the descriptive fact that the user does not know how to move a file cannot be directly used. UC must first determine that this description applies to the mv command before it can be used for deducing the user's level of expertise. In UC, descriptions of the purpose of commands are interpreted by UC's UNIX planner [WILE86]. After UC's UNIX planner has determined the command referred to by the command description, KNOME can use this information in deducing the user's level of expertise.

Goal inferences apply when UC can deduce the user's goals. If UC can infer that the user wants to know something, ?x, then KNOME can deduce that the user does not know ?x and also does not know the answer to ?x. For example, if the user asks "How can I delete a file?" then UC can deduce that the user's goal is to know how to delete a file. From this goal, KNOME can deduce that the user does not know how to delete a file. Then, after UC computes that a plan for deleting a file is to use the rm command, KNOME can infer that the user does not know this planfor relation between deleting files and the rm command. Finally, since the

planfor relation is central to UNIX commands, KNOME can deduce that the user does not know rm in the sense that the user is not familiar with rm (see Section 2.3).

In UC, the initial deduction about the user's goal is made by UC's goal analysis component, PAGAN [WILE86]. PAGAN computes the user's plans and goals from the dialog, and can handle phenomena such as indirect speech acts. In some cases, PAGAN utilizes KNOME's model of the user's knowledge in deducing the user's goals. For example, if the user asks, "Do you know how to compact a file?", then the choice between the direct interpretation (the user wants to know if UC knows how to compact a file) and the indirect interpretation (the user wants to know how to compact a file) depends partly on whether or not KNOME believes the user already knows the compress command. If KNOME does believe that the user is likely to know compress, then that will bias PAGAN toward the direct interpretation that the user is testing UC (not uncommon), and vice-versa.

This process may seem circular, since PAGAN's analysis of the user's goal depends on KNOME, while KNOME's analysis of the user's knowledge depends on PAGAN. However, KNOME's initial evaluation of whether the user knows is only a prediction from a partial user model. This prediction does not determine PAGAN's interpretation, because PAGAN must also take into account other factors in its analysis. One such factor is the frequency of usage of the direct versus indirect interpretations. PAGAN is biased toward adopting the more frequent usage. Another factor is the the context of the dialog. When UC is configured in knowledge acquisition mode as UCTeacher [MARJ85], then the direct interpretation (i.e. the user is testing UC's knowledge), is inherently more likely than the indirect interpretation. KNOME's beliefs about the state of the user's knowledge is only one factor in PAGAN's analysis, which then leads KNOME to its own inference about the state of the user's knowledge.

Besides questions on how to do something, Goal deductions apply to many other kinds of queries such as:

What is a directory?
→ the user wants to know what is a directory (goal)
→ the user does not know what is a directory
→ the user does not know the directory concept

Can you tell me how to find out who is on the system?
→ the user wants to know who is on the system (goal)
→ the user does not know who is on the system

What does ruptime do?
→ the user wants to know the effects of the ruptime command (goal)
→ the user does not know the effects of the ruptime command
→ the user does not know the ruptime command

All of the above queries represent Goal deductions, since PAGAN can deduce that the user wants to know some information. Based on this initial deduction, KNOME can infer that the user does not know the information that is sought.

Usage deductions are made when the user applies some plan/procedure correctly, or uses some specialized terminology correctly. Examples include:

I tried typing "rm foo," but I got the message, "rm: foo not removed."
→ the user knows the rm command

How can I find out the inode of a file?
→ the user knows the inode concept

I tried typing "del foo", but I got the message "del: Command not found."
→ the user knows the DOS del command
→ the user does not know the UNIX rm command

In the first example, KNOME can deduce that the user knows the rm plan. In the second example, KNOME deduces that the user knows the meaning of the term "inode." This is the main type of deduction used in ITS in which the system observes the user at work solving problems. This type of deduction is suggested by [RICH83] for consulting situations.

In general, whenever the user uses specialized terminology correctly, KNOME infers that the user must know the concept denoted by the terminology. In UC, terms include UNIX command names and operating system terminology such as "file protection", "directory," and "inode." Checking that the user used the terms *correctly* is not very sophisticated in KNOME. The first check is simply that the term must be used in an English sentence that UC can parse and understand. However, the fact that the user used a term in an understandable sentence does not automatically mean that the user knows the meaning of the term. For example, if the user asks, "What does rm do?", KNOME should not infer that the user knows the rm command. The same principle applies when the user states, "I don't know the rm command." In these types of sentences, KNOME actually first infers that the user does know rm and then reverses its inference later based on the fact that the user has the goal of knowing the effects of rm, which is a Goal type inference. In cases when KNOME has evidence both that the user knows something and that s/he does not know it, KNOME always assumes that the user doesn't know, since it is always possible that the user just seems to know and in fact doesn't know.

The last example is somewhat more complex than the previous examples. In this case, KNOME first infers that the user knows the DOS del command. Then, since the DOS del command is used to delete files and the user mistakenly tried to use the del command, KNOME can deduce that the user does not know the UNIX command for deleting files, that is, the rm command. In general, whenever the user tries to use a foreign command in UNIX, then KNOME assumes that the user does not know the corresponding UNIX command. This correspondence is found by looking at the goal of the foreign command and finding a UNIX command that is a plan for the same goal.

Background deductions are made when the system learns some relevant fact about the user's background. Examples include:

I am a fourth year computer science graduate student.

→ the user is a member of the 4TH-YEAR-CS-GRAD stereotype

→ the user is LIKELY to be an EXPERT

In my CS2 class, ...

→ the user is a member of the CS2-STUDENT stereotype

→ the user is LIKELY to be a BEGINNER

In the first example, KNOME infers that the user is a member of the 4TH-YEAR-CS-GRAD stereotype. This allows KNOME to further infer that the user is LIKELY to be an expert, because most 4th year CS graduate students are experts. In the second example, UC's Concretion mechanism infers from "my CS2 class" that the user is a CS2-STUDENT. Based on this, KNOME then infers that the user is a member of the CS2-STUDENT stereotype, and so is likely to be a beginner.

To make Background deductions, KNOME uses a collection of rules, one for each stereotype related to the user expertise levels. For example, the rule for CS2-STUDENT is: if the user is a CS2-STUDENT, the the user is LIKELY to be a BEGINNER.

Query-Reply deductions are made after the system asks whether or not the user knows some fact. The user's answer usually provides information about whether or not the user knows the fact. For example, if a system were to ask the user, "Do you know the rcp command?" and the user answers "Yes," then the system can infer that the user knows the rcp command. This type of deduction has not been implemented in KNOME, since the current version of UC does not ask the user whether the user knows particular facts.

No-Clarify deductions show how a system might obtain information based on what the user does *not* say. If the system uses some terminology that the system is not sure of the user knows, then the system can infer that the user does know this terminology provided that the user does not ask for a clarification and the meaning of the terminology is not evident from the context. Such deductions are more complex than the previous ones, since they require that the system keep track of the conversation. This type of deduction has not been implemented in KNOME, since UC tries to avoid the use of terminology unfamiliar to the user.

In KNOME, deductions are made using a rule-based system. The actual rules were shown in Table II, with the exception of the Background type of inference. The Background type inference is actually implemented by a collection of rules. For every type of user background that gives some clue about the user's level of expertise in using UNIX, there is a rule. Since there are potentially many types of user backgrounds that are relevant, there will be a corresponding number of such rules. For example, if the user is a CS2 student, then that is a relevant piece of background information which provides some clue about the user's level of expertise in using UNIX. On the other hand, the facts that the user is a scuba diver, likes blueberry pies, or is a bachelor are irrelevant. The particular Background rules that have been implemented in KNOME are shown in Table 3, which also lists the auxiliary inferences which support the major inferences that were shown in Table 2.

Table 3. Other rules used to infer what users know

user is a fourth year CS graduate student	→ user is likely to be an expert
user is a CS2 student	→ user is likely to be a beginner
user does (not) know ?x, where ?x is a description, and ?y is the referent of ?x	→ user does (not) know ?x
user does (not) know a planfor of command ?x	→ user does (not) know ?x
user does (not) know	→ user does (not) know ?x

Such rules are implemented using *if-detected daemons* [CHIN87], which are daemons that detect specific configurations of knowledge in UC's KODIAK knowledge bases. For example, the first rule shown above would normally fire after PAGAN has inferred that the user wants to know something. Then the action part of the rule allows KNOME to infer that the user does not know some fact.

These classes of deductions were found to be important in the domain of a natural language consultation system. Other domains may provide other clues as to what a user knows. For example, in a domain where the user provides speech input, a system may be able to make deductions about the user's knowledge when the user mispronounces technical words.

3.4. Combining Evidence

After collecting evidence that the user does or does not know certain facts, the system must combine the evidence to determine the user's level of expertise. Facts are particular pieces of information about what the user does or does not know as described in the previous section. Note that a single user statement or query may produce several facts. Each unique fact is treated equally in determining the user's level of expertise. When KNOME deduces that the user does or does not know something more than once (perhaps from different user statements), only the first such fact is used in deducing the user's level of expertise. The rest only reconfirm the first fact, so they do not contribute to determining the user's level of expertise.

At the start of a session, KNOME has very little idea about the level of expertise of the user. KNOME's beliefs about the user's level are encoded as ratings of the likelihood that the user is a member of a candidate stereotype category. As evidence is gathered during the dialog, KNOME continuously adjusts its ratings concerning the likelihood that the user might be a member of any particular stereotype. Eventually, KNOME gathers enough evidence to peg the expertise level of the user. Typically, this takes about three interchanges. After making this decision, KNOME

does not change its estimation of the user's level. This works well in UC where typical sessions are short and it is advantageous to quickly guess the user's level of expertise. A more flexible approach is to keep a running account of likelihoods as in SC-UM [NESS87], a user modeling system that is based on the methodology demonstrated in KNOME. However, a system that does not commit to a particular interpretation of the user's level of expertise cannot avoid the need to store information that can be predicted from the user's stereotype. This becomes inefficient as more is learned about the user.

There are two types of conclusions that may be drawn from any particular fact about what the user knows or does not know. First, the evidence may be enough to eliminate some stereotypes from consideration. An example is when the user does not know a simple fact such as the rm or ls commands. In these cases, KNOME can rule out the possibility that the user may be an intermediate or an expert, since all intermediates and experts know all simple facts (see Table 1). So, if the user does not know a simple command like rm or ls, then the user could not possibly be more advanced than a beginner.

Even when the evidence is not enough to rule out a category, the system can still infer that the category is either more likely or less likely. KNOME does this by increasing or decreasing the likelihood rating of candidate categories.

Likelihood ratings are combined using the following linear scale: FALSE, VERY-UNLIKELY, UNLIKELY, SOMEWHAT-UNLIKELY, UNCERTAIN, SOMEWHAT-LIKELY, LIKELY, VERY-LIKELY, TRUE. The UNCERTAIN rating is neutral. It implies that the system does not believe that membership in that category is either likely or unlikely. Likelihood ratings combine linearly according to the scale. For example, a rating of LIKELY when combined with another LIKELY rating produces a TRUE rating. Also a SOMEWHAT-LIKELY rating combined with a VERY-UNLIKELY rating produces an UNLIKELY rating.

To see why KNOME might change a likelihood rating, consider what happens when KNOME determines that a user does not know some mundane level fact. In this case, KNOME infers that the user cannot be an expert, since experts know all mundane facts. Also, KNOME considers it somewhat less likely that the user might be an intermediate, since intermediates know most mundane level facts. Similarly, the user is considered somewhat more likely to be a beginner, since beginners know a few mundane facts. Finally, since the stereotypical novice does not know any mundane facts, the user is considered more likely to be a novice.

Table 4 shows the basis for these deductions. Given that a user does or does not know a fact of some difficulty level, and given that a particular user stereotype knows none/afew/most/all of the facts of that difficulty level, then KNOME can deduce that the user belongs to that stereotype with the additional likelihood shown in Table 4.

Combining Table 4 with Table 1, yields Tables 5 and 6. Given the particular stereotypes used in KNOME, these two tables show the deductions that are made when KNOME determines that a user does (Table 5) or does not (Table 6) know some fact that has the difficulty level indicated. Deductions consist of eliminating categories (FALSE) or increasing/decreasing the likelihood ratings of categories.

Table 4. Deductions when user does (not) know some fact

Stereotype knows difficulty level	Likelihood (user ∈ stereotype)	
	User does know fact	User does not know fact
NONE	FALSE	LIKELY
AFEW	SOMEWHAT-UNLIKELY	SOMEWHAT-LIKELY
MOST	SOMEWHAT-LIKELY	SOMEWHAT-UNLIKELY
ALL	LIKELY	FALSE

Since these two tables are easily derivable from the previous two, KNOME does not actually store the information of these two tables internally. Table 1 is represented declaratively as a KODIAK network that is accessible to the rest of UC. Only Table 4 is stored internally in KNOME. This table is used when needed to derive the likelihood rating differences shown in Tables 5 and 6.

Table 5. Deduction when user knows some fact

User stereotype	Difficulty level of fact			
	simple	mundane	complex	esoteric
Novice	SOMEWHAT UNLIKELY	FALSE	FALSE	FALSE
Beginner	SOMEWHAT-LIKELY	SOMEWHAT-UNLIKELY	FALSE	—
Intermediate	LIKELY	SOMEWHAT-LIKELY	SOMEWHAT-UNLIKELY	—
Expert	LIKELY	LIKELY	SOMEWHAT-LIKELY	—

When the likelihood rating of a stereotype reaches TRUE, it is selected as the user's category. The user's category can also be deduced by elimination. Candidate stereotypes are eliminated when their likelihood drops to FALSE. When all but one candidate have been eliminated, the remaining candidate is selected. During the period before the final decision, KNOME works under the provisional assumption that the user belongs to the category with the highest current likelihood. In the case of a tie (i.e. when there are two or more candidate categories with the same highest likelihood rating), the lower level of expertise is chosen to represent the user temporarily. Using the lower level helps prevent KNOME from overestimating the user before KNOME is sure about the user's level of expertise. Underestimating the user's level may result in more verbose descriptions, but does not lead KNOME to

Table 6. Deduction when user does not know some fact

User stereotype	Difficulty level of fact			
	simple	mundane	complex	esoteric
Novice	SOMEWHAT-LIKELY	LIKELY	LIKELY	SOMEWHAT-LIKELY
Beginner	SOMEWHAT-UNLIKELY	SOMEWHAT-LIKELY	LIKELY	—
Intermediate	FALSE	SOMEWHAT-LIKELY	SOMEWHAT-LIKELY	—
Expert	FALSE	FALSE	SOMEWHAT-UNLIKELY	—

leave out information that the user may not know. Also, inferences that are made based on these provisional stereotype assumptions are marked as having greater uncertainty.

3.5. Examples

To see how KNOME deduces a user's level of expertise, consider the following traces from the UC sessions shown in Figures 5 and 6. At the start of either session, UC does not have any information about either user. In such cases, UC starts with a slight bias toward believing that the user is a beginner, because more UC users tend to be beginners. Hence the likelihood rating for the user being a beginner is SOMEWHAT-LIKELY, whereas the likelihood ratings for the user being a novice, intermediate or expert are all UNCERTAIN (the neutral likelihood rating).

In Figure 5, KNOME deduces that the user is a novice, because the user does not know either the more or lpr commands, which are both simple commands, nor the rwho command, which is of mundane difficulty. When KNOME first deduces that the user does not know the more command, KNOME can draw a number of inferences based on the fact that the user does not know a simple command. Since experts and intermediates know all simple commands (see Table 1), KNOME rules out the possibility that the user might be an intermediate or an expert. Also, since beginners know most simple facts, KNOME lowers the likelihood rating for the user being a beginner from SOMEWHAT-LIKELY to UNCERTAIN. Likewise, since novices know only a few simple facts, KNOME raises the likelihood rating for the user being a novice to SOMEWHAT-LIKELY.

When KNOME next determines that the user does not know the simple command, lpr, KNOME makes additional inferences. KNOME raises the likelihood rating of the user as a novice to LIKELY and lowers the likelihood rating of the user as a beginner to SOMEWHAT-UNLIKELY. Finally when KNOME finds that the user does not know the mundane command rwho, the likelihood rating for the

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.

Hi.
How can I help you?

# What will display a file on my terminal?

KNOME: Asserting *USER* does not know ACTION12?
KNOME: Asserting *USER* does not know PLANFOR40
KNOME: Asserting *USER* does not know UNIX-MORE-COMMANDO
KNOME: Since UNIX-MORE-COMMANDO is a UNIX-MORE-COMMAND,
       asserting *USER* does not know UNIX-MORE-COMMAND
KNOME: UNIX-MORE-COMMAND has difficulty SIMPLE, so deducing:
KNOME: ruling out *USER* = INTERMEDIATE
KNOME: ruling out *USER* = EXPERT
KNOME: *USER* is SOMEWHAT-LIKELY to be NOVICE
       => likelihood(*USER* = NOVICE) = SOMEWHAT-LIKELY
KNOME: *USER* is SOMEWHAT-UNLIKELY to be BEGINNER
       => likelihood(*USER* = BEGINNER) = UNCERTAIN

Use more.
For example, to display the file named foo, type 'more foo'.

# How can I print a file on the lineprinter?

KNOME: Asserting *USER* does not know UNIX-DLPR-COMMANDO
KNOME: Since UNIX-DLPR-COMMANDO is a UNIX-DLPR-COMMAND,
       asserting *USER* does not know UNIX-DLPR-COMMAND
KNOME: UNIX-DLPR-COMMAND has difficulty SIMPLE, so deducing:
KNOME: *USER* is SOMEWHAT-LIKELY to be NOVICE
       => likelihood(*USER* = NOVICE) = LIKELY
KNOME: *USER* is SOMEWHAT-UNLIKELY to be BEGINNER
       => likelihood(*USER* = BEGINNER) = SOMEWHAT-UNLIKELY

Use lpr.
For example, to print the file named foo, type 'lpr foo'.

# What does rwho do?

KNOME: Asserting *USER* does not know STATE11?
KNOME: Asserting *USER* does not know HAS-EFFECT23
KNOME: Asserting *USER* does not know UNIX-RWHO-COMMANDO
KNOME: Since UNIX-RWHO-COMMANDO is a UNIX-RWHO-COMMAND,
       asserting *USER* does not know UNIX-RWHO-COMMAND
KNOME: UNIX-RWHO-COMMAND has difficulty MUNDANE, so deducing:
KNOME: asserting *USER* = NOVICE

rwho is used to list all users on the network, list their tty,
and list their login time.
```

Figure 5. UC session with an novice

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type 'D' or '(exit)'.

Hi.
How can I help you?

# Can you tell me how to find out the inode of a file?

KNOME:  Asserting *USER* knows INODE1
KNOME:  Since INODE1 is an INODE,
        asserting *USER* knows INODE
KNOME:  INODE has difficulty COMPLEX, so deducing:
KNOME:  ruling out *USER* = NOVICE
KNOME:  ruling out *USER* = BEGINNER
KNOME:  *USER* is SOMEWHAT-UNLIKELY to be INTERMEDIATE
        => likelihood(*USER* = INTERMEDIATE) = SOMEWHAT-UNLIKELY
KNOME:  *USER* is SOMEWHAT-LIKELY to be EXPERT
        => likelihood(*USER* = EXPERT) = SOMEWHAT-LIKELY
KNOME:  Asserting *USER* does not know ACTION15?
KNOME:  Asserting *USER* does not know PLANFOR390
KNOME:  Asserting *USER* does not know UNIX-LS-l-COMMANDO
KNOME:  Since UNIX-LS-l-COMMANDO is a UNIX-LS-l-COMMAND,
        asserting *USER* does not know UNIX-LS-l-COMMAND
KNOME:  UNIX-LS-l-COMMAND has difficulty COMPLEX, so deducing:
KNOME:  *USER* is SOMEWHAT-LIKELY to be INTERMEDIATE
        => likelihood(*USER* = INTERMEDIATE) = UNCERTAIN
KNOME:  *USER* is SOMEWHAT-UNLIKELY to be EXPERT
        => likelihood(*USER* = EXPERT) = UNCERTAIN

Type 'ls -l'.

# What does rwho do?

KNOME:  Asserting *USER* does not know UNIX-RWHO-COMMANDO
KNOME:  Since UNIX-RWHO-COMMANDO is a UNIX-RWHO-COMMAND,
        asserting *USER* does not know UNIX-RWHO-COMMAND
KNOME:  UNIX-RWHO-COMMAND has difficulty MUNDANE, so deducing:
KNOME:  ruling out *USER* = EXPERT
KNOME:  only one candidate left, so asserting *USER* = INTERMEDIATE

rwho is like who, except rwho is for all users on the network.
```

Figure 6. UC session with an intermediate

user being a novice rises from LIKELY to TRUE. TRUE is the threshold (nothing can have a higher likelihood than TRUE), so KNOME concludes that this user is a novice.

In Figure 6, KNOME deduces that the user is an intermediate. First, KNOME deduces that the user knows what an inode is, since the user mentions "the inode of a file." Since knowing about inodes is a complex fact, KNOME is able to eliminate the novice and beginner categories as possibilities. KNOME also lowers the likelihood rating of the user as an intermediate and raises the likelihood rating of the user as an expert. However, these changes in likelihood ratings are canceled out by KNOME's deduction from the same user query that the user does not know the ls -i command, which is of complex difficulty. Next, KNOME deduces that the user does not know the rwho command, which is of mundane difficulty. This allows KNOME to eliminate the possibility that the user might be an expert, leaving KNOME to conclude that the user is an intermediate, since that is the only possibility left.

Even before KNOME has completely determined the expertise level of the user, KNOME's partial user model is still useful to the other components of UC. In the dialog with the novice, UC chooses to provide the user with examples of command-formats based on KNOME's initial deduction that the user is SOMEWHAT-LIKELY to be a novice. On the other hand, UC does not provide examples of simple command-formats to the intermediate user who presumably would already know such simple formats. Also, UC is able to use a simile to explain to the intermediate user the rwho command in terms of the who command. The novice user was given a complete explanation of rwho, since KNOME believed that the user would not know the who command and so would not understand the simile.

4. Modeling UC's Knowledge

Besides modeling what users know, KNOME also models what UC itself knows. Such a model allows KNOME to differentiate between cases where UC lacks knowledge and where the user has a misconception.

4.1. Open vs. Closed World Models

An important problem in AI knowledge bases is to define the limitations of the knowledge base. One solution is to use a closed world model, which states that the knowledge base knows everything (see [REIT78]). In a closed world model, anything that is neither in the knowledge base nor deducible from the knowledge base is deemed false. Such a model works only for completely self-contained microworlds such as airline reservation systems that know all airline flights. For real world applications, a closed world model is untenable, since real world knowledge bases cannot know everything even in just their own area of expertise. So a system with a closed model would be prone to giving out false information such as claiming

that objects that the system did not happen to know about do not exist, claiming that actions that the system could not figure out cannot be done, etc.

The other extreme is an open world model, which states that anything that is neither in the knowledge base nor directly deducible from the knowledge base is not known. This model has a significant problem with ruling out things that do not exist. For example, a knowledge base might list all the participants in any particular relation. Using a closed world model, the system would know that these are the only participants of that relation. However, in an open world model, the system cannot rule out that there might not be other participants unless the knowledge base explicitly encodes for all other possible participants that they do not participate in that relation. Since there are usually many more such negative facts than positive facts, encoding such facts in a knowledge base is extremely inefficient.

Neither model is quite adequate. A closed world model allows a system to rule out spurious hypotheses easily, but is prone to ruling out true facts that were not in the knowledge base. An open world model allows the system to be non-committal about hypotheses that are not covered by its knowledge base, but does not allow it to rule out spurious hypotheses when the system *does* have complete information. The ideal system needs to combine the best of both models. It needs to be assertive in ruling out spurious hypotheses when the system does have complete information, yet remain non-committal when it does not have complete information. To do this, a system needs to know the limitations of its own knowledge.

4.2. Meta-Knowledge

UC uses an open world model augmented by knowledge about the coverage limitations of the knowledge base. By using an open world model, UC is able to remain non-committal about information that is not in its knowledge base. This allows UC to profess ignorance about things outside its area of expertise. On the other hand, when UC does have complete information about an area, this is explicitly encoded in KNOME's model of UC's knowledge. In a sense, this is a model of what UC itself knows. Such knowledge is called *meta-knowledge*.

The term meta-knowledge was previously used for AI knowledge bases by [BARR77, DAVI77]. [BARR77] argues that AI knowledge bases need metaknowledge, i.e. the system's knowledge about "the extent and limit of available knowledge, what facts are relevant in a given situation, how dependable they are, and how they are to be used." [DAVI77] describes meta-knowledge in the TEIRESIAS system and its use in knowledge acquisition and in selecting which rules to try applying first through the use of *meta-rules*. [SMID83] use meta-level reasoning to find all of the solutions to a problem. Although they use a closed world assumption, they suggest that other systems for which the closed world assumption is not valid can encode limits on the number of solutions and use these limits to cut off inference. Such limits can be considered a type of meta-knowledge.

In KNOME, meta-knowledge is not primarily used for the same purposes as in TEIRESIAS; rather it is primarily used in dealing with user misconceptions. So KNOME's meta-knowledge describes the coverage limitations of UC's knowledge

base, Chapter III, Section 4 describes how user misconceptions are detected and how meta-knowledge is used in correcting the user's misconceptions.

4.3. Representation of Meta-Knowledge

Meta-knowledge in KNOME is represented explicitly as a collection of KODIAK statements about what UC knows. Examples of meta-knowledge include the fact that UC knows that it knows all file manipulation commands and the fact that UC knows all possible side effects for all known simple commands, most known mundane commands, and a few known complex commands. The fact that UC does not know all of the side effects for known commands is due to not enough programming and the knowledge limitations of UC's programmers rather than any inherent limitation of UC.

Figure 7 shows the representation of KNOME's meta-knowledge about simple commands. KNOW1 states that UC knows all of the effects of simple commands, and KNOW2 represents the fact that UC knows all of the options of simple commands.

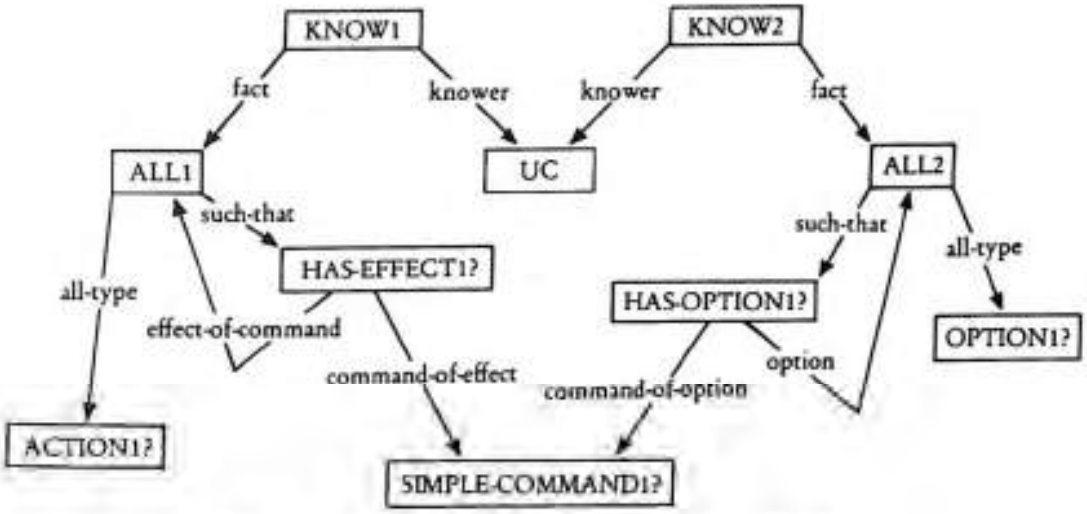

Figure 7. KODIAK representation of some of UC's meta-knowledge

Besides the explicit meta-knowledge modeled by KNOME, UC also has meta-knowledge that is represented implicitly within the KODIAK representation language. For example, any aspectual of a relation can be constrained to be filled by only certain classes of objects. This effectively constrains the way in which different objects may be related to other objects in KODIAK. Of course, these constraints are sometimes violated when users speak metaphorically.

4.4. Other Approaches to User Misconceptions

Other systems that have dealt with user misconceptions use one of two common approaches. The first approach compiles an a priori list of possible user misconceptions for use in detecting misconceptions. This is common among ITS (Intelligent Tutoring Systems) (e.g. [BROW78, STEV79, SLEE81, JOHN84, REIS85, ANDJ85b]). These systems are not very flexible, since they cannot deal with any user misconceptions that are not listed among their precompiled a priori lists.

Another approach attempts to detect classes of misconceptions. Among the best of these is Kaplan's CO-OP natural language database-query system [KAPS83], which handles a class of faulty user presumptions. An example from CO-OP is shown in Figure 8.

User:	Who advises projects in area 36?
CO-OP:	I don't know of any area #36.

Figure 8. CO-OP example with hedged response to misconception

The user in the CO-OP example believes that there is an area 36. In processing the user's query, CO-OP finds that there is no area 36 listed in its database, so it detects a possible user misconception: the user may mistakenly presume that there is an area 36 when in actuality there might not be an area 36. Note that CO-OP's reply does not actually *correct* the user's misconception, rather it *hedges*, claiming that it does not know of any area #36. By hedging, CO-OP cannot correct the user's misconception, but can only *suggest* to the user that the user might be mistaken. CO-OP is unable to take corrective action, because it does not assume either an open or closed world model for its database. As a result, it cannot tell from the fact that there is no area 36 in its database whether there really is no area 36, or if area 36 was just left out of its database. Unlike UC, in which the KNOME subcomponent models UC's own knowledge with meta-knowledge, CO-OP does not have a model of its own database.

A corrective response is shown in Kaplan's example of a cooperative human, shown in Figure 9.

User:	Which students got a grade of F in CS105 in Spring 1980?
Human:	CS105 was not given in Spring 1980.

Figure 9. Kaplan's example of a cooperative human respondent

Because CO-OP does not adopt either an open or closed world model, it would not be able to deny that CS105 was given in Spring 1980 as in Kaplan's example of a cooperative human. CO-OP would only be able to suggest a misconception by saying something like, "I don't know of any CS105 in Spring 1980." To be able to actually correct the user, CO-OP would need to know that the absence of a CS105 among the Spring 1980 courses implies that there was no CS105 offered then. That is, CO-OP would need meta-knowledge. Since CO-OP was designed to be easily transportable among different databases, it did not have such non-transportable meta-knowledge.

The DAIQUIRY module of HAM-ANS [MARB87, MARB88] extended the misconception approach of CO-OP to include conceptual failures in which the user asks about objects not modeled by the database. Because DAIQUIRY, like CO-OP, did not have meta-knowledge, it too could only hedge its replies to the user. For example, it would reply, "The system has no knowledge about 'SHIP'," when SHIP was not in its conceptual hierarchy.

[MAYS80b, WEBB83] report efforts to deal with user misconceptions that can be found by constraint violations on relations. For example, if the TEACH relation is constrained to relate only FACULTY to COURSES, then the system can detect a misconception if the user asks about UNDERGRADUATES that TEACH. This type of relational constraint can be considered a kind of implicit meta-knowledge. It differs from the explicit meta-knowledge of KNOME in that relational constraints can only be used to detect misconceptions about what can be the case, and not misconceptions about what is the case. For example, relational constraints can be used to deduce that only commands can have command-options, but not whether any particular command has any particular command-option.

After a user misconception has been detected, it must be corrected. [MCCO85a, MCCO*] discuss strategies for correcting object-oriented misconceptions. [QUIL*] discusses correcting plan-oriented misconceptions in AQUA. AQUA assumes a closed world model for its knowledge base of plans, called an "advisor model." As a result of this assumption, AQUA can only model perfect advisors that have complete knowledge of the domain. If AQUA's advisor model were incomplete, then AQUA would tend to mislead the user. For example, if an obscure side effect of a plan were inadvertently left out of AQUA's knowledge base, then AQUA would mislead the user to believe that the plan did not have this a side-effect. Some form of meta-knowledge would be required to allow AQUA to model an advisor without perfect knowledge of the domain.

4.5. Detecting Misconceptions in UC

User misconceptions are detected by UC during the processing of the user's query. Currently, UC only handles relational misconceptions, that is, misconceptions in which the user believes a relation holds between two objects when in fact, such a relation cannot hold or does not happen to hold between those particular objects. UC does not handle object-oriented misconceptions, such as those for which [MCCO85a] discusses correction strategies.

In processing the user's query, UC checks to see whether all relations mentioned by the user in the user's query have a counterpart in UC's knowledge base. For example, if the user asks, "What does ls -e do?", then UC's parser/understander understands part of this as a HAS-OPTION relation relating an instance of the ls command and an instance of the -e option. While looking for the effects of the command, UC finds that there is no corresponding HAS-OPTION relation between the ls command and the -e option in UC's knowledge base. At this point, UC suspects that the user may have a misconception.

In order to see whether the user actually has a misconception, or whether UC just lacks information about this particular option of this command, UC checks with the KNOME subcomponent, which models the limitations of UC's knowledge base using meta-knowledge. If KNOME has meta-knowledge that UC knows all of the options of this command, then UC knows that the user has a misconception. On the other hand, if the options of this command are not covered by KNOME's meta-knowledge, then UC cannot assume that the user has a misconception. Instead, UC must assume that it lacks information about this command/option combination.

4.6. Correcting Misconceptions

After UC has determined either that the user has a misconception or that UC lacks knowledge, UCEgo decides how UC should reply to the user. When the user has a misconception, UCEgo tries to correct this misconception by denying that what the user mistakenly believes is the case. On the other hand, if UC determines that it lacks information, then UCEgo apologizes to the user for not knowing the answer to the user's query. These two types of responses are shown in the UC session of Figure 10.

```
Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type "D' or '(exit)'.

Hi.
How can I help you?

# What does who -b do?

who does not have a -b option.

# What does ruptime -t do?

I'm sorry, I do not know that.
```

Figure 10. UC session showing different UC responses to possible misconceptions

In the first query, UC corrects the user's misconception that who has a -b option. It does this by first noticing that the user's usage of "who -b" translates into a HAS-OPTION relation between a UNIX-WHO-COMMAND and a -b-OPTION. There is no equivalent HAS-OPTION relation in UC's knowledge base, so UC suspects a possible user misconception. To see if this is the case or if UC just lacks knowledge about this particular option, UC consults the meta-knowledge stored in KNOME. The appropriate meta-knowledge in this case is the fact that UC knows all the options of all simple commands. Since who is a simple command, and since the -b-OPTION is not listed among the options of UNIX-WHO-COMMAND in UC's knowledge base, KNOME can infer that there is no such option for who. Next, UCEgo decides that UC should correct the user's misconception by denying the existence of a -b option for who.

On the other hand, in the second query, UC professes ignorance about the -t option of the ruptime command. As in the previous query about who, UC detects a possible misconception when it does not find a -t option listed for ruptime in its knowledge base. However, in this case, ruptime is a complex command, so the previous meta-knowledge does not apply. There is no meta-knowledge about the options of complex commands (due to not enough programming by UC's implementors rather than any inherent limitation of UC), so UC cannot tell if ruptime has a -t option. In order to be polite, UCEgo apologizes to the user for not knowing.

5. Exploiting KNOME

KNOME is used by several different components of UC. This section will provide a survey of these applications, since a detailed analysis is beyond the scope of this chapter.

One application of KNOME is in goal analysis (see [WILE86]). KNOME can be used to choose between different interpretations of the user's statements. For example, consider the user's query "Do you know how to delete a file?" There are two possible interpretations of this query: a direct interpretation where the user wants to know if UC knows how to delete a file; and an indirect speech act interpretation where the user wants to know how to delete a file. If the goal analyzer knows that the speaker already knows how to delete a file, then the system can rule out the indirect interpretation. Similarly, if the system believes that the user does not know how to delete a file, then the indirect interpretation is more likely.

Another use for the user model is to keep UC from telling the user something that the user already knows, and to express information in terms of concepts with which the user is already familiar. For example, in expressing the answer, the UCExpress component of UC prunes those parts of the answer that the user already knows. So in the following UC dialog, UCExpress gives an example of the format of the vi command to a novice, since the novice likely does not know the simple-file-format. To beginners and other non-novices, UCExpress does not provide the example, since UC can assume that the user will know that to use vi, one types "vi" followed by the name of the file to edit.

User: How can I edit a file?
UC to novice: Use vi.
For example, to edit the file named foo, type 'vi foo'.
UC to beginner: Use vi.

KNOME is also used by UCExpress when determining whether or not to use an analogy in explaining the usage of a command. Consider the following UC dialog. UCExpress needs to know that the user already knows the uptime command before it can choose to use the analogy.

User: What does ruptime do?
UC: ruptime is like uptime, except ruptime is for all machines on the network.

UCEgo [CHIN87], the agent component of UC, uses KNOME when deciding whether or not to provide additional information that the user may need (cf. [JOSH84a, JOSH84b] for other theoretical approaches). For example, in the UC dialog shown in Figure 11, UCEgo does not actually answer the user's question.

User: Who is on the system?

UC: To find out who is on the system, type 'who'.

Figure 11. UC volunteers information that the user does not know

UC does not actually tell the user who is on the system (e.g. by saying "John, Mary, and David are on the system"). Instead, UCEgo decides to tell the user how to find out this information, namely use the who command. However, if the user already knows the who command, then UC's answer would be superfluous. So UCEgo first consults KNOME to see if the user lacks such knowledge before providing the additional information.

6. Conclusion

6.1. Summary

The user modeling system of KNOME is extremely simple, yet it provides enough functionality for the needs of UC. KNOME can infer what the user knows from a consulation dialog with the user. Based on only a few such facts about what a user does or does not know, KNOME can also *predict* many other default facts about what the user might or might not know. KNOME's stereotypes serve to efficiently group those default inferences that commonly co-occur. Using only a few such stereotypes, KNOME is able to provide enough differentiation among potential

users so that UC can use these differences in answer expression, goal analysis, and detection of situations when the user lacks knowledge or has misconceptions. The double-stereotype system is not only space efficient, but also allows easy addition of knowledge to KNOME. When a new fact is added to UC, the knowledge engineer needs only to classify the new fact according to its difficulty level.

Although no attempt has been made yet to apply the methodology to other areas, it would seem that many of the techniques should transfer well. Stereotype categories are used in UC in similar fashion to the way humans use categories, that is as reference points for reasoning about individual members (see [ROSC77, ROSC78, ROSC83]). Thus stereotypes should be useful for approximate reasoning wherever human categorization has proven to be useful. The double-stereotype system extends the use of computer stereotype categories to cover more of human categorization, that is, to include categories for knowledge as well as categories/stereotypes for people/users. The double-stereotype system can be applied straightforwardly to other domains where a computer system needs to model the relation between different classes of users and different classes of information. In cases where the user population is homogeneous or where the information is homogeneous, then a double-stereotype system would not be applicable.

The methodology used to deduce the user's stereotype in KNOME transfers in two ways to other domains. First, the method used to infer individual facts about what the user knows is applicable to any natural language system that needs to infer facts about the user's knowledge based on what the user says. On the other hand, the methodology used to combine such evidence is highly dependent on a double-stereotype system. It can, however, be used in any double-stereotype system where the system can deduce particular relations between the user's stereotype and the modeled information.

6.2. Current Status

KNOME is currently implemented in FRANZ LISP and Common LISP in a version of KODIAK [WILE87], which is a network style knowledge representation language featuring objects and relations, multiple inheritance, and other epistemological tools.

6.3. Problems

Currently, KNOME does not address the problem of whether the user knows something after UC has informed the user. The difficulty is that the user may not understand the system's presentation, in which case the user modeling system should not infer that the user knows the new information. A drawback of systems limited to natural language interactions is that these systems cannot watch the user's reactions to deduce whether or not the user is having trouble understanding the system's presentation. Human consultants are able to switch explanation strategies midway through a presentation when they notice their clients' facial expressions showing confusion. A computer consultant would have to wait for the user to

respond with language in order to get feedback. In such cases, the best that a user modeling system can do may be to assume that the user has understood (perhaps with a lower certainty factor), until the modeler gets evidence to the contrary. Then, if the user does not ask for a clarification in the next exchange, the modeler can increase the certainty of its belief that the user has indeed understood.

Another deficiency in KNOME is that it does not contain very many specific rules for inferring what specific facts a user might know based on other specific facts that the user knows. For example, if a user knows the rwho command, then it is very likely that the user will also know the who command. KNOME has avoided these types of explicit rules in favor of the more efficient, though less specific mechanism of double- stereotypes. The double-stereotypes allow KNOME to combine a small number of explicit facts to first infer the user's level of expertise and from this, the user's knowledge of large sets of other facts. Although this powerful general mechanism works well, there are still cases when explicit rules will give more specific information about what a user knows with a higher degree of certainty. The GUMS general user modeling shell [FINI*] makes good use of these explicit rules.

A possible deficiency with KNOME's method of combining evidence is that KNOME marks the user as belonging to a particular stereotype after only a short time (typically 3 interchanges). After this, KNOME does not alter its conception of the user's level of expertise. This is reasonable in the UC domain, because UC sessions are typically short. However, this would not be acceptable for longer term user models. If a system needs to keep track of a user's level of expertise over periods long enough that the user's expertise may increase appreciably, then the system needs to be able to change its judgment of the user's level of expertise. This process is easier than one might suppose, since a user's level of expertise tends to grow monotonically (it seems reasonable to ignore the phenomenon of forgetting).

Currently, KNOME only has a single range of user expertise and a single range of knowledge difficulty. This is sufficient for UC, since UC only covers commands related to the UNIX file system. In order to extend UC to cover other areas of UNIX such as using the vi or emacs editors, KNOME would need additional ranges of user expertise levels and information difficulty levels. Problems which need to be addressed with multiple ranges include how the levels in one range might relate to the levels of other ranges. For example, a system may be able to predict that an expert in using vi is likely to be an expert in using the UNIX file system. On the other hand, there may be no such relations between familiarity with emacs and familiarity with the UNIX file system, since the emacs editor is common to many other operating systems.

Another area which KNOME does not address completely is how to model users of other operating systems. Such users may be able to transfer some amounts of expertise to UNIX. These users can benefit from analogies between UNIX and the operating system that they know. On the other hand, such users sometimes incorrectly apply analogies and often mix up commands. Adding additional stereotypes for such users would enhance KNOME's ability to provide proper analogies and to detect improper analogies or command usage. [MACM83] has shown that one user modeling system can adequately model different operating systems (UNIX,

TOPS-20, and VM). However, no one has investigated how knowledge of the user's expertise in one domain transfers to another domain (correctly or incorrectly) or how to exploit such knowledge (e.g. to provide analogies).

Other areas not addressed by KNOME include applying the techniques demonstrated in KNOME for other purposes such as selection of terminology in generation (cf. [LEHM*]). The approach used in KNOME would also be useful in the selection of different explanation strategies as in the TAILOR system [PARI*].

7. References

[ALLE80], [ANDJ85b], [BARR77], [BROW78], [BURT76], [CARB83], [CARB*], [CARR77], [CHIN83], [CHIN86], [CHIN87], [DAVI77], [FINI*], [HOEP84b], [JOHN84], [JOSH84a], [JOSH84b], [KAPS83], [KASS*], [KAYD85], [KOBS85b], [KOBS85d], [KOBS86a], [LEHM*], [LITM84], [MACM83], [MARB87], [MARB88], [MARJ85], [MATT86], [MAYS80b], [MCCO85], [MCCO85a], [MCCO*], [MORI85b], [MORJ*], [NESS87], [PARI*], [QUIL*], [REJS85], [REIT78], [RICH79b], [RICH83], [RICH*], [ROSC77], [ROSC78], [ROSC83], [SEAR69], [SLEE81], [SLEE82c], [SMID85], [STEV79], [WAHL*], [WEBB83], [WILE84], [WILE86], [WILE87], [ZADE65], [ZADE82]

5 Detecting and Responding to Plan-Oriented Misconceptions[1]

Alexander Quilici

Abstract

This chapter[2] discusses the problem of detecting and responding to plan-oriented misconceptions in advice-seeking dialogs, concentrating on the problems of novice computer users. A cooperative response is one which not only corrects the user's mistaken belief but also addresses the missing or mistaken user beliefs that led to it. Providing a cooperative response is presented as a process of the advisor trying to explain why he doesn't hold the user's belief and presenting this explanation to the user. An explanation is shown to correspond to a set of advisor beliefs that together contradict the user's belief, and searching for an explanation to determine whether various abstract configurations of advisor beliefs hold. The search for an explanation makes use of both domain-specific advisor beliefs and a taxonomy of domain-independent explanations for potential user misconceptions. A taxonomy for misconceptions involving plan applicability conditions, preconditions, and effects is presented, along with a method of organizing the necessary advisor planning knowledge.

1. Introduction

A novice computer user having trouble performing a task often describes his problem to an expert, who is then expected to explain its cause and provide its solution. Consider, for example, the following advice-seeking dialog between a novice UNIX user and his more experienced counterpart.

> **User:** I tried to remove a file with the "rm" command. But the file was not removed and the error message was permission denied. I checked and I own the file. What's wrong?

---

[1] This chapter is an extended and revised version of [QUIL88].

[2] The work reported here was supported in part by a grant from the Lockheed Software Technology Center (Austin, Texas). Special thanks go to Mike Gasser, John Reeves, and Ron Sumida for fighting their way through several earlier incoherent and uninteresting versions of this chapter. Comments by the editors of this book and a collection of anonymous reviewers have also greatly improved its organization and content.

Advisor: To remove a file, you need to be able to write into the directory containing it. You do not need to own the file.

User: How do I make that directory writeable?

Advisor: To make a directory writeable, use the "chmod +w" command.

User: Wait a second while I try it. No, "chmod +w" prints the error message "permission denied".

Advisor: To make a directory writeable, you need to own it.

User: So to remove a file, I have to own the directory that contains it.

Advisor: No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

User: So how do I remove the file?

Advisor: Send mail to whoever has write permission on the directory, asking him to remove the file for you.

The advisor in such a dialog must be able to recognize missing or mistaken user beliefs about plan applicability conditions, preconditions, and effects. The advisor above recognizes two user misconceptions: an initial incorrect belief that owning a file is a precondition to removing it, and a subsequent incorrect belief that the precondition is owning the directory in which it resides. This advisor also notices several gaps in the user's knowledge: The user has no plan for making a directory writeable, does not know why the advisor's plan for doing so failed, and has no plan for removing a file that isn't in a writeable directory.

The advisor must also be able to respond to both missing and mistaken user beliefs. A cooperative advisor response to a *missing* user belief simply provides that belief. This advisor provides the missing beliefs that the plan for making a directory writeable is to use the "chmod +w" command, that owning a directory is a precondition to making it writeable, and that the plan for removing a file when all else fails is to send mail requesting its removal to some person who can write into the directory in which the file resides.

For a *mistaken* user belief, however, the advisor must not only point out that the belief is incorrect and provide a correction, but must also address the missing or mistaken beliefs that are the source of this misconception. Above, this is done by pointing out that the actual precondition is being able to write into the directory that contains it, and by explaining that owning that directory is necessary only if it is not already writeable.

This chapter examines the problem of detecting and responding to mistaken user beliefs. We consider misconceptions involving plan applicability conditions (whether a particular plan should be used to achieve a goal), enablements (whether a particular state must exist before a plan can achieve a goal), and effects (whether a state will exist as a result of a plan's execution). This problem can be broken into several subproblems:

1. Mapping the user's natural language problem description into a set of user beliefs.
2. Determining which of these beliefs are incorrect.

3. Inferring the missing or mistaken user beliefs that might have led to these incorrect beliefs.
4. Selecting the advisor beliefs to present to the user as a conversationally-cooperative response.
5. Mapping these advisor beliefs into a natural language response.

This chapter provides a computational model of (2), (3), and (4). Its input is an initial user model composed of a set of potentially incorrect user beliefs. Its output is a set of advisor beliefs that constitute a cooperative response to any mistaken user beliefs. The advisor selects these beliefs by trying to explain why he doesn't hold the user's mistaken beliefs.

The rest of the chapter presents this explanation-based approach to misconception detection and response. We begin with an overview of our approach and of our representation for plan-oriented user and advisor beliefs. We then examine the process by which the necessary explanations are found and the knowledge needed to find these explanations. As part of this discussion, we provide a taxonomy of explanations for the types of plan-oriented beliefs we consider, and show how the advisor's planning knowledge can be organized to facilitate the process of explanation finding. Finally, we compare our approach with other work in detecting and correcting user misconceptions, and discuss its implementation and limitations.

2. An Explanation-Based Approach

How can an advisor decide whether a particular user belief is mistaken and then understand how the user came to believe it? And furthermore, how can the advisor determine the contents of a cooperative response to the user's misconception?

We suggest an explanation-based approach to accomplish these tasks. To understand why the user holds a mistaken belief, the advisor tries to *explain* why he does *not* hold this user belief. The explanation consists of a set of advisor beliefs that together contradict the user's incorrect belief. These beliefs constitute the advisor's response. The underlying assumption is that the user would not hold his mistaken belief if he knew the reasons why the advisor does not hold it.

We illustrate our approach by showing how the advisor arrives at the response found in this exchange from our introductory dialog.

User: So to remove a file, I have to own the directory that contains it.

Advisor: No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

The user's possibly incorrect belief is that a file cannot be removed without owning the directory in which it resides.

The advisor tries to explain why he does not hold this user belief. The explanation he finds consists of two related beliefs. The first is his belief that a file cannot be removed without write permission on the directory containing it. The other is

his belief that owning the directory is necessary only if one does not already have write permission on it. Together these beliefs contradict the user's belief that owning the directory is always necessary to remove a file. The contradiction is that the advisor believes that the user's claimed precondition for removing a file is merely a precondition for achieving one of its preconditions.

Once the advisor finds this explanation, he presents it to the user. This response corrects the user's misconception by pointing out that the user's claimed precondition for removing a file is incorrect, by providing the actual precondition for removing a file, and by providing the relationship between the user's presumed precondition and the actual precondition. The user's being unaware of this relationship led to the user's misconception.

In effect, our approach has the advisor infer a set of beliefs that contradict the user's erroneous beliefs. Our assumption is that the user's misconception resulted because he was unaware of these advisor beliefs.

Our approach derives from work in explanation-based story understanding [SCHR86, DYER83, WILE83, SCHR77]. The basic idea is that to understand a particular input, such as a person's action, we have to explain why it has occurred. One way to explain a person's action is to relate it to known goals the person is trying to achieve. Suppose, for example, that a story understander reads that a hungry character bought a restaurant guidebook [WILE83]. One explanation for this action is that hungry people want to eat, to eat you have to be near food, and to be near food you have know where it is and then go there. The guidebook says where the food is. This explanation can be constructed either by using rules to build a reasoning chain or by applying preexisting schemas that capture the relationship.

Our task is to try to understand why the user holds a particular belief, a task similar to that faced by explanation-based story understanders. Since we have a similar task, we use a similar approach.

3. Representing User and Advisor Beliefs

The mistaken user beliefs that we consider involve plan applicability conditions, enablements, and effects. In this section we describe how we represent these beliefs. In essence, we utilize existing frameworks for representing planning knowledge, except that we are careful to distinguish between user and advisor beliefs.

Traditional planning systems [SACE74, FIKE71] represent an agent's planning knowledge as a database of operators associated with applicability conditions, preconditions, and effects. Since these systems have only one agent, the planner, the entries in the database are implicitly assumed to represent that agent's beliefs. However, because user misconceptions occur when the user's planning knowledge differs from the advisor's, systems that deal with user misconceptions must explicitly distinguish between advisor beliefs about what the user knows and advisor beliefs about what the advisor knows.

Our representation for beliefs [ABEL79, ABEL73] is similar to that of existing systems that keep track of the possibly contradictory knowledge of multiple

participants [ALVA86, POLL86b, FLOW82]. A **belief** relation represents an advisor's belief that an actor maintains that a particular plan applicability condition, precondition, or effect holds. The actor is either the user or the advisor.

belief(user, R)	Advisor believes that user maintains *R*
belief(advisor, R)	Advisor believes that advisor maintains *R*

In this chapter we do not discuss beliefs involving other relationships, such as a belief that an object has a particular property. In addition, for readability we do not use the **belief** predicate here but instead precede a list of planning relationships with either "the user believes" or "the advisor believes".

3.1. Representing Planning Relationships

The planning relation can be one of the relations between actions and states shown below. Here *A* denotes an action, which is either a primitive operator whose execution results in a set of state changes, or a plan, which is a sequence of these operators. *S*, *S1*, and *S2* denote states, which are descriptions of properties of objects.

causes(A,S)	Executing A has S an effect
!causes(A,S)	Executing A does not have effect S
enables(S1,A,S2)	S1 is necessary for A to have S2 as an effect
!enables(S1,A,S2)	S1 is unnecessary for A to have S2 as an effect
applies(A,S)	A is a correct or normal plan for achieving goal state S
!applies(A,S)	A is not a plan for achieving S
precludes(S1,S2)	S1 and S2 cannot exist simultaneously
!precludes(S1,S2)	S1 and S2 can exist simultaneously
goal(A,S)	Actor A wants to achieve S

These relationships are derived from existing representations. SPIRIT's [POLL86b] representation for planning knowledge uses **gen** to represent a state resulting in an action and **cgen** to represent a state resulting in an action only if some other state exists. **causes** and **enables** are identical in semantics to **gen** and **cgen**. **applies**, which has no analog in SPIRIT, is similar to the **intends** relation in BORIS [DYER83]. The difference between **causes** and **applies** is in whether the action is *intended* to cause the state that results from its execution to exist. **causes** represents cause-effect relations which are nonintentional – the effect is not necessarily a user goal – while **applies** represents a cause-effect relation between an action (sequence) or plan which is intended to achieve a desired state (a goal). An action **causes** a state whenever the state results from its execution. But an action **applies** to a state only when an actor believes the action causes the desired state to occur.

To see why this distinction is necessary, consider two actions that can be used by a user who wants to remove one of his files: typing "rm" followed by the file's name, and typing "rm" followed by an asterisk. Both have removing the file as one of their effects, but the latter also causes all other files to be removed as well, an

effect that is *not* the user's *goal*. Only typing "rm file" **applies** to removing a file, although both actions have an effect that **causes** the file to be removed.

To further illustrate the semantics of these relations, we use them to represent the first exchange in our example dialog.

User: I tried to remove a file with the "rm" command. But the file was not removed and the error message was permission denied. I checked and I own the file. What's wrong?

Advisor: To remove a file, you need to be able to write into the directory containing it. You do not need to own the file.

Three of the user's beliefs in this exchange are: (1) the "rm" command is used when one wants to remove a file, (2) one has to own a file to remove it, and (3) an error message resulted when the plan was executed. In terms of these planning relations, the user's beliefs are:

applies(using "rm file", the file's removal)
enables(owning the file, using "rm file", the file's removal)
causes(using "rm" on the user's file, an error message)

The advisor holds several similar beliefs, except that he believes that to remove a file it is necessary to have write permission on the directory containing it. In terms of the planning relationships, the advisor's beliefs are:

applies(using "rm file", the file's removal)
enables(directory write permission, using "rm", the file's removal)
causes(using "rm" on the user's file, an error message)

(The chapter is not concerned with representing notions such as "the file's removal" or "write permission on the directory containing the file". The details of the representation for such things may be found in [QUIL85].)

The user and advisor in this exchange share one belief that we have not yet represented. This belief is that using "rm" did not cause the user's file to be removed. To represent beliefs that a state does not result from an action, that a plan is not applicable to a goal, or that a state is not an enablement condition of an action having another state as a result, we use **!causes**, **!applies**, and **!enables**, respectively. The belief above is represented with **!causes**, a belief that "mkdir" is not used to remove a file is represented with **!applies**, and a belief that "rm" does not require owning the directory containing the file is represented with **!enables**.

!causes(using "rm" on the user's file, the file's removal)
!applies(using "mkdir file", the file's removal)
!enables(owning the directory, using "rm", the file's removal)

Examining user-advisor exchanges makes it is apparent that we must also be able to represent the notion that a state's existence caused a planning failure. Consider the following exchange.

User: I accidentally hit the up arrow key and it deleted 20 unanswered mail messages. How can I get them back?

Advisor: Hitting the up arrow does not delete your messages, but does result in your being disconnected from the etherplexer. You could not access your mail messages because they were moved to "mbox". The mail program requires that your mail messages be in "mailbox".

Here the advisor believes that the user's mail messages are inaccessible because they are not in the location the mail program expects them to be. The belief that the mail program requires the mail messages to be in the file "mailbox" can be represented using **enables**. The advisor's belief that the mail messages being in the file "mbox" prevents the mail program from accessing is represented with **precludes**, which captures the notion that two states are mutually exclusive.

enables(messages in "mailbox", use "mail", display messages)

precludes(messages in "mbox", messages in "mailbox")

precludes and **!precludes** relations between states can be inferred using rules such as "an object cannot be in two places at once".

One other relation we find useful is **goal**, which represents a belief that an actor wants to achieve a particular state. In the example above, the advisor believes that one goal of the user is accessing his mail messages.

goal (user, access user's mail messages)

Most user-modeling systems use a similar relation to explicitly represent that a state is a user's goal.

3.2. Summary of the Representation

The main focus of our work is in trying to detect and respond to user misconceptions. To do so, it is necessary to have some representation for user and advisor planning knowledge. Our representation is based on that used by traditional planning systems. The most important difference is that we take care to distinguish between things the advisor believes and things the advisor thinks the user believes. We also distinguish between actions that are intended to achieve a state and actions that happen to have a particular state as one of their effects. And we find it necessary to represent beliefs that two states cannot exist at the same time and that achieving a particular state is a goal of the user.

4. Explanation-Based Misconception Recognition and Response

Our approach to recognizing and responding to a potentially incorrect user belief revolves around the advisor trying to explain why he does not hold a particular user belief. But what exactly is an explanation? And what knowledge does the advisor need to find one? And, finally, how is an explanation found?

4.1. Explanations as Sets of Beliefs

An explanation is a set of advisor beliefs that justify his not sharing a user's belief. To find an explanation the advisor tries to find a set of advisor beliefs that contradict the user's belief.

For instance, the user in our earlier example incorrectly believes that owning a directory is a precondition for removing a file.

enables(own directory, use "rm", the file's removal)

The advisor holds two beliefs that together contradict this user belief. The first is the advisor's belief that the actual precondition is write permission on the directory containing the file. The other is his belief that owning a directory is a precondition to obtaining write permission on the directory.

enables (writeable directory, use "rm", the file's removal)
enables(own directory, use "chmod", obtain writeable directory)

Together these beliefs let the advisor infer that owning a directory is not a precondition of removing a file, the negation of the user's belief.

!enables(own directory, use "rm", the file's removal)

Thus, these beliefs constitute an explanation for the user's mistake: the user does not realize that owning a directory is merely a precondition to obtaining write permission on the directory, which is the actual precondition to removing the file.

4.2. Required Advisor Knowledge

Two types of knowledge are needed to locate these explanatory advisor beliefs: (1, a set of domain-specific beliefs, (2) a set of abstract explanation patterns that let the advisor recognize contradictory beliefs.

4.2.1. Domain-Specific Beliefs

The first type of required knowledge is a set of domain-specific beliefs about plan applicability conditions, preconditions, and effects. Examples of these include beliefs that "rm" is used to remove a file, and that it is necessary to have write permission on the directory containing the file. Without these types of beliefs it would be impossible for the advisor to correct user misconceptions about the preconditions for removing a file. This category of knowledge also includes beliefs such as a belief that "rm" is not used to remove a directory. These negated beliefs – **!applies**, **!enables**, **!causes**, and so on – are especially useful in detecting misconceptions. An advisor with the explicit belief that "rm" is not applicable to removing a directory, can easily detect that a user belief that "rm" is applicable to removing a directory is incorrect (although he may still have to do some work to understand why the user holds this belief).

These domain-specific beliefs are assumed to derive from past advisor experiences or past advisor interactions with users. An advisor who successfully uses "rm"

to remove a file will believe that using "rm" is applicable to the goal of removing a file. An advisor who uses "rm" to try to remove a directory and has it fail will believe that "rm" is not applicable to removing a directory. The negated beliefs correspond to the bug lists kept by many tutoring and planning systems [ANDJ85b, BURT82, STEV82, BROW78].

4.2.2. Potential Explanations

The other type of knowledge is a taxonomy of potential explanations. Each type of planning relation – **applies**, **enables**, and **effects** – is associated with a set of potential explanations that allow the advisor to recognize when his beliefs contradict the user's. Each potential explanation is an abstract pattern of planning relationships. To find a set of contradictory beliefs that constitute an explanation for why the advisor does not hold a particular user belief, the advisor classifies the belief and then tries to verify one of the potential explanations associated with that class of belief. The idea is that to verify a potential explanation, the advisor tries to prove, either by memory search or by deductive reasoning, that each of these planning relationships hold.

5. A Taxonomy of Potential Explanations

The potential explanations for why the advisor *does not* hold an instance of a plan-oriented belief are shown below. Each of these potential explanations suggests that to find a belief that contradicts the user's, the advisor must try to verify some small set of beliefs. This section presents a taxonomy of these explanations for each of the three types of beliefs.

5.1. Explanations for Incorrect Applies

There are four ways to recognize that the advisor does not hold a user belief that a plan is applicable to a goal; that is, a belief of the form **applies**(*Ap*, *Sp*). These are shown in Figure 1.

The first, *Achieves Other Goal*, applies to one of our earlier examples. The explanation is that the user's plan has no effect that achieves his goal, that his plan actually achieves some other goal, and that some other plan achieves his goal.

User: I tried to display my file with the "ls" command but it just printed the file's name.

Advisor: The "ls" command is not used to display the contents of files, the "more" command is. "ls" is used to list the names of your files.

The user's incorrect belief is that using "ls" displays a file. This belief is contradicted by the advisor's beliefs that using "ls" does not display the contents of files, that "ls" is applicable to listing the names of files, and that "more" can be used to display a file's contents.

Class of mistake	Potential explanation	English description
Plan Achieves Other Goal	!causes(Ap,Sp) applies(Ap,S) causes(A, Sp)	No effect achieves goal Plan applies to other goal Some other plan achieves goal
Plan Missing Needed Effect	!causes(Ap,Sp) applies(A, Sp) causes(A,S) !causes(Ap,S)	No effect achieves goal Other plan achieves goal That plan has an effect the user's plan does not have
Unachievable Plan Enablement	enables(S,Ap,Sp) !causes(Ao,S)	Some state enables Plan No action achieves this state
Plan Thwarts User Goal	applies(A, Sp) causes(Ap,S) precludes(S, Sp) goal(user, Sp) !causes(A,S)	Other plan achieves goal User plan has effect that thwarts the user's goal Other plan does not have that affect

Figure 1. Potential explanations for incorrect applies

The second explanation, *Plan Missing Needed Effect*, is that the user's plan P1 does not have one of the effects that the plan P2 (that achieves his goal) has.

User: I tried to remove my directory and I got an error message "directory not empty". But "ls" didn't list any files.
Advisor: Use "ls -a" to list all of your files. "ls" cannot be used to list all of your files because "ls" does not list those files whose names begin with a period.

The user's mistaken belief is that "ls" lists all file names. This belief is contradicted by the advisor's belief that "ls" does not have an effect that causes it to list files whose names begin with a period, an effect that the plan that achieves the goal (ls -a) has.

The third explanation, *Unachievable Plan Enablement*, is that there is no plan that achieves one of the user's plan's enablements.

User: So to read Margot's mail, all I have to do is "more ~flowers/mail".
Advisor: No, only "flowers" can read her mail.

The user mistakenly believes that his plan of using "more" to examine Margot's mail file will allow him to read her mail. This belief is contradicted by the advisor's beliefs that one of the enablements of reading a user's mail is that you have to be that user, and that no plan achieves this enablement.

The final explanation, *Plan Thwarts User Goal*, suggests that the user is unaware that another plan achieves the user's goal without an additional effect that the user's plan has.

User: To list files whose names begin with a number, I pipe "ls" to "grep [0-9]".

Advisor: Use "ls [0-9]*" instead. It is more efficient.

The user's mistaken belief is that piping "ls" to "grep" is the most appropriate plan for listing files whose names begin with a digit. This belief is contradicted by the advisor belief that the plan of using "ls [0-9]*" not only achieves the user's goal but also does not thwart his other goal of using his time efficiently.

5.2. Explanations for an Incorrect Enables

Just as there are several different explanations for why the advisor does not share a user belief that a plan is applicable to a goal, there are also several different explanations for why the advisor does not share a user belief that a state is a precondition to a plan achieving a goal; that is, a user belief of the form **enables**(*Se*, *Ap*, *Sp*). These explanations are shown in Figure 2.

Class of mistake	Potential explanation	English description
Enablement For Subgoal	enables(S,Ap,Sp) enables(Se,A,S)	State that is actual enablement Is enabled by user's enablement
Not Always Enablement	enables(S,Ap,Sp) causes(A,Sp) !enables(Se,A,Sp)	Other state is necessary Plan achieves user's goal without claimed enablement
Enablement Too Specific	causes(A,Se) causes(A,S) enables(S,Ap,Sp) causes(Ao,S) !causes(Ao,Se)	User enablement results from action that achieves real enablement Other action achieves real enablement without user's enablement as effect

Figure 2. Potential explanations for incorrect enablement

The first, *Enablement For Subgoal*, explains the user's mistake in our introductory exchange. The explanation is that the user is unaware that his precondition is not a precondition of the goal itself, but of one of its preconditions.

User: So to remove a file, I have to own the directory that contains it.

Advisor: No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

The user's belief is contradicted by the advisor's beliefs that owning a directory is a precondition for achieving write permission on it, and that having write permission is a precondition for removing a file.

The second explanation, *Not Always Enablement*, is that some plan achieves the user's goal without the user's claimed precondition.

User: So I can only edit files when I'm on a smart terminal?

Advisor: Only if you edit with "re". "vi" works fine on a dumb terminal.

The user's incorrect belief is that it is necessary to have a smart terminal to edit a file. This belief is contradicted by the advisor's belief that plans exist that allow him to edit but do not require a smart terminal.

The last explanation, *Enablement Too Specific*, is that the user's precondition is less general than the actual precondition for achieving his goal.

User: So I have to remove a file to create a file?

Advisor: You do not have to remove a file to create a file. You must have enough free space. Removing a file is only one way to obtain it. You could also ask the system administrator for more space.

The user mistakenly believes that it is necessary to remove an existing file before a new file can be created. This belief is contradicted by the advisor's belief that the precondition is sufficient space for the new file, which can be achieved either by executing a plan for removing a file or by executing the plan of requesting more space.

5.3. Explanations for Incorrect Causes

One final class of user misconception is an incorrect belief that a particular state results from a plan's execution; that is, a user belief of the form **causes**(*Ap*, *Sp*). The explanations for this class are shown in Figure 3.

Class of mistake	Potential explanation	English description
Effect Requires Enablement Unfulfilled	enables(S,Ap,Sp) !causes(A,S)	State required for Action and no way to achieve State
Effect Inferred From Other Effect	causes(Ap,So) precludes(So, S) precludes(Sp, S)	Action causes other effect That effect precludes a state that is precluded by user's effect
Effect From Another Plan	causes(Ap,So) causes(A, Sp)	Action used to cause other effect Some other action caused it

Figure 3. Potential explanations for incorrect effect

The first, *Effect Requires Unfulfilled Enablement*, suggests that the user is unaware that a particular state is required for the plan to have the claimed effect.

User: I was cleaning out my account when I accidentally deleted all the command files by typing "rm".

Advisor: You can't delete the command files with "rm" unless you are the system administrator.

The user incorrectly believes that typing "rm" removed several system files. The advisor's contradictory belief is that it is necessary for the user to be the system administrator for this effect to occur.

The second, *Effect Inferred From Other Effect*, accounts for another one of the earlier examples. It suggests that the user has inferred an incorrect plan effect from an actual plan effect.

User: I accidentally hit the up arrow key and it deleted 20 unanswered mail messages. How can I get them back?

Advisor: Hitting the up arrow does not delete your messages, but does result in your being disconnected from the etherplexer. You could not access your mail messages because they were moved to "mbox". The mail program requires that your mail messages be in "mailbox".

The user incorrectly believes that one effect of hitting uparrow was that his mail messages were deleted and are now inaccessible. This contradicts the advisor's belief that one effect of hitting uparrow is that files are moved to a different location, which makes them seem inaccessible.

The third, *Effect From Another Plan*, accounts for the following example. The explanation is that the user is unaware that the user's action actually has a different effect.

User: I deleted a file by typing "remove".

Advisor: No, typing "rm" deleted it. Typing "remove" deletes a mail message from the mail program.

The user's mistaken belief is that typing "remove" deletes a file. This contradicts the advisor's belief that typing "remove" actually throws away old mail messages.

6. A Detailed Process Model

When presented with a potentially incorrect user belief, the advisor must answer two questions: Is the belief mistaken? And, if so, why does the user believe it? To do so, the advisor first tries to determine whether he holds the user's belief. If he does not, the advisor assumes the user's belief is mistaken and tries to find an explanation for why the user holds this belief. The taxonomy guides this search for an explanation.

6.1. Finding an Explanation

The advisor finds an explanation by hypothesizing and trying to confirm one of the potential explanations associated with the user's type of belief. The advisor

1. Classifies the user's belief according to its type: **applies**, **enables**, or **effects**.
2. Selects one of the potential explanations associated with that class of belief.
3. Instantiates this particular potential explanation with information from the user's belief.
4. Tries to verify each of the planning relationships within the potential explanation. If all can be verified, this potential explanation is the desired explanation.
5. Repeats the process until one of the potential explanations associated with this belief's type is verified or all potential explanations have been tried and have failed.

The result of this process is that the advisor has verified that he holds a particular set of beliefs that contradict the user's belief. These beliefs constitute the desired explanation.

6.2. Verifying an Advisor Belief

The advisor's verifying whether or not he holds beliefs about particular planning relationships constitutes an important part of finding an explanation. To verify a belief, the advisor first searches his memory for the desired piece of planning knowledge. If it isn't found, the advisor then tries to deduce that the relationship holds. Once the advisor has proved that the planning relationship holds, either by search or by reasoning, that piece of knowledge is noted to be an advisor belief.

Consider, for example, the process of verifying that the advisor holds a belief that owning a directory is not a precondition of removing a file. If this fact is already known from past experience, the advisor will recognize it during memory search. If not, the advisor can try to deduce it. One rule that applies here says that "if a state S is not one of the known states that are preconditions to an action A for achieving a goal state, then assume that *S* is not a precondition." Here this means that if owning a directory is not among the known preconditions for removing a file, assume it is not a precondition for removing a file. In effect, this is the closed world assumption: If the advisor cannot show that a belief holds, assume its negation.

6.3. An Example

This section shows how the advisor processes the user belief that owning a directory is a precondition of removing a file.

enables(user own directory, use "rm", the file's removal)

The advisor first determines whether he holds this belief. He verifies that he does not by noting that owning a directory is not a known precondition of removing a file and applying the closed-world assumption. The advisor then tries to

explain why he does not hold this user belief. He notices that this belief can be classified as a belief that some state *Sp* (owning the directory) is a precondition to achieving some other state *Sg* (removing a file). The taxonomy provides several potential explanations for this type of belief. One potential explanation is that *Sp* is actually a precondition of achieving a state *S*, which is a precondition to achieving *Sg*. In this case, instantiating *Sp* and *Sg* leads the advisor to try to verify that he holds these two beliefs:

enables(*S*, use "rm file", the file's removal)
enables(owning a directory, *A*, *S*)

These beliefs are verified when the advisor finds that having write permission on a directory is a precondition to removing a file, and that owning a directory is a precondition to obtaining write permission on the directory (the advisor verifies these beliefs either by finding the belief relationships in memory or by deducing them). These matching beliefs contradict the user's belief, confirming it as a misconception and providing a possible explanation for why the user holds it. Finally, the advisor presents the resulting beliefs to the user.

6.4. The Point of Potential Explanations

Having a taxonomy of potential explanations lessens the amount of reasoning the advisor must do to detect and respond to user misconceptions.

To see why, consider an advisor trying to understand how the user arrived at the mistaken belief that a precondition of removing a file is owning the directory containing it. The advisor is trying to find some connection between the user's enablement and removing a file. The potential explanations suggest how to find specific, likely-to-be-useful connections. For example, the potential explanation *Enablement for Subgoal* suggests examining whether achieving any of the preconditions of removing a file requires owning a directory.

Without a set of potential explanations it becomes necessary to reason from a set of rules that describe likely differences between user and advisor beliefs. One rule might be that a user may incorrectly attribute an enablement of one action to another action. Another rule might be that a user may incorrectly attribute the result of one action to another action. From a set of such rules the advisor must somehow deduce the cause of the user's mistake. The potential explanations turn the problem of finding an explanation into one of guided memory search rather than reasoning from first principles.

7. Accessing Advisor Planning Knowledge

The advisor verifies potential explanations by searching his planning knowledge for the appropriate belief relationships. The standard way of organizing planning knowledge is a database in which each plan is associated with applicability conditions, enablements, and effects. Using our representation for planning knowledge,

this corresponds to having each plan associated with a set of **applies**, **enables**, and **causes** beliefs. For example, some beliefs about "rm" are that "rm" is applicable to removing a file; that two enablements of using "rm" to remove a file are having write permission on the directory containing the file, and making sure that the file is not a currently executing program; and that one effect of using "rm" to remove a file is that the file is removed. The advisor has similar sets of beliefs about other plans. As demonstrated earlier, however, these beliefs – which essentially capture the advisor's knowledge about planning successes – are not sufficient. The advisor also needs knowledge about planning failures.

7.1. Organizing Knowledge About Planning Failures

The advisor has two types of knowledge about planning failures associated with each plan. The first type involves negated plan-oriented beliefs and the explanations for why these beliefs are held. These beliefs involve goals to which a plan does not apply (**!applies**), states that do not enable a plan's achieving a goal (**!enables**), and the states that do not result from an action's execution (**!causes**). This means that along with the above knowledge about "rm", the advisor has:

1. *Beliefs about attempts to use "rm" to achieve other goals, and explanations for why those attempts failed.* The advisor may believe that "rm" is not an applicable plan for removing a directory. The advisor's explanation for this belief is that he believes "rm" has no effect that removes a directory and is instead used to remove files, and that another command, "rmdir" is used to remove a directory (*Plan Achieves Different Goal*).
2. *Beliefs about states that are not enablements of "rm", and explanations for why those states are not necessary.* The advisor may believe that a file that is being displayed can be removed. The explanation for this belief is that the advisor believes that he has previously removed a displayed file and that the enablement is instead simply that the file cannot be an executing program (*Not Always Enablement*).
3. *Beliefs about states that do not result from "rm", and explanations for why those states do not occur.* The advisor may believe that "rm" does not query the user before removing a file. The explanation for this belief is that "rm" is known to produce no output, and that "rm -i" queries the user before removing the file (*Effect From Another Plan*).

These beliefs and explanations are stored along with the rest of the advisor's knowledge of each plan. Accessing these stored beliefs allows the advisor to immediately recognize when a user's belief is contradicted without doing any reasoning. The explanation patterns are used only if a canned explanation cannot be found.

The other type of knowledge associated with each plan describes the results of failing to hold correct beliefs when the plan was used. These include beliefs about what happens when a plan is applied to an incorrect goal, or is applied without various plan enablements being achieved, or produces unexpected effects. This information is maintained as lists of effect/failed-enablement, effect/failed-application pairs, and effect/related-effect pairs. Some of these belief pairs for "rm" include

the belief that a "permission denied" message results from failing to have write permission on the directory containing the file; that a "file is a directory" message results when "rm" is incorrectly applied to the goal of removing a directory; and that a "permission denied" message means that the file was not removed. This organization means that an advisor given an unusual plan effect can retrieve the failed enablement or incorrect plan application that led to it, as well as any other effects that are likely to have occurred with it.

7.2. Retrieving Advisor Planning Knowledge

Although the advisor possesses a great many beliefs, only a few apply to the potential explanation for any particular incorrect user belief. "rm", for example, has many enablements: the user must have write permission on the directory containing the file, the file being removed cannot be a program in use, the file system containing the file must be mounted, and so on. But if the advisor knows the user's use of "rm" resulted in a permission denied message, only the first enabling condition is relevant, and the other enablements should be ignored. On the other hand, if the advisor knows the result was the message "text file busy", only the second condition is relevant. The point is that somehow the advisor must be able to determine which of these beliefs are relevant to his search for an explanation. Ideally, when the advisor verifies beliefs that are part of any potential explanation, he first examines the likely-to-be-relevant beliefs, and then his more general database of planning knowledge.

To find a set of relevant advisor beliefs, the advisor examines each of the beliefs in the user model. If a user belief corresponds to one of the plan's standard advisor beliefs (that is, beliefs about the plan's usual applicability conditions, enablements, or effects), the corresponding advisor belief is added to the set. The user belief above that "rm" is used to remove a file means that the similar advisor belief is relevant. Otherwise, the advisor uses the user belief as an index into the plan's failure-oriented advisor beliefs, and adds any indexed beliefs to the set. The user belief above that "rm" printed a permission denied message indexes the advisor beliefs that having write permission on the directory containing the file is an enablement to removing a file, which implies that these beliefs are relevant.

7.3. An Example

Consider the user model we developed earlier for the first exchange in our example dialog.

applies(use-rm, remove a file)
!causes(use-rm, file removed)
causes(use-rm, permission denied)
enables(own file, use-rm, remove a file)

The advisor constructs a set of relevant advisor beliefs by using each of these beliefs as indices into his knowledge base. The user's belief that "rm" is used to

remove a file is a standard advisor belief, and is therefore automatically added to the set. The user's belief that "rm" does not cause a file to be removed is not a standard belief, and is used to try to index other advisor beliefs. But it does not index any beliefs. The user's belief that "rm" causes a permission denied belief is also not a standard advisor belief and it, too, is used to try to index other advisor beliefs. It succeeds, indexing a pair of beliefs: that having write permission enables removing the file (effect/failed-enablement), and that when "rm" prints permission denied, the file is not removed (effect/related-effect). Finally, the last user belief, that owning a file enables removing a file, is used as an index but retrieves no beliefs. The result is that the following advisor beliefs are marked relevant.

applies(use-rm, remove file)
enables(write directory, use-rm, write permission on a directory)
!causes(use-rm, file removal)
causes(use-rm, permission denied)

This means that when the advisor later tries to verify various potential explanations, write permission will show up as the relevant enablement.

7.4. Related Work in Memory Organization

Unlike traditional planning systems, our advisor uses knowledge of past planning failures. Our organization of this knowledge is based on earlier work in organizing past experiences and indexing them appropriately for efficient search and retrieval [KOLO86, KOLO84, SCHR82]. These systems organized knowledge of a class of episodes around the differences between each episode and the class prototype. For example, specific diplomatic trips were indexed around general knowledge of diplomatic trips by differences in location, participants, purpose, and so on. A particular diplomatic experience was accessed by providing its location, participants, purpose, or some combination of these features.

Our approach is similar. A plan's normal behavior corresponds to the prototypic episode, and its different plan failures correspond to the other episodes. Specifying some information about the failed episode (such as its results) allows the advisor to retrieve its cause (the particular failed enablement or bad plan application).

8. Related Work

Two approaches have been used to detect and correct misconceptions. The first approach is used by many intelligent tutoring systems. These systems [ANDJ85b, BROW78, BURT82, STEV82] locate mistaken beliefs in a database of error-explanation pairs and provide the associated explanation. A basic problem with this approach is that, because there is no information about the underlying causes of the errors, these systems can handle only those misconceptions known in advance.

The other approach avoids the difficulty inherent in enumerating all possible misconceptions within a domain by using strategies that address an entire class

of misconceptions. The user's misconception is classified according to the abstract reasoning error likely to have led to it. This approach shares many features with recognizing abstract thematic situations (such as irony) in narratives, where such situations are defined in terms of abstract planning errors made by the narrative characters [DYER88, DOLA86, DYER83]. Once an appropriate strategy is found, it can be used to generate advice (in narratives, this advice may be in the form of adages). In advisory systems, this approach has been applied to both object- and plan-oriented misconceptions.

8.1. Object-Oriented Misconceptions

ROMPER [MCCO*] corrects user misconceptions dealing with whether an object is an instance of a particular class of objects or possesses a particular property.

User: I thought whales were fish.

ROMPER: No, they are mammals. You may have thought they were fish because they are fin-bearing and live in the water. However, they are mammals since, while fish have gills, whales breathe through lungs and feed their young with milk.

ROMPER classifies a user's misconception as either a misclassification or misattribution and then selects one of several strategies associated with each class of misconception to generate a response. Each strategy addresses a different type of reasoning error, and is selected based on ROMPER's own beliefs about objects and its model of the user's relevant beliefs. One such strategy is useful when the advisor believes that *X isa Z*, the user mistakenly believes that *X isa Y*, and the advisor believes that *X* and *Y* share certain attributes. The strategy suggests presenting these shared attributes as a possible reason for the misclassification, and pointing out the unshared attributes that lead the advisor to believe that *X isa Z*.

Despite dealing with a very different class of misconceptions, ROMPER's approach is similar to ours.

8.2. Plan-Oriented Misconceptions

Two efforts have examined detecting and responding to plan-oriented misconceptions.

Joshi, Webber, and Weishedel [JOSH84a] suggest using a strategy-based approach to provide cooperative responses to problematic planning requests. They consider "How do I do *X*?" questions in which *X* can be inferred to be a subgoal of a more important goal *Y*.

User: How can I drop cs577?

System: It is too late in the quarter to drop it. But you can avoid failing by taking an incomplete and finishing your work next quarter.

They provide several strategies, listed below, for selecting the contents of a reasonable response, with strategy selection based on the advisor's beliefs about which plans achieve a particular goal and the achievability of their preconditions.

	Situation	Response
(1)	Unachievable Precondition E of X	Provide E
	(a) Plan P achieves Y	Provide P
	(b) No plan to achieve Y	Point this out
(2)	X doesn't help achieve Y	Point this out
	(a) Plan P achieves Y	Provide P
	(b) No plan to achieve Y	Point this out
(3)	Plan P better way to achieve Y	Provide P
(4)	X only way to achieve Y	Point this out
(5)	Plan P involving uncontrollable event E achieves Y	Provide P

One such strategy, which is useful when the advisor believes that *X* cannot be achieved because of an impossible-to-achieve precondition, is to point out the troublesome precondition and suggest an alternate plan that achieves *Y*.

Our work differs from theirs in several respects. The main difference is that they focus on correcting the user's misconception instead of trying to explain why it occurred. Only one strategy above is concerned with providing an explanation that addresses the source of a user misconception (in this case, an inappropriate plan). The other strategies describe situations in which achieving X is inappropriate and an alternate plan for Y exists and should be presented to the user as a correction. In addition, they do not consider responding to incorrect beliefs about plan preconditions or effects.

The other effort, SPIRIT [POLL86a, POLL86b] tries to detect the inappropriate plans underlying queries made by users of a computer mail program and the mistaken user beliefs underlying those plans.

User: I want to prevent Tom from reading my file. How can I set the permissions on it to faculty-read only?

System: You can make the file readable by faculty only using "set permission". However, Tom can still read it because he's the system administrator.

User misconceptions about the applicability and executability of plans are detected by reasoning about the likely differences between the advisor's beliefs and the user's, with various rules used to infer these differences. One such rule, used to detect the source of the misconception above, states that an advisor who believes that an act has a particular result under certain conditions can infer that the user has a similar belief missing one of the required conditions.

SPIRIT has a task similar to ours but takes a very different approach, trying to determine the cause of the user's error through reasoning from first principles rather than memory search. In addition, SPIRIT cannot detect or respond to mistakes involving plan applicability conditions or preconditions. Finally, SPIRIT does not specify how knowledge of the cause of the user's mistaken belief affects the information to be included in a cooperative response, something that naturally is beyond our model.

8.3. UNIX Advisors

Finally, there are two other related research efforts, UC [WILE86, WILE84] and SC [KEMK86], that address providing advice to novice UNIX users. Neither system, however, detects or responds to misconceptions. Instead, both are concerned with tailoring a response to a question to reflect the user's level of expertise. UC's user modeling component, KNOME [CHIN*], analyzes a user's questions to determine which stereotypical class the user belongs to and then uses this information to provide more details and possibly more examples to less experienced users.

Novice: What does the "rwho" command do?

UC: Rwho lists all users on the network, their tty, and their login and idle times.

Expert: What does the "rwho" command do?

UC: Rwho is like who, except rwho lists all users on the network.

SC's user modeling component, SCUM [NESS87], takes an approach similar to UC's, also using stereotypical information. These approaches are complementary to ours.

9. Implementation Details

The theory discussed in this chapter is implemented in AQUA, a computer program under development at UCLA. The latest version of AQUA is implemented in Prolog and runs on Apollo workstations. Given a set of user beliefs involving plan applicability conditions, preconditions, or effects, AQUA determines which of these user beliefs are incorrect and what missing or mistaken user beliefs are likely to have led to them, and then produces a set of advisor beliefs that capture the content of the advisor's response. AQUA's domain of expertise is in the basic plans used to manipulate and access files, directories, and electronic mail. It has been used to detect and respond to at least two different incorrect user beliefs in each class of misconception that we have identified.

10. Limitations and Future Work

Our approach to determining why a user has a particular incorrect belief has been to let potential explanations direct the search for the advisor beliefs that serve as an appropriate explanation. Our focus has been on discovering and representing these explanations. The limitations of our approach arise in areas we have ignored, each of which is an interesting area of research.

10.1. Building the User Model

Most of our work has involved automatically inferring the sources of mistaken user beliefs, given an initial set of user beliefs. Unfortunately, users rarely explicitly state their beliefs, leaving the advisor the difficult task of inferring them. Consider our introductory exchange.

User: I tried to remove a file with the "rm" command. But the file was not removed and the error message was permission denied. I checked and I own the file. What's wrong?

Advisor: To remove a file, you need to be able to write into the directory containing it. You do not need to own the file.

Here the advisor must infer the user's beliefs that (1) using "rm" is applicable to removing a file, that (2) using "rm" did not cause the file's removal, that (3) using "rm" resulted in an error message, and that (4) owning a file is a precondition to removing it. How can these beliefs be inferred, assuming that their components (plans, goals, and state descriptions) have been recognized?

Our approach to recognizing these components has been to apply a set of belief inference rules to each mentioned or inferred plan, goal, and state. Each of these rules describes a situation to check for, and suggests a user belief that may be inferred in that situation. For example, one such rule is "if the user executes a plan, and has achieving a particular state as his goal, assume the user believes that plan applies to that state." This rule helps infer the first user belief from the user's executing "rm" to try to remove a file. The second belief is inferred from the rule that "if an utterance describes the nonexistence of a state that is a believed result of an action, assume that the user believes that the action did not cause the state." A similar rule helps infer the third belief.

Inferring the final belief, that owning a file is a precondition to its removal, is a difficult task. Because there are a potentially-infinite number of incorrect user beliefs about the preconditions of removing a file, the advisor cannot simply match owning a file against a list of incorrect preconditions. Because the user may have been discussing other plans and other goals the advisor cannot simply assume that any utterance after a plan's failure refers to its preconditions. Instead, the advisor must infer this user belief from the knowledge that the user did some sort of verify-action, the knowledge that one plan for dealing with a plan failure is to try to verify that the enablements of the plan have been achieved, and the knowledge that both owning the file and having write permission are different instantiations of having sufficient permission. The rule used to infer this belief is "if the user tries to verify a state S, and both S and a known precondition are instances of a more general state, assume that the user believes that S is a precondition of the user's goal."

These rules are useful and provide a feel for how different user beliefs can be recognized – making more plausible our assumption that an initial user model exists – but they fall far short of being a serious process model for belief recognition. Creating such a model is one of our most pressing research concerns. Recognizing these components of these beliefs (the user's plans and goals) is also an active area of research [CARB*, KAUT86, GOOM86, WILE86].

10.2. Other Classes of Misconceptions

We are currently studying how well the classes of misconceptions described here account for responses to misconceptions in domains other than the problems of novice computer users, such as the domain of simple day-to-day planning. In addition, we are examining other classes of planning misconceptions. For example, to respond to an incorrect user belief such as "rm" cannot be used to remove a file, the advisor needs potential explanations for why an action does not apply to a particular goal state.

We do not yet know whether our approach is suitable for generating responses to misconceptions that are not directly related to plan-goal interactions, such as mistakes in referring to an object. Consider the following exchange:

User: Diana is up but I cannot access my file.

Advisor: Your files are on Rhea, not Diana. They moved your files yesterday because your file system was full.

Here the user's problem is that he is incorrectly using "Diana" to refer to the machine his files are on. We are examining whether our approach is extendable to respond to these types of user misconceptions.

10.3. Response Generation

The response we provide is a set of advisor beliefs that is as complete as possible. We make no attempt to use knowledge about other user beliefs to modify our response to provide only the most relevant beliefs. However, if the advisor can infer that a user knows that his plan has failed (perhaps because of the error message a command produces), he need not inform the user that his plan is incorrect. One straightforward way to extend our model is to have the advisor filter out those beliefs he can infer the user has.

The advisor should use information about the user to tailor the response based on the user's level of expertise. Recall the following exchange.

User: I tried to remove my directory and I got an error message "directory not empty". But "ls" didn't list any files.

Advisor: Use "ls -a" to list all of your files. "ls" cannot be used to list all of your files because "ls" does not list those files whose names begin with a period.

An advisor who knows the user is a novice might want to augment his response with an explanation that "-a" is a command option and that command options cause changes in the normal behavior of commands. Several researchers are working on tailoring the response to the user based on knowledge about the user's expertise [CHIN*, PARI*].

10.4. Acquiring and Using Planning Knowledge

Currently, to verify a potential explanation, the advisor searches his memory for relevant beliefs. While we have specified the types of planning knowledge the advisor needs and suggested how this knowledge should be organized to facilitate rapid retrieval of relevant beliefs, we have not shown how this information can be automatically acquired and appropriately organized. Similarly, we have assumed the advisor possesses an abstract taxonomy of potential explanations for planning failures, but have not specified how this taxonomy is acquired. Initial work in recognizing and indexing abstract configurations of planning relations is discussed in [DOLA86, DOLA85].

In addition, the advisor may be required to verify a belief that he does not already possess somewhere in memory. Suppose the advisor is trying to verify that write permission is needed to remove a file. The advisor may be able to deduce this from his beliefs that write permission is needed to make changes on objects and that removing a file involves making a change to a directory. This requires more powerful reasoning capabilities than simple memory search.

11. Conclusions

We have presented an explanation-based approach to the problem of recognizing and responding to user misconceptions. The advisor determines the source of an incorrect user belief by finding an explanation for why he does not hold the user's belief. The explanation is found by hypothesizing and trying to verify a small set of potential explanations associated with each type of user belief. In essence, the model uses information about likely sources of different classes of user misconceptions to recognize user mistakes and infer their underlying causes.

This approach is attractive for several reasons. First, because it has information about classes of abstract misconceptions, it can handle misconceptions of which it has no prior knowledge, as long as they fall into one of these classes. Its small set of potential explanations account for a large number of specific user mistakes. Second, the model circumvents the need for general deductive reasoning by using knowledge of the types of misconceptions users are likely to make. However, the model can also use its knowledge of specific misconceptions and their sources to rapidly produce responses to misconceptions it has seen before, as do tutoring systems. In addition, the system can easily be augmented to use general reasoning when it is confronted with a misconception that cannot be explained by any of its potential or canned explanations, as does SPIRIT. Finally, and perhaps most importantly, our approach leads to responses similar to (and sometimes more informative than) those of the many UNIX advisors we have observed.

12. References

[ABEL73], [ABEL79], [ALVA86], [ANDJ85b], [BROW78], [BURT82], [CARB*], [CHIN*], [DOLA85], [DOLA86], [DYER83], [DYER88], [FIKE71], [FLOW82], [GOOM86], [JOSH84a], [KAUT86], [KEMK86], [KOLO84], [KOLO86], [MCCO*], [NESS87], [PARI*], [POLL86a], [POLL86b], [QUIL85], [QUIL88], [SACE74], [SCHR77], [SCHR82], [SCHR86], [STEV82], [WILE83], [WILE84], [WILE86]

6 Plan Recognition and Its Use in Understanding Dialog [1]

Sandra Carberry

Abstract[2]

Analysis of naturally occurring dialog suggests that human information-providers often use knowledge acquired from the dialog to understand subsequent utterances and provide cooperative, helpful responses. One of the most important ways in which an information-seeking dialog can be assimilated is by inferring the task-related plan motivating an information-seeker's queries. This chapter presents two process models. The first dynamically constructs a model of an information-seeker's plans and goals from an ongoing dialog; and the second reasons on this component of a user model to understand one class of problematic utterance. The chapter further addresses the problem of detecting and recovering from discrepancies between the system's model of the user and the actual plan under construction by the user. It proposes a four-phase approach to detecting and recovering from disparate models, and argues for an enriched user model that differentiates among its components on the basis of the support it accords each component as a correct and intended part of the user's plan.

1. Introduction

Analysis of naturally occurring dialog suggests that when two individuals participate in an information-seeking dialog, the information-provider often uses the information exchanged during the dialog and her knowledge of the domain to hypothesize a model of the speaker, and that this model is adjusted and expanded as the dialog progresses. There are many ways in which an information-seeker can be modeled, but one of the most significant of these is the inference of the underlying task-related plan motivating the information-seeker's queries. It appears that human information-providers often attempt to infer this plan and use their acquired knowledge to interpret subsequent utterances and provide useful responses.

---

[1] This chapter is a revised and extended version of [CARB88]

[2] Some of this work was partially supported by a grant from the National Science Foundation, IST-8311400, and a subcontract from Bolt Beranek and Newman Inc. of a grant from the National Science Foundation, IST-8419162.

We are interested in a class of information-seeking dialogs in which the information-seeker is attempting to construct a plan for a task that will be executed after the dialog terminates. These dialogs are representative of a large percentage of interactions with database management systems, decision support systems, and expert systems. Typical tasks include expanding a company's product line, purchasing a home, or pursuing a degree at a university. One aspect of our research has been to develop a strategy for dynamically constructing a model of an information-seeker's underlying task-related plan from an ongoing dialog and for tracking his focus of attention in this plan. Since the system's beliefs about the user's plans and goals provide a context for understanding subsequent dialog, we will refer to this part of a user model as a *context model*.

In an information-seeking dialog, the information-provider is engaged in helping the information-seeker construct a plan for his task. The information-seeker must communicate whatever information he believes necessary for the information-provider to fulfill her helping role. But naturally occurring dialog is both imperfect and incomplete. A speaker may inadvertently use incorrect terminology in constructing the language representation of his intended query, or he may short-cut its complete specification. Our analysis of naturally occurring dialog indicates that human information-providers often use knowledge acquired from the dialog to understand many of an information-seeker's faulty utterances and allow the dialog to continue without interruption. We believe that a natural language system should operate in a similar manner since, if its communication is to be regarded as natural, the system must be able to handle the full spectrum of utterances that humans understand with relative ease.

Two kinds of utterances that fall in this category are those that are imperfect and those that are incomplete. Part of this research has been concerned with investigating how a user model can aid in understanding these. We have developed a method, based on Grice's theory of meaning and maxim of relevance, for handling pragmatically ill-formed utterances. It uses the task-related plan inferred for an information-seeker to suggest variants of an ill-formed utterance that might represent his intentions or at least satisfy his perceived needs. We have also developed a method for understanding intersentential elliptical utterances that occur during the course of an information-seeking dialog. Our strategy uses discourse expectations and our model of the speaker's plan to identify the discourse goal that he is pursuing via an elliptical fragment and to interpret his elliptical utterance relative to his task-related plan [CARB85].

Section 2 of this chapter briefly reviews related work in user modeling, and Section 3 presents our strategy for dynamically inferring a model of the user's plan from an ongoing dialog. Section 4 describes our mechanism for reasoning on this user model to understand utterances that are problematic for current natural language systems. These ideas have been implemented and tested in the IREPS system (Intelligent REsPonse System). This is an ongoing research effort whose objective is a robust natural language interface to information systems. The component that infers the user's underlying task-related plan is called TRACK. To move from one domain to another, only the corpus of domain-dependent plans and goals must be reconstructed; the heuristics and processing strategies are completely transportable.

Section 5 describes our current research on relaxing restrictive assumptions present in previous work on plan recognition and developing a more robust plan inference framework. We present a four-phase approach to handling the problem of possible disparity between the system's context model and the actual plan under construction by the user, and argue for an enriched model that differentiates among its components according to the support it accords each component as a correct and intended part of the user's plan. Section 6 relates our research on plan recognition to other work on user modeling.

Throughout this chapter, the information-seeker and information-provider will be referred to as IS and IP respectively.

2. Plan Recognition in Dialog Systems

Early work in dialog understanding concentrated on apprentice-expert dialogs, during which an expert guided an apprentice in performing a task. Grosz [GROS77a] formulated heuristics for recognizing shifts in focus of attention in the task structure and presented a strategy that used the knowledge currently focused on by the dialog participants to identify the referents of definite noun phrases appearing in an utterance. Robinson [ROBI80, ROBI81] extended Grosz's work and developed a process model for determining the referents of verb phrases, such as variants of *do* in the utterance

"I've done it."

However, in apprentice-expert dialogs, the overall task that the apprentice i attempting to perform is known at the outset of the dialog, and the ordering of actions and subtasks in the plan being executed strongly influences the dialog between expert and apprentice. This differs from the kind of information-seeking dialog that we are investigating, in which the information-seeker is attempting to construct a plan for a task that will be executed after the dialog is complete. In such dialogs, the information-seeker's utterances are not tightly constrained by the order in which actions in the task will eventually be executed, as in apprentice-expert dialogs.

Allen [ALLE80, PERR80] inferred the goal underlying a speaker's utterance in the context of an information agent in a train setting. This inferred goal was used to account for extra helpful information included in the agent's response, and the inference path connecting the speaker's utterance and the inferred goal was used to interpret indirect speech acts. However, Allen's domain was very restricted; the only domain goals were *meeting a train* and *boarding a train*, each of which could be accomplished by a few primitive steps, and his system was primarily concerned with utterances that might occur at the outset of a dialog. In more complex domains, the information-seeker's complete plan will consist of a hierarchy of subplans and subgoals which accomplish his overall goal. Such a complete plan is not immediately evident from a single utterance, and individual utterances must be related to one another to build the user's plan as the dialog progresses.

Sidner [SIDN83b, SIDN85] and Litman [LITM86] developed enhanced models of plan inference. However, both were concerned with dialogs that were initiated

in order to begin or continue execution of an underlying task (display of structures on a graphics terminal and meeting/boarding a train), and the dialogs were therefore constrained by the order in which individual actions in the task must be executed. In addition, Sidner investigated how discourse markers aid in recognizing the speaker's intent, and Litman studied a meta-plan framework and its use in handling clarification subdialogs.

Pollack [POLL86a] has recently proposed that plans be viewed as mental phenomena. She contends that, in order to comprehend an utterance and relate it to the user's plan, the system must reason about the configuration of beliefs and intentions that it should ascribe to the user. This will be discussed further in Section 5.

3. Inferring and Modeling the Task-Related Plan

In order to reason about what the user wants to accomplish, the system must have knowledge about the goals that a user might pursue in a domain and plans for accomplishing those goals. We view a plan as the means by which an agent can accomplish a non-primitive task-related goal. Using an extension of the STRIPS formalism [FIKE71, ALLE80], we represent a plan as a structure containing applicability conditions, preconditions, a plan-body, and effects. Applicability conditions and preconditions both represent constraints that must be satisfied before a plan can be executed. However, an agent can plan to satisfy preconditions, whereas it is anomalous to plan to satisfy applicability conditions; the latter determine whether it is reasonable to even consider a particular plan for achieving a desired goal. The plan-body contains a set of subgoals and the effects represent the results of successfully executing the plan. Arguments in plans are either constants, represented as uppercase strings, or typed variables, represented as lowercase strings preceded by the character "_" and followed by the characters ":&" and an uppercase string giving the variable's type. Figure 1 presents a sample plan used by TRACK. These plans are hierarchical, since many of the subgoals in a plan are non-primitive and therefore have associated plans which may be substituted for them. Thus a plan can be expanded to any desired degree of detail.

```
Learn-Material(_agent:&PERSON, _sect:&SECTIONS, _syl:&SYLLABI)
    Plan-Body:
        Learn-From-Person(_agent:&PERSON, _sect:&SECTIONS, _fac:&FACULTY)
            where Teaches(_fac:&FACULTY, _sect:&SECTIONS)
        Learn-From-Text(_agent:&PERSON, _txt:&TEXTS)
            where Uses(_sect:&SECTIONS, _txt:&TEXTS)
    Effects:
        Learn-Material(_agent:&PERSON, _sect:&SECTIONS, _syl:&SYLLABI)
```

Figure 1. Plan for learning course material

At the outset of an information-seeking dialog, the system has little knowledge about the information-seeker's (IS) purpose in requesting information. In most cases, a complete plan for IS cannot be constructed during the first part of a dialog. Instead, potential goals must be inferred from individual utterances and integrated into the overall plan structure, thereby incrementally expanding and instantiating the plan as the dialog progresses.

Oftentimes there will be many domain goals that a single utterance might address. For example, if a student wants to know what time Dr. Smith arrives in the morning, he may want to either visit Dr. Smith or call him in his office. Furthermore, even if we can identify a single domain goal addressed by an utterance, there may be many ways in which that goal could be incorporated into an overall plan. For example, if a student asks whether Political Science 210 is offered in the spring, we might infer that the student wants to take Political Science 210. But how should this goal be built into the student's overall plan? Perhaps he is considering taking Political Science 210 in order to satisfy a breadth requirement or perhaps he wants to earn a minor. So the issue that must be addressed in dynamically inferring IS's underlying task-related plan from an ongoing dialog is the following: *how can we identify which of many candidate goals is the actual goal which IS is addressing with a particular utterance, and how can we determine where this particular goal fits into IS's overall plan?*

Two factors appear to provide a basis for a solution. The first is the cooperative nature of the dialogs we are studying. The dialog participants are expected to maintain a meaningful exchange, during which the relationship of individual utterances is explicitly or implicitly conveyed. The second factor is the organized nature of naturally occurring information-seeking dialogs, as exhibited in dialog transcripts. These two factors produce a structure in information-seeking dialogs. As a result, we can formulate focusing heuristics that specify how individual utterances should be related to the existing dialog context, as represented by the plan inferred for IS and his current focus of attention in that plan.

Thus our approach is the following:

1. Hypothesize from an individual utterance a set of domain-dependent candidate focused goals that may represent the information-seeker's focus of attention in the task.
2. Use focusing heuristics to select the candidate focused goal most appropriate to the existing dialog context and incorporate it into the model of the information-seeker's plan.

3.1. Hypothesizing Candidate Focused Goals And Plans

The first stage of processing analyzes an utterance without considering the preceding dialog. Plan-identification heuristics are used to hypothesize a set of domain-dependent goals and associated plans that might represent that aspect of the task on which IS's attention is currently focused. These heuristics are extensions of inference rules proposed by Allen [ALLE80]. For example, if IS wants to know the values of an argument that cause a proposition to be true, then that proposition or its

negation may be relevant to the plan that IS is considering. Therefore any goals whose plans might prompt such a request become *candidate focused goals* and their associated plans become *candidate focused plans*. Thus, if IS asks,

"Who is teaching Section 10 of French 112 in the fall?"

then IS wants to know the values of the argument _fac:&FACULTY such that the proposition

Teaches(_fac:&FACULTY, FRENCH112-10-FALL)

is true. The plan for learning the material of a section of a course (Figure 1) contains the proposition

Teaches(_fac:&FACULTY, _sect:&SECTIONS)

as a constraint on a subgoal in its body. Substituting FRENCH112-10-FALL for _sect:&SECTIONS produces the proposition

Teaches(_fac:&FACULTY, FRENCH112-10-FALL)

addressed by IS's utterance. Making this substitution throughout the plan in Figure 1 produces a plan for learning the material of Section 10 of French 112 in the fall. Therefore the goal

Learn-Material(IS, FRENCH112-10-FALL, _syl:&SYLLABI)

becomes a candidate focused goal, and the plan produced by substituting FRENCH-112-10-FALL for _sect:&SECTIONS in Figure 1 becomes a candidate focused plan. The goal

Learn-From-Person(IS, FRENCH112-10-FALL, _fac:&FACULTY)
where
Teaches(_fac:&FACULTY, FRENCH112-10-FALL)

is the most recently considered subgoal in this candidate focused plan; thus it provides the greatest expectations for future utterances.

3.2. Context Processing

The second stage relates an utterance to the context established by the preceding dialog. We use a tree structure called a *context model* to represent the task-related plan inferred for IS from the preceding dialog. Each node in this tree represents a goal that the system believes IS is considering achieving and, except for the root, is a descendant of a higher-level goal whose associated plan contains the subgoal represented by the child node. One node in the tree is marked as the current focus of attention and indicates that aspect of the task on which IS's attention is currently centered. The path from the root of the context model to the current focus of attention is called the *active path* and represents the global context, or sequence of progressively lower-level goals that led to the subgoal under consideration by IS.

Initially, there is no existing context; each candidate focused goal and its plan become the root of a context model and are marked as current focused goals and

plans. If there is only one context model and its root goal appears as part of only one domain-dependent plan, then we have further knowledge about what IS wants to do and can add this higher-level plan as the new root of the context model, with the old root as its child. We continue expanding the tree upward until more than one higher-level plan is possible. Thus only that part of the user's task-related plan that the system believes IS intended it to recognize is built into the user model. Section 5 discusses more robust user modeling, in which default inference rules are used to expand the system's model of the user's plan, and addresses the problem of detecting and recovering from errors that might be introduced into the model.

As each new utterance occurs, it must be related to the established context. A set of focusing heuristics are used to determine the most likely relationship between one of the hypothesized candidate focused plans and the context model, and to expand the context model to include it. Grosz [GROS77a] introduced the concept of focusing in her work on identifying the referents of definite noun phrases in apprentice-expert dialogs. She noted that the focus of the discourse followed the plan for performing the apprentice's task. Our information-seeking dialogs differ from apprentice-expert dialogs in that the dialog is not constrained by the order of execution of the actions in the overall plan. However, we do find structure in the dialogs we are studying, and it is the basis for our focusing heuristics. This structure appears to be caused by two factors.

The first is the organized nature of naturally occurring information-seeking dialogs. Dialog transcripts indicate that humans generally ask all their questions that are relevant to a plan for one subgoal before moving on to ask questions about a plan for another subgoal of the overall task. One possible explanation for this behavior is that it may require less mental effort than switching back and forth among partially constructed plans for different subgoals.

The second factor producing structure in our dialogs is their cooperative nature. Since the dialogs are cooperative and both participants must be focused on the same subset of knowledge in order to understand one another [GROS77a], we expect IS to shift topic slowly between consecutive utterances, and to adhere to the focusing constraints espoused by McKeown [MCKE85a]. McKeown expanded on focus rules proposed by Sidner [SIDN81a] to explain how speakers should organize their utterances when faced with a choice of topic. In particular, McKeown claims that a speaker should move to a recently introduced topic if he has something further to say about it; otherwise he will have to reintroduce the topic at a later time. Similarly, the speaker should choose to finish discussion of the current topic before switching back to a previous one.

Our focusing heuristics rely on these expectations about possible shifts in focus of attention in IS's underlying task-related plan to identify which candidate focused plan is most relevant to the established dialog context and to determine how it fits into the context model. The following ordered list gives the focusing heuristics' preferences on the relationship between a candidate focused plan and the context model. Each relationship is illustrated under the assumption that the tree shown in Figure 2a is the context model immediately preceding the utterance, with node C (marked by an asterisk) representing the current focused goal/plan and node G representing the most recently considered subgoal in the current focused plan.

1. The candidate focused plan is part of the expansion of a plan for the most recently considered subgoal in the current focused plan; for example, if Figure 2b is an expansion of the context model shown in Figure 2a, then node C1 might represent such a candidate focused plan.
2. The candidate focused plan is part of an expansion of the current focused plan; for example, if Figure 2b is an expansion of the context model shown in Figure 2a, then node C2 might represent such a candidate focused plan (where C2 is part of a plan for the goal at node F, which is in turn part of a plan for the goal at node C).
3. The candidate focused plan is part of the expansion of a plan for a goal along the active path, with preference given to goals that are closest to the current focused goal on the active path. For example, if Figure 2b is an expansion of the context model shown in Figure 2a, then nodes C3 and C4 would both be part of the expansion of a plan for a goal along the active path; but if nodes C3 and C4 both represent candidate focused plans, then node C3 would be preferred, since it appears in an expansion of the plan associated with node B, which is closer to the current focused goal (represented by node C) than is node A.
4. The candidate focused plan is a plan whose expansion contains the goal associated with the root of the context model; Figure 2c illustrates such a relationship, where node C5 represents a candidate focused plan.
5. The candidate focused plan is part of the expansion of a higher-level plan, and this expansion also contains the goal associated with the root of the context model; Figure 2d illustrates such a relationship, where node C6 represents a candidate focused plan.

In applying each rule, we use a breadth-first expansion of plans, so that the resulting shift in focus of attention will be as small as possible. For example, if nodes C7 and C8 in Figure 2e both represented candidate focused goals/plans, the second rule in the above list would prefer C7 to C8, since C7 is closer to the existing focus of attention in the dialog.

3.3. An Example

To illustrate this plan inference process, let us consider a dialog segment containing four utterances by IS. Suppose IS begins with the statement:

IS: "I want to major in computer science."

Since IS states that he wants to achieve a goal, majoring in CS, TRACK's plan identification heuristics hypothesize

Satisfy-Major(IS, BA, CS)
and
Satisfy-Major(IS, BS, CS)

as candidate focused goals and their associated plans as candidate focused plans. Since there is no way of choosing between these, two context models would be built,

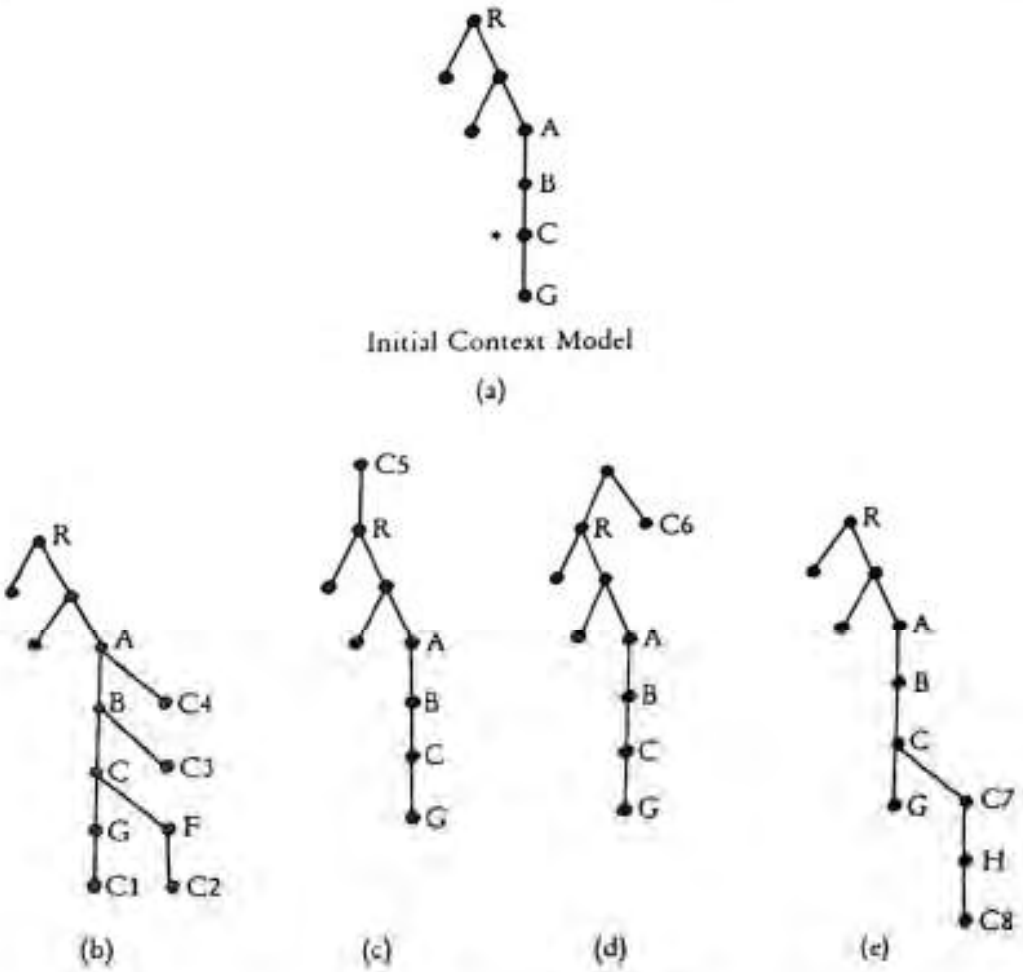

Figure 2. Examples of relationships between candidate plans and context model

each with one of the candidate focused goal/plan pairs as its root (Figure 3). The resulting current focused plan in each context model is preceded by an asterisk.

* Satisfy-Major(IS, BA, CS)	* Satisfy-Major(IS, BS, CS)

Figure 3. Two context models

Suppose that IS's next utterance is the query

"What are the prerequisites for taking CS180?"

Since IS wants to know the preconditions for the plan associated with the goal of taking CS180 (the introductory course for majors and minors in computer science), TRACK hypothesizes

Earn-Credit(IS, CS180, _ss:&SEMESTERS, _cr1:&CREDITS)

and its associated plan as the candidate focused goal/plan pair. The focusing heuristics must determine how this relates to the preceding dialog, as represented by the context model. The strongest expectation is that IS is continuing with some aspect of the current focused plan; since the preceding utterance did not address any particular goal in this plan, there is no *most recently considered* subgoal. Since taking CS180 appears in an expansion of the plans for majoring in computer science, TRACK expands the context models as shown in Figure 4 and marks the plan for earning credit in CS180 as the new current focus of attention.

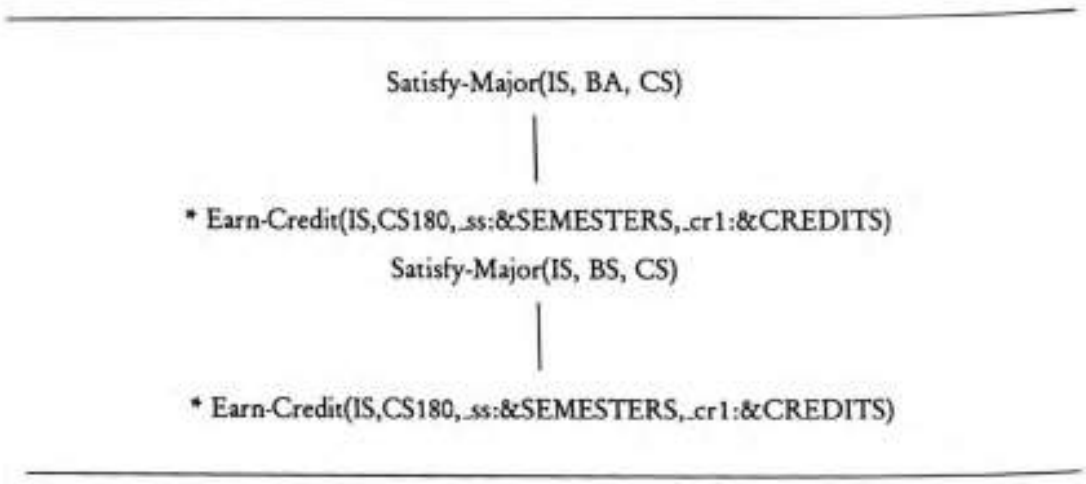

Figure 4. Context models after two utterances

Suppose that IS's next query is

IS: "What courses must I take in order to satisfy the foreign language requirement?"

Since IS is asking about the argument (courses) of a subgoal (taking courses) that is part of a plan for achieving a second goal, TRACK hypothesizes the second goal and its associated plan

Satisfy-Language-Req(IS)

as the candidate focused goal/plan pair. The focusing heuristics must now determine how this relates to the preceding dialog, as represented in the context model. The strongest expectation is that IS will continue with some aspect of the current focused plan. However, the candidate focused plan does not appear in an expansion

of the current focused plan, indicating that IS has shifted focus to another aspect of the overall task. In fact, none of the first four focusing heuristics find a relationship between the candidate focused plan and the context model. However, the last focusing heuristic finds that there is a goal,

Obtain-Degree(IS, BA)

whose associated plan can be expanded to include both the candidate focused plan and the context model whose root is Satisfy-Major(IS, BA, CS), indicating that IS has shifted his attention to another subtask (satisfying the foreign language requirement) of a higher-level plan (obtaining a Bachelor of Arts degree) of which the old current focused plan (obtaining a computer science major) is also a part. Therefore, this goal becomes the root of a new context model, as shown in Figure 5; Satisfy-Language-Req(IS) is marked as the new current focus of attention, as indicated by the asterisk preceding it. The other previous context model, whose root was Satisfy-Major(IS, BS, CS), is discarded, indicating that the second query has led us to deduce that IS wants to pursue a Bachelor of Arts degree. Note that our plan inference process makes what Pollack [POLL86a] terms the *appropriate query assumption* – namely, that IS does not ask queries that are inappropriate to his intended goal. This aspect of our plan inference process will be discussed further in Section 5, where we pursue a more robust plan recognition paradigm.

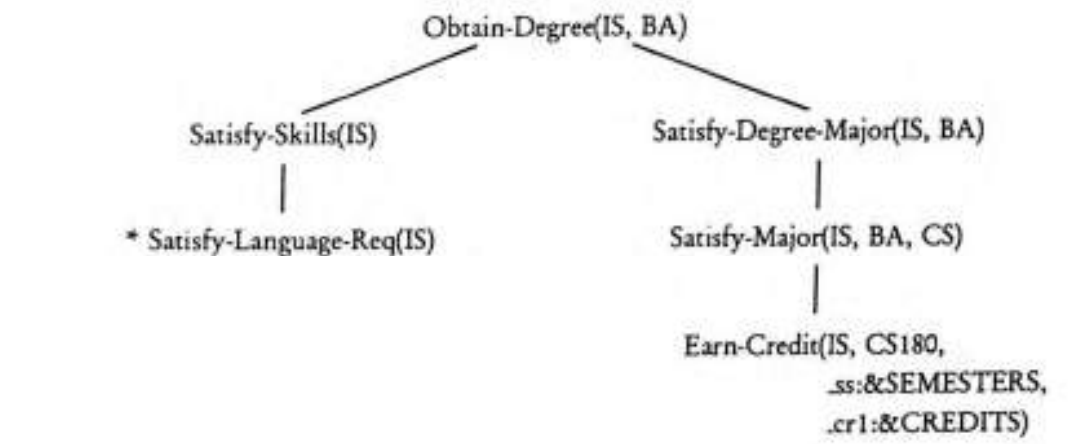

Figure 5. Context model after three utterances

Suppose that IS's next query is

IS: "Who is teaching Section 10 of French 112 in the fall?"

As described earlier, since IS is asking about the teacher of a particular section of a course, he may be considering the subgoal of learning from that teacher; this sub-

goal appears in a plan for learning the material of a course, and therefore TRACK hypothesizes the plan associated with the goal

Learn-Material(IS, FRENCH112-10-FALL, _syl:&SYLLABI)

as one of the candidate focused plans. The focusing heuristics find that this candidate focused plan appears in an expansion of the most recently considered subgoal (taking courses) in the current focused plan, and therefore it is selected as the new focus of attention and the context model is expanded to include it (Figure 6).

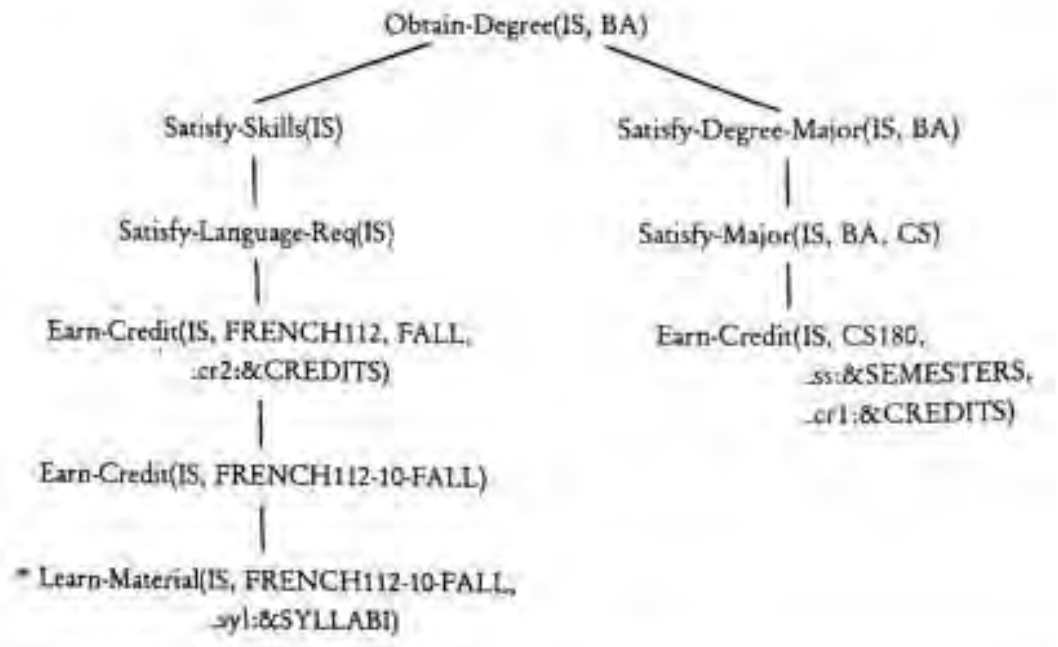

Figure 6. Context model after four utterances

In this manner, our plan inference process dynamically infers from an ongoing dialog the underlying task-related plan motivating an information-seeker's queries and tracks his focus of attention in this plan structure.

4. Application of User Models

The context model is one component of a comprehensive user model, representing the system's acquired beliefs about the plan an information-seeker is trying to construct. The possible expansions of this plan provide expectations about information that IS might want, and these expectations can often be used to repair and

disambiguate IS's subsequent utterances. We have developed strategies that use our context model to handle two forms of problematic input: pragmatically ill-formed utterances and intersentential ellipsis. This section describes our approach to the first of these; our framework for handling ellipsis is described in [CARB85].

4.1. Pragmatic Ill-Formedness

Every listener, whether human or machine, has a model of the world that is used to interpret utterances. An utterance can be syntactically and semantically well-formed, but violate the structural properties of the listener's world model. We shall say that such an utterance is *pragmatically ill-formed*.

Consider, for example, the query

IS: "What is the area of the Special Cargo Hold on the Europa?"

whose semantic representation will contain the proposition

Area(EUROPA-SPECIAL-CARGO-HOLD, .areaval:&SQ-FT)

A query similar to this appears in a dialog transcript of an information-seeker attempting to load cargo onto ships using the REL natural language interface [THOM80]. The system was unable to understand the query since the query erroneously presumed that these storage locations had an *area* attribute in the associated database. If a human information-provider had a similar problem in understanding the utterance, or considered the meaning of *area* ambiguous, he might be able to use the context established by the preceding dialog to identify what the information-seeker really wanted to know. For example, if IS's goal is to load cargo of the appropriate type into the various cargo holds, then he probably wants to know the *remaining capacity* of the Special Cargo Hold. On the other hand, if his goal is to assign ships to routes in order to handle the expected cargo shipping requirements, then IS probably wants to know the *total capacity* of the Special Cargo Hold; similarly, if his goal is to assign workers to fill the storage holds, with one worker assigned to handle all cargo holds located in the same section of the ship, then IS probably wants to know the *location* of the Special Cargo Hold.

Another example of a pragmatically ill-formed query illustrates the *missing joins* problem.

IS: "Who is teaching Section 10 of French 112 in the fall?"

IP: "Dr. Walker."

IS: "When's Mitchel meet?"

A semantic representation of this query contains the proposition

Meeting-Time(MITCHEL, .time:&MEETING-TIMES)

Suppose that in the system's world model, faculty teach sections of courses, chair committees, and present colloquia, and each of these may have a scheduled meeting time, but there is no direct relationship between faculty and times. Then the above

query will appear pragmatically ill-formed. Although this utterance might be an abbreviated version of any of the queries

"When does the section of French 112 taught by Dr. Mitchel meet?"
"When does the committee chaired by Dr. Mitchel meet?"
"When does the colloquium given by Dr. Mitchel meet?"

a human information-provider would be likely to recognize from the above dialog that IS wants the meeting time of the section of French 112 taught by Dr. Mitchel, and respond accordingly.

4.2. Understanding Pragmatically Ill-Formed Queries

Our analysis of naturally occurring dialog suggests that human listeners often reason on their model of the speaker to remedy many of his faulty utterances and allow the dialog to continue without interruption. We contend that a natural language system should operate in a similar manner since, if its communication is to be regarded as natural, the system must be able to handle the full spectrum of utterances that humans understand with relative ease. A number of researchers have investigated the problem of handling pragmatically ill-formed queries [SOWA76, CHAN78, MAYS80b, KAPS79], but their strategies were deficient in that they considered the queries in isolation, without using a model of the preceding dialog to address the speaker's intentions.

Grice's theory of meaning [GRIC69, GRIC57] and maxim of relation [GRIC75] suggest that the listener's beliefs about what the speaker is trying to do should be used to recognize the intent behind an ill-formed query. According to Grice's theory, a listener should believe that the speaker thinks the listener can infer the intended meaning of an utterance – otherwise the speaker would not have made it. So given a pragmatically ill-formed query, a cooperative listener should attempt to deduce these intentions. Grice's maxim of relation suggests that the speaker's utterance is relevant to the existing dialog context, so the listener should use this context to attempt to deduce the speaker's intended meaning and enable the dialog to continue without interruption.

Our strategy is based on this theory of meaning and intention. It uses the context model to suggest substitutions for the erroneous proposition appearing in the semantic representation of IS's pragmatically ill-formed query, thereby producing semantic representations for one or more revised queries, all of which are relevant to what IS is trying to accomplish. If more than one revised query is proposed, then it must be determined whether any of these is significantly more likely than the others to represent the speaker's intentions or satisfy his perceived needs. Two criteria appear appropriate for comparing suggested revised queries. The first is the relevance of the revised query to the current focus of attention in the dialog. Since we have contended that some shifts in focus of attention in the plan structure are more likely than others, it is reasonable to hypothesize that the more expected the shift in focus of attention that would result from a revised query, the more likely is that query to represent the speaker's intentions. The second criteria for comparing

suggested revised queries is the similarity of a revised query to the speaker's actual utterance. The more the speaker's utterance is altered, the less likely it is to represent what he meant to say.

Therefore our pragmatic ill-formedness processor contains a suggestion mechanism and a selection mechanism. The suggestion mechanism proposes revised queries, all of which are relevant to IS's underlying task-related plan, and the selection mechanism uses the criteria of relevance and semantic similarity to select, from among multiple suggestions, the revised query deemed most likely to represent the speaker's intentions or satisfy his perceived needs.

4.3. Repair Strategy

Our suggestion mechanism uses two sets of substitution heuristics, one for making simple substitutions of a property, relation, function, or object class for that used by the speaker, and a second set for expanding relational paths to handle the missing joins problem. As an example of a simple substitution, suppose the dialog preceding the query

"What is the area of the Special Cargo Hold on the Europa?"

indicates that IS's current focused goal within his overall plan is to load cargo of the appropriate type into the various cargo holds. A subgoal in the plan associated with this goal would be

Load-Type-Cargo(IS, _item:&CARGO, _storearea:&STORAGE-AREA)
where
Is-Type(_item:&CARGO, _cartype:&CARGO-TYPE)
Cargo-Type(_storearea:&STORAGE-AREA, _cartype:&CARGO-TYPE)

and a plan for this subgoal would contain the precondition

Greater(_remcap:&CUBIC-FT, _itemsize:&CUBIC-FT)
where
Volume(_item:&CARGO, _itemsize:&CUBIC-FT)
Remaining-Capacity(_storearea:&STORAGE-AREA, _remcap:&CUBIC-FT)

specifying that the storage area must have room for the cargo item. Our property substitution heuristic would examine this plan and suggest substituting either of the propositions

Cargo-Type(EUROPA-SPECIAL-CARGO-HOLD, _cartype:&CARGO-TYPE)
and
Remaining-Capacity(EUROPA-SPECIAL-CARGO-HOLD,
_remcap:&CUBIC-FT)

for the erroneous proposition

Area(EUROPA-SPECIAL-CARGO-HOLD, _areaval:&SQ-FT)

appearing in the semantic representation of IS's query, producing suggested revised semantic representations equivalent to the two queries

IS: "What is the cargo type of the Special Cargo Hold on the Europa?"

IS: "What is the remaining capacity of the Special Cargo Hold on the Europa?"

More formally, this heuristic is represented by the following rule:

If IS's proposition erroneously presumes that a member Obj of CLASS1 has a property Att1, then replace property Att1 with property Att2 if the following conditions hold:

1. a proposition specifying property Att2 on a member Obj2 of CLASS1 appears in an expansion of IS's context model.
2. Obj1 and Obj2 unify (Either Obj1 in the semantic representation of IS's utterance or Obj2 in the plan proposition refers to a general member of CLASS1, or both refer to the same specific member of CLASS1).

In the context of our student advisory dialog, suppose a student wants to pursue an independent study project; such projects can be directed by full-time faculty but not by faculty who are *extension* or *on sabbatical*. The student might erroneously follow the utterance

"I want to take an independent study project."

with the pragmatically ill-formed query

"What is the classification of Dr. Smith?"

In a university world model, only students have a classification attribute; this attribute can have values such as Arts&Science-1989, Engineering-1988, and Business-1990. Faculty have attributes such as rank, status, age, and salary. Pursuing an independent study project under the direction of Dr. Smith has the precondition that Dr. Smith's status be *full-time* or *part-time*. Our substitution mechanism would analyze the plan for taking an independent study course, and the property substitution rule would suggest substituting the proposition

Status(DR.SMITH, _statusval:&STATUSVALUES)

for the erroneous proposition

Classification(DR.SMITH, _classval:&CLASSVALUES)

appearing in the semantic representation of the student's query, resulting in a suggested revised semantic representation equivalent to the query

"What is the status of Dr. Smith?"

As an example of the second set of heuristics, the path expansion heuristics, consider again the query

"When's Mitchel meet?"

following the dialog that produced the context model shown in Figure 6. As mentioned earlier, the semantic representation of this query contains the erroneous proposition

Meeting-Time(MITCHEL, _tme:&MEETING-TIMES)

indicating a direct relationship between faculty and times. Our path expansion heuristics will analyze and expand the context model shown in Figure 6 and note that a plan for the goal

Earn-Credit(IS, FRENCH112, FALL, _cr2:&CREDITS)

can include a path containing the sequence of goals

Earn-Credit-Section(IS, _sec1:&SECTIONS)
where
Is-Section-Of(_sec1:&SECTIONS, FRENCH112)
Section-Offered(_sec1:&SECTIONS, FALL)

Learn-Material(IS, _sec1:&SECTIONS, _syl1:&SYLLABI)
where
Is-Syllabus-Of(_sec1:&SECTIONS, _syl1:&SYLLABI)

Learn-From-Person(IS,_sec1:&SECTIONS,_fac:&FACULTY)
where
Teaches(_fac:&FACULTY, _sec1:&SECTIONS)

Attend-Class(IS,_plc:&MEETINGPLCS,_day:&MEETINGDAYS,
_tme:&MEETINGTIMES)
where
Is-Meeting-Place(_sec1:&SECTIONS, _plc:&MEETINGPLCS)
Is-Meeting-Day(_sec1:&SECTIONS, _day:&MEETINGDAYS)
Is-Meeting-Time(_sec1:&SECTIONS, _tme:&MEETINGTIMES)

One path expansion heuristic notes that the propositions

Teaches(_fac:&FACULTY, _sec1:&SECTIONS)
Is-Meeting-Time(_sec1:&SECTIONS, _tme:&MEETINGTIMES)

both appear on this path in the expanded plan, and suggests substituting the conjunction of the propositions

Teaches(MITCHEL, _sec1:&SECTIONS)
Is-Meeting-Time(_sec1:&SECTIONS, _tme:&MEETINGTIMES)

for the erroneous proposition appearing in the semantic representation of IS's query, resulting in a suggested revised semantic representation equivalent to the English query

"When do sections taught by Mitchel meet?"

Although we have only illustrated substituting a conjunction of two propositions for the erroneous proposition in the user's query, the path expansion heuristics can propose expansions of any length.

Other heuristics and other parts of a plan will suggest other substitutions in addition to the ones shown in our examples. The important point is that the revised queries resulting from these suggestions are all relevant to the plan that IS is constructing. As mentioned earlier, relevance to the current focus of attention and similarity to the speaker's actual utterance are used to select from among multiple suggestions. We use focusing heuristics, similar to those used for constructing the context model, to measure relevance of a revised query to the current focus of attention in the dialog, and generalization hierarchies for properties, relations, functions, and object classes to measure the semantic similarity of a substituted term and the term that it replaces. In the example,

"What is the area of the Special Cargo Hold on the Europa?"

both suggested revised queries have approximately the same relevance to the current dialog but, of the two properties *cargo type* and *remaining capacity*, *remaining capacity* is much closer semantically to the property *area* used by the speaker. Therefore our selection mechanism chooses

"What is the remaining capacity of the Special Cargo Hold on the Europa?"

as the most appropriate query representing IS's needs.

This approach is superior to previous strategies because it uses a model of the speaker to identify and address his perceived intentions and needs in making an utterance. As such, it not only reasons on the user model to suggest possible interpretations relevant to the user's goals and plans, but it also limits consideration to those interpretations that are reasonable given the established dialog context.

5. Improving Plan Recognition

Our research has shown how an information-seeker's underlying task-related plan can be dynamically inferred from an ongoing dialog, and how the resulting context model can be used to achieve better communication. However, research on plan recognition has made four critical assumptions which enable the system to avoid errors in forming its beliefs about the user:

1. IS has no misconceptions about the task domain.[1]

[1] Allen's and Pollack's models are exceptions, as described later.

2. IS's queries always address aspects of the task within the system's limited knowledge. These systems maintain the *closed world assumption* [REIT78].
3. IS's statements and queries are correct and not misleading.
4. The system's inference mechanisms do not introduce errors into the context model.

These assumptions represent unrealistic constraints on real-world dialogs and must be removed. The first assumption, called the *valid plan assumption* by Pollack [POLL87], limits the kinds of beliefs IS can already have about the domain – namely, it says that IS's knowledge may be incomplete but not erroneous. But IS is interacting with the system because IS does not know enough about the domain to construct his task-related plan by himself. Therefore, since IS is not an expert in the area, it is to be expected that some of his beliefs about the domain may be false, contradicting the first assumption. An implication of the valid plan assumption is what Pollack terms the *appropriate query assumption* – namely, that IS knows enough about how to solve his problem that he always asks relevant questions.

The second assumption limits the questions IS can ask to those which the system can answer. But even an expert system has limited domain knowledge. Furthermore, in a rapidly changing world, knowledgeable users may have more accurate information about some aspects of the domain than does the system. For example, a student advisory system may not be altered immediately upon changing the teacher of a course. A cooperative system should recognize its limited knowledge and reason with it to provide whatever pertinent, helpful information it can.

The third assumption restricts IS to utterances that are clear, precise, and accurate. For example, it eliminates the possibility that IS might say he is a junior, when in fact he is 3 credits short of junior standing, thereby leading the system to erroneously infer that IS is eligible for certain programs or awards. But human information-seekers are often imprecise, especially when they are not aware that small perturbations in the data can be significant.

The fourth assumption says that the system never makes an error in inferring IS's plan. But even in the simplest cases, the system must hypothesize how individual utterances relate to one another. Such decisions select from among multiple possibilities and are a potential source of error.

Pollack [POLL87] argues against plan inference systems making the first two assumptions, because they prevent the system from inferring plans which the user believes he can pursue but which are novel (to the system) or invalid. However, there is another implication of relaxing the appropriate query assumption that is not considered by Pollack: IS may ask an irrelevant question that seems perfectly reasonable to the system, thereby leading the system to develop incorrect beliefs about IS's objectives. Consider, for example, a student advisory system. If only BA degrees have a foreign language requirement, the query

"What courses must I take to satisfy the foreign language requirement in French?"

may lead the system to infer that IS is pursuing a Bachelor of Arts degree. If only BS degrees require a senior project, then a subsequent query such as

"How many credits of senior project are required?"

is problematic. Either the second query is inappropriate to IS's goal of obtaining a Bachelor of Arts degree [POLL87], or the system's context model does not accurately reflect what IS wants to do. Note that in both cases the user has a misconception; but in the latter case, the misconception went undetected and was allowed to introduce errors into the system's context model.

Traditional natural language plan inference systems also make the third and fourth assumptions, which together with the first two, guarantee that the underlying plan inferred by the system and the task-related plan under construction by IS are never at variance with one another. If we want systems capable of understanding and appropriately responding to naturally occurring dialog, natural language interfaces must be able to deal with situations where those assumptions are not true.

Grosz [GROS81] claimed that miscommunication can occur if both dialog participants are not focused on the same subset of knowledge. Joshi [JOSH82] contended that successful communication requires that the mutual beliefs of the dialog participants be consistent. Extending this to inferred plans, we claim that a successful cooperative dialog requires that the system's beliefs about IS's plan be consistent with what IS is actually considering doing. But clearly it is unrealistic to expect that the system's model will always be correct, given the different knowledge bases of the two participants and the imperfections of communication via dialog.

Thus we need a repair mechanism that attempts to detect inconsistencies in the models and repair them whenever possible. This view is supported by the work of Pollack, Hirschberg and Webber [POLL82b]. They suggested that expert-novice dialogs could be viewed as a negotiation process, during which not only an acceptable solution is negotiated but also understanding of the terminology and the beliefs of the participants. The context model is one component of the system's beliefs, as is its belief that this model accurately reflects the plan under construction by IS.

In the remainder of this chapter, we outline a four-phase approach to the problem of handling plan disparity. This approach was motivated by an analysis of naturally occurring dialog and is an extension of the hypothesize and revise paradigm proposed by Schmidt, Sridharan, and Goodson [SCHM78] for inferring a user's goal by observing his non-communicative actions. This approach requires a richer context model than maintained by current systems, and we argue that the support that the system accords a component as a correct part of the model should affect its hypothesis about the source of error once plan disparity is suggested.

5.1. Related Work

Several research efforts have addressed problems related to plan disparity. Kaplan [KAPS79], McCoy [MCCO*], and Quilici [QUIL*] investigated misconceptions about domain knowledge and proposed responses intended to remove the misconceptions. However, such misconceptions may not be exhibited when they first influence the information-seeker's plan construction; in such cases, disparate plans may result, and correction will entail both a response correcting the misconception and further processing to bring the system's context model and the plan under construction by the information-seeker back into alignment.

Allen's [ALLE80] plan inference system did not eliminate invalid plans, but weighted them less favorably than valid plans. However, his model did not consider how potential user misconceptions might affect the partial plan inferred by the system.

Pollack [POLL86a] studied removal of the *appropriate query assumption* of previous planning systems. She proposed a richer model of planning that regarded plans as mental phenomena and explicitly reasoned about the information-seeker's possible beliefs and intentions. She addressed the problem of queries that indicated the information-seeker's plan was inappropriate to his overall goal, and attempted to isolate the erroneous beliefs that led to the inappropriate query. However, queries deemed inappropriate by the system may signal phenomena other than that the query is inappropriate to what the user really wants to do. For example, the information-seeker may have shifted focus to another aspect of the overall task without successfully conveying this to the system, the system's context model may have been in error prior to the query, or, as noted by Pollack [POLL87], the information-seeker may be addressing aspects of the task outside the system's limited knowledge.

Pollack was concerned with issues that arise when the information-seeker's plan is incorrect due to a misconception exhibited by the current query. She assumed that, immediately prior to the user making the problematic query, the system's partial model of the user's plan was correct. We argue that since the system's inference mechanisms are not infallible and communication itself is imperfect, the system must contend with the possibility that its inferred model does not accurately reflect the user's plan. Previous research has failed to address this problem.

Schmidt, Sridharan, and Goodson [SCHM78] proposed a hypothesize-and-revise paradigm for inferring a user's goal by observing his non-communicative actions. They formulated a set of revision critics for altering a plan upon observing actions that conflict with expectations, but failed to provide any principles or mechanism for selecting the appropriate revision. Although the model of plan recognition for an office environment formulated by Carver, Lesser, and McCue [CARV84] attempted to repair the inferred plan when actions inconsistent with it were observed, it did not reason about where the plan might be wrong but merely backtracked to select another interpretation.

5.2. An Approach to Robust Plan Recognition

Our goal is to have the system's context model reflect the plan under construction by the information-seeker. But errors may enter the system's context model if the user's statements are inaccurate or misleading, if the system uses default inference rules, or if the system uses focusing heuristics to relate new utterances to the existing dialog context. In addition, models that are less specific than intended by the information-seeker may result if the user fails to adequately communicate his intentions (or the system fails to recognize these intentions). Furthermore, the system's limited domain knowledge may hamper its recognition and modeling of a valid plan being constructed by the user. As mentioned earlier, Pollack [POLL86a] has addressed the problem of recognizing and reasoning about invalid plans; our

concern here is with modeling the user's plan when his planning is correct, but for some reason the system's model fails to reflect that plan.

Rarely will an information-seeker be aware that the system has constructed an incorrect model of what he wants to accomplish, and even less often will he know exactly where the model is wrong. Instead, the information-seeker may express surprise with the system's response or in some way convey that his needs are not being satisfied. The system must then determine the cause of this dissatisfaction.

Our research indicates that detecting and recovering from disparate plan structures requires a four-phase approach:

1. Detecting clues to disparity between the system's model of the user and the user's actual goals and plans.
2. Reasoning on the system's model and the system's domain knowledge to hypothesize the source of these disparities.
3. Responding to the user and negotiating with him to isolate the errors.
4. Appropriately repairing the user model.

This approach is motivated by the way humans appear to respond in similar situations, as determined through an analysis of dialog transcripts. Therefore a natural language system that pursues this strategy will be viewed as acting naturally by human users.

The next sections discuss each of these phases. The dialogs shown in Figures 7 and 8, which we will refer to as the *IRA example* and the *Kermit example* respectively, will be used to motivate and illustrate the strategy.

---

(1) **IS:** "I'm a retired government employee but I'm still working. I'd like to start out an IRA for myself and my wife — she doesn't work."

(2) **IP:** "Did you work outside of the government last year?"

(3) **IS:** "Yes I did."

(4) **IP:** "There's no reason why you shouldn't have an IRA for last year."

(5) **IS:** "I thought they just started this year."

(6) **IP:** "Oh no. IRA's were available as long as you are not a participant in an existing pension."

(7) **IS:** "Well, I do work for a company that has a pension."

(8) **IP:** "Ahh. Then you're not eligible for 81."

Figure 7. Individual retirement account (IRA) dialog

(1) **IS:** "I wish I could transfer files between the Vax and my PC."

(2) **IP:** "Kermit lets you do that."

(3) **IS:** "How do I get Kermit?"

(4) **IP:** "The computing center will give you a copy if you bring them a floppy disk."

(5) **IS:** "How late is the University Bookstore open?"

Figure 8. File Transfer Via Kermit Dialog

The first dialog[4] was transcribed from a radio talk show on investments[5], and the second is similar to a dialog that took place between two people in a university. Utterances are numbered for later reference.

5.2.1. Detection

As claimed earlier, since the system is presumed to be a cooperative dialog participant, it must be on the lookout for plan disparity. We have identified four sources of clues to the existence of such disparity:

1. The discourse goals of IS, such as expressing surprise or confusion.
2. Conflicts between inferences drawn from IS's current utterance and the existir context model.
3. Relevance of IS's current utterance to the context model.
4. Abrupt shift in focus of attention in the context model.

IS can express surprise or confusion about the system's response, thereby cuing the possibility of plan disparity. For example, in the IRA dialog, plan disparity is suggested when IS, in utterance (5), expresses confusion at IP's previous response. On the other hand, IS's query may contradict or appear irrelevant to what the system believes is IS's overall task, leading the system to suspect that its context model may not reflect IS's plan. Or IS's query may require so sharp an unsignaled shift in focus as to suggest possible problems in the system's model of what IS wants to accomplish; the strongest expectations are for speakers to address aspects of the task closely related to the current focus of attention [SIDN81a, MCKE85a, CARB83].

The Kermit dialog illustrates a case in which plan disparity is suggested by an abrupt shift in focus of attention. Upon completion of utterance (4), the system's model of IS's plan might be represented as

[4] Minor alterations have been made to the dialog to remove restarts and extraneous phrasing.

[5] These transcripts were provided by the Department of Computer and Information Science of the University of Pennsylvania.

Transfer-Files(IS, KERMIT, VAX, PC)
|
Have(IS, KERMIT)
|
Get-Copy-On(IS, _x:&DISK, KERMIT)
|
Have(IS, _x:&DISK)

Suppose that the system does not know how to obtain floppy disks. Then from its limited knowledge, IS's next query,

"How late is the University Bookstore open?"

will not appear to address an aspect of the plan inferred for IS, or any expansion of it. The system could just respond by

1. answering the direct question, if possible, ignoring its ramifications.
2. responding *I don't know*, if the direct answer is not available.

However, cooperative information-providers are expected to try to understand the import of a query and provide as helpful a response as they can.

Grice's maxim of relation [GRIC75] suggests that IS believes the query is relevant to the overall dialog. Several possibilities exist. IS may be shifting focus to some aspect of a higher-level task that includes transferring files as a subaction. One such higher-level task might be to compose a document using the Scribe text formatting system, and the aspect queried by the new utterance might be the purchase of a Scribe manual from the university bookstore. In this case, the subtask of the overall task represented by the existing context model might be the transfer of files containing the document so that they can be modified using a PC editor.

On the other hand, focusing heuristics and the absence of discourse markers [SIDN83b] suggests that the new query is most likely to be relevant to the current focus of attention. So the system should begin trying to determine how IS's utterance might relate to the currently focused subtask in the context model, and consider the possibility that the context model might contain errors or that IS's domain knowledge might exceed the system's or be erroneous.

5.2.2. Hypothesis Formation

We have found that the knowledge acquired from the dialog and how it was used to construct the context model are important factors in hypothesizing the cause of disparity between the system's context model and the actual plan under construction by the information-seeker. Natural language systems must employ various techniques such as focusing heuristics and default rules for understanding and relating dialog in order to do the kind of inferencing exhibited in dialog transcripts and provide the most helpful responses. But confidence in individual components of the resultant context model appears to be important in hypothesizing errors. We contend that the system's context model should be enriched, so that its representation of the plan

inferred for the user differentiates among its components according to the support that the system accords each component as a correct and intended part of that plan. The system can then reason on this enriched context model to hypothesize the most likely sources of suspected disparities.

We currently see four kinds of support for plan components:

1. Whether the system believes that IS intended to communicate the component.
2. Whether the system has inferred the component on the basis of its own domain knowledge, which the system cannot be certain IS is aware of.
3. The kinds of rules used to add each component to the model (for example, default rules that select one component from among several possibilities, or heuristics that suggest a shift in focus of attention), and the evidence for applying the rules.
4. Whether the system's domain knowledge supports, contradicts, or is neutral regarding inclusion of the component as part of a correct overall plan.

The first three are important factors in formulating a hypothesis regarding the source of disparity between the system's model and IS's plan. If the system believes that IS intends the system to recognize from his utterance that G is a component of his plan, then the system can add G to its context model and have the greatest faith that it really is a component of IS's plan. Therefore such components are unlikely sources of disparity between the system's context model and IS's plan.

Components that the system adds to the context model on the basis of its domain knowledge will be strongly believed by the system to be part of IS's plan, but not as much as if the system believes IS intended to communicate them. These components resemble *keyhole recognition* rather than *intended recognition* [COHE81]. Keyhole recognition is the inference of an agent's goals and plans by unobtrusively observing him, as if through a keyhole; in such cases, the agent has no intent to communicate his goals to the observer. Intended recognition is the inference of those goals and plans that an agent *intends* to convey. Intended recognition is essential in communicative situations [COHE81], since the listener must identify the intended meaning of a speaker's utterance; but it appears that in user modeling and response generation, keyhole recognition should also play a significant role. Our analysis of naturally occurring dialog suggests that humans often use keyhole recognition in order to expand their beliefs about what the information-seeker is trying to do and how he should go about doing it, beyond what the information-seeker explicitly intended to convey. However, since the information-seeker may not have intended to communicate these goals and plans, they are more likely sources of error than components which IS intended that the information-provider recognize.

Consider, for example, a student advisement system. If CS180 is an introductory course restricted to majors in computer science and electrical engineering, then the system might infer from the utterance

"Can you tell me what time CS180 meets?"

not only that IS wants to know the meeting time for CS180, but also that IS is a computer science or electrical engineering major. However, IS may *intend* the system to recognize the first goal, but it is questionable whether he actually *intends* the system to recognize that he is pursuing a major in computer science or electrical

engineering. This latter inference is based on the system's beliefs about who can take CS180 – knowledge that IS may not have. Therefore, since IS may not have intended to communicate these components, they are more likely sources of error than components which he intended the system to recognize.

The particular rules used to add a component to the context model should affect the system's faith in that component as part of IS's overall plan. Consider again the IRA example in Figure 7. Utterance (4) can be explained by postulating that IP has applied the default inferences that since IS is retired but still working, he is only working part-time, is therefore not covered by a pension program, and is therefore eligible for an IRA during the year in question (at that time, rules on IRA's were different). IS's next utterance expresses confusion at IP's response, thereby cuing the possibility of plan disparity. In utterance (6), IP identifies one of these components as part of IP's context model, and attempts to justify its inclusion.

Analysis of naturally occurring dialogs such as that in Figure 7 suggest that natural language systems should use mechanisms such as default inference rules and focusing heuristics to expand the context model and provide a more detailed and tractable arena in which to understand and respond to subsequent utterances. However, these rules select from among multiple possibilities and therefore produce components that are more likely sources of error than components added as a result of intended recognition or inferences made on the basis of the system's domain knowledge.

Whether the system's domain knowledge supports, contradicts, or is neutral regarding inclusion of a component into the context model is important in deciding whether the user's query might address an aspect of the task outside the system's limited domain knowledge. Consider again the Kermit dialog shown in Figure 8. If the system can creatively construct a plan that may reasonably be ascribed to IS, is relevant to the current focus of attention, and which the system's knowledge supports or views neutrally (for example, floppy disks are an educational item and the bookstore sells many other items of an educational nature, so perhaps one can obtain a floppy disk by purchasing it from the bookstore), then the system can hypothesize that the cause of the plan disparity may be that IS has more extensive domain knowledge.

5.2.3. Response Generation and Negotiation

Webber [WEBB86] distinguishes between answers and responses. She defines an answer as the production of the information or execution of the action requested by the speaker, but a response as

"the respondent's complete informative and performative reaction to the question which can include ... additional information provided or actions performed that are salient to this substitute for an answer."

Our analysis of naturally occurring dialog indicates that humans *respond*, rather than *answer*, once disparate models are detected. This third phase often entails additional actions, including a negotiation dialog to ascertain the cause of the discrepancy and enable the models to be *squared away* [JOSH82] so that they are once again in alignment. Otherwise, the participants might lack confidence that they understand the other's utterances or that their own responses are being understood. In

the ensuing negotiation dialog, each participant may attempt to identify errors by making relevant assertions, asking questions, or justifying suspect portions of his own model.

The IRA example illustrates a negotiation dialog. In utterance (6), IP identifies a suspect component of her context model (that IRA's were available to IS in 1981) and justifies it. IS's next utterance informs IP that the assumption that led IP to infer that IS was eligible for an IRA is incorrect. IP then notifies IS that IP recognizes the error and that her context model has been repaired. The information-seeking dialog then resumes.

Negotiation is also appropriate when a query addresses an aspect of the task outside the system's limited domain knowledge. The system should attempt to understand IS's utterance in terms of the system's limited knowledge and provide any pertinent helpful information supporting or contradicting the user's plan. However, the system must inform IS of its limited knowledge in order to avoid misleading him by appearing to implicitly support his task-related plan. Thus the system's response takes the form of negotiation with the user as to whether the plan being constructed by the user is likely to work.

Consider again the Kermit dialog presented in Figure 8, and suppose that the system does not know how to obtain floppy disks but has reasoned that IS may be thinking about trying to purchase one from the university bookstore. The system can now respond to IS. This reply should of course contain a direct answer to IS's posited question. But this alone is insufficient. In a cooperative information-seeking dialog, IS expects the information-provider to assimilate the dialog and relate utterances to IS's inferred underlying task in order to provide the most helpful information. If the system limits itself to a direct response, IS may infer that the system has related IS's current utterance to his task-related plan and that the system's knowledge supports it – that is, that the system also believes IS can purchase a floppy disk at the bookstore. Joshi's revised maxim of quality [JOSH82] asserts that the system's response must block false inferences. In addition, as a helpful participant, the system should include whatever evidence it has for or against the plan component proposed by IS. An appropriate response would be:

> "The University Bookstore is open until 4:30PM. But I don't know whether it sells floppy disks. However, it does sell many other items of an educational nature, so it is perhaps a good place to try."

5.2.4. Model Repair

Once the cause of model disparity is identified, both participants must adjust their models to remove the disparities. If the system's model is in error, the system must alter its context model, as determined through the negotiation dialog. The system may communicate to IS the changes that are being made, so that IS can assure himself that the models now agree. For example, utterance (8) in the IRA dialog conveys that IP has modified her beliefs to include the correction provided by IS in utterance (7) and that, as a result, IP deduces that IS is not eligible for an IRA in 1981. On the other hand, if IS's model is in error, the system may inform IS of

any information necessary for IS to construct an appropriate plan and achieve his goals.

However, if the disparity was caused by IS's queries addressing an aspect of the task beyond the system's limited domain knowledge, then the system's model should be expanded to reflect what it believes to be IS's plan structure, noting however that its own knowledge offers only limited support for these components. This is the fourth kind of support that an enriched context model should include: differentiation among components according to whether the system's domain knowledge supports, contradicts, or is neutral regarding inclusion of the component as part of a correct overall plan. For example, upon determining that IS's domain knowledge may exceed the system's in the Kermit dialog, the system should expand its existing model to incorporate the acquired knowledge about how IS believes floppy disks can be obtained. The plan components creatively constructed can be added to the system's model, but as components proposed by IS and not fully supported by the system's knowledge. In this manner, the system can assimilate new utterances that exceed its limited domain knowledge and develop an expanded context model which serves as *knowledge* that can be referred back to in the ensuing dialog. This will enable the system to understand (within its limited knowledge) how IS plans to accomplish his objectives, and permits it to use this knowledge to understand subsequent utterances and provide helpful information.

5.3. Future Directions

We have outlined a four-phase approach to handling disparate plan structures, motivated by an analysis of naturally occurring dialog. We have further argued for an enriched context model that differentiates among its components according to the support it accords each component as a correct and intended part of the user's plan. Our current research is focusing on representing the knowledge that should affect the system's support for the components of an inferred plan, and on formulating heuristics for hypothesizing the source of error in the system's context model. This work should produce a more robust and natural user modeling framework.

6. Constructing and Exploiting Other Components of a User Model

There are many aspects of user modeling besides plan recognition that can aid in understanding and generating dialog. Lehman and Carbonell [LEHM*] have designed a framework that enables a natural language system to dynamically adjust to a user's idiosyncratic use of language. Their strategy extends the system's grammar to produce a customized parser that can be viewed as a model of how the user speaks.

Several researchers have exploited models of the user's knowledge and expertise. Chin designed a user modeling component called KNOME [CHIN*] that constructed a model of a user's knowledge about, and expertise in using, the UNIX

operating system. KNOME attempts to stereotype the user based on the user's queries to UC, a natural language help system for UNIX. This acquired knowledge about the user's expertise is then used in a variety of ways, including detecting user misconceptions and generating useful responses.

Paris was also concerned with exploiting a model of the user's domain knowledge. Her research indicated that texts directed toward naive users typically provide a different kind of information than those directed toward expert users. Her system, TAILOR [PARI*], used its beliefs about the user's domain knowledge to generate appropriate descriptions of physical objects.

Both McCoy and Quilici have addressed the problem of responding to misconceptions that a user exhibits about the domain. McCoy [MCCO*] was concerned with misconceptions about objects and their properties, and how the user's perspective on the domain should influence the corrective response. Quilici [QUIL*] was concerned with identifying the source of user difficulty in using the UNIX operating system and generating responses that corrected relevant user misconceptions.

Cohen and Jones [COHE*] have formulated a framework for reasoning on a user model in a system for educational diagnosis, in order to generate responses appropriate to the background and goals of individual users. Morik [MORI85b] has employed a user model to address user preferences in the context of a real estate system.

These research efforts can benefit from the work described in this chapter, and similarly our work can make use of many of the results of research in other areas of user modeling. For example, knowledge acquired from dynamically modeling the user's plans and goals may be useful in understanding ill-formed utterances [RAMS87] and could be incorporated into an adaptive parser. McCoy assumes that the system is given the user's perspective on the domain, and plan recognition can provide much of that perspective. On the other hand, if the system determines that a pragmatically ill-formed query is caused by a serious user misconception, the strategies proposed by McCoy and Quilici can be used to generate an appropriate corrective response. In addition, the system's beliefs about the user's expertise and knowledge of the domain will be important in hypothesizing whether a pragmatically ill-formed query is caused by a misconception or is simply the result of careless use of language.

7. Conclusions and Current Research

A cooperative natural language system must attempt to infer the underlying task-related plan motivating the information-seeker's queries and use this plan to provide cooperative, helpful responses. The system's model of this plan, which we call a *context model*, is one component of a user model. We have presented a strategy for dynamically inferring the context model from an ongoing dialog, and have shown how this model can be used to handle one class of problematic utterances – those that violate the pragmatic rules of the underlying world model.

However, the assumptions underlying current plan inference systems are unrealistic and must be removed. We have argued that an appropriate approach to handling

the resultant problem of disparity between the system's context model and an actual valid plan under construction by the information-seeker is an extension of the hypothesize and revise paradigm, consists of four phases, and requires an enriched context model that differentiates among its components according to the support it accords each component as a correct and intended part of the information-seeker's plan. Since this approach is motivated by the way humans appear to respond in similar situations, a natural language system that pursues this strategy will be viewed as acting naturally by human users.

8. References

[ALLE80], [CARB83], [CARB85], [CARB88], [CARV84], [CHAN78], [CHIN*], [COHE81], [COHE*], [FIKE71], [GRIC57], [GRIC69], [GRIC75], [GROS77a], [GROS81], [JOSH82], [KAPS79], [LEHM*], [LITM86], [MAYS80b], [MCCO*], [MCKE85a], [MORI85b], [PARI*], [PERR80], [POLL86a], [POLL87], [POLL82b], [QUIL*], [RAMS87], [REIT78], [ROBI80], [ROBI81], [SCHM78], [SIDN81a], [SIDN83b], [SIDN85], [SOWA76], [THOM80], [WEBB86]

7 Learning the User's Language: A Step Towards Automated Creation of User Models

Jill Fain Lehman and Jaime G. Carbonell

Abstract

No matter how comprehensive the coverage of a natural language interface, users will invariably provide unparsable input: idiosyncratic phrases, linguistically deviant utterances, or sentences simply beyond the linguistic sophistication of the interface. Although individual users differ significantly from each other in their preferred linguistic expression, they show surprising consistency in multiple interactions over time. Addressing both observations, a general framework is developed for automated adaptation to the preferred and idiosyncratic linguistic behavior of individual users. In essence, a method for flexible recovery and completion of partial parses is developed, with subsequent individualized extensions to a base grammar. Thus, the system adapts to the user by growing its grammar dynamically to acquire her preferred forms of expression and recognize them directly in future interactions.

1. Introduction: Adaptable Interfaces

To be effective, user interfaces require some measure of user training, whether by explicit instruction or implicitly through a gradual familiarization process. Thus, developing interfaces for novice users provides a great challenge in that the design must minimize the need for training and familiarization time. Natural language interfaces traditionally strive to bridge the user-system gulf by exploiting the user's existing familiarity with his or her own natural language.[1] However, even natural language interfaces fall short of immediate utility by naive users; no existing interface can deal with all the complexities of full natural language, including the full range of potentially relevant vocabulary, all non-standard turns of phrase, extra-grammatical utterances [CARN83d], abbreviations and other elided expressions. Thus, a user must learn the extent and limitations of the natural language interface, mostly by trial and error.

[1] Natural language exhibits many advantages (e.g., expressivity, human comprehensibility) and disadvantages (e.g., verbosity, complexity of machine interpretation) over other interface modalities such as menu selection, formal command interfaces, and graphic-oriented techniques. Whereas detailed comparative analyses are beyond the scope of this chapter, we note that the user modeling and adaptation techniques proposed here may well be adaptable to multi-media interfaces, speech, and any other free-form communication medium.

There are three possible methods of compensating for the fact that present interfaces do not achieve close to full coverage of the English language or the domain of discourse:

- **The user beware** – The simplest approach is to face the fact that many utterances will not be interpreted and expect the user to tolerate such a mismatch and learn the sublanguage of the system. One might hope that such adaptation could be rapid, but the empirical evidence discussed later in this chapter and elsewhere [CARN83a] suggests otherwise; frustration and disenchantment are the observed reactions.
- **The ultimate language interpreter** – Whereas the long-term goal of natural language processing is to interpret the full range of syntactic and semantic constructions possible in natural language while accommodating a measure of grammatical deviance [CARN83d, FAIN85], the state of the art does not approach such total performance. Moreover, as lexicons, grammars and semantic knowledge bases increase in size, and as more grammatically deviant constructions are searched upon failing an exact parse, the response time of any comprehensive system must increase considerably. Near real-time response, however, is necessary in a habitable interface. Also problematic is the reality that with increasing grammar size and increasing ability to recognize deviant forms, the number of spurious interpretations that the user would never consider sensible rises dramatically.
- **The user-adaptable interface** – Instead of asking the user to adapt, or asking the developer to build the ultimate system with complete coverage, a radically different approach is to build a system capable of adapting to the linguistic usage patterns of the user. In this approach we start with a kernel grammar (including semantics and lexicon) that is extended dynamically when the user provides uninterpretable input. Such extensions occur both by a combination of selective relaxation and augmentation of the kernel grammar and by focused clarificational dialog. Extensions to the grammar are stored in an individualized user model[2]; progressively fewer extensions are required as the system's linguistic coverage extends to that of the individual user (rather than to the union of all possible users, as in the previous alternative). This alternative is clearly superior for users who interact frequently enough with a system for it to create a rich linguistic user model.

The remainder of this chapter discusses in some detail the adaptation model in which a kernel grammar is gradually extended to match the linguistic usage patterns of each individual user. In addition, empirical evidence from *hidden-operator* experiments is presented that suggests:

- User performance, measured in speed and accuracy of task completion, increases significantly as the system adapts to the linguistic usage patterns of individual

[2] Thus, the grammar serves as a user model in a sense distinct from that of Schuster [SCHU85]. Her system, VP^2, used a static grammar in each of a native and second language to help detect and explain errors in second language use.

users. The same performance advantages accrue in a non-adaptive version of the interface only when the user's natural linguistic style conforms closely to the chosen kernel, and thus little adaptation by the user is required.

- The size of each resulting extended and user-customized grammar is much smaller than an equivalent complete grammar, i.e. a grammar that could tolerate all expected grammatically deviant forms. Moreover, the user customizations do not overlap greatly, showing diversity of usage across users, but consistency within users. This is exactly the empirical profile that makes our user customization process a tractable solution to the demands of the task.

2. Foundations: Least-Deviant-First Parsing and MULTIPAR

Our approach to adaptive parsing is based in part on ideas first explored in the flexible parser MULTIPAR [FAIN85] [MINT85]. MULTIPAR was designed to allow for the particular types of extragrammatical behavior frequently displayed by natural language interface users, for example, simple spelling errors, some word-order inversions, missing case markers, missing determiners, and semantic inconsistencies.

Like most natural language parsers, the system's performance can be described as search through the virtual space of possible parses for a given input. However, since ungrammatical constructions were permitted, the system could not limit search by rejecting a partial parse when a syntactic or semantic constraint was violated. Instead, each type of *deviation* from grammaticality had an associated recover strategy that compensated for it (e.g. by inserting a case marker where one w: missing) so that the parse could continue. Since use of a recovery strategy incurre some loss of confidence in the resulting meaning structure, deviant partial parses were inserted into a queue and explored in a *least-deviant-first* manner. More serious deviations from a grammatical parse were examined only if no parse was found with less serious deviations. Grammatical parses, of course, were attempted first.

In the next sections we examine the design of MULTIPAR more closely and discuss the mechanisms by which least-deviant-first exploration of the search space was accomplished.

2.1. Organization of Knowledge in MULTIPAR

In anticipation of our discussion of knowledge structures for adaptive parsing, it is useful to distinguish between semantic and syntactic knowledge in MULTIPAR. Semantic knowledge is completely contained in the set of *entity definitions* that define the actions and objects in the domain. An entity [HAYE84] is a generalized caseframe; a simplified example, taken from MULTIPAR's operating system domain, is given below. The entity shown encapsulates the system's knowledge about moving a file or directory from one place in a file structure to another. Note that entity definitions may be embedded according to the semantic composition princi-

ple, i.e. values of slots in entities may be terminal symbols or they may be other entities satisfying a set of local semantic constraints and global semantic predicates.

```
(EntityName   MoveCommand
      SemanticCases (
            Object (FileObjDesc[3] or DirectoryObjDesc)
            Source (DirectoryObjDesc or LogicalDeviceObjDesc)
            Destination (FileObjDesc or DirectoryObjDesc)
            Location (DirectoryObjDesc or LogicalDeviceObjDesc))
      Constraints (
            (Required Object Destination)
            (Destination FileObjDesc ) Object FileObjDesc)
            (Object DirectoryObjDesc ) Destination DirectoryObjDesc)
            (Object DirectoryObjDesc ) Source LogicalDeviceObjDesc)
            (Object DirectoryObjDesc ) Location LogicalDeviceObjDesc))
      SurfaceForms (
            (SFName ImperativeCaseframe-Canonical
                  Head (movehead)
                  DirectObject Object
                  Cases (
                        (Preposition (sourcepreps)
                         Bind Source)
                        (Preposition (destpreps)
                         Bind Destination)
                        (Preposition (locpreps)
                         Bind Location))))
      InstanceTemplate (
            Action 'MOVE
            Deviations Deviations
            Source (
                  IsA (IsA in Object)
                  Name (Name in Object)
                  Extension (Extension in Object)
                  Directory (Directory in Object or
                              Directory in Source or
                              Directory in Location)
                  LogicalDevice (LogicalDevice in Object or
                                 LogicalDevice in Source or
                                 LogicalDevice in Location)
                  ObjDesc (Description in Object)
                  SourceDesc (Description in Source)
                  LocDesc (Description in Location))
                  Destination (
                        ...)))
```

[3] FileObjDesc means file object description; other abbreviations are similar.

There are two sources of semantic information in the entity definition:

- **SemanticCases** define the basic structure of the entity in terms of the other entities that are its components. In the example, the entity MoveCommand has an object (the file or directory to be moved), a source (the directory or logical device to move it from), a destination (the file or directory to move it to), and a location (the directory or logical device in the context of which the move takes place). The semantic cases do not tell MULTIPAR how to recognize the entity in an utterance; rather they denote what other entities may or must be referenced in the input as part of recognizing the entity defined.
- **Constraints** specify, for any instance of an entity, both the predicates on individual cases and the relations that are required to hold between cases. The first constraint of MoveCommand indicates that the Object and Destination cases are required to be present for any instance of this action. The second constraint means that if the Destination case is filled by a file, then the Object case must also be filled by a file. The remaining constraints have analogous interpretations. MULTIPAR enforces constraints of both types insofar as it is able. In other words, it prefers parses in which the constraints are satisfied, but will accept (as a grammatical deviation) utterances in which the constraints are violated.

The location of syntactic information in MULTIPAR is more diffuse. Specification of the acceptable syntax can be found partly in the remaining fields of the entity definition. In particular:

- **SurfaceForms** act as a bridge between domain semantics and syntax; the parsing strategies they point to (through the SFName attribute) recognize the syntactic forms that indicate a reference to this action or object. In general, an entity may have more than one SurfaceForm, each of which corresponds to a different form of syntactic expression for the same underlying semantic case. There is, however, a 1-to-1 correspondence between SFName and parsing strategy in the implementation.[4]

  The remaining attributes (Head, DirectObject, etc.) are specific to the SurfaceForm and are best thought of as the values provided to parameters in the parsing strategy. Symbols like ⟨movehead⟩ are non-terminals in a grammar used by the DYPAR pattern-matcher [CARN81, BOGG85] and expand in the course of computation (e.g. ⟨movehead⟩ → move | transfer). SurfaceForm slot fillers that are not surrounded by "⟨⟩" are SemanticCase names. When a case name is present, the strategies appropriate to parsing entities of that type are invoked subject to the Constraints. Consider the DirectObject case in the SurfaceForm shown. The strategy that uses this form will be passed an instance of the MoveCommand entity definition and an input segment to work on. When the strategy reaches the point at which it expects the direct object, it retrieves the SemanticCase name from the SurfaceForm. Since the Object case can be filled by either a FileObjDesc or DirectoryObjDesc, the strategy will try to parse the current subsegment as each (unless the Destination has already been filled with a FileObjDesc as per

[4] Parsing strategies are simply functions written in Commonlisp.

the first constraint). To do this it calls those strategies corresponding to the SurfaceForms in the entity definitions of FileObjDesc and DirectoryObjDesc respectively. The result of the call may be an InstanceTemplate (see below) that is then bound to the case name.

- The **InstanceTemplate** binds syntax to meaning representation. The slot fillers associated with each attribute act primarily as a map between the values bound to the SemanticCases during the parse and a canonical meaning structure for the utterance. In this way, "move [c410jf90] foo to my directory" produces the same meaning structure as "move foo from [c410jf90] to my directory," even though in the former the directory name would be found when parsing the Object as a FileObjDesc and in the latter it would be found when parsing the Source as a DirectoryObjDesc.

While the syntactic information in the SurfaceForms and InstanceTemplate is fairly explicit, there is a great deal of syntactic knowledge left implicit in MULTIPAR's parsing and recovery strategies. It is the parsing strategy that knows how to organize the information given in the SurfaceForm. The parsing strategy dictates, for instance, the order in which the cases may appear in the input. The parsing strategy also contains the recovery strategy whose purpose is to compensate for the user's deviation from a particular, expected grammatical construction. The recovery for a violated expectation is placed at the point of the expectation.

The final piece of implicit linguistic knowledge we consider is the tolerance level associated with a recovery action. Tolerance levels, or *flexibility increments*, will be discussed more fully in the next section. Here we note only that they encode statically the relative severity of each of the specific types of deviation permitted by MULTIPAR.

2.2. The Control Structure

If the least-deviant parse is to be found efficiently, the exploration of alternative paths in the search space must be controlled carefully. For example, spelling correction should be tried before hypothesizing a missing word, hypothesizing one word before hypothesizing five. Moreover, a strategy failing to find a correct parse for some utterance does not mean that the recovery actions should be invoked immediately. It may be that another strategy will find a grammatical and semantically consistent parse; if so, that strategy should be tried before any recovery actions are attempted.

To control the exploration of the search space, competing alternatives within a parsing strategy are explicitly specified using SPLIT statements. When a SPLIT statement is encountered, the computation divides into separate branches; each branch has a *flexibility increment* indicating the additional degree of ungrammaticality tolerated by the associated continuation. The values of the increments form a partial ordering on the severity of the deviations for which MULTIPAR can compensate. A partial parse is generated along each branch, and each computation may proceed from the SPLIT statement independently of the others.

```
(Split (+0 continuationA)
       (+1 continuationB)
       (+3 continuationC)
            ...)
```

Thus, a SPLIT statement such as the one above produces a three-way branch in the search tree. ContinuationA does not result in a gain in flexibility of the associated parse: this path would be followed unless the particular grammatical expectations it represents are not satisfied by the utterance. Continuations B and C, if they are ever pursued, will result in gains of one and three respectively; each represents the violation of a grammatical expectation but the recovery action in continuationC creates more uncertainty about the validity of the resulting meaning structure.[5] All branchpoints in the parse deposit their untried combinations in the same global agenda, indexed by their global *flexibility level* (explained below). The structure of a global agenda looks like:

```
[Agenda (0 ContinuationA ContinuationD ...)
        (1 ContinuationB ...)
        (3 ContinuationC ContinuationE ...)
        (5 ContinuationF ...)
                ...]
```

MULTIPAR starts with an agenda containing one or more top-level strategies at the 0-flexibility level, and a global flexibility level set to 0 (i.e. full grammaticality). As each strategy is applied it can post additional continuations in the agenda at level 0 or higher. Continuations are posted when SPLIT statements are encountered, and the flexibility levels at which they are posted correspond to the sum of the global flexibility level and the flexibility increment of the continuation. When a continuation completes execution it is removed from the agenda.

MULTIPAR pursues all the continuations at the 0-flexibility level. Ideally, exactly one of these yields a complete and consistent parse, indicating that the input was grammatical and unambiguous. If more that one parse is generated, we have true ambiguity, which must be resolved by external means. If no parse is generated at the 0-flexibility level then the input was ungrammatical and MULTIPAR goes on to the next-higher flexibility level in the agenda, selecting this one as the new global flexibility level. The process is repeated at this flexibility level, perhaps resulting in more continuations being posted on the agenda. If no parse is found, the global flexibility level is increased again. In this fashion, the search for a possible parse continues, seeking the least deviant interpretations first. If the global flexibility level reaches some preset threshold, too much uncertainty has accrued and MULTIPAR abandons its parsing attempts.

[5] The non-determinism provided by the SPLIT statement is also exploited for handling normal ambiguities. When a strategy finds that a parse is ambiguous, a separate branch is created for each alternative parse. +0 flexibility increments are used for each locally-ambiguous but semantically-correct continuation.

3. CHAMP: Design for an Adaptive Parser

In MULTIPAR the loci at which expectations could go unmet were predetermined at fixed junctures in the grammar; these fixed locations corresponded to only a subset of the possible recovery points required by an input. In some sense, when we built MULTIPAR we simply extended our notion of what we would consider part of the grammar to include those classes of extragrammatical constructions most often produced by interface users. Our goal in building CHAMP,[6] an adaptive parser, is to create a system that acquires an idiosyncratic grammar with minimal ambiguity and maximal coverage via experience with a user. In essence, we need to remember the recovery actions performed in the presence of an unfamiliar utterance, and then augment the grammar for that particular user with a representation that can parse the same structure directly. In this way a kernel grammar can be extended to handle new forms, or variants of existing forms, whether they be true grammatical deviations, or grammatically valid extensions to a kernel whose coverage was less than complete. It is clear that the model of deviance and recovery embedded in MULTIPAR is inadequate to this task, for expectations can go unmet at any point in the input, and no methods for automatically extending the grammar were provided.

There is a powerful assumption implicit in the approach to adaptation that we describe. Specifically, the method can achieve our goal only if the individual user is reasonably consistent in his or her linguistic expressions over time. If the user's grammar is not *self-bounding* then the system's idiosyncratic language model will continue to grow to intractable proportions. In Section 4 we formulate the idea of *self-bounding* linguistic behavior more precisely and examine empirical evidence in support of the phenomenon. We note here only that linguistic regularity, like other forms of stereotypy, does appear with task repetition. Thus we limit the applicability of our approach to tasks requiring fairly frequent user interaction.

In order to take advantage of regularity in language use we must generalize the key concepts of deviation and recovery, starting from the following definitions:

- **Kernel grammar**: the lexicon, caseframe concept definitions, default semantic values, and syntactic forms that are present in the system prior to interaction with any user.
- **System canonical form**: any syntactic form in the kernel grammar.
- **Deviation**: the violation of an expectation as embodied in a syntactic form. Exactly four types of deviation other than misspelling are possible: the lack of expected text (deletion error), the existence of unexpected text (insertion error), the presence of text instead of the text expected (substitution error), and the presence of expected text in an unexpected location (transposition error). Note that misspellings correspond exactly to these four categories but the constituents are letters in words, rather than constituents in sentences [DURH83].
- **Recovery action**: one of insertion, deletion, substitution, transposition or spelling correction. Each action compensates for one type of deviation. The recovery

[6] CHAMeleonic Parser: it adapts to its environment.

Form: schedule | add [a | an] meeting | appointment | ⟨meal⟩ [⟨marked-case⟩]*

0 deviations: "Schedule lunch with Bob at 12"

1 deviation: "There is a meeting at AISys at 7: p.m."

Recovery:
1. substitute "schedule" for "There is"

2 deviations: "Meeting with Philip on Thursday, June 12 10 to 11: a.m."

Recovery:
1. insert "Schedule" before "Meeting"
2. insert "from" before "10"

3 deviations: "I have to be at AISys for a meeting at 7: p.m."

Recovery:
1. substitute "schedule" for "I have to be"
2. transpose "at AISys" to post-nominal position
3. delete "for"

4 deviations: "There is a business meeeting at AISys on June 11 7: p.m.to 8: p.m."

Recovery:
1. substitute "schedule" for "There is"
2. spelling correct "meeeting"
3. insert "from" before "7:00"
4. either substitute "meeting" for "business meeting" or delete "business"

Figure 1. Sample deviant sentences and recover actions

action for a deletion error is insertion; for an insertion error, it is deletion; transposition and substitution are their own inverses. Sample sentences at various levels of deviation and the recovery actions that correct them are shown in Figure 1.[7]

- **User canonical form**: a new syntactic form created in response to the successful application of one or more recovery actions in the presence of deviant input. These forms are stored in the *user model*, and used in parsing subsequent input more rapidly and with greater confidence.
- **User model**: the union of system and user canonical forms that grows in a self-bounded manner with increasing user interaction.

Using these definitions we can describe *adaptive parsing* in the following manner. For each utterance, we first attempt a grammatical parse, i.e. we try to under-

[7] The form given at the top of Figure 1 translates to: the word *schedule* or *add* optionally followed by the word *a* or *an* followed by the word *meeting* or *appointment* or *breakfast*, *lunch* or *dinner* followed by zero or more occurrences of a marked case.

stand the utterance using the current lexicon, defaults, and system canonical and user canonical forms. In the presence of an utterance that is deviant in this context, the general recovery actions may be applied. If the recovery is successful, a new form is constructed and added to the model.

In essence, the parsing process is one in which we look for the simplest combination of deviations that "explains" the input and store some generalization of that explanation as a derived user canonical form. We constrain the search for a meaning representation both by our method of indexing forms, which proposes candidates accounting for more of the utterance first, and by ordering the subsequent search space in a least-deviant manner, as in MULTIPAR. In addition, related forms, (a system canonical form and those user canonical forms derived from it) are ordered both by recency and frequency of use over time for purposes of guiding the search.

How does this approach achieve our goal? In the initial interactions with a user, we expect the number of deviant utterances to be fairly large, although the degree of initial disparity between the kernel grammar and the user's grammar clearly depends on both the choice of system canonical forms and the user's idiosyncrasies. Those utterances that can be explained without too much uncertainty (for example, containing at most two deviations) will give rise to user canonical forms. Note that an utterance that is more than two deviations away from every form in the user model at a particular point in time may become accessible through the adaptation process. This happens if, for example, a form lying between the grammar and the desired utterance is learned first; although previously unparsable, the desired utterance may now be within two deviations of the enlarged grammar. Thus, in Figure 1, if the one and two deviation forms had already been learned, then the four deviation form would be learnable with respect to the user model at that point; only recovery actions two and four would be necessary.

By extending the grammar in response to real instances of behavior, we increase the system's linguistic coverage in useful directions while incorporating only the ambiguity intrinsic to the user's idiolect. Having found that different users exhibit different preferences and idiosyncrasies, with little overlap in their acquired user canonical forms (see Section 4.3), our method of individually tailoring extensions to the grammar into unique user models proves an appropriate solution.

If linguistic behavior is self-bounding with frequent use, then eventually the user's grammar will stabilize. During this process, the likelihood that an existing form serves as an adequate explanation of an utterance increases until most of the learning behavior of the system consists of reordering rather than increasing the search space. While a user's behavior remains constant, the system's performance should asymptotically approach the optimal performance given any stable inherent ambiguity in the user's grammar. Should linguistic behavior begin to vary again (as when a novice user becomes more expert and moves away from full sentences and toward a command language), there would follow a period of decreased performance as the system learns new forms, then a return, with stabilization, to faster response.

It is important to note that the quality and utility of the "explanation" one can derive varies with the system's coverage, which is in turn a function of the history of experiences with a user. Yet self-corrective mechanisms exist within the

adaptive parser to minimize any detrimental effect a particular history might create. Consider the movement of the user's grammar over time. If a new form is a useful generalization of a true linguistic regularity, its predictive strength will reinforce its likelihood of use. Conversely, if a derived form is a poor explanation of an utterance, or reflects only a transient idiosyncrasy in the user's grammar, then after a time it will no longer be proposed by the indexing process; in effect, it will have been forgotten.

Having discussed the process of adaptation in a general way, we turn now to the specific declarative structures and control mechanisms around which the design is organized. In Section 3.4 we will bring the declarative and the procedural together in a detailed example of the system at work.

3.1. Representing the Task Domain

We have chosen as a task domain for the adaptive parser a subset of the activities of an executive assistant: maintaining a schedule of meetings and events, and arranging airline reservations when those events require travel. The number of actions and objects relevant to the domain is fairly small. In keeping a calendar, the actions are limited to viewing some portion of the current schedule or adding, deleting, or changing the value of an entry. In addition to meetings and events, objects in the domain include the different kinds of time associated with an event (e.g. start, end, duration), and types of locations, participants, and topics. For airline reservations the list looks similar: scheduling, changing and canceling flights, arrival and departure times and locations, flight numbers and passenger names.

Although the number of actions and objects is small, the domain has the typical degree of linguistic complexity. Consider the following hypothetical utterances which show the possible initial variation between different users expressing the same request:

User 1: Schedule a meeting at 2 with the Robotics Group for an hour.
User 2: Add a Robotics Group meeting from 2 p.m. to 3 p.m.

User 1: Show me my afternoon meetings.
User 2: Display every meeting after 12:30 p.m.

User 1: Cancel the one at 3.
User 2: Delete the 3 p.m.meeting.

User 1: Reschedule the meeting at 4 to tomorrow at 10 a.m..
User 2: Change the starting time of today's 4 p.m. meeting to tomorrow at 10.

User 1: Change the location to John's office.
User 2: Change the location of the meeting at 10 a.m. tomorrow to John's office.

Even for a single user reference to the same events may change over time [RICH84]:

Day 1: Schedule a meeting at 2 with the Robotics Group for an hour.
Show me my afternoon meetings.
Cancel the one at 3.

Reschedule the meeting at 4 to tomorrow at 10 a.m..
Change the location to John's office.

Day 14: Schedule Robotics Group, @ 2, 1 hr
Show afternoon
Cancel @ 3
Reschedule @ 4 to tomorrow @ 10 a.m.
Relocate to John's office

Day 28: Sc Robotics Group 2-3
Sh p.m. (or, just, "p.m. ?")
C 3
Res 4 tomorrow 10 a.m.
Rel John's ofc

As in MULTIPAR, we represent the domain concepts using caseframes. The general form is similar; an explanation of the fields follows the example:

```
(FrameName                Add
    Isa                   action
    Forms                 addforms
    InterSlotConstraints  (endtime ) starttime)
    Cases
        (c-object
            Isa (meeting seminar class meal trip)
            History nil)
        (c-date
            Isa date
            History nil)
        (c-start
            Isa hour
            History nil)
        (c-end
            Isa hour
            History nil)
        InstanceTemplate
            (action
                Map 'add
                Defaults nil
                Constraints nil)
            (object
                Map (and    (object Map c-object.isa
                                    Defaults nil
                                    Constraints (required))
                            (ref    Map c-object.ref
                                    Defaults nil
                                    Constraints ((ref = indefdesc)))
                            (specifiers   Map c-object.specifiers
                                          Defaults nil
                                          Constraints nil)))
```

```
(day
        Map (or c-date.day c-object.day)
        Defaults ((always? *currentday*))
        Constraints (required (day )= *todaysday*))) ...
(starttime
        Map (or c-start c-object.starttime)
        Defaults ((always? *currentstart*))
        Constraints (required))
(endtime
        Map (or c-end c-object.endtime)
        Defaults ((always? *nextstart*))
        Constraints (required)))
```

Some of the slots, such as **FrameName** and **IsA**, have straightforward interpretations. Others, such as **Cases** and **InstanceTemplate**, have counterparts in the entity definition examined in Section 2.1. The remaining slots are new, however, and worthy of note.

Recall that the InstanceTemplate is a way of packaging domain information in a canonical form. As is evident from the **Map** field, that information may be provided by a variety of bindings. For this reason the **Constraints** are attached to the slots in the InstanceTemplate rather than the cases (similarly, we talk about **InterSlot-Constraints** rather than InterCaseConstraints). In addition to private constraints, a slot may have associated **Defaults**. If, for example, the user fails to specify the day on which the object should be added to the calendar, the parser will first check the Constraints field to see if the information is required, then will look for a valid default in the current context. The kernel definition of Add instructs the parser to ask the user if the last day referred to is an appropriate value. As the system gains experience with the user, the value of a default or the context under which it applies may be modified by this kind of interaction.

Also new are the closely related fields **Forms** and **History**. The filler of the Forms slot is a pointer to the list of system canonical and user canonical forms that recognize references to this concept. As we try to interpret an utterance as a reference to an Add action, we will access the definitions of Add's casefillers. But these definitions are themselves caseframes that index their own forms. Just as we encode the utility of each Add form in the ordering of the Forms list, so we encode the utility of constituent forms in a case's History list. In fact, the ordering provided by the History list may override the ordering given in the Forms list of the nested caseframe. This is desirable if, for example, the user relies primarily on the system canonical form SC-*i* to refer to a meeting, but in the context of SC-1 (an addform) always uses SC-*j* to denote the object case. Currently we use the simple *move-to-front* algorithm to balance the influences of recency and frequency on the ordering:

1. Initially the Forms list contains only the appropriate kernel forms in a random order.
2. When a user canonical form is derived it is add to the front of the list.
3. When a known form successfully parses an utterance, it is moved to the front of the list.

Thus, as some forms fall into disuse and others dominate, the order of form names is altered to take advantage of the user's emerging idiosyncrasies. In this way we can also use Forms and History to control the amount of work required to understand a novel form; since a deviant utterance may partially index many forms, we use historical information to help select a candidate form for adaptation.

3.2. Representing the Learning Domain

Since the adaptive parser is learning primarily about language in a domain rather than about the domain *per se*, it is useful to distinguish between the task domain (activities of an executive assistant) and the learning domain (language). The caseframe representation discussed in the last section is used to describe the task domain; it is comparable to MULTIPAR's entity definitions in its encoding of the semantic knowledge in the system. To make the problem of adaptation tractable, we need declarative structures that capture the learning domain as well. These structures contain knowledge about language and syntax that was left implicit in MULTIPAR.

There are two sets of structures that work together to permit adaptation. First, there are the definitions attached to words and phrases in the lexicon. Two sample definitions are shown below. In the first entry from the lexicon, "add " is seen to be an element of the class of addwd. The entry for "schedule" would also show this membership. In contrast, "June" is more than just a lexical item. It is, by itself, a legitimate instance of a monthform – a system canonical form that recognizes references to months. Thus, the definition of June contains a fully instantiated InstanceTemplate for a month concept.

```
(add
      instanceof        (wd addwd)
      def               nil)

(June
      instanceof        (forms monthforms)
      definition        (month June))
```

"Addwd" and "monthforms" tie the lexical entries into the second important knowledge structure in the learning domain: the caseframe for canonical forms. A simplified example is shown below.

```
(FormName               Act1
      Belongsto         addforms
      Type              kernel
      Parent            nil
      Children          nil
      Strategy
            step1 (find-case m-dateforms c-date)
            step2 (find-case m-hourforms c-start)
            step3 (required find-terminal addwd)
            step4 (required find-case ggforms c-object))
```

Act1 is a system canonical form that recognizes one syntactic method of referring to the Add action. Because it is a kernel form, Act1 has no **Parent**; any user canonical form derived from it is stored as one of its **Children**. The link between the Add caseframe and Act1 is through the token "addforms" which is found both in Add's Forms list and Act1's **Belongsto** slot.

The key to adaptation lies in the explicit representation of the parsing **Strategy**. The parser itself is essentially a knowledge free interpreter that executes the "find procedure" in each step. When a step fails, the parser uses the general recovery actions to add continuations to the agenda. The general form for a strategy step has four parts (see step4 above): the first position is empty or contains the word "required" indicating that the step must consume input. The second position contains the name of one of the four step types from which all strategies are constructed; in addition to find-case and find-terminal (shown above), there is find-subform (a variant of find-case) and find-primitive (a variant of find-terminal). The third position is filled by the class name of the form or word expected at this point in the utterance. Finally, the fourth position may be empty or contain the name of a case to which to bind information returned by the find procedure.

3.3. Adaptation: Detecting and Recovering from Deviation

In order to learn an idiosyncratic language, we have moved away from a grammar-specific set of deviations and recovery strategies and toward a generalized notion of deviation and recovery which corresponds to finding the simplest explanation for any violated expectation that preserves the meaning of the utterance. The process of understanding is divided into three phases. In the first phase, the utterance is segmented into known and unknown token sequences. The result of segmentation is a representation of the input that reflects both grouping of lexical items and lexical level ambiguity.

Once segmentation is complete, the system begins the bottom-up, or voting, phase of the parse. The purpose of this phase is not to assign meaning but to provide evidence to the top-down phase about which canonical forms are most likely to account for the input. The kernel is small enough that all system canonical forms could be tried at the start, but as the user model grows this approach becomes impractical. The bottom-up procedure indexes those forms that seem to be able to cover the input; as a class of forms is predicted (for example, actionforms at the start of the top-down phase), members of the form class are ordered depending upon the number of nodes in the segmentation for which the voting phase found evidence. Syntactic forms that account for little of the input must necessarily lead to violated expectations; one way in which CHAMP differs from MULTIPAR is in being able to decide not to explore a form at level 0. The output from the voting procedure also helps the top-down phase "look ahead" at critical points in the parse to see, for example, whether there is evidence for a missing pre-nominal case in the post-nominal position.

Voting is possible because of the connections that are maintained between knowledge structures. When the steps in Act1's strategy (see previous section) are loaded into the system, the form class or word class is updated. For example (Act1 3)

is created as an annotation to the class addwd, (Act1 1) for m-dateforms, etc. When a node is created during segmentation its class membership is copied from the word or phrase definition in the lexicon; since the class annotation points into the strategy structures, the lexical referents in the utterance have been tied to the basic form steps that recognize them. Recall that the forms themselves are members in a form class hierarchy (thus, Act1 isa addform and addforms are actionforms). By following these links, evidence can be propagated upward, allowing each strategy to inherit the nodes covered by its constituents.

Finally, we reach the top-down phase where parsing commences. If the sentence is non-deviant with respect to the current user model, the parsing process amounts to straightforward caseframe instantiation. To begin, CHAMP takes the most-voted-for actionform and executes each step in the form's strategy in turn. If the step type seeks to fill a case, then the evidence list is consulted for the most-voted-for form in the case's form class, and the parser is called recursively. Parsing continues in this way until all of the input has been accounted for; the result is an instantiated InstanceTemplate for the appropriate calendar action.

If the sentence cannot be understood with the current grammar and default knowledge, at some point in the parsing process a deviation is detected. Since every word is now a possible locus for deviation, every strategy step must be permitted to fail, i.e., every strategy step is implicitly surrounded by a SPLIT statement that recovers from the general deviations represented by the failure of a find procedure. This uniformity permits us to remove the recovery mechanism from the strategies and place it in the interpreter. Thus, instead of SPLIT statements, the adaptive parser relies on an agenda like that described in Section 2.2 and two new control structures: the POCS (Path Of Computation Stack) and the LSQ (Leftover Step Queue). The purpose of these structures will become clear as we explore exactly how detection and recovery are accomplished.

Of the four error types, two may occur only when a find-terminal or find-primitive procedure fails. In a substitution error, the user has replaced the expected word or phrase with a different lexical item intended to achieve the same reference. In an insertion error, the user has added extraneous text at the current point in the input (see Figure 2). At the moment at which the expectation is violated, it is impossible to tell which of these errors has occurred. Since some kind of deviation is present, we must create a continuation which recovers from each. A continuation is a "snapshot" of the current computation that will permit processing to continue. It contains the current configuration of the LSQ and POCS, the name of the current strategy, the number of the current step, the type of deviation detected, and the unused portion of the segmented input.

If we are trying to explain the deviation as a substitution error, we create a continuation noting the step in which the substitution occurs and selecting a word from the expected word class as a synonym for the unknown segment. The step is pushed onto the POCS and the unused segment is updated. If a substitution continuation is restarted the step will have appeared to succeed. If this path is ultimately successful, a new entry will be created for the lexicon indexed by the previously unknown word or phrase and defined by copying the entry of its synonym.

To create an insertion error, the snapshot notes the inserted segmentation node and the number of the failed step. In addition, the unused portion of the input

Form: schedule | add [a | an] meeting | [(marked-case)]*
Error Type: Example (deviant input italicized)
Insertion: *Please* schedule a meeting on Tuesday with John. Substitution: Schedule an *appointment* on Tuesday with John. Deletion-terminal: Add a meeting *Tuesday* with John. Deletion-case: Add a on Tuesday meeting with John. Transposition: *On Tuesday* add a meeting with John.

Figure 2. Sample sentences with single deviations

is updated. Thus, if an insertion continuation is restarted the current step will be retried at a new point in the utterance. If the explanation turns out to have been correct a new lexical entry for this phrase will be added to the lexicon, with its own word class. A user canonical form is derived from the form being executed when the deviation was detected. It is created with a find-terminal step expecting a member of that class inserted at the appropriate point. Finally, an annotation is added to the class name pointing into the new strategy; in this way the bottom-up phase will vote for the new form if the inserted segment reoccurs.

Transposition and deletion errors are possible explanations when either a find procedure seeking a terminal or a find procedure seeking to fill a case fails (again, see Figure 2). For these deviations, detection is more problematic than above, for when a strategy step fails, each of the following is a possibility:

1. The step can succeed using text further on in the input. This corresponds to the initial detection of a transposition error. The step is pushed onto the LSQ, the step counter advanced and processing continues.
2. The text satisfies a step that we had previously placed on the LSQ. For steps on the LSQ that may succeed at this point in the input, a continuation is placed on the next level of the agenda in which the the transposition is noted, and the step is removed from the LSQ. If the continuation is restarted, the transposed step is executed first. If the path eventually succeeds, a user canonical form reflecting the change in step order is derived from the form in which the transposition occurred. The form class is updated as well.
3. The step will never be satisfied. This cannot be detected until the parser finishes executing the current strategy. If all the steps in the LSQ are optional or try to fill cases for which default values exist in the user model then no deviation has occurred; the current path can continue at the current level. But for each step on the LSQ that is required and for which there is no default information, either a *case deletion* or *terminal deletion* has occurred.

   A terminal deletion causes a continuation to be posted at the next level of the agenda with the step removed from the LSQ and the deletion noted. If the continuation is restarted, the success or failure of the path relies on the remaining contents of the LSQ. If the path succeeds, the user canonical form built in this situation removes the deleted step from the parent form.

A case deletion requires interaction with the user for its resolution; it generally signals the need to acquire default information. Thus, the current step is set to correspond to the the first required case on the LSQ and a continuation noting the need for user help is posted at the next level. If this continuation is restarted the system will query the user for the missing information. The success or failure of this path relies on the success of the interaction and the remaining contents of the LSQ. If the error can be resolved in this way, a new default for the case in the current context may be added to the concept frame recognized by the incomplete form.

3.4. An Annotated Example

To make clear how the components we have described combine and interact to learn a user's style, we present the following extended example. We will rely on the definition of Add and Act1 given previously as well as the assumption of suitable definitions for constituent cases and their forms.

The kernel form Act1 is adequate to parse the imperative sentence:

"On July 10 at 7 pm schedule a meeting with John".

Consider, however, the system's behavior when a user who prefers declarative phrasing types the reasonable variant:

"On July 10 at 7 pm there will be a meeting with John".

1. The utterance is segmented:

```
(G1 G2 G3 (OR G4 G5) G6 G7 G8 G9 G10 G11 G12)

G1: ST 1 END 1 DEF (WD DTMKR))
G2: ST 2 END 2 DEF (FORM MONTHFORMS))
G3: ST 3 END 3 DEF (IS NUMBER))
G4: ST 4 END 4 DEF (WD HRMKR))                  ) "at" can mark a
G5: ST 4 END 4 DEF (WD LOCMKR))                 ) location or hour
G6: ST 5 END 5 DEF (IS NUMBER))
G7: ST 6 END 6 DEF (WD DAYNITEWD))
G8: ST 7 END 9 DEF (IS TOKSEQ))                 ) "there will be"
G9: ST 10 END 10 DEF (WD ARTICLEWD))
G10: ST 11 END 11 DEF (WD MEETINGWD))
G11: ST 12 END 12 DEF (WD PRTMKR))
G12: ST 13 END 13 DEF (FORMS STUDENTFORMS))
```

2. The bottom-up phase creates the following evidence for actionforms. The sub-lists correspond to the step numbers within the strategy that may consume the given segmentation node. Since Act1 and Act2 are distinguishable only by the word class of their verbs (add vs. delete), their evidence lists are identical.

```
ACT1: ((G1 4 2) (G2 4 2) (G3 4 2 1) (G4 4 1) (G5 4)
       (G6 4 2 1) (G7 4 1) (G9 4) (G10 4) (G11 4) (G12 4))
```

ACT2: ((G1 4 2) (G2 4 2) (G3 4 2 1) (G4 4 1) (G5 4)
(G6 4 2 1) (G7 4 1) (G9 4) (G10 4) (G11 4) (G12 4))

3. The top-down phase begins and Act1 is chosen from the list of most-voted-for action forms. The parser begins execution of Act1's strategy with step1 (find-case m-dateforms c-date). The find-case looking for a marked dateform executes successfully; nodes G1, G2, and G3 are consumed, corresponding to "On July 10." The instantiated InstanceTemplate returned by the find procedure is bound to c-date and pushed onto the POCS along with the step number.
4. Step2 (find-case m-hourforms c-start) is executed. Since Act1 does not permit the location of the object to precede the verb (as in "At CMU schedule a meeting"), the sense of "at" given by node G4 is used. Nodes G6 and G7 are also accounted for and the resulting structure and the step number are pushed onto the POCS.
5. In step3 (required find-terminal addwd), the call to the procedure find-terminal fails. The parser must now try to compensate for a violated expectation using the general recovery actions. An insertion continuation is posted to level 1 of the agenda in which the current step number remains 3 but the unused segment begins with G9. A substitution continuation in which node G8 is tied to the class addwd is also added to level 1. A final level 1 continuation explains the error as a deletion of the required verb: the current step advances to 4 but the unused segment remains the same. Anticipating a transposition error, a continuation is posted at level 0 in which step3 is on the LSQ and step4 is current. Because a deviation was detected, the current branch of the computation returns failure.
6. The parser checks level 0 of the agenda for other forms or suspended computations that will give a grammatical interpretation to the input. It finds the anticipated transposition of the verb and restarts this path. Step4 also fails to consume G8 and a number of level 2 continuations are posted. (No anticipated transposition of step4 will be posted to level 0 because it is the last step).
7. At this point the parser must try any forms covering the same number of segmentation nodes as Act1. It will find only Act2 (a deleteform) and processing identical to the above will ensue. When the level 0 transposition for Act2 fails, the level 0 bucket will be empty, the global flexibility value will be incremented, and we will move to level 1 of the agenda.
8. The insertion continuation for Act1 is restarted with the unused segment of the input beginning at G9. Step3 is retried at this position but it also fails, creating snapshots at level 2.
9. The substitution continuation for Act1 is restarted with "there will be" (node G8) considered as a synonym for "schedule". The parser successfully executes step4 (required find-case ggforms c-object) with the unused segment beginning at G9. C-object is bound to the result and the unused portion of the input becomes nil.
10. Having completed the strategy, the parser makes certain that the LSQ contains no required steps, then returns successfully. The InstanceTemplate for the add frame is used to create the specific instance of the action referred to in the

utterance. The top-level control notes that all the input has been consumed, meaning that this path has led to a consistent interpretation of the input at level 1.

11. The agenda must still be checked for competing interpretations. The deletion continuation for Act1 fails when step4 is executed at node G8. The insertion and deletion continuations for Act2 will fail as above. The substitution continuation for Act2 will fail after the object case is filled because the deletion action requires a definite article to introduce the noun phrase.
12. With level 1 exhausted, we are confident we have found the least-deviant interpretation of the input. Depending upon the degree of deviation of the utterance and the severity of the effect of performing the action, the system may check the effect of its interpretation with the user. In this example, the system would show the additional entry on July 10th and ask the user the yes/no question, "Is this what you wanted?" (If the answer was no, the system would go on to level 2).
13. When the user responds affirmatively, the parser finds the deviation notation for this path in order to update CHAMP's knowledge structures. The substitution notation indicates that node G8 should be added to the lexicon as a member of the class addwd. The new entry looks like:

```
((there will be)
      instanceof        (wd addwd)
      def               nil)
```

14. Next the history of the path is recorded: the token "Act1" is moved to the front of the add frame's form slot. Similarly, the form and history slots for the constituent concepts referred to by the utterance are updated.
15. Finally, the parser returns the instantiated InstanceTemplate to the database calendar routines which perform the update.

The next time the user tries to add an object to the calendar using the declarative "there will be" phrasing, the bottom-up phase will advance Act1 as the unique preferred form and step3 of that strategy will execute successfully at level 0.

4. Hidden Operator Experiments with Professional Secretaries

In prior discussion (Sections 1 and 3) we identified two conditions that limit the applicability of adaptation in user modeling. We review these now and add a third:

1. Within-user consistency – The tractability of the particular computational model we have presented requires that the user's grammar converge on a stable subset of natural language. We have called this asymptotic movement toward a fixed set of utterances "self-bounding behavior." If the user's past behavior is not an indicator of his future behavior, adaptation as we have described it is no longer useful because the user's grammar grows without bound.

2. Across-user variance – If all users employ the same subset of language, i.e. if there is no natural idiosyncrasy of expression, then the correct approach to interface design may not require adaptation. By "employ the same subset of language" we mean that each user employs (almost) all the forms in the subset and only forms in the subset. In this situation, a methodology such as that of Good *et al* [GOOD84] would be preferred; through a generate-and-test cycle one builds a single system incorporating the complete common subset. Note, however, that even if all users converge eventually on a consistent subset, if their paths to that point are highly idiosyncratic, adaptation may still ease the way. Our data indicate disparate linguistic subsets across users.
3. Partial user adaptability – To the degree that a user can adapt to the system's limitations, such adaptation should be exploited, allowing the user and system to "meet in the middle." However, our experiments show that most of the adaptation should be performed by the system, permitting the user to focus on her task and relieving her of the cognitive burden of learning the boundaries of the interface's sublanguage.

The behavioral assumptions that underlie our approach are apparent: we believe that with frequent interaction a user will develop stereotypical and idiosyncratic language patterns; further, a system that permits the natural expression of these patterns through adaptation will result in better task performance than a system that forces the user to adapt to it. In the series of hidden-operator experiments described in the remainder of this section, we found that in conditions of frequent use the expected behaviors were clearly evident. The underlying assumptions for automated adaptation were indeed borne out.

4.1. The Base-Adaptive Experimental Condition

In a hidden-operator experiment[8], users were told that they would be typing the input to a *natural language learning interface* that would increase its knowledge of English while helping them keep an on-line calendar for a busy professor/entrepreneur. In each of nine sessions, the user was given between ten and twelve pictorial representations of changes to be made to the calendar. Two examples of these *subtasks* are shown in Figure 3. She was told to interpret each picture and effect the change by typing her commands as if she were "speaking or writing to another person". Although no time limit was enforced, users were told that it was more important to attempt than to accomplish each subtask. They were given the general rule that three tries were sufficient effort before moving on to the next picture.

If the user's utterance was deemed either parsable or learnable (containing at most two deviations) by the experimenter, the user was told that the intended action had been carried out and the calendar and language model were updated as appropriate. If the utterance was too disparate with respect to the current model of

[8] In a hidden-operator experiment, the user believes she is interacting with a computer system, but, in reality, the feedback is produced by the experimenter at a remote terminal. In this way, alternate designs of an interface and all its capabilities can be tested prior to actual construction.

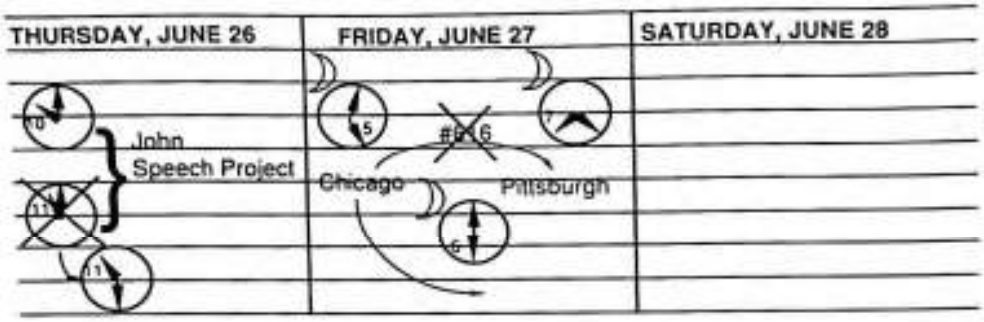

Figure 3. Sample pictorial stimuli

the user's grammar, the user was generally told which parts of the utterance were understood and asked to "please try again."

4.2. Control Conditions

One counterargument to an adaptive approach maintains that the user will naturally adapt to the system's limitations faster than the system can usefully adapt to the user's idiosyncrasies. To evaluate this claim we designed two variations on the basic experimental condition.

In the No Adaptation condition, the experiment was conducted as outlined except that users were not told the system was a learning interface and the kernel grammar was never updated. Although the system remained more permissive of extragrammaticality than the average natural language interface, the boundaries of the grammar were rigidly fixed.[9] Any increase in performance would therefore be attributable to the user's adaptability.

The Echo-Adaptive condition was included in response to arguments by Slator *et al* that users want to learn, and will learn, a mnemonic command syntax and domain-specific vocabulary [SLAT86]. When given the graphics command language equivalent to their natural language input, his users were able to gradually integrate the terser forms into their grammar. It was not known whether individuals would show the same propensity in interactions with an adaptive system. Users in the Echo-Adaptive condition were given the same instructions as those in the Base-Adaptive condition except that they were told the system would display a paraphrase of their utterance in an "internal form" that they were free to incorporate into their commands or to ignore as they saw fit.

[9] Because we felt it important to keep the criteria for parsability the same under all conditions, we extended the kernel grammar for this condition in one respect: use of articles was made optional. This is the most common ungrammatical construction displayed by interface users. If we had not extended the grammar in this way, most sentences would have involved more than two deviations and the results from this condition would have been comparable primarily to session one of the base condition.

None of the users who participated had prior experience with natural language interfaces. Each was employed as a secretary or executive assistant in either a business or university environment. Although not every user maintained a calendar for her employer, each had kept a personal calendar for at least one year.

Sessions were run at 24 hour intervals (approximately) for all users except User 1 whose sessions were run twice a day with a five hour break. The number sessions of varied in the two control conditions in accordance with the availability of the users. Table 1 summarizes the relevant user information.

Table 1. Summary of user information

User	# years other calendar	# years own calendar	Employer	Condition	# sessions
User 1	2.25	1	univ-psych	Base-Adapt	9
User 2	0	1.33	company	Base-Adapt	9
User 3	1.5	2	company	Base-Adapt	9
User 4	2	4	company	Base-Adapt	9
User 5	0	4	univ-compsci	Echo-Adapt	3
User 6	0	10	univ-compsci	No Adapt	3
User 7	0	"always"	company	Echo-Adapt	5
User 8	4.5	40	univ-english	No Adapt	5

4.3. Results and Discussion

The results for users of the adaptive system indicate a high degree of self-limiting behavior, converging upon a user-specific recurrent subset of English. With the exception of case ordering – which was strongly influenced by the stimuli – there was virtually no overlap of user canonical forms at the sentential level; non-sentential similarities were limited primarily to missing determiners and some common phrase marker substitutions. Figure 4 shows the approximate number of changes to the grammar over time for each of the four Base-Adaptive condition users, Figure 5 shows the same information for the Echo-Adaptive condition. Both figures indicate by user and session the number of new constructions per sentence (cps) which is calculated as the total number of changes to the grammar divided by the total number of parsable sentences. The number of unparsable sentences in a session is given in parentheses next to the fraction. Cross-hatched areas of Figure 4 indicate sessions following the introduction of supplementary instructions to perform the task as quickly as possible. The purpose of these instructions was to encourage the user toward a terse, possibly idiosyncratic, command language.

Figures 4 and 5 show both the importance of how the initial grammar is chosen and the difference in the rates at which language stabilization occurs. Observe that User 1's grammar requires the least learning and stabilizes quickly. She began with terse utterances and a vocabulary that was quite close to the kernel grammar chosen

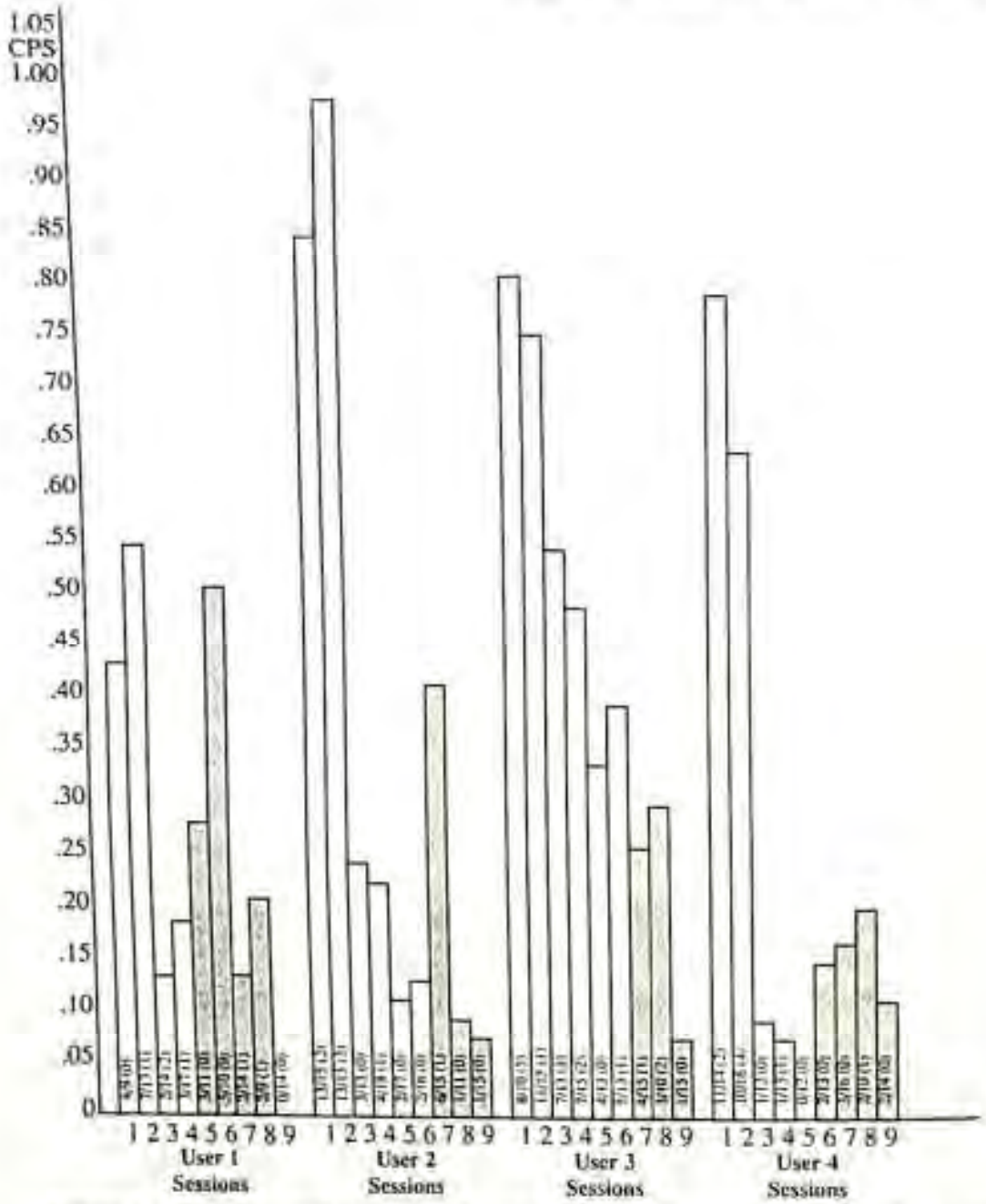

Figure 4. Self-limiting behavior in the Base-Adaptive condition as measured by the number of new constructions/number of parsable sentences (CPS)

for the system. User 2's grammar showed less initial resemblance but she employed it consistently and thus stabilized quickly as well. User 3 had a tendency to be both polite and verbose, so her utterances were more likely to contain new forms and stabilization occurred more slowly. User 4 showed the most rapid stabilization. It is interesting to note that although the new forms she introduced in sessions one

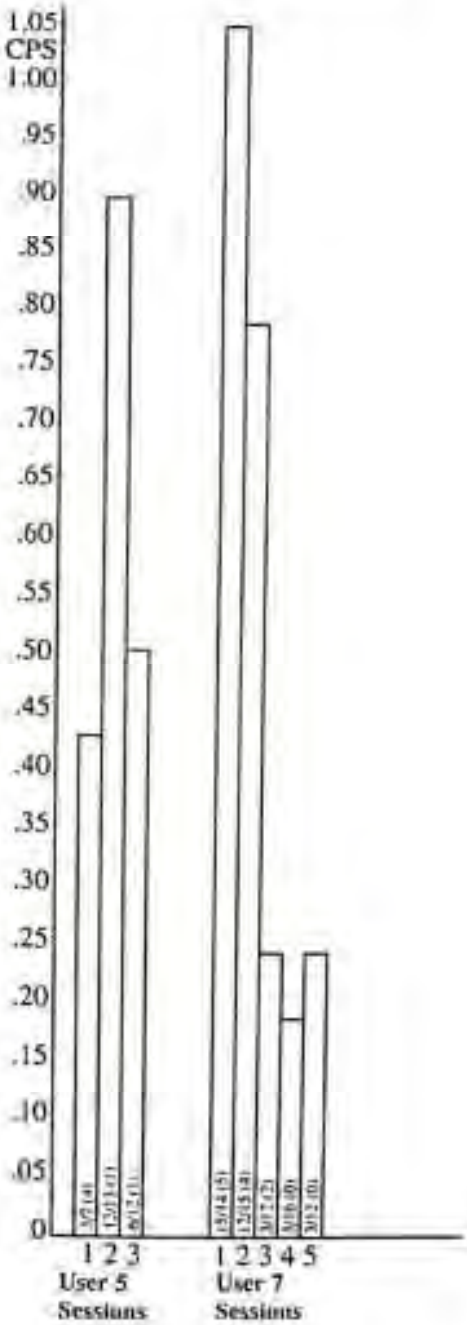

Figure 5. Self-limiting behavior in the Echo-Adaptive condition as measured by the number of new constructions/number of parsable sentences (CPS)

and two were not terse, they were quite "natural," i.e. they were used consistently and without modification in subsequent sessions.

User 5's behavior is of limited interest as she became ill after the third session and her participation was discontinued. User 7's graph shows much the same trend as that found in the Base-Adaptive condition. Neither User 5 nor User 7 showed any integration of the command language paraphrase, putting our results at odds

with Slator's general claims. Our experience did, however, replicate that of Good *et al* [GOOD84] and Cobourn [COBO86]: there is a strong assumption on the part of users that if the system can generate it, the system can parse it. Thus, although neither user in the Echo-Adaptive condition chose to employ the command language provided, many of our users displayed idiosyncrasies that were clearly based on the system's output. User 7, for example, came to construct all instances of the schedule command (and only this command) using the template: ⟨start-time⟩ – ⟨end-time⟩ ⟨date⟩ ⟨meeting or meal⟩, as in "12:00 – 1:30 June 11 lunch with Andy". Utterances thus constructed deviate from a kernel form only by the deletion of the verb and indefinite article; thus, the user canonical form is easily learned. In addition, these utterances closely conform to the template used by the experimenter to seek confirmation of an interpretation, e.g.:

```
Do you want:
      12:00-1:30          lunch, Andy
```

Table 2. Evolution of an element in User 2's grammer

	Session								
	1	2	3	4	5	6	7	8	9
SC: from <time> to <time>	-	2/6	-	5/9	4/4	2/3	-	6/6	1/3
UC1: from <time> till <time>	2/4	-	-	-	-	-	-	-	-
UC2a: beginning at <time> and ending at <time>	2/4	3/6	6/7	-	-	-	-	-	-
UC2b: beginning at <time> ending at <time>	-	1/6	1/7	-	-	-	-	-	-
UC3: <time> to <time>	-	-	-	4/9	-	1/3	1/1	-	2/3

To illustrate what we mean by a self-limiting grammar, the evolution of a particular referring phrase is shown in Table 2. The five forms that User 2 employs to refer to a time interval are given in the left portion of the figure. The actual use of a form in each session is shown as a fraction of the number of opportunities for its use. Initially, the system grammar contains only the system canonical form (SC). Using the general recovery actions, the experimenter equates the user canonical (UC) forms as they appear. Note the transience of UC1 and UC2 and the eventual reliance on the terse system canonical form and an even terser user derivative (UC3).

Figures 6 and 7 display the results for the No Adaptation condition. Recall that the purpose of this variation was to study the claim that people adapt well enough to obviate the need for system adaptation. We measure user adaptation in

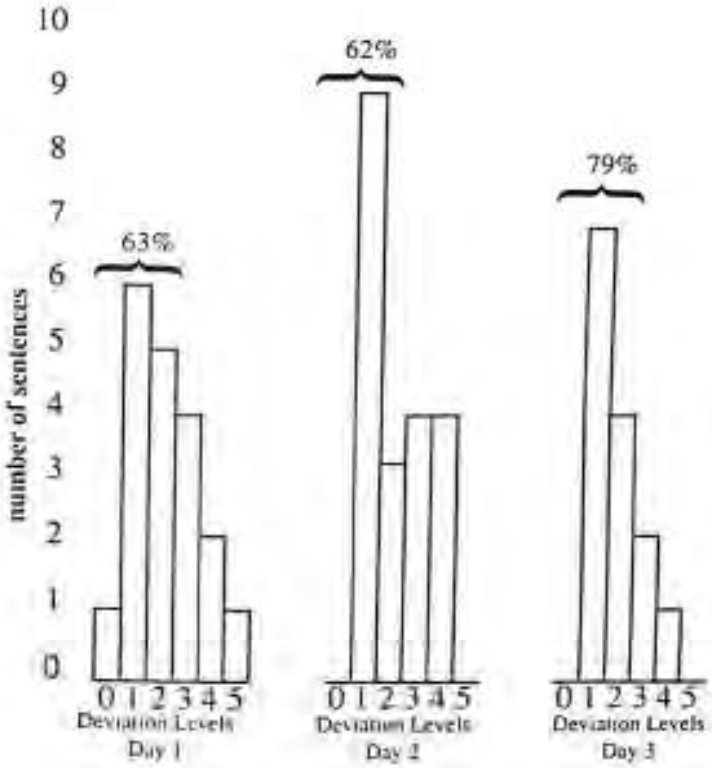

Figure 6. Number of utterances parsable at each deviance level in No Adaptation condition, User 6

this condition by categorizing each utterance by the minimum number of deviations required to parse it. If the user is adapting to the limitations of the system we should see a general increase in parsable sentences (those containing zero, one or two deviations). Further, grammatical utterances (zero deviations) should come to dominate. The latter conjecture is based on the belief that slower response time and more work are negatively reinforcing. Since the interpretation of utterances with one or two deviations took longer and had to be confirmed by the user, finding forms in the kernel grammar should have been preferred.

The graph for User 6 clearly shows the increase in parsable sentences. Her behavior is interesting in two respects. First, it was reminiscent of User 1's in that User 6's grammar was fairly close to the system kernel, especially with respect to her vocabulary, and she tended to rely on relatively terse forms. As a result, there were, in general, fewer loci for deviation. When an utterance did meet with failure, her next try almost always succeeded. Second, only one of her 54 sentences was without deviation, refuting our previous conjecture that grammatical utterances would be sought. Note, however, that deviant forms that are terse can be checked fairly quickly against the grammar so the penalty paid in time for non-grammaticality is not very great. Still, it must be admitted that User 6 did not seem to mind the extra work involved in confirming the system's interpretation of her command.

We will defer discussion of User 8 until we have examined two additional measures of performance. Table 3 shows the number of subtasks completed by each

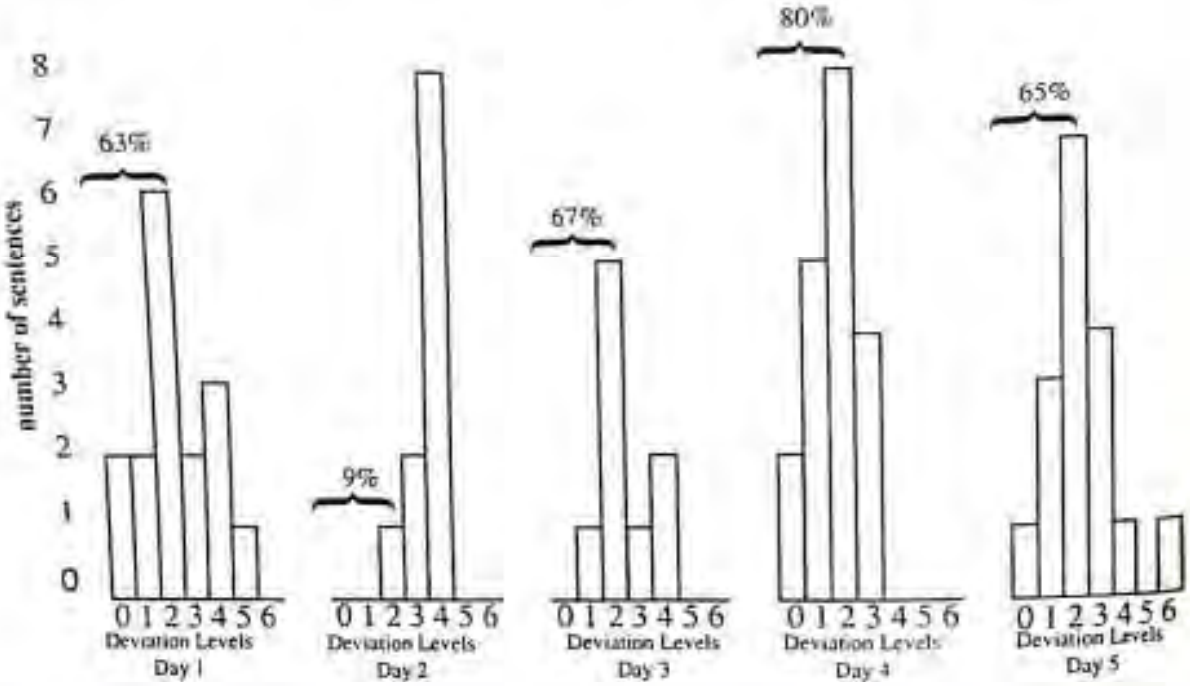

Figure 7. Number of utterances parsable at each deviance level in No Adaptation condition, User 8

user in each session. Table 4 shows the related metric of time taken per subtask. With respect to the former measure, users other than User 8 generally managed to effect the changes in each subtask they tried.[10] They rarely required more than three linguistic attempts per subtask in the initial sessions and averaged slightly more than one attempt per subtask in later sessions.

The general decrease in task time is a predictable result of self-limiting behavior. Users in the adaptive conditions accomplish the task in less time as the grammar of the simulated system comes to reflect their own and more sentences are judged parsable rather than learnable. When the supplementary instructions are given (Base-Adaptive condition only) and the users are encouraged to employ simpler, terser forms, task times increase as the new forms are learned. When these forms stabilize, response times fall to new lows since the shorter forms take less time to check against the grammar. Because she is consistent in her use of what worked in the past, and because her initial grammar conformed so closely to the kernel, User 6 shows the same general trend in time spent as those in the adaptive conditions.

User 8 shows neither consistent decline in the time taken each session nor in the number of unparsable sentences (see Figure 7). The combination of her work experience and the content of her utterances leads us to conclude that little of her

[10] There were four sessions in which a user skipped all or part of one or more subtasks. In these cases no utterance relating to the subtask appears in the log file.

Table 3. Number of completed subtasks, arranged by user and session ([#] indicates number of subtasks skipped)

			Session (total number of subtasks possible)								
			1(10)	2(12)	3(12)	4(10)	5(10)	6(10)	7(11)	8(10)	9(11)
Adaptation	Base	User 1	7[1]	11	12	10	10	8[2]	11	10	11
		User 2	10	12	12	10	10	10	11	10	11
		User 3	8	11	12	10	10	10	11	10	11
		User 4	10	12	12	10	10	10	11	9.5[.5]	10.5[.5]
	Echo	User 5	6.5	9	12						
		User 7	10	12	12	10	10				
No Adaptation		User 6	10	12	12						
		User 8	7	1	4.5	5	6				

behavior can be attributed to task misunderstanding. Put simply, she was unable or refused to adapt.

Her linguistic style, like that of User 3, can be characterized as verbose. In response to a typical utterance the system would tell her which segments it could parse and ask her to try again. She would then try typing exactly those segments just echoed, with no connecting text. When the terse form met with failure she would gravitate back to overly explanatory forms.

User 8's performance in session two was so poor (in 31 minutes she produced one parsable sentence with two deviations) and her frustration so great that she was given hints prior to beginning session three about how to use the system more effectively.[11] Although the help appeared to improve her performance in sessions three and four, in session five there is still a preponderance of unparsable forms. Contrast this with the behavior of users in the other conditions, most of whom had cut their number of unparsable utterances to zero by session five. Finally, note her low values for the number of tasks completed. These were due largely to the fact that she never managed to find a parsable form for an entire class of subtasks (those involving the "change" action). Taken as a whole, User 8's performance is a strong counterargument to the notion that everyone finds adaptation natural.

[11] Specifically, she was told to try to type simple, but fully grammatical sentences and to think of the system as someone to whom she was writing instructions. She was the only user who was given this aid.

Table 4. Number of minutes/subtasks arranged by user and session (* indicates approximate value due to damage to log files; + indicates supplementary instructions given this session)

		Session 1	2	3	4	5	6	7	8	9
Adaptation, Base	User 1	5.5*	3.0	2.5	2.8	1.7+	2.4	1.6*	2.1*	1.0
	User 2	5.0	3.5	2.0	3.5	2.2	2.1	2.2+	1.3	1.3
	User 3	6.3	3.2	2.7	3.3	1.7	2.3	1.6+	2.0	1.0
	User 4	3.4	3.3	1.7	2.1	1.1+	1.6	1.5	1.5	1.1
Adaptation, Echo	User 5	8.5	5.3	3.3						
	User 7	4.7	3.8	2.5	3.3	1.5				
No Adaptation	User 6	4.3	3.7	2.0						
	User 8	6.3	31.0	7.1	5.0	5.3				

5. Concluding Remarks

Natural language interfaces can, in principle, be made much more effective by automated adaptation to individual users. Analysis of data from hidden-operator experiments demonstrated that there are significant differences among the active linguistic patterns generated spontaneously by different users, but that each user is fairly consistent across sessions spanning several days. These observations suggest that as an integral part of individualized user models, one should store the linguistic preferences and idiosyncrasies of each user. Such information allows for direct and accurate interpretation of the user's commands and queries in the given interface. Individualized user models, however, must be acquired automatically in order to be of any practical significance – the designer cannot hand-analyze the behavior of each individual user and hand-tailor the kernel grammar to be consistent with the observed behavior.

This chapter developed a method for automatic user adaptation starting from a kernel grammar of system canonical forms. In brief the process may be summarized as follows:

- The system begins with the kernel grammar. If the user is known to the system, and user canonical forms have been synthesized for her, these are loaded atop the kernel, forming the complete user model.

- As user inputs are processed in the interface, the system attempts to parse them using the (possibly augmented) kernel grammar and recovery strategies as needed.
  - If exact (non-deviant) parsing is possible, the user's intention is understood and no recovery or adaptation methods are required.
  - If exact parsing is not possible, recovery strategies are attempted in a least-deviant-first manner. These strategies are based on lexical spelling correction or manipulating grammatical constituents by composing the following four simple operations: insertion, deletion, substitution and transposition. If the application of a small number (typically two) of such operations results in a parse, the user is asked to confirm the system's interpretation, and the correct action is carried out.
- If a user-confirmed deviant parse occurred, the transformed grammar construct is subjected to a conservative generalization process, based on recent explanation-based learning methods [MITC86], then added to the individual user model.

Presently these methods are being developed and refined.[12] The MULTIPAR project demonstrated the feasibility of several key concepts: focused parse recovery, least-deviant-first search methods in the space of possible partial interpretations, and the parse-continuation computational architecture as an effective implementational substrate. However, MULTIPAR was unable to modify the grammar or change its behavior with experience (whether globally or tailored to an individual user). These limitations were caused in small part because the parse recovery mechanisms were encoded as arbitrary LISP functions, and in large part because MULTIPAR defined the notion of deviation in a very narrow manner. The five grammar adaptation operations discussed in this chapter provide a conceptually simpler and (allowing for composition) more powerful recovery method. Such a set of strategies performs well-defined operations on the current grammar on an "on demand" basis, i.e. when attempting to parse a grammatically deviant input. The grammar constructs that result are expressed in a declarative language to facilitate generalization and further adaptation.

Acknowledgements. This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA order No. 4976, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520 and in part by scholarship funding provided by the Boeing Company. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, Boeing or the US government. The authors would like to acknowledge the contributions of Steven Minton who designed and implemented MULTIPAR's control structure, the secretarial assistants from Carnegie-Mellon University, Lexeme Corporation and UNILOGIC, LTD. who participated in the experiments, and Marion Kee and Ben McLaren who helped conduct the experiments.

[12] The knowledge structures and algorithms discussed in Section 3 have been implemented in CHAMP, but the generalization process — though applied successfully in other projects at CMU — remains to be implemented and tested.

6. References

[BOGG85], [CARN81], [CARN83a], [CARN83d], [COBO86], [DURH83], [FAIN85], [GOOD84], [HAYE84], [MINT85], [MITC86], [RICH84], [SCHU85], [SLAT86]

Part III

Exploiting User Models

Introduction

In Part III, a number of possible applications of user models in dialog systems are examined in closer detail. These include the exploitation of user models for tailoring object descriptions to the user's level of expertise, correcting user misconceptions with respect to taxonomic object classification, monitoring a user's impression of objects in an evaluation-task dialog, and adapting an expert system's response behavior to the background knowledge of its users. Besides the domains for which the described systems stand as representatives, there exists a large number of other possible applications, some of which have been described in Parts II and IV. For surveys on systems not reported on in this volume, see [WAHL*, KASS88].

Cécile Paris investigates how assumptions about the user's level of expertise can be employed by a dialog system for tailoring descriptions of complex physical objects. The author compared a number of encyclopedias for children and adults and found that different types of descriptions are used in each of these two text types, which are obviously aimed at two different types of readership. For readers who can be expected to possess some background knowledge about the domain, objects or concepts are described in terms of their subparts and the properties of these subparts. This pattern of description has been identified in studies on text organization by [MCKE85a, MCKE85b] who called it a 'constituency schema'. Descriptions for "naive" readers, on the other hand, essentially describe the processes associated with the operation of the object. This pattern is called a 'process schema' by Paris. The author explains her findings by assuming that readers with background knowledge – in contrast to "naive readers" – can be expected to figure out how parts fit together to form an object capable of performing a function. Providing such information would thus contradict Grice's [GRIC75] maxim of quantity and should therefore be avoided.

Paris' analysis suggests that the user's level of domain knowledge affects not only the *amount*, but also the *kind* of information provided in descriptions. The author suggests that, based on a user model, dialog systems should also employ these two distinct strategies for generating descriptions which are well-adapted to the user. Since users are hardly ever either completely naive nor completely expert, she also suggests that a dialog system should *combine* both strategies, for example in describing complex physical systems: Whenever the user model indicates that the user possesses local expertise about an involved object which is to be described, the constituency schema can be employed; otherwise the process schema must be used. Paris outlines the points in each description schema at which a switch to the other schema is possible.

The proposed principles have been implemented by the author in TAILOR, a system which – roughly – takes a request for an object description as its input, accesses its user model, and generates a conceptual representation of an object description adapted to the user's level of expertise. After its concepts have been replaced by lexemes, this description is passed on to a generator which unifies it with a functional grammar to produce English sentences. The user model of TAILOR contains individual assumptions about the user's experience with respect to the domain of discourse. These assumptions are represented by a list of those items in the system's knowledge base which are known to the user, and by information about whether the user understands the basic concepts underlying the domain of discourse. The operation of the program is illustrated by a detailed example. In a final section, Paris briefly presents heuristics that might be used to acquire this user model.

A frequent type of user misconception is the misclassification of objects (i.e. their subsumption under a wrong superconcept in a concept hierarchy). Another type is the belief that objects have certain attributes or attribute values which they do not actually have. **Kathleen McCoy**'s contribution deals with how to behave after such misconceptions have been detected and classified as belonging to one of these categories. For each type of user misconception, she proposes three strategies for generating appropriate responses. The generated response patterns are then transformed into natural language responses using McDonald's [MCDO80] MUMBLE system.

In all response strategies, the similarity of objects plays an important role. Object similarity is defined via the similarity of the *attributes* of these objects. However, not all attributes are taken into account, but only those which are *salient* within the current dialog context. McCoy discerns three types of salience, each stemming from a different source: static salience encoded directly in the generalization hierarchy, dynamic salience gained by an attribute being explicitly mentioned in a discourse, and perspective salience gained by viewing the domain objects through the active perspective.

Perspective salience turns out not to be a substructure of the concept hierarchy, but to form an independent structure (namely a set of highlighted attributes) which is orthogonal to the concept hierarchy. Detailed examples from the field of financial advice are given to illustrate the effect of perspective on attribute salience and on one of the above-mentioned response strategies. Suggestions are made as to what influences the choice of perspective in a given dialog.

The chapter by **Anthony Jameson** is concerned with the modeling, monitoring and exploitation of a rather complex belief type, namely the user's belief about the user model (in the notation of [KOBS*], system beliefs are here of type S (3.1.1.) v (3.1.3.) U (3.1.1.) v (3.1.3.) S (3.1.1) U (1.1.1) v (2.1.1.)), and about the motivation of the system. The issues that arise in connection with this sort of belief can be most easily studied in everyday dialogs between human participants. A speaker generally not only exploits some model of the listener's beliefs, interests, and/or goals in deciding what to say; s/he also takes into account the fact that the listener will have

some *image* of this model and the way it is being used. The listener employs this image, which may be vague and/or inaccurate, to make substantive inferences that go beyond the conventional meaning of the speaker's dialog contributions.

Jameson's specific examples and his simulation model IMP are restricted to a specific kind of dialog setting, namely *evaluation-oriented dialog*. In this setting, the goal of one participant (the listener) is to evaluate some object along one or more dimensions, and that of the other participant (speaker) is to provide the information the listener needs. (In the specific examples used in the chapter, the system simulates a job applicant who is describing herself to a personnel interviewer.)

The issues raised and the model's performance are discussed in a sequence of increasingly complex situations. The first two show how the speaker, given a simple sort of information about the listener's values and expectations, can volunteer information which should produce a realistic object evaluation according to the listener's standards – or, if the speaker so desires, an evaluation which is either positively or negatively biased. (Note that non-cooperative motivation is seldom dealt with in research on dialog systems.) If the listener is aware of the values and expectations the speaker ascribes to him/her, s/he can go beyond the literal meaning of an utterance by considering alternative utterances that the speaker *would* have produced if certain things had been true about the object. This *pragmatic interpretation* of dialog contributions can greatly increase the efficiency of communication; but it can yield incorrect inferences if the listener's image of the speaker is not entirely accurate - e.g. if the listener does not recognize that the speaker is biased. By choosing his/her dialog contributions carefully, the speaker can correct a discrepancy between his/her image and his/her actual criteria for generating dialog contributions - or, if s/he desires, intentionally create and/or maintain such a discrepancy. The task of creating and maintaining an image becomes more complicated in a situation where the speaker is not entirely free to decide what aspects of the object to mention but must respond to questions of the listener.

The most complex situations considered by Jameson show that there exist two sorts of uncertainty concerning the image the speaker is projecting. First, the listener will often be quite uncertain about what evaluation standards the speaker ascribes to him. In some cases (though not all), the result is that the listener will be less inclined to make pragmatic interpretations of what the speaker says. Second, the speaker may of course be quite uncertain about the assumptions the listener is making about him/her. This uncertainty needn't cause problems as long as it is matched by a similar degree of uncertainty on the part of the listener; but if the listener has definite assumptions about the speaker's model of him/her and the speaker does not know what these are, the speaker will often not know what s/he can say without risking puzzlement and misunderstanding.

The most general implication of this chapter for practical user modeling is that it is not sufficient to devise techniques for making and exploiting correct inferences about the user. The system (and its designer, respectively) must also anticipate the specific inferences that the user will make on the basis of the system's behavior (a number of methods by which the system can do this are discussed in [WAHL*] under the heading 'anticipation feedback'). Jameson presents potential methods of conveying to the user a realistic image of the system's user model and its manner

of exploitation – or at least an image which is sufficiently indefinite to discourage the user from misinterpreting the system's behavior.

In the final chapter of this part, **Robin Cohen** and **Marlene Jones** explore the possible role of a user model in an envisaged large-scale expert system for educational diagnosis which is to support the assessment of school students' learning difficulties. Such an assessment, in general, includes an analysis of the student's developmental history, medical and behavioral information and, above all, the result of a number of educational tests concerning both general abilities and achievement in areas in which the student's performance is deemed to be inadequate. The system should be able to report and explain its results to widely different users, including experienced psychologists, parents, resource room teachers, regular classroom teachers, principals, etc.

Cohen and Jones investigate how a user model can assist in adapting system responses to the background knowledge and the goals of these diverse user groups. They propose that, in order to be able to do so, the user model must contain assumptions about the user's domain knowledge (including knowledge about diagnostic procedures, test interpretation rules and possible remedial activities); about his/her familiarity with the student's background (family facts, medical history) and test results concerning the student as well as their interpretation by the system; and about the user's goals and preferences, both on a global and on a speech act level. A simple stereotype approach (cf. [RICH*]) is not sufficient for modeling all these assumptions since background knowledge, for instance, might vary considerably within the same user group. An individual user model must therefore also be constructed (which possibly supplements and overrides the selected stereotype).

Such an envisaged user model would enable the diagnosis system to not just *answer* user questions, but also to add relevant information that the user might not know (e.g. about the student's background), but which is necessary for him/her to understand how the system arrived at this answer. In explanations, on the other hand, the system can leave out facts which the user can be assumed to know already. A model of the user's goals and preferences is necessary, for example, in order to take into account his/her bias with respect to what tests to administer (e.g. psychologist: those with which s/he is familiar; principal: those which can be used in funding formulae; parents: as few as possible to minimize stress). Cohen and Jones also outline possible enhancements of their model, particularly with respect to the recognition and treatment of user plans and user misconceptions, and with respect to incompatible appraisals of specific psychological methods.

8 The Use of Explicit User Models in a Generation System for Tailoring Answers to the User's Level of Expertise [1]

Cécile L. Paris

Abstract

A question-answering program providing access to a large amount of data will be most useful if it can tailor its answers to each individual user. In particular, a user's level of knowledge about the domain of discourse is an important factor in this tailoring if the answer provided is to be both informative and understandable to the user. In this research, we address the issue of how the user's domain knowledge can affect an answer. By studying texts, we found that the user's level of domain knowledge affected the kind of information provided and not just the amount of information, as was previously assumed. Depending on the user's assumed domain knowledge, a description can be either parts-oriented or process-oriented. Thus the user's level of expertise in a domain can guide a system in choosing the appropriate facts from the knowledge base to include in an answer. We propose two distinct descriptive strategies that can be used in a question-answering program, and show how they can be mixed to include the appropriate information from the knowledge base, given the user's domain knowledge. We have implemented these strategies in TAILOR, a computer system that generates descriptions of devices. TAILOR uses one of the two discourse strategies identified in texts to construct a description for either a novice or an expert. It can merge the strategies automatically to produce a wide range of different descriptions to users who fall between the extremes of novice or expert, without requiring an a priori set of user stereotypes.

1. Introduction

The tailoring of answers according to a person's domain knowledge frequently occurs in human communication. For example, an explanation of how a car engine works aimed at a child will be different than one aimed at an adult, and an explanation adequate for a music student is probably too superficial for a student in mechanical engineering. We have found further evidence of this phenomenon in nat-

[1] This research was supported in part by the Defence Advanced Research Projects Agency under contract N00039-84-C-0165, and the National Science Foundation grant ISI-84-51438.

urally occurring texts [COLL62, BRIT63, NEW67, WEIS73, CHEV78, ENCY82, BRIT84], where the information presented to readers varies with their assumed level of general knowledge.

Likewise, a question answering program that provides access to a large amount of data to many different users could be more effective if it could customize its answers to each user, retrieving from its knowledge base the facts that are most appropriate and useful for a given user. Much research to date has focused on tailoring an answer depending on a user's goals, but customizing an answer depending on what the user knows about the domain of the question has been overlooked. This is an important factor in tailoring an answer if the answer provided is to be both informative and understandable to the user. The answer should not provide information that is obvious to the user [GRIC75]. However, if the answer assumes knowledge that the user does not have, it may be very hard (if not impossible) for the user to understand the answer [WILS86, DAVS84]. In this paper we show the feasibility of incorporating the user's domain knowledge, or *level of expertise*, into a generation system and address the issue of how this factor might affect an answer.

Through an analysis of texts directed towards audiences at the two ends of the knowledge spectrum (naive and expert), we found that two distinct discourse strategies were used in texts describing objects. We postulate that the writers' choice of strategy might be based on the assumed domain knowledge of the expected readers. If so, then the reader's level of knowledge about the domain affects the *kind* of information provided as opposed to just the *amount* of information. In previous generation systems, the user's domain knowledge affected only the amount of detail provided in a response [WALL82].

In this work, we show how the two discourse strategies found in texts can be used to provide answers to users whose domain knowledge falls anywhere along a knowledge spectrum, from naive to expert. We have implemented these strategies in TAILOR, a computer system that generates descriptions of physical objects.

TAILOR uses one of the two discourse strategies identified in texts to construct a description for either a novice or an expert. It can merge the strategies automatically in a systematic way to produce a wide variety of different descriptions for users who fall between the extremes of novice and expert. This means TAILOR is able to generate descriptions to a whole range of users, rather that just for an *a priori* set of user stereotypes.

1.1. Previous Work on User Modeling in Question-Answering Programs

In studying the factors involved in tailoring the content of an answer to a user, research to date has focused mainly on the problems of inferring and using user goals, plans and beliefs [APPE82a, APPE85, CARB83, CARB88, CARB*, MCKE85c], recognizing and dealing with misconceptions [KAPS82, MCCO83, MCCO86, MCCO88, MCCO*, QUIL88, QUIL*], and superposing various stereotypes [RICH79b, RICH*]. The issue addressed here differs from these because we are not concerned with the users' goals in asking the question, nor with correcting

their view of the domain, but rather with providing an answer that is optimally informative given how much the user knows about the domain. We are not interested in *building a user model* using stereotypes (as was Rich), but in *determining an answer* based on a user model involving user types. As in [MCCO86, MCCO88], we are more concerned about using information from the user model to generate an answer than building the model itself.

While the need for a model of the user's domain knowledge in question answering systems has been noted by various researchers [LEHN77, MCKE85b], few programs have actually had one. The HAM-ANS system [HOEP84b] has a model of the user's knowledge, but this knowledge is mainly used for anaphora resolution and production. In our work, we are more interested in studying how a user's knowledge affects the *content* of an answer as opposed to its *phrasing*. Wallis and Shortliffe [WALL82], who have used the naive/expert distinction in providing an answer (or explanation), did so mainly by giving more or less detail, without addressing the issue of whether the level of detail was the only important factor to vary. The issue we confront in this work is identifying the role played by a user's level of knowledge in determining the content of an answer. The UNIX consultant (UC) [CHIN86] uses a user's knowledge level about the UNIX system to provide help to its users. UC, however, uses stereotypes for both the user and the knowledge base (set of unix commands). Stereotypes for the knowledge base include "simple," "mundane" and "complex." UC matches the user type against the command type to decide on the answer. In this work, we are looking at a different kind of domain, the domain of complex physical objects, in which this categorization of the knowledge base is not possible. For example, there is no reason to believe that knowing about microphones indicates more expertise about the domain of complex devices than knowing about telescopes. Another approach is thus required. Furthermore, we would like to be able to tailor answers to users whose domain knowledge level falls anywhere along a knowledge spectrum without necessarily having to classify users in several different stereotypes. Finally, we are more concerned in this work with exploiting the user model whereas Chin is concerned with building it.

1.2. The Domain

In our work, we are mainly concerned with describing complex devices such as telescopes, telephones and disk drives to users with different levels of expertise. Our choice of domain has been motivated by RESEARCHER, a program being developed at Columbia University. RESEARCHER reads, remembers, and generalizes from patent abstracts written in English [LEBO83, LEBO85, LEBO86]. The resulting knowledge base is organized in a generalization hierarchy. The abstracts describe complex devices in which spatial and functional relations are important. In this domain, the amount of information contained in the knowledge base is very large and the information can be very detailed. Moreover, the knowledge base contains several different kinds of information: spatial, functional, and attributive (properties attached to objects). A program can choose from among facts representing different kinds of information about an object, and facts at different levels of detail in the knowledge base, making the decision process a complicated one.

A request for the description of an object cannot be translated into a simple database query and thus cannot be answered by a straightforward retrieval from the knowledge base [TENN78, MCKE85b]. There are no clear constraints on what information should be included in the answer. Since the amount of information contained in the knowledge base is very large and the information very detailed, a program cannot just state all the facts contained in the knowledge base about the object as there will typically be too many. Rather, it needs to select a subset of facts to present to the user. As the answer will be composed of several facts, a generation program needs to organize these facts in order to construct from them a coherent text [MCKE85b]. When a generation system can choose among many facts, a user model representing what the user presumably knows about the domain can guide the system in choosing facts that the user understands and does not already know (and cannot easily infer), thereby improving the resulting answer.

Descriptions are important because they can be used to answer other types of questions. For example, to compare two objects, it may be necessary to describe each of them. Furthermore, with a knowledge base of physical objects, users are likely to request descriptions. Generating descriptions is thus a good first step towards developing a fuller question-answering program.

In order to focus on how the level of expertise affects a description, we have not considered how the goal of the questioner could affect the description. It is clear that in a sophisticated question-answering program the user goal should also play an important part. An answer for a user whose goal is to buy an object should include different kinds of information than an answer for a user who wants to repair this object. Detecting and using the user's goal to provide an appropriate response has been the focus of extensive research [APPE85, CARB83, MCKE85c]. In this work, being more concerned with the role played by the user's domain knowledge, we simply assume that users want to find some information about an object. A description should provide meaningful information about an object and allow the user to build a mental functional model of the object. Therefore, we assume that the goal of a description is to help the user construct a mental functional model of the object under consideration.

2. Identifying What Needs to Be in the User Model

Our goal is to provide a characterization of the role of the user's domain knowledge in generating descriptions that is computationally usable by a generation system. Even though we will not be addressing the problem of how to *determine* how much the user knows about the domain, we still have to ask what *kinds* of knowledge a user can possess about a domain that can affect generation and that can be explicitly represented in a user model. Instances of these kinds of knowledge will be the information contained in our user model. Having identified what needs to be in the user model, we will take the user model as given, and study how a system can use the information about a user's domain knowledge contained in the user model to tailor the answer to each user.

Knowledge is multidimensional, in the sense that it is possible to be knowledgeable about a domain along several dimensions, such as historical, economical or functional. In identifying the kinds of knowledge a user might have, however, we restrict ourselves to the types of knowledge that are explicitly represented in terms of our knowledge base.

Analysis of natural language texts suggests the existence of at least two kinds of domain knowledge that affect the type of descriptions that can be provided in our domain:

- *knowledge about specific items in the knowledge base.* We define "knowing" about an object to mean knowing about the existence of the object, its purpose and how this purpose is achieved (that is, how the various subparts of the object work together to achieve it). Knowing about an object thus means understanding the functionality of the object and the mechanical processes associated with it.
- *knowledge about various basic underlying concepts.* In a domain of complex physical objects, such concepts include electricity and voltages.

We define an *expert* user as one whose knowledge about the domain includes functionality of objects and mechanical processes. An expert user knows all the underlying basic concepts and the majority of the generalized objects contained in the knowledge base for a particular domain. An expert user does not necessarily know about *all* the objects contained in the knowledge base, however, and thus might ask questions about objects he or she does not know about. Given an object that is new but similar to a known one, an expert user has enough domain knowledge to infer how the parts of this new object work together to perform a function. For example, the expert user might already know about particular instances of this object, or, on the other hand, about generalizations of the object. A *naive* user is one who does not know about specific objects in the knowledge base and does not necessarily understand the underlying basic concepts.

A user is not necessarily naive or expert, however. The level of expertise is really a continuum, with most users falling along this knowledge spectrum. For example, a user may know about several objects in the knowledge base. Such users would not be considered naive, but, as there are many objects they do not know in the domain, they would not be considered experts either. In this work, instead of rating a user as having some intermediate level of domain knowledge, we use explicit parameters to indicate the user's knowledge. These parameters are a list of items in the knowledge base which are known to the user and information about whether the user understands the underlying basic concepts.

Figure 1 shows examples of user models in TAILOR. In (a), the user has local expertise about the microphone and understands the concept of electricity. Lack of expertise is assumed about other objects and concepts. In (b), the user is a naive user, and the user model is empty. In (c), the user has local expertise about both the microphone and the telephone. The concepts of electricity and magnetism are also understood. We retain the terminology *naive* and *expert*, for users at the two ends of the knowledge spectrum, only as a short-hand notation.

Our emphasis is on studying how object descriptions can be varied when the parameters contained in the user model vary. We will not try to categorize a user's

(a) The user has local expertise about the microphone and understands the concept of electricity:
Local expertise: *microphone*
Basic concepts: *electricity*

(b) The user is a ⊏ naive user:
Local expertise: *nil*
Basic concepts: *nil*

(c) The user has local expertise about the microphone and the telephone and understands the concepts of electricity and magnetism:
Local expertise: *microphone, telephone*
Basic concepts: *electricity, magnetism*

Figure 1. Representing the user model explicitly

domain knowledge, attempt to determine what levels exist between the two extremes, or categorize a user into one of several user types. By explicitly representing elements of a user's knowledge, we gain the flexibility needed to tailor descriptions to specific users falling anywhere along the knowledge spectrum, without requiring a predefined set of stereotypes.

3. Two Descriptions Strategies Found in Texts: Constituency Schema and Process Trace

3.1. The Textual Analysis

To develop effective strategies for tailoring a description to a particular level of expertise, we began by studying descriptions in a variety of texts: adult encyclopedias [BRIT84, COLL62] and junior encyclopedias [BRIT63, NEW67, ENCY82], manuals [CHEV78, WEIS73] and high school textbooks (i.e., [BAKE57, VERW62]). This range of texts was chosen because it provided a good source of descriptions[2] and because these texts seem to address audiences at the two ends of the knowledge spectrum: naive and expert. Texts from adult encyclopedias are directed at an audience much more knowledgeable in general than the audience addressed by high school textbooks and junior encyclopedias. Likewise, the Chevrolet manual is aimed at knowledgeable users (professional mechanics) while the other manual claims to be directed towards novices. These texts thus constitute a good starting point for studying the differences between the descriptions given to naive users

[2] We hope that by choosing several sources, stylistic differences on our results are minimized. We studied about fifteen examples from each encyclopedia and textbook and a few from the manuals.

and those given to experts in a domain. We studied descriptions of devices, taking the description of the same object in all sources whenever possible. This allows for good comparisons of the texts, both in terms of content and organization. The descriptions we have studied are generally several paragraphs in length.

Given the desire to focus on how a reader's assumed domain knowledge can affect a description, encyclopedia texts have an added advantage besides providing examples of descriptions. They are directed to a general audience and people turning to them do so for a variety of reasons. Yet, they all read the same texts and thus acquire the same information, regardless of their goal for reading these texts. Encyclopedia texts are not directly aimed to help achieve particular goals. Rather, they provide general information intended to be useful to achieve any goal a reader may have. Thus an encyclopedia must provide its readers with some information about an object, without knowing or attempting to address a reader's specific goals. This property of encyclopedias gives us the opportunity to focus on how the level of expertise affects a description, without considering how the goals and specific beliefs of the reader could affect it.

We analyzed the different texts using methods developed by other researchers [HOBB78b, HOBB80, MANN84, MCKE85b], decomposing paragraphs in terms of their primitive rhetorical structure in an attempt to find consistent structures in the texts. We found that the texts fell into two groups: most of the descriptions in the adult encyclopedias entries and in the car manual for mechanics were organized around object subparts and their properties, while the descriptions in the junior entries and in the car manual for novices traced process information.

3.2. Texts from Adult Encydopedias

The texts from the adult encyclopedias and the Chevrolet manual can be characterized in terms of the discourse strategy the *constituency schema*, one of the textual structures posited in [MCKE85b]. In her work on natural language generation, McKeown studied the problems of what to say when there are many facts to choose from and how to organize a text coherently. To study this problem, she examined texts and transcripts in order to determine whether there were standard patterns of discourse structure used in naturally occurring texts. Where patterns of discourse structure could be identified for various discourse goals, these patterns could be used to guide a generation system in choosing and organizing facts to construct a text.

McKeown analyzed the texts by classifying each sentence as one of a set of rhetorical predicates. Rhetorical predicates characterize the structural purpose of sentences and have been discussed by a variety of linguists [SHEP26, GRIM75, HOBB78a]. Some examples are *constituency* (description of subparts or subtypes), *attributive* (providing detail about an entity or event) and *analogy*. McKeown found that, in the texts she studied, some combinations of predicates were more likely to occur than others, and that, for each discourse situation (such as providing a *definition*), some combination was the most frequent. McKeown encoded these standard combinations as schemas that are associated with a particular discourse situation. One of these schemas is the constituency schema which is used to describe an object

(or concept) in terms of its subparts and their properties. The constituency schema, shown in Figure 2 below, can be characterized by the four following steps:

1. [Identify the object as a member of some generic class, using the *identification* predicate][3]
2. Present the constituents of the item to be defined (subparts or sub-entities), corresponding to the *constituency* predicate
3. Present characteristic information about each constituent in turn, corresponding to the *depth-attributive* predicate
4. [Present additional information about the item to be defined, corresponding to the *attributive* predicate.]

[Identification (description of an object in terms of its superordinate)][4]

Attributive* (associating properties with an entity) / Cause-effect*

Constituency (description of subparts or subtypes.)

[Depth-identification / Depth-attributive

[Particular Illustration / Evidence]

[Comparison; Analogy]]+

[Attributive / Explanation / Analogy]

Figure 2. The constituency schema

The descriptions from the adult encyclopedias and the car manual for mechanics followed the pattern shown in this schema: the descriptions are organized around the parts of the object. An example of such a description is show in Figure 3. This entry, describing the telephone, is taken from [COLL62].

In the first sentence, the telephone is described in terms of its subparts: the transmitter, the receiver and the housing. This proposition corresponds to the constituency predicate, the second step of the schema. The first step of the schema, identification in terms of a generic class, was omitted. Following this sentence, the two main subparts are described in turn: the transmitter, in sentences 2 through 8

[3] The steps included in square brackets are optional.

[4] We are using McKeown's notation: "[]" indicate optionality, "/" indicates alternatives, "+" indicates that the item may appear 1 or more times, and "*" indicate that the item may appear 0 or more times. Finally, ";" is used to indicate that the propositions could not be clearly classified as corresponding to one predicate. We changed McKeown's schema slightly by adding the *identification* predicate as an option for the first predicate of the schema.