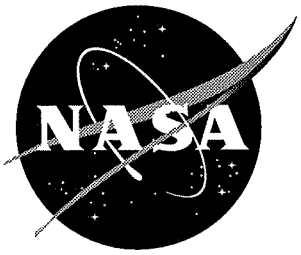
https://ntrs.nasa.gov/search.jsp?R=19990041500 2020-06-15T21:40:06+00:00Z

NASA/CP-1999-208986



Intelligent Agents and Their Potential for Future Design and Synthesis Environment

Compiled by
Ahmed K. Noor
University of Virginia
Center for Advanced Computational Technology, Hampton, Virginia

John B. Malone
Langley Research Center, Hampton, Virginia

February 1999

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart or peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

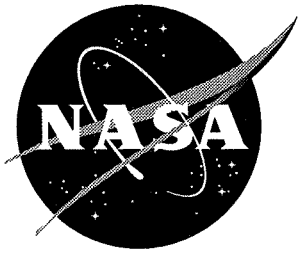
- Email your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Telephone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/CP-1999-208986



Intelligent Agents and Their Potential for Future Design and Synthesis Environment

Compiled by
Ahmed K. Noor
University of Virginia
Center for Advanced Computational Technology, Hampton, Virginia

John B. Malone
Langley Research Center, Hampton, Virginia

Proceedings of a workshop sponsored by the National Aeronautics and Space Administration and the University of Virginia Center for Advanced Computational Technology, Hampton, VA, and held at NASA Langley Research Center, Hampton, Virginia
September 16–17, 1998

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

February 1999

Notice for Copyrighted Information

This document contains material copyrighted by the party submitting it to NASA—see the copyright notice affixed thereto. It may be reproduced, used to prepare derivative works, displayed, or distributed only by or on behalf of the Government and not for private purposes. All other rights are reserved under the copyright law.

The use of trademarks or names of manufacturers in this report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

Available from the following:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Preface

This document contains the proceedings of the Workshop on Intelligent Agents and Their Potential for Future Design and Synthesis Environment, held at NASA Langley Research Center, Hampton, Virginia, Sept. 16-17, 1998. The workshop was jointly sponsored by the University of Virginia's Center for Advanced Computational Technology and NASA. Workshop attendees came from NASA, industry and universities. The objectives of the workshop were to assess the status of intelligent agents technology and to identify the potential of software agents for use in future design and synthesis environments. The presentations covered the present status of agent technology and several applications of software agents.

Certain materials and products are identified in this publication in order to specify adequately the materials and products that were investigated in the research effort. In no case does such identification imply recommendation or endorsement of products by NASA, nor does it imply that the materials and products are the only ones or the best ones available for this purpose. In many cases equivalent materials and products are available and would probably produce equivalent results.

Ahmed K. Noor
Center for Advanced Computational Technology
University of Virginia
Hampton, Virginia

John B. Malone
NASA Langley Research Center
Hampton, Virginia

Page intentionally left blank

Contents

Page intentionally left blank

Attendees

1. Dr. Khaled Abdel-Tawab
   Center for Advanced Computational Technology/Univ. of Virginia
   Mail Stop 201
   NASA Langley Research Center
   Hampton, VA 23681
   (757) 864-1992; Fax (757) 864-8089
   Email:k.i.abdel-tawab@larc.nasa.gov

2. Ms. Danette Allen
   Mail Stop 488
   NASA Langley Research Center
   Hampton, VA 23681
   (757) 864-7364; Fax (757) 864-7944
   Email: b.d.allen@larc.nasa.gov

3. Dr. Richard R. Antcliff
   Mail Stop 254
   NASA Langley Research Center
   Hampton, VA 23681
   (757) 864-4606; Fax (757) 864-1707
   Email: r.r.antcliff@larc.nasa.gov

4. Dr. Manuel Aparicio
   IBM, Intelligent Agent Services
   P.O. Box 12195
   4205 South Miami Blvd.
   Research Triangle Park, NC 27709
   (919) 254-7622
   Email: aparicio@us.ibm.com

5. Mr. Don E. Avery
   Mail Stop 367
   NASA Langley Research Center
   Hampton, VA 23681
   (757) 864-1947; Fax (757) 864-4449
   Email: d.e.avery@larc.nasa.gov

6. Prof. William P. Birmingham
   Electrical Engineering and Computer Science Department
   The University of Michigan
   1101 Beal Avenue
   Ann Arbor, MI 48109
   (734) 936-1590; Fax (734) 763-1260
   Email: wpb@eecs.umich.edu

7. Prof. David C. Brown
   Dept. of Computer Science
   Worcester Polytechnic Institute
   100 Worcester Road
   Worcester, MA 01609
   (508) 831-5618; Fax (508) 831-5776
   Email: dcb@cs.wpi.edu

8. Mr. Dennis M. Bushnell
   Mail Stop 110
   NASA Langley Research Center
   Hampton, VA 23681
   (757) 864-8987; Fax (757) 864-8980
   Email: d.m.bushnell @larc.nasa.gov

9. Ms. Abigail C. Chapin
   Computer Science Department
   University of Virginia
   Charlottesville, VA 22903
   (804) 970-5096
   Email: acc2a@virginia.edu

10. Mr. Adam J. Cheyer
    SRI International
    333 Ravenswood Ave. EJ217
    Menlo Park, CA 94306
    (650) 859-4119; (650) 859-3735
    Email: cheyer@ai.sri.com

11. Mr. Simon S. Chung
    Mail Stop 157A
    NASA Langley Research Center
    Hampton, VA 23681
    (757) 864-7337
    Email: s.s.chung@larc.nasa.gov

12. Dr. Thomas M. Eidson
    High Technology Corporation
    28 Research Drive
    Hampton, VA 23666
    (757) 865-0818; Fax (757) 865-6766
    Email: teidson@htc-tech.com

13. Mr. Carl R. Elks
    Mail Stop 130
    NASA Langley Research Center
    Hampton, VA 23681
    (757) 864-2078
    Email: c.r.elks@larc.nasa.gov

14. Mr. Travis Emmitt
    University of Virginia
    1734 Franklin Drive
    Charlottesville, VA 22911
    (804) 984-2706
    Email: emmitt@virginia.edu

15. Mr. Yamir E. Encarnacion
    1301 North Courthouse Road #1209
    Arlington, VA 22201
    (703) 527-6567
    Email: yamir@worldnet.att.net

16. Prof. Tim Finin
Dept. of Computer Science and Electrical Engineering
Univ. of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
(410) 455-3522; Fax (410) 455-3969
Email: finin@cs.umbc.edu

17. Dr. Stanley Franklin
Institute for Intelligent Systems
Math Sciences, Dunn Bldg. #373
University of Memphis
Memphis, TN 38152
(901) 678-3142; Fax (901) 678-2480
Email: stan.franklin@memphis.edu

18. Mr. William T. Freeman
Mail Stop 188E
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2945; Fax (757) 864-8911
Email: w.t.freeman@larc.nasa.gov

19. Dr. George G. Ganoe
Mail Stop 328
NASA Langley Research Center
Hampton, VA 23681
(757) 864-1940; Fax (757) 864-1975
Email: g.g.ganoe@larc.nasa.gov

20. Ms. Thea M. Ganoe
Center for Advanced Computational Technology/Univ. of Virginia
Mail Stop 201
NASA Langley Research Center
Hampton, VA 23681
(757) 864-8391; Fax (757) 864-8089
Email: t.m.ganoe@larc.nasa.gov

21. Mr. Brantley R. Hanks
Mail Stop 367
NASA Langley Research Center
Hampton, VA 23681
(757) 864-4322; Fax (757) 864-4449
Email: b.r.hanks@larc.nasa.gov

22. Dr. Charles E. Harris
Mail Stop 188M
NASA Langley Research Center
Hampton, VA 23681
(757) 864-3447; Fax (757) 864-7729
Email: c.e.harris@larc.nasa.gov

23. Prof. James Hendler
Computer Science Department
University of Maryland
College Park, MD 20742
(301) 405-2696; Fax (301) 405-8488
Email: hendler@cs.umd.edu

24. Dr. Jerrold M. Housner
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2907; Fax (757) 864-8912
Email: j.m.housner@larc.nasa.gov

25. Prof. Michael N. Huhns
Dept. of Electrical and Computer Engineering
University of South Carolina
301 South Main Street
Columbia, SC 29208
(803) 777-5921; Fax (803) 777-8045
Email: huhns@sc.edu

26. Mr. Stephen C. Irick
Mail Stop 432
NASA Langley Research Center
Hampton, VA 23681
(757) 864-7078; Fax (757) 864-7009
Email: s.c.irick@larc.nasa.gov

27. Dr. Anupam Joshi
Dept. of Computer Science and Electrical Engineering
Univ. of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
(410) 455-2590; Fax (410) 455-3969
Email: joshi@cs.umbc.edu

28. Prof. Larry Kerschberg
ISE Dept., MSN4A4
George Mason University
4400 University Drive
Fairfax, VA 22032
(703) 993-1661; Fax (703) 993-1638
Email: kersch@gmu.edu

29. Ms. Kara A. Latorella
Mail Stop 152
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2030; Fax (757) 864-7793
Email: k.a.latorella@larc.nasa.gov

30. Mr. Marc Latorella
Penn State University
Mail Stop 152
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2030
Email: mdl135@psu.edu

31. Mr. Henry A. Lieberman
Media Lab.
Massachusetts Inst. of Technology
20 Ames Street 305 A
Cambridge, MA 02139
(617) 253-0315; Fax (617) 253-6215
Email: lieber@media.mit.edu

32. Mr. Kenneth N. Lodding
Computer Sciences Corporation
3217 North Armistead Avenue
Hampton, VA 23666
(757) 766-8244; Fax (757) 766-2571
Email: k.n.lodding@larc.nasa.gov

33. Ms. Christine G. Lotts
Mail Stop 238
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2911
Email: c.g.lotts@larc.nasa.gov

34. Dr. Kwan-Liu Ma
ICASE
Mail Stop 403
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2195; Fax (757) 864-6134
Email: kma@icase.edu

35. Dr. Moinuddin Malik
Center for Advanced Computational
Technology/Univ. of Virginia
Mail Stop 201
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2492; Fax (757) 864-8089
Email: mmalik@puma.larc.nasa.gov

36. Mr. Brian H. Mason
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-4895
Email: b.h.mason@larc.nasa.gov

37. Dr. Neal N. McCollom
Lockheed Martin Tactical
Aircraft Systems
Mail Zone 2275
P.O. Box 748
Fort Worth, TX 76108
(817) 763-3378; Fax (817) 763-3689
Email: neal.n.mccollom@lmco.com

38. Mr. Alan R. McCoy
Center for Advanced Computational
Technology/Univ. of Virginia
Mail Stop 201
NASA Langley Research Center
Hampton, VA 23681
(757) 864-8518; Fax (757) 864-8089
Email: a.r.mccoy@larc.nasa.gov

39. Dr. Mark A. Motter
Mail Stop 152
NASA Langley Research Center
Hampton, VA 23681
(757) 864-6978; Fax (757) 864-7793
Email: m.a.motter@larc.nasa.gov

40. Dr. Vivek Mukhopadhyay
Mail Stop 159
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2835; Fax (757) 864-9713
Email:
v.mukhopadhyay@larc.nasa.gov

41. Mr. Michael L. Nelson
Mail Stop 158
NASA Langley Research Center
Hampton, VA 23681
(757) 864-8511; Fax (757) 864-8342
Email: m.l.nelson@larc.nasa.gov

42. Prof. Ahmed K. Noor
Director, Center for Advanced
Computational Technology
University of Virginia
Mail Stop 201
NASA Langley Research Center
Hampton, VA 23681
(757) 864-1978; Fax (757) 864-8089
Email: a.k.noor@larc.nasa.gov

43. Dr. Peter Norvig
Mail Stop 269-1
NASA Ames Research Center
Moffett Field, CA 94035
(650) 604-6207
Email: pnorvig@mail.arc.nasa.gov

44. Dr. F. G. Patterson, Jr.
NASA Headquarters
Code FT
300 E Street, S.W.
Washington, D.C. 20546
(202) 358-2171; Fax (757) 358-4164
Email: pat.patterson@hq.nasa.gov

45. Mr. Daniel L. Page
Computer Sciences Corporation
Mail Stop 157B
NASA Langley Research Center
Hampton, VA 23681
(757) 864-9361
Email: d.l.page@larc.nasa.gov

46. Ms. Jeanne M. Peters
Center for Advanced Computational Technology/Univ. of Virginia
Mail Stop 201
NASA Langley Research Center
Hampton, VA 23681
(757) 864-1989; Fax (757) 864-8089
Email: j.m.peters@larc.nasa.gov

47. Ms. Elizabeth B. Plentovich
Mail Stop 261
NASA Langley Research Center
Hampton, VA 23681
(757) 864-1919; Fax (757) 864-8093
Email: e.b.plentovich@larc.nasa.gov

48. Dr. Alan T. Pope
Mail Stop 152
NASA Langley Research Center
Hampton, VA 23681
(757) 864-6642; Fax (757) 864-7793
Email: a.t.pope@larc.nasa.gov

49. Mr. Joe Rehder
Mail Stop 139
NASA Langley Research Center
Hampton, VA 23681
(757) 864-4481; Fax (757) 864-9715
Email: j.j.rehder@larc.nasa.gov

50. Mr. James L. Rogers
Mail Stop 159
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2810; Fax (757) 864-9713
Email: j.l.rogers@larc.nasa.gov

51. Ms. Andrea O. Salas
Mail Stop 159
NASA Langley Research Center
Hampton, VA 23681
(757) 864-5790; Fax (757) 864-9713
Email: a.o.salas@larc.nasa.gov

52. Mr. Manuel D. Salas
ICASE
Mail Stop 403
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2174; Fax (757) 864-6134
Email: salas@icase.edu

53. Dr. Chris A. Sandridge
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2816
Email: c.a.sandridge@larc.nasa.gov

54. Mr. William J. Seufzer
College of William and Mary, and Computer Sciences Corp.
Mail Stop 157D
NASA Langley Research Center
Hampton, VA 23681
(757) 864-9014
Email: w.j.seufzer@larc.nasa.gov

55. Mr. David W. Sleight
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-8427
Email: d.w.sleight@larc.nasa.gov

56. Ms. Kathryn Stacy
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681
(757) 864-6719; Fax (757) 864-8910
Email: k.stacy@larc.nasa.gov

57. Dr. Olaf O. Storaasli
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2927; Fax (757) 864-8912
Email: o.o.storaasli@larc.nasa.gov

58. Dr. Z. Peter Szewczyk
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2913; Fax (757) 864-8912
Email: z.p.szewczyk@larc.nasa.gov

59. Mr. Michael D. Uenking
Mail Stop 152
NASA Langley Research Center
Hampton, VA 23681
(757) 864-6954; Fax (757) 864-7793
Email: m.d.uenking@larc.nasa.gov

60. Mr. James A. Villani
Logistics Management Institute
2000 Corporate Ridge
McLean, VA 22102
(703) 917-7304; Fax (703) 917-7198
Email: jvillani@lmi.org

61. Dr. John T. Wang
Mail Stop 240
NASA Langley Research Center
Hampton, VA 23681
(757) 864-8185; Fax (757) 864-8912
Email: j.t.wang@larc.nasa.gov

62. Dr. Tamer M. Wasfy
Center for Advanced Computational Technology/Univ. of Virginia
Mail Stop 201
NASA Langley Research Center
Hampton, VA 23681
(757) 864-1984; Fax (757) 864-8089
Email: t.wasfy@larc.nasa.gov

63. Ms. Shahani M. Weerawarana
Dept. of Computer Sciences
Purdue University
3074 High Ridge Road
Yorktown Heights, NY 10598
(914) 243-0186
Email: markus@cs.purdue.edu

64. Mr. Earl R. Wingrove
Logistics Management Institute
2000 Corporate Ridge
McLean, VA 22102
(703) 917-7387; Fax (703) 917-7066
Email: ewingrov@lmi.org

65. Mr. Robert E. Yackovetsky
AST Program Office
NASA Langley Research Center
Hampton, VA 23681
(757) 864-3894
Email:
r.e.yackovetsky@larc.nasa.gov

66. Mr. Long P. Yip
Mail Stop 254
NASA Langley Research Center
Hampton, VA 23681
(757) 864-3866; Fax (757) 864-1707
Email: l.p.yip@larc.nasa.gov

67. Dr. Thomas A. Zang
Mail Stop 159
NASA Langley Research Center
Hampton, VA 23681
(757) 864-2307; Fax (757) 864-9713
Email: t.a.zang@larc.nasa.gov

51-61
032515
370160
20p

Overview of Intelligent Software Agents

Ahmed K. Noor
Center for Advanced Computational Technology
University of Virginia
Hampton, VA

Page intentionally left blank

Outline

The fascination with non-human agents dates back to the beginning of recorded history. Popular notions about androids, humanoids, robots, cyborgs and science fiction creatures permeate our culture and form the backdrop against which software agents are perceived. As a result of technological advances in computer hardware, networking, communciations, and modeling and simulation, the new paradigm of parallel, distributed, collaborative and immersive computing is emerging. One of the consequences of this paradigm is a significant increase in software complexity. Hence, there is a need for intelligent software that will not only respond to requests, but anticipate, adapt and actively seek ways to support diverse, geographically dispersed teams. This presentation provides an overview of intelligent agent technology. The outline is given in Fig. 1. First, a definition of intelligent agents is given, and some of their attributes are described. The parent disciplines and the technologies are listed. A classification and some of the applications of intelligent agents are discussed. The future potential of intelligent agents and their use in the design and synthesis environment is outlined. A list of some of the information sources on intelligent agents is given.

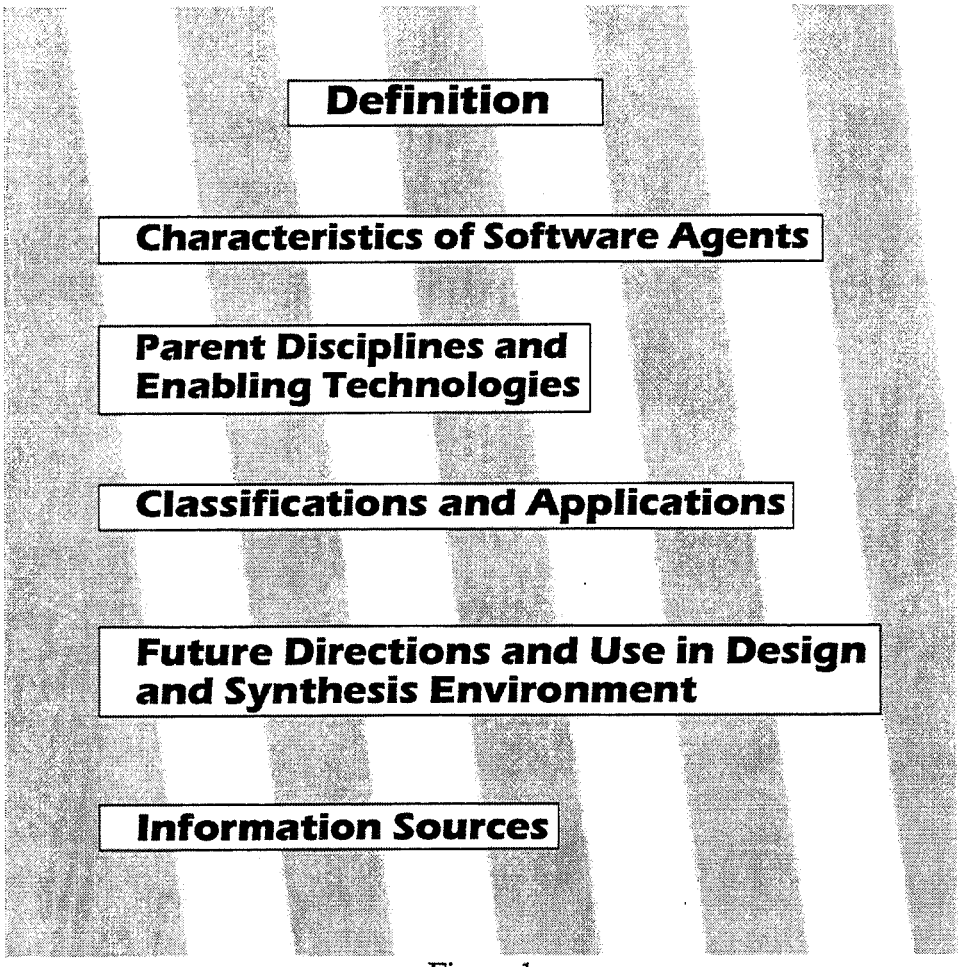


Figure 1

Definition

Due to the inter-disciplinary character of agents, it has not been possible to agree on a generally accepted, comprehensive definition of an intelligent agent. At the highest level, three major categories of agents can be distinguished: human agents, hardware agents, and software agents (Fig. 2). All agent categories have the common feature that they, to a large extent, independently perform tasks on behalf of their contracting party (or user) for which specialized knowledge is needed, or which consist of many time-intensive individual steps. The functional definition for agents used herein is: a software/hardware agent resides in an environment, uses sensors to identify certain aspects of the environment and executes commands that affect the environment. Intelligent software agents act on behalf of people, take initiatives and make suggestions. Today's passive software is referred to as software tools.

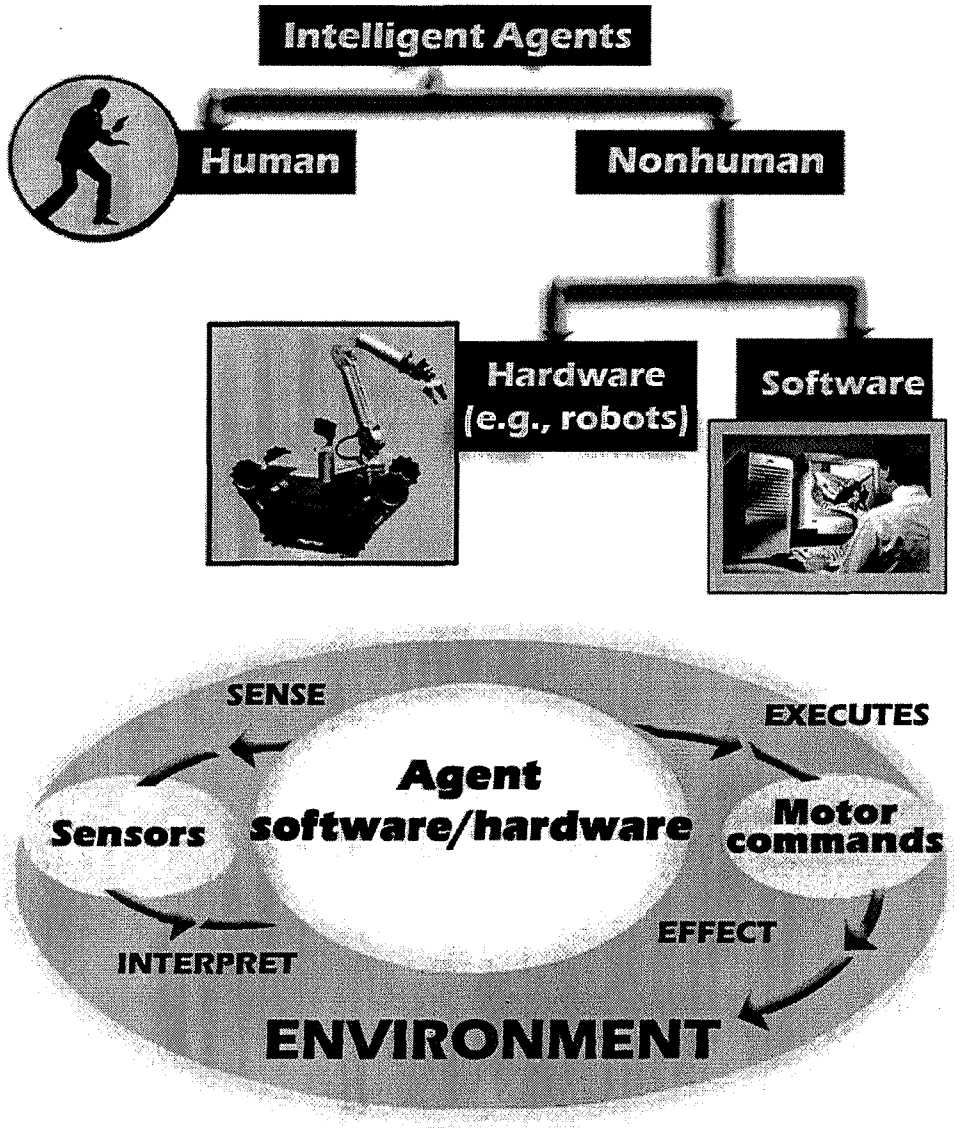


Figure 2

Characteristics of Software Agents

Some of the characteristics of a software agent are shown in Fig. 3. These are:

reactive - responds in a timely fashion to changes in the environment.

autonomous - exercises control over its own internal state. Exhibits self-starting behavior.

proactive - exhibits goal-directed behavior by taking initiative.

adaptive - changes its behavior based on previous experience.

inferentially capable - has reasoning capability based on a knowledge base and is able to learn.

mobile - able to navigate within electronic communication networks, and migrate from one host platform to another.

communicative - communicates with other agents and humans.

collaborative - has the ability to cooperate with other agents for complex tasks that exceed the capability of a single agent.

The first six characteristics are particular to the agent, and the last two pertain to its interaction with the environment.

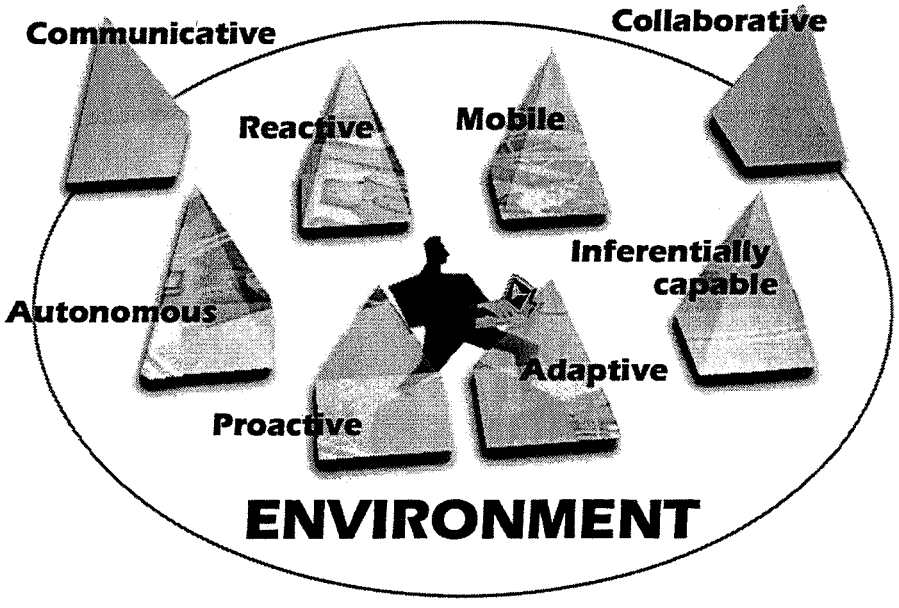


Figure 3

Mobile Agents

The concept of a mobile agent emerged from communication protocol between computers, in use since the late 1970's. It is based on remote procedure calls (RPC). Each call involves a request sent from the user to the server, and a response sent from the server to the user (Fig. 4).

An alternative to RPC is the remote programming (RP). The user computer sends to the server a mobile agent. The agent, not the user computer, orchestrates the work on the server.

Ongoing interaction in RPC requires ongoing communications. By contrast, in RP, it does not.

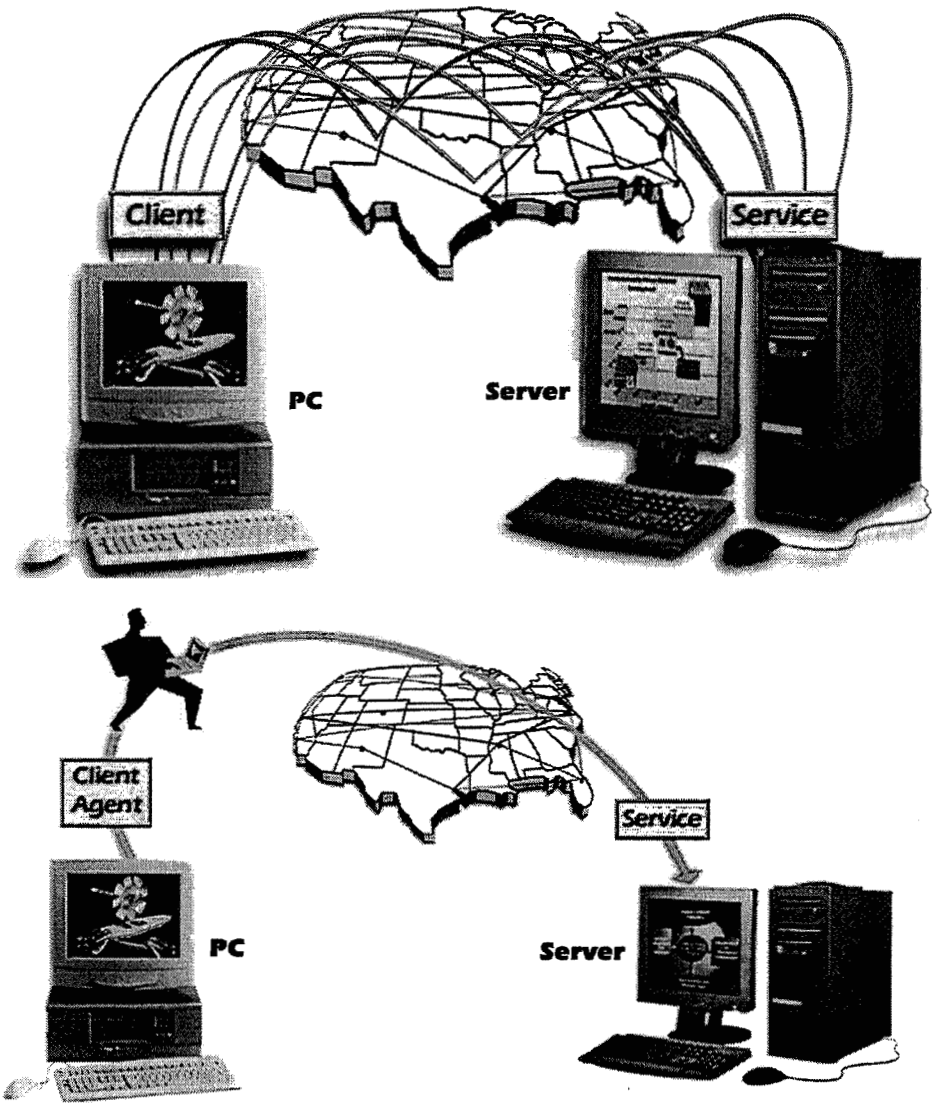


Figure 4

Parent Disciplines and Enabling Technologies

The development of agent technology is strongly influenced by three disciplines: classical AI planning systems, control theory and cybernetics, and cognitive psychology (and neuroscience). A number of technologies can facilitate the development and use of intelligent agents, namely:

- AI knowledge-based and expert system
- Object-oriented software development, and
- Soft computing.

An example of the first is the Cyc knowledge base, inference engine, and application modules, which use the CycL representation language. Agents can be structured such that their world view, communications, and internal mechanisms are based on a strong notion of modularity, avoidance of single points of failure, relative autonomy, and various types of reuse via object-oriented methodologies of design, development and execution. Soft computing provides agents with the ability to: a) reason under uncertainty, and with imprecise or incomplete data (via the use of fuzzy logic); b) discern patterns, learn and generalize (via the use of neural networks); and c) best fit or deal with the situation at hand (via the use of genetic algorithms).

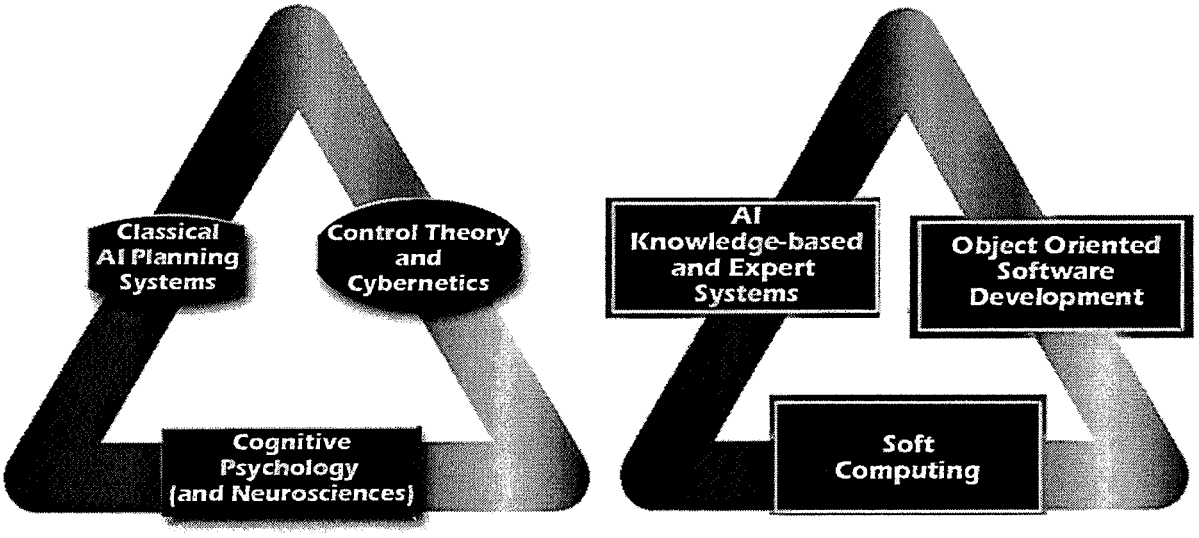


Figure 5

Intelligent Agents

Intelligent agents can be described in terms of the space defined by the three dimensions of agency, intelligence and mobility (as proposed by Gilbert, et al. at IBM).

> *Agency* is the degree of autonomy and authority vested in the agent. It can be measured by the nature of interaction between the agent and the environment it resides in. At a minimum, an agent must run asynchronously. More advanced agents interact with data, applications and other agents.
>
> *Intelligence* can be defined as the degree of reasoning and learned behavior. At a minimum, there can be a set of preferences in the user's statement of goals and the tasks delegated to the agent. Higher levels of intelligence include a user model and reasoning. Still higher levels are systems that learn and adapt to their environment, both in terms of the user's objectives, and the resources available to the agent.
>
> *Mobility* refers to the degree to which agents travel through the network. Mobil scripts may be composed on one computer and shipped to another for execution. Mobil objects are transported from computer to computer in the middle of execution and carrying accumulated state data with them. Figure 6 shows the domain of expert systems and fixed-function agents in the same three-dimensional space. Intelligent agents can be thought of as advanced proactive expert systems.

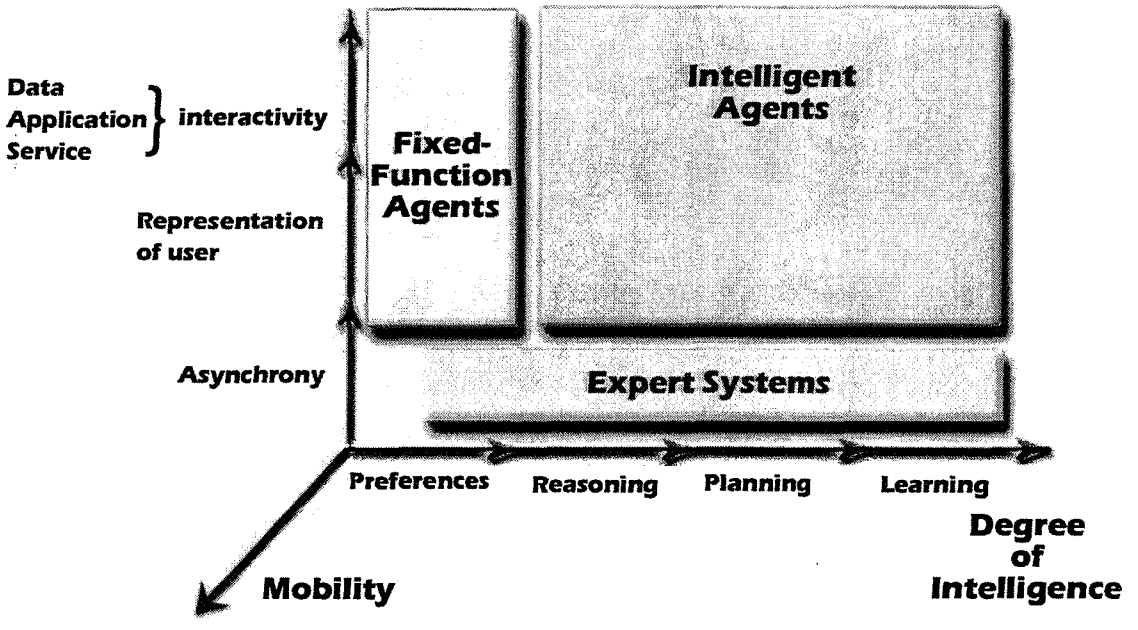


Figure 6

Technologies Used for Developing Agent Applications

Three categories of technologies are used in developing agent applications: languages, ontologies, and support computing technologies. *Languages* can be divided into object-oriented/scripting languages and agent communication languages (ACL). The latter are used for applications containing multiple agents. Several examples of the first category of languages are listed in Fig. 7. The figure also shows the degree of typing and the number of instructions that can be executed per statement for various scripting languages. An example of an ACL language is the Knowledge Query and Manipulation Language (KQML) which is an evolving standard ACL being developed as part of the DARPA Knowledge Sharing Effort (KSE). The KQML language can be viewed as consisting of three layers: the content, message and communication layers.

Ontologies describe the way in which knowledge is organized, and the vocabulary used to describe the domain's concepts. Examples of the *support computing technologies* that facilitate the design and implementation of agent applications are client-server technology, and the Common Request Broker Architecture (CORBA).

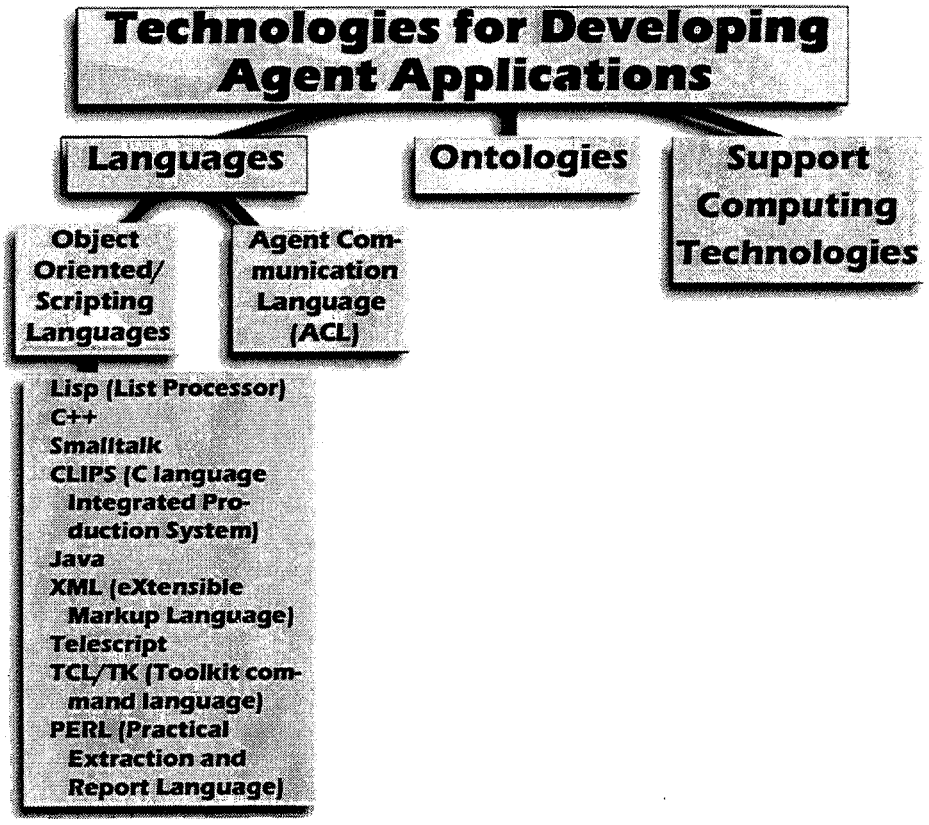


Figure 7

Agent Classification

As with the definition of an agent, several classifications have been proposed for software agents. These classifications are described herein. The first is based on the attributes exhibited by the agent. *Interface agents* are autonomous and inferentially capable. *Collaborative learning agents* are collaborative and inferentially capable, and *smart agents* are autonomous, inferentially capable and collaborative.

The second classification uses three criteria: intelligence, mobility and number of agents. With respect to degree of intelligence, agents are classified into simple and complex. The latter exhibit a highly intelligent behavior. With respect to mobility, agents are classified into stationary and mobile. The number of agents associated with a system forms the third classification criterion. Single agents are not capable of contacting other agents, even when they reside in their environment. By contrast, multi-agent systems consist of a number of agents that can communicate or even cooperate with each other.

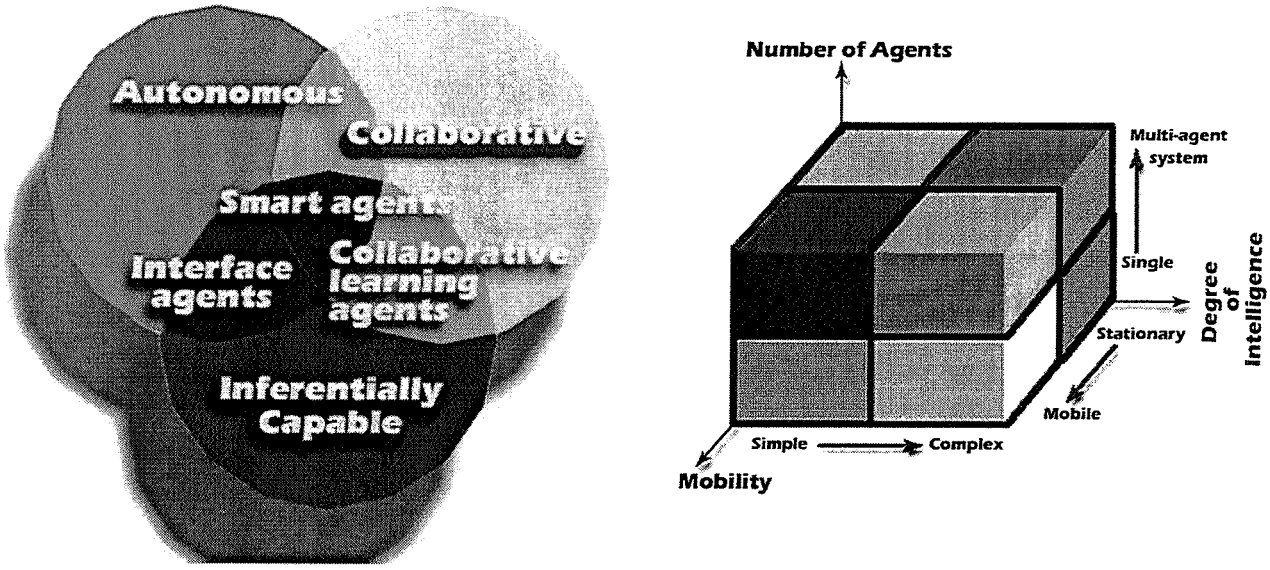


Figure 8

Agent Classification/Applications

The third classification is based on the particular applications for which the agent is used. Five categories of agents can be identified, namely:

- *Interface agents* - used for intelligent tutoring and intelligent help, as well as for workflow automation.
- *Information agents* - used for search, retrieval and filtering, as well as for advising and focusing.
- *Cooperation agents* - including meeting facilitators and management of group processes.
- *Product development/mission synthesis agents.* These include assistants and work-flow automation agents and the agents used for collaborative, distributed product development (resource selection, mediators and recommenders).
- *Computing, networking and communication agents.* These include management agents for configurable computing and active networks.

Examples of some agents in the aforementioned categories are described subsequently.

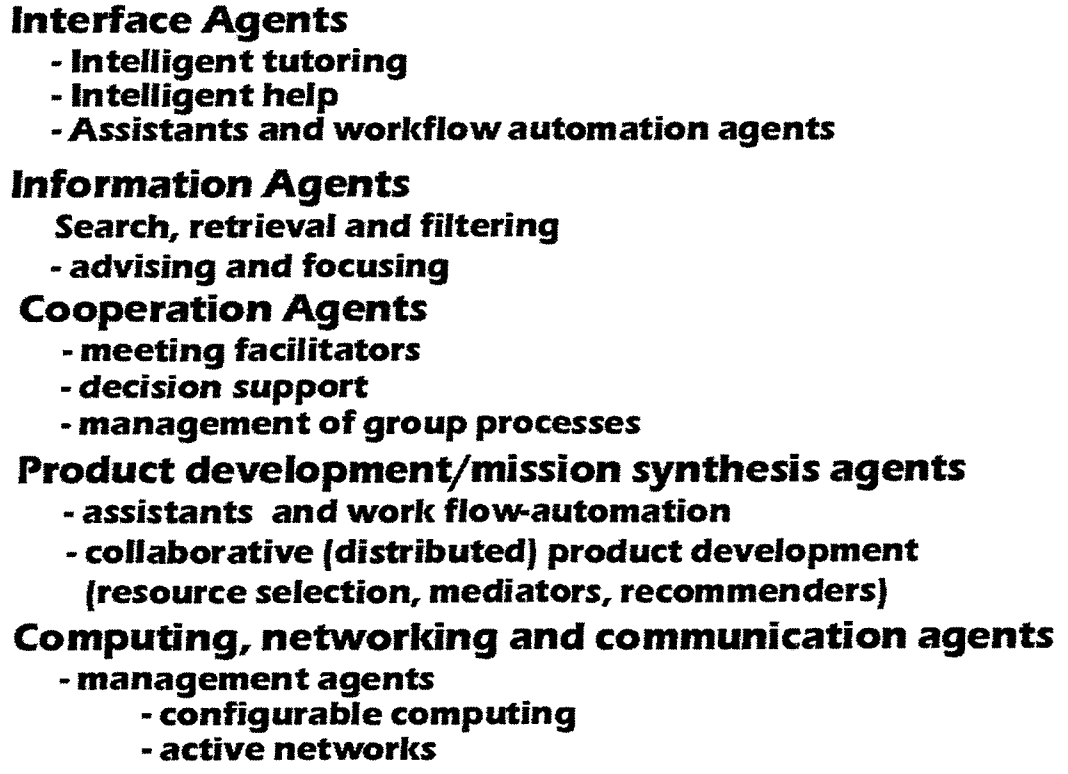


Figure 9

Intelligent Agents in Human-Computer Interfaces

Figure 10 shows the evolution of human-computer interfaces and the involvement of intelligent agents in them. During the period of the 1950's through the 1970's, static interfaces were used in the form of teletype style and full-screen text and light pen. The system designer built the interface and the user had to learn how to use it. This was followed in the 1980's and early 1990's by more flexible interfaces - windows, mouse and graphical tablet. The flexibility was restricted to simple changes (colors, size, or positions of windows). In the 1990's, windows, mouse, graphical tablets, adaptive multimedia (audio, video and animation) interfaces were introduced. The adaptation covered both the communication and functionality and included: user-initiated self adaptation, user-controlled self-adaptation, computer-aided adaptation and system initiated adaptation. The trend is now moving towards intelligent interfaces, which integrates adaptive interfaces with intelligent agents for making intelligent help and tutoring available to the user. In the future, intelligent agents will be used in adaptive/reconfigurable interfaces that take advantage of the advances made in cognitive neuroscience to couple humans with the computing facilities and hence, maximize their performance. Among the adaptive/reconfigurable interfaces are the neural interfaces which use brain waves to sense the actual state of attention and alertness of the user.

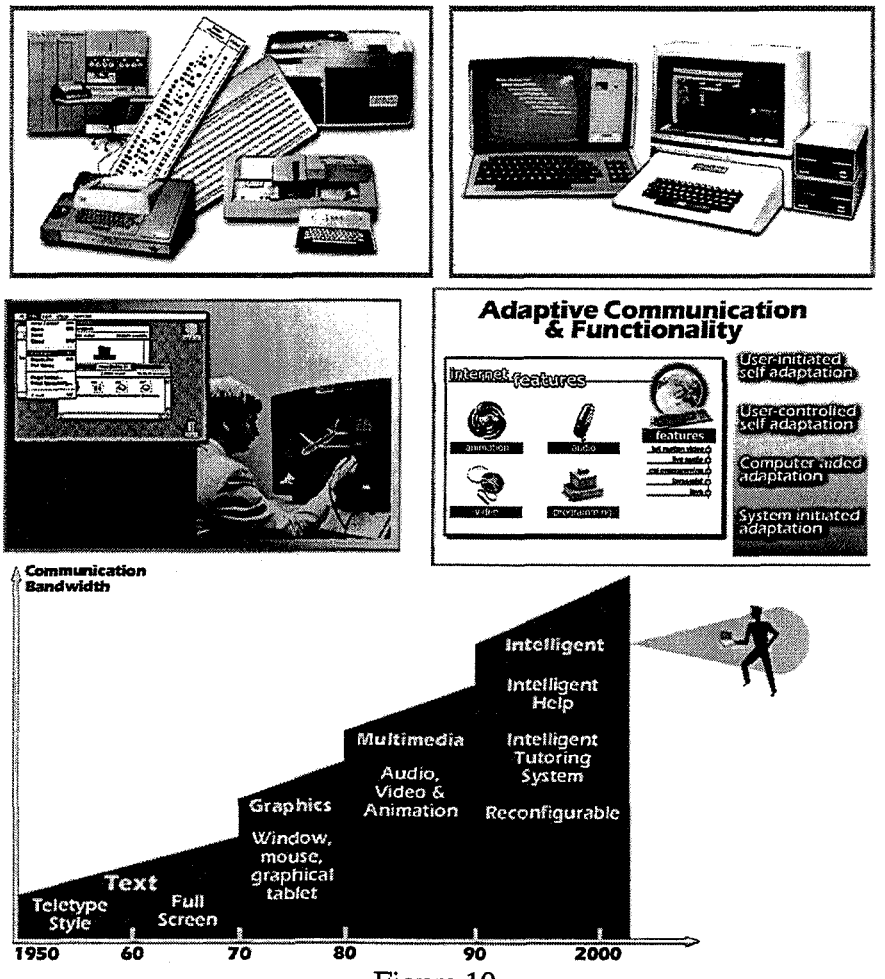


Figure 10

Intelligent Agents in Learning Technology

Figure 11 shows the evolution of learning technology and the extent of using intelligent agents. Computer-based technology (CBT) systems of the 1960's and 1970's were initially passive. Later developments in that period included learner modeling and more elaborate computer-learner interfaces. The addition of expert systems to CBT resulted in Intelligent Tutoring Systems (ITS) of the 1980's. ITS had explicit models of tutoring and domain knowledge, and were more flexible in their response than CBT systems. However, they were developed for "information transfer" and were not change-tolerant. The advent of intelligent agents, which enabled the learner to manipulate cognitive artifacts from several perspectives or viewpoints, led to the interactive learning systems (ILS) of the 1990's. Examples of these systems include on-line courses, interactive learning systems and intelligent evaluation facilities. The current trend is towards collaborative distributed learning systems extensively using intelligent agents in the learning environment. Such dynamic environments will provide the learners access to other ideas and concepts, allow them to express their viewpoints and incrementally adapt initial viewpoints to more informed and mastered concepts.

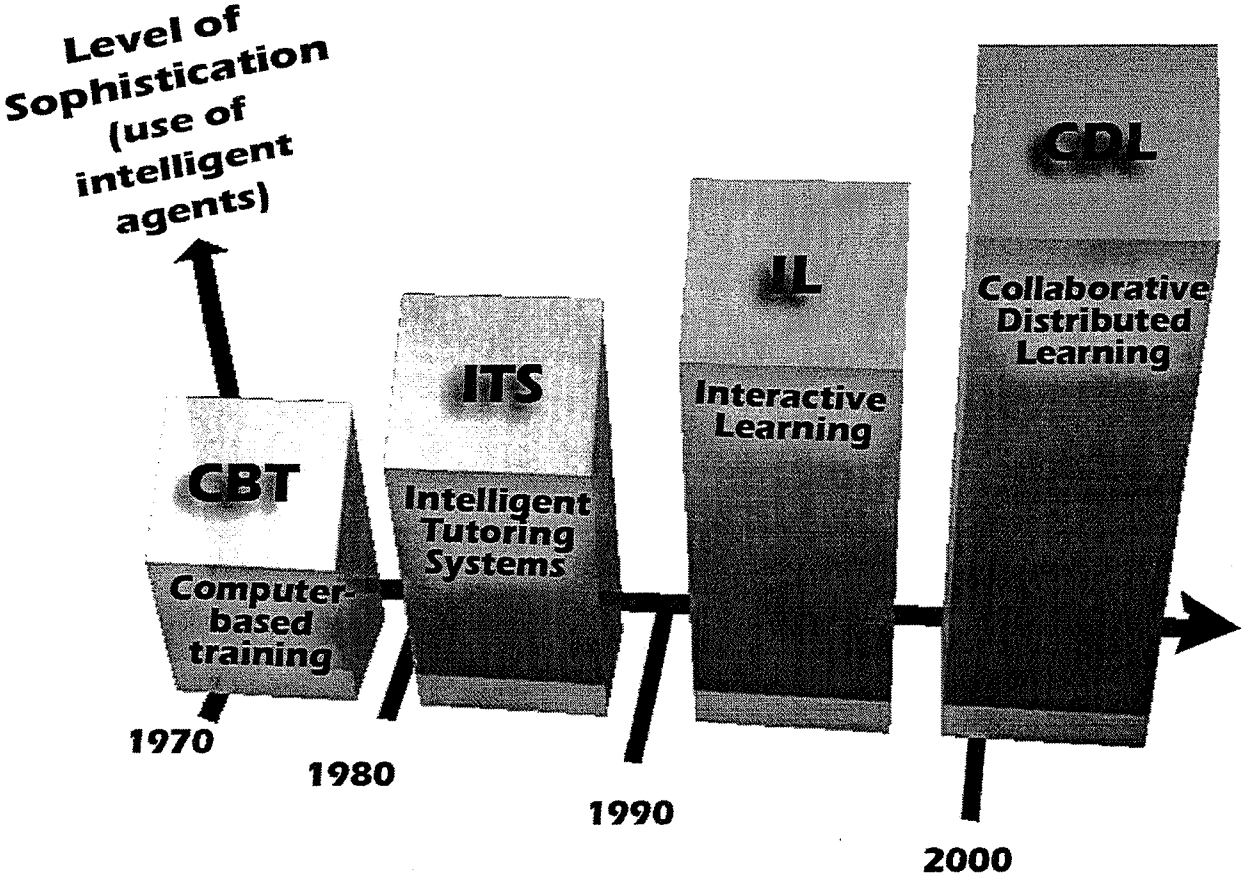


Figure 11

Intelligent Agents for Information Retrieval and Filtering

Depending on the extent of use and the degree of intelligence of the agents used, the hierarchy of search engines (shown in Fig. 12) can be identified. The simple search engines represent a low level of search tools. The intelligent agents used in these tools store all the information found in a database, which can be central or distributed. Examples of simple search engines are AltaVista, WebCrawler, Excite, HotBot, InfoSeek, Lycos and OpenText. The next set of search engines in the hierarchy are the *pseudo meta search engines*, which provide the user with known search engines that serve as a starting point in the individual search for information. An example of such a search engine is the Configurable Unified Search Engine (CUSI).

Meta search engines provide automation of the simultaneous query to several simple search engines. They provide the user with the results of the associated search query in a compressed and improved form compared with the simple search engine. Examples of meta search engines are MetaCrawler, SavvySearch, MetaGer, and Inquirus. Advances in intelligents agents and other technologies will lead to customized (personalized) search engines which provide the user with the information in a variety of formats including text, equations, images, animations, and video.

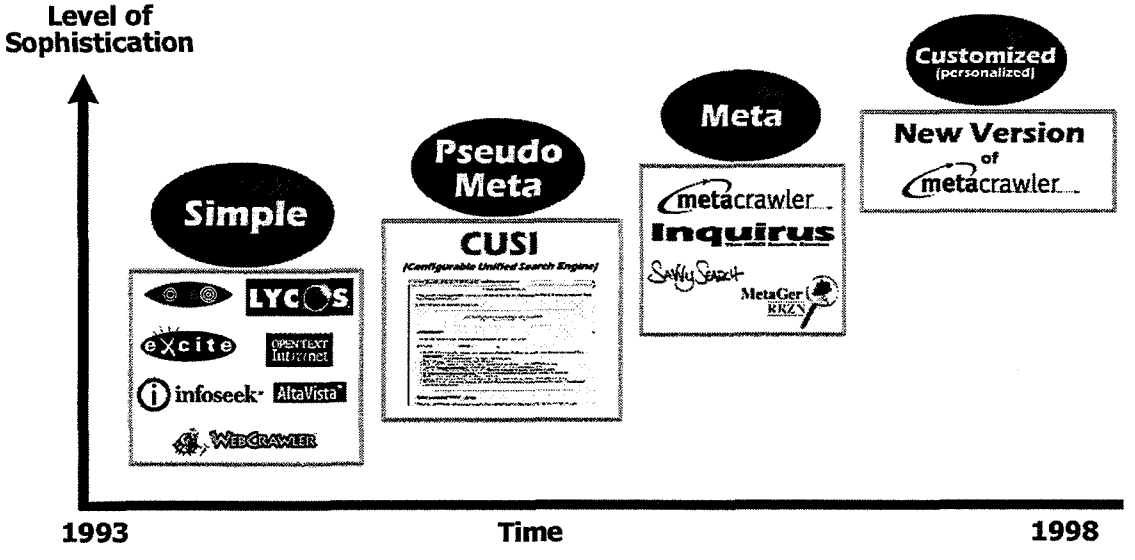


Figure 12

Intelligent Agents in CAD/CAE Systems

Figure 13 shows the evolution of CAD/CAE systems. In the 1950's, engineering design was primarily a pencil and paper activity, and the computing engine was the slide rule (used in conjunction with design manuals). The first major improvement was the development of computer-aided drafting and wire frame models. This merged CAD systems with solid modeling facilities, and subsequently, to the current virtual product systems with embedded simulation capabilities for the entire life cycle of the product. The addition of intelligent agents will transform the systems into knowledge-enriched virtual product development systems. The benefits obtained include higher productivity, better product quality, and a broader design to provide an integrated system solution.

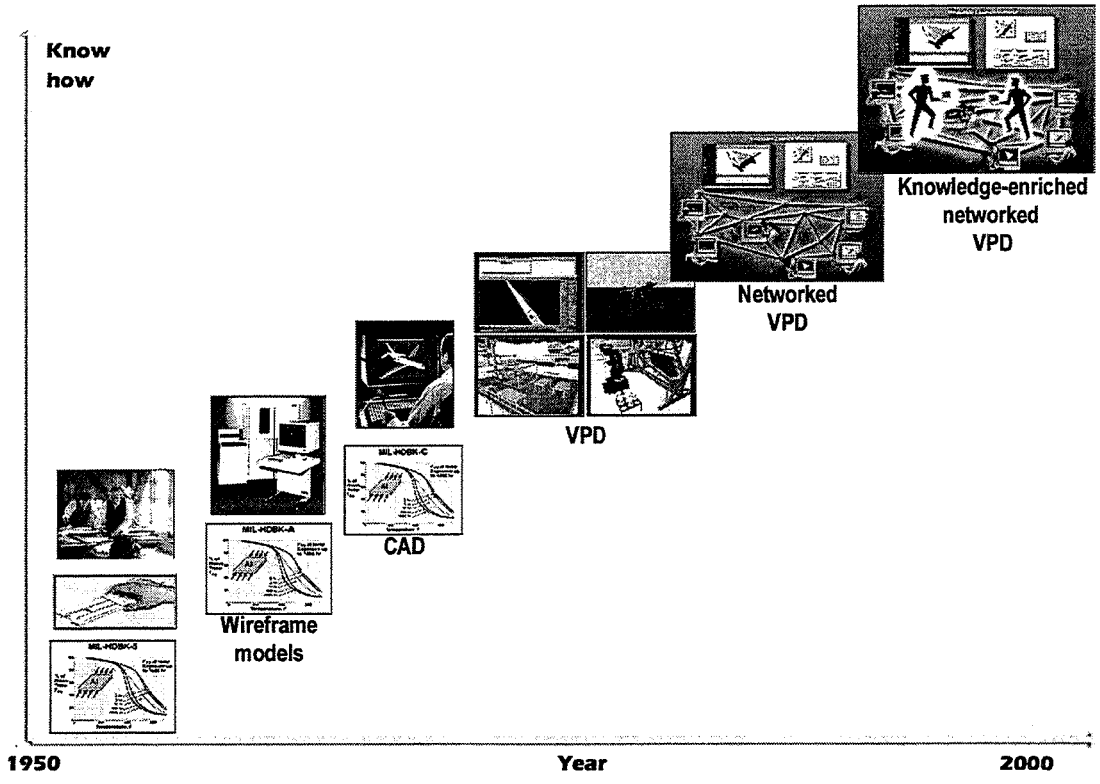


Figure 13

Future Directions

In the future, agents will become pervasive within computing and communication systems. They will form the central components of future information-based engineering systems. Specifically, software agents will:

- Have increased intelligence, flexibility and independence. Specifically, they will:
  - Exhibit inference capability and be able to reason about goals
  - Be trainable, easy to configure and program via goal statements or natural language
  - Provide unprecedented levels of functionality (via combining hierarchical multiagent systems with AI constructs and infrastructure utilities).
- Provide personalized advice and services
- Keep diverse teams informed, focused and organized
- Revolutionize collaborative engineering processes and scientific research. They will enable the development of knowledge work teams (or distributed minds – see Fig. 14).

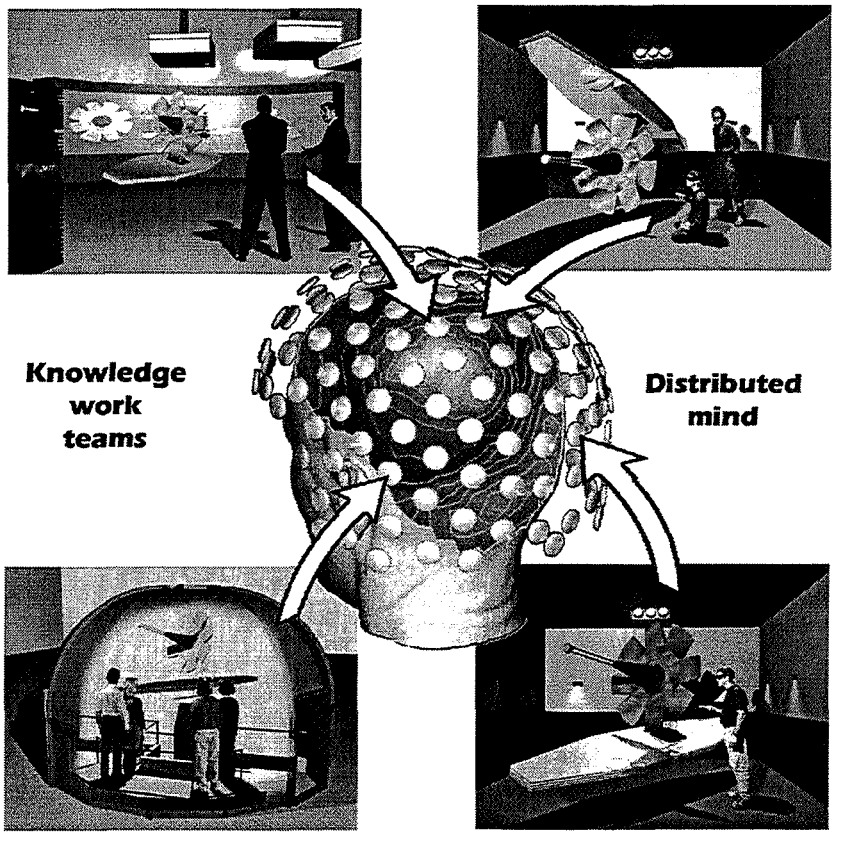


Figure 14

Intelligent Synthesis Environment (ISE)

The potential benefits of collaborative distributed environments for virtual product development and mission synthesis have led several organizations to initiate programs to realize these benefits. Among the government programs are:

- Simulation Based Design (SBD)
- Rapid Design Exploration and Optimization (RaDEO) of the Defense Advanced Research Projects Agency
- Partnership for Advanced Computational Infrastructure (PACI) of the National Science Foundation
- Knowledge and Distributed Intelligence (KDI) of the National science Foundation
- System Integration for Manufacturing Applications (SIMA) of the National Institute of Standards and Technology
- Intelligent Synthesis Environment (ISE) concept being developed by NASA and the University of Virginia.

Figure 15 shows the five major components of ISE: human-centered computing, infrastructure for distributed collaboration, rapid synthesis and simulation tools, life-cycle integration and validation, and cultural change in the creative process.

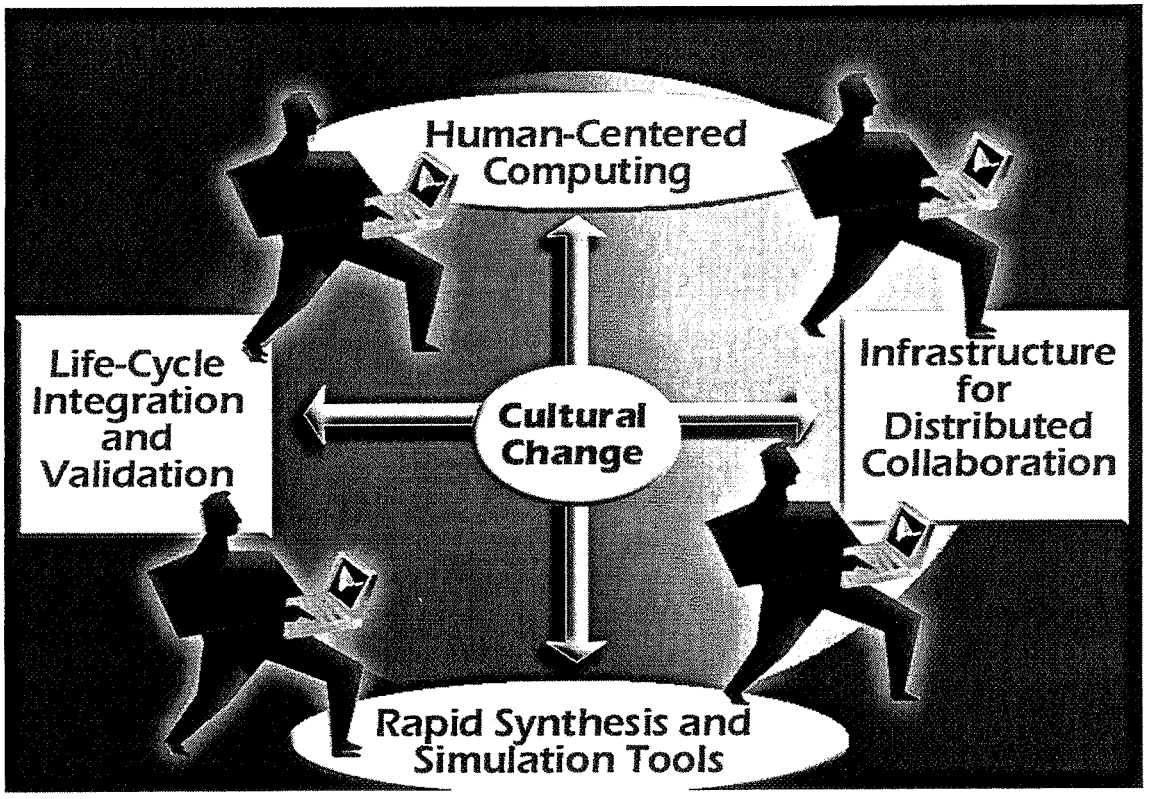


Figure 15

Potential Role of Intelligent Agents in ISE

Extensive use of intelligent agents will be made in ISE. Specifically, the following uses of intelligent agents in each of the five components can be identified:

- In the human-centered computing component, intelligent agents are used to overcome the limitations of the current user-interface approaches, e.g., verbots (virtual humans), can enable natural language understanding as well as adaptive/reconfigurable interfaces.
- In the infrastructure for distributed collaboration component, intelligent agents:
  - Simplify the different tasks associated with distributed computing (e.g., communication and network managers can act as global resource managers).
  - Organize and automate the work flow for diverse teams.

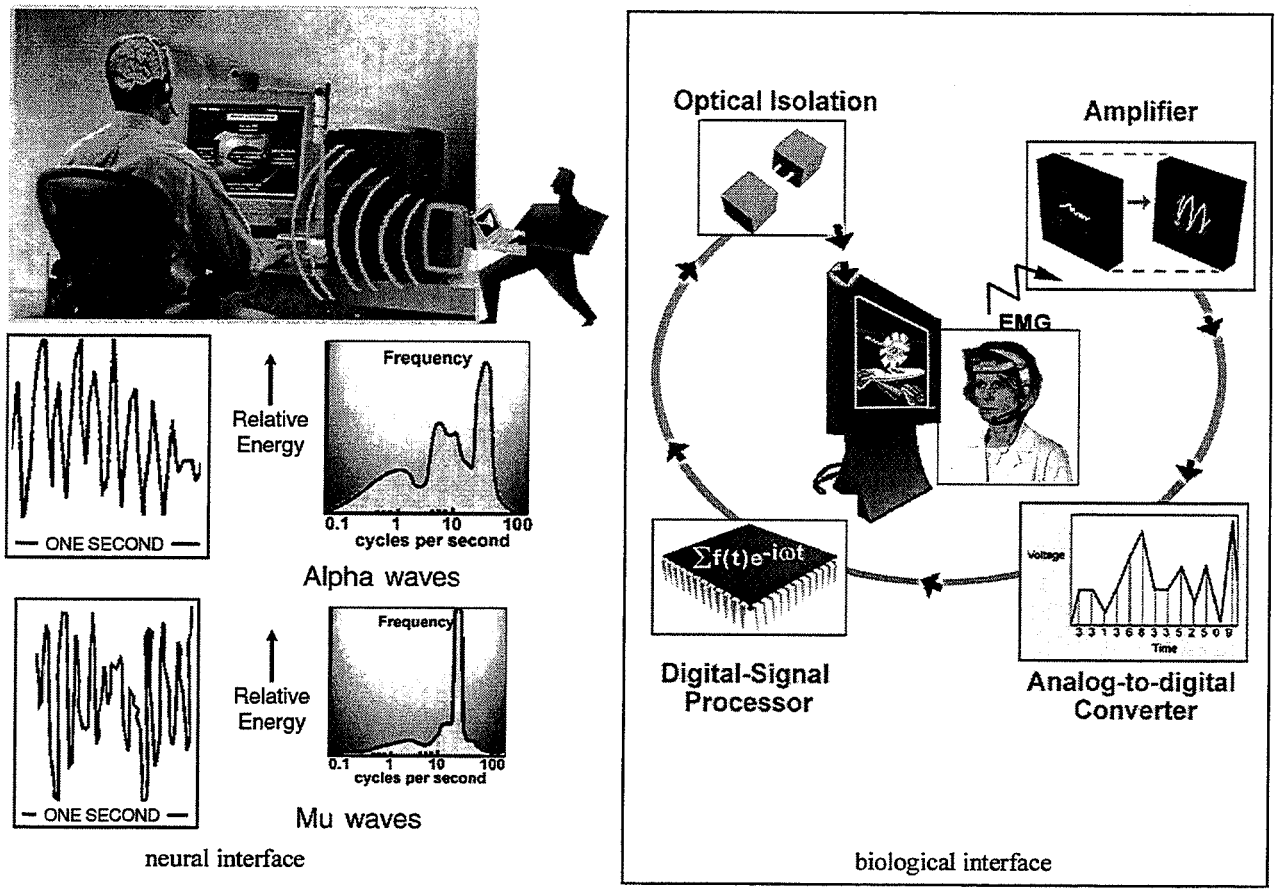


Figure 16

Potential Role of Intelligent Agents in ISE

- In the rapid synthesis and simulation tools component, intelligent agents act as modeling, analysis and design advisors.
- In the life cycle integration and validation component, intelligent agents:
  - Check data integrity, distribute and provide constraints
  - Query broad spectrum of databases
  - Improve database performance and support data security.
- In the cultural change component, intelligent agents manage knowledge networks to support collaborative distributed learning environments.

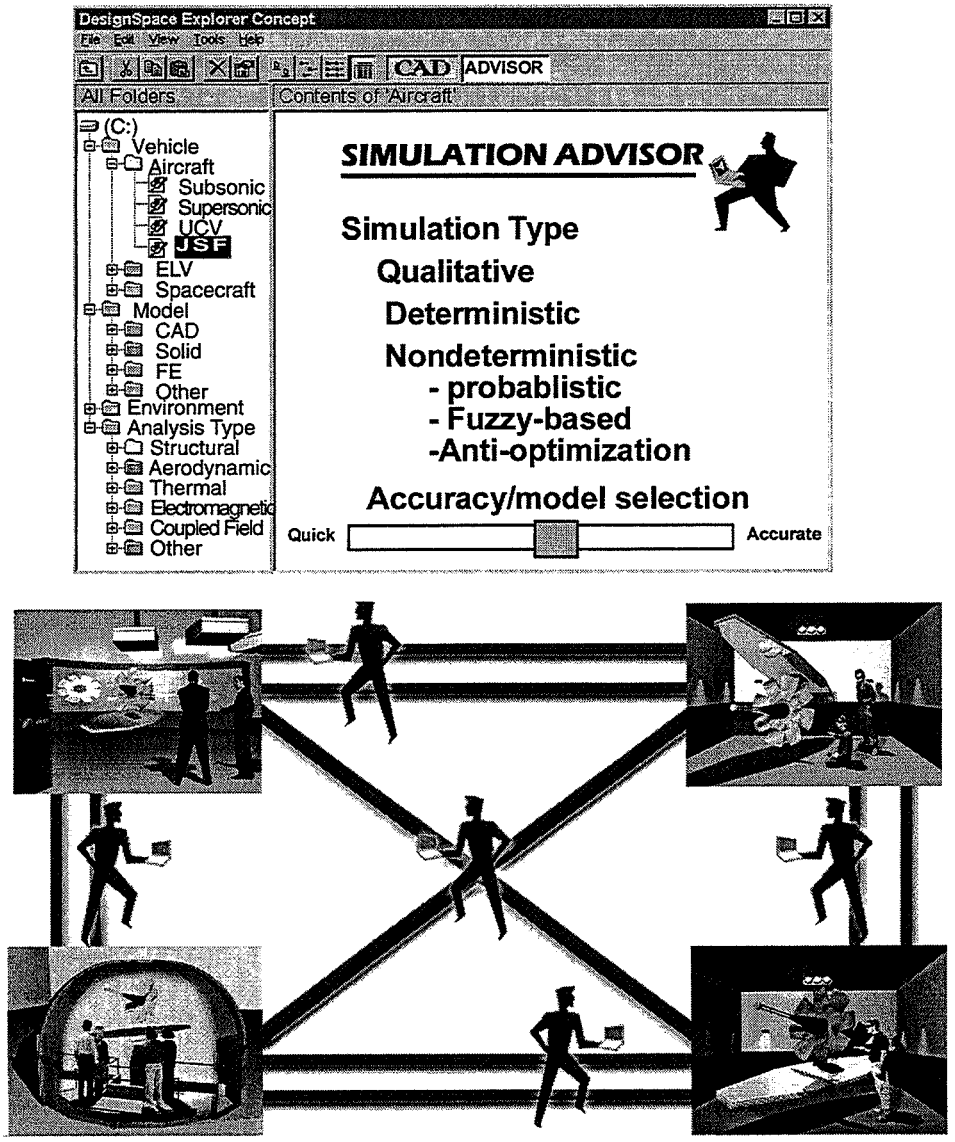


Figure 17

Information Sources

Extensive resources are available on agent technology.

- Books and monographs, including several published in 1998.
- Overview and survey papers, online articles, News Webletter and bibliographies.
- Workshop and conference proceedings.
- Journals, research groups, societies and companies.
- Internet repository. URL address: http://www.cs.umbc.edu/agents/

Figure 18

52-61
032516
370163
18p

The Role of Intelligent Agents in Advanced Information Systems

Larry Kerschberg
Center for Information Systems Integration
and Evolution
George Mason University
Fairfax, VA

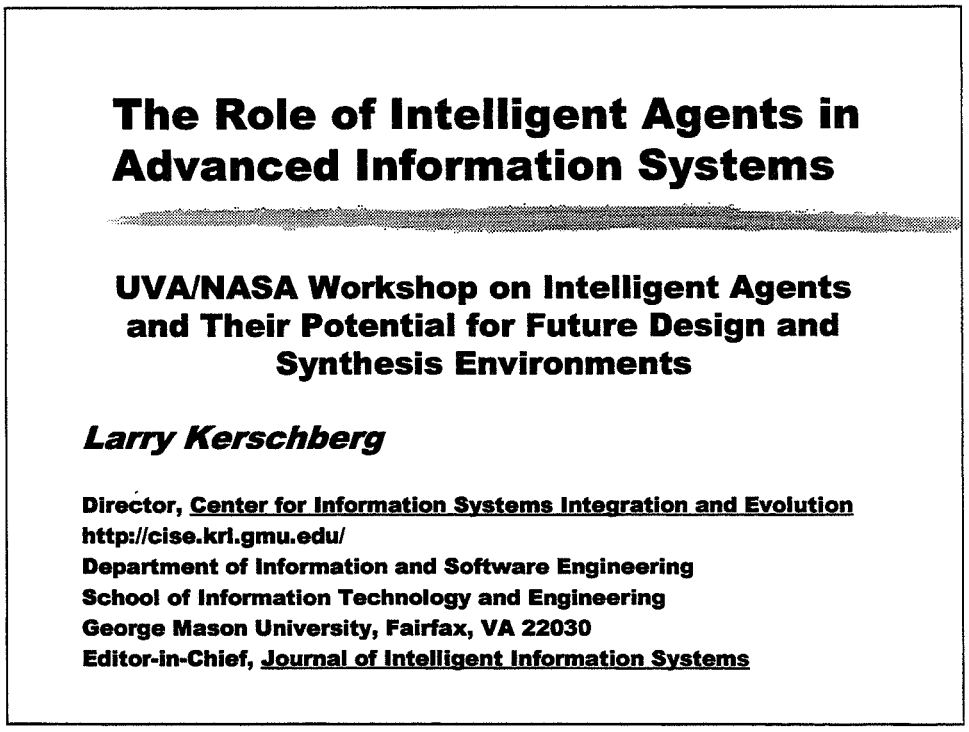
Page intentionally left blank

The Role of Intelligent Agents in Advanced Information Systems

Larry Kerschberg
Professor of Information and Software Engineering
Director, Center for Information Systems Integration and Evolution
George Mason University
Fairfax, VA 22020-4444

kersch@gmu.edu
Phone: 703-993-1661
Fax: 703-993-1638
http://cise.krl.gmu.edu/~kersch/

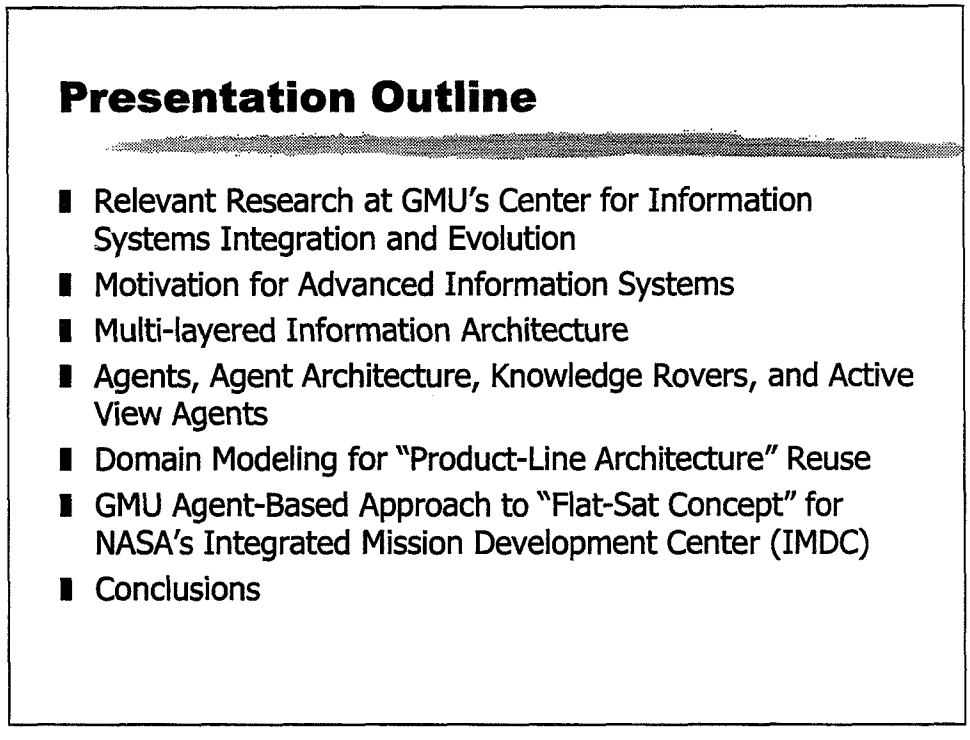
This research has been sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and a NASA "Flat-Sat" grant from Goddard Space Flight Center.



Presentation Outline

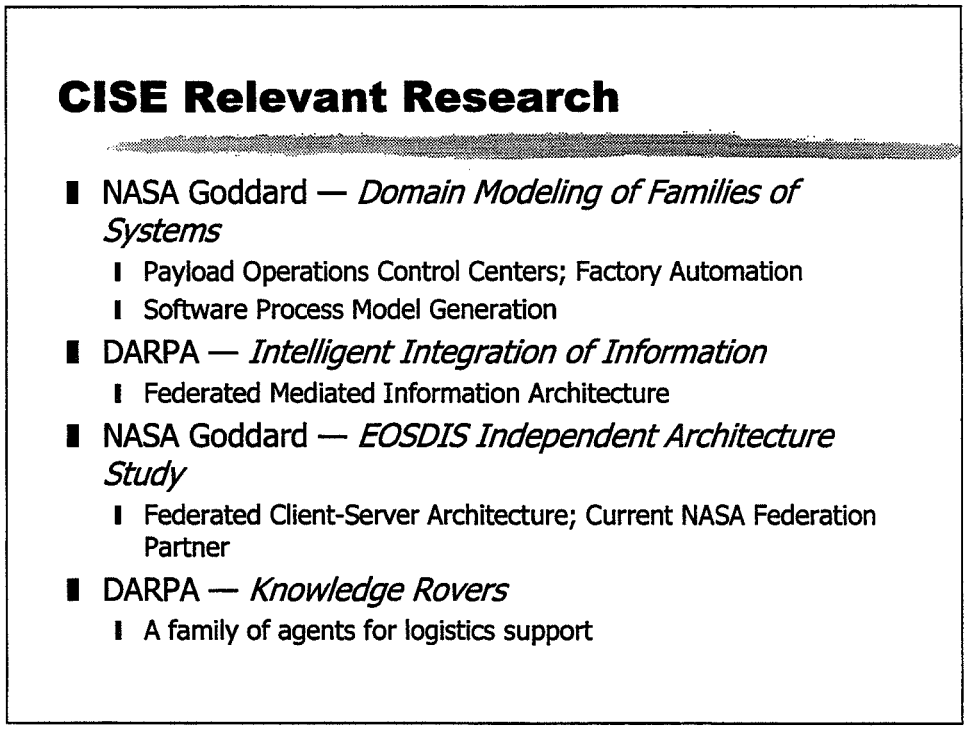
In this presentation we review the current ongoing research within GMU's Center for Information Systems Integration and Evolution (http://cise.krl.gmu.edu). We define characteristics of Advanced Information Systems, discuss a family of agents for such systems, and show how GMU's Domain Modeling tools and techniques can be used to define a "Product Line Architecture" for configuring NASA Missions.

These concepts can be used to define Advanced Engineering Environments such as those envisioned for NASA's new initiative for Intelligent Design and Synthesis Environments.



CISE Relevant Research

GMU's Center for Information Systems Integration and Evolution (CISE) has been involved in a number of research and development projects that influence the ideas presented in this talk. These include: 1) Domain Modeling for Families of Systems (joint work with Dr. Hassan Gomaa, and sponsored by Mr. Walt Truszkowski of NASA Goddard, 2) DARPA's Program on the Intelligent Integration of Information which motivated our work into federated and mediated data, information and software architectures, 3) The GMU Independent Architecture Study for EOSDIS in which we proposed a federated approach to ECS, and which is now being implemented by NASA, and 4) DARPA's Advanced Logistics Program in which we formulate a "Knowledge Rover" architecture consisting of a family of configurable software agents to support an enterprise information architecture.



Key Features for Advanced Information Systems

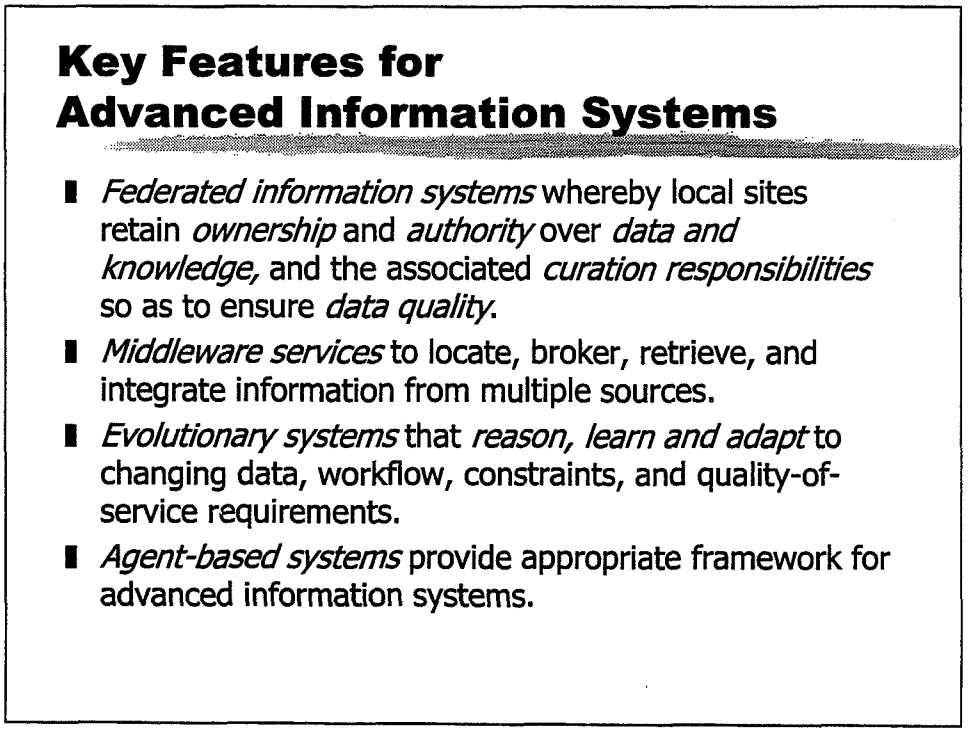
Advanced Information Systems must be active and evolutionary. We feel that intelligent software agents can provide the services needed for this class of systems.

These systems will be distributed and federated so as to share data and knowledge among the participants. Issues of data quality, timeliness, completeness and applicability of the data will all have to be dealt with.

The major contribution of agents, we feel, will be at the "middleware" layer so that users can be assisted in finding data, integrating it, and using it for decision-making. Here we propose the notion of Curator Agent (or several of them) to constantly review the information in a repository as to its quality, pedigree, context, and other attributes so that users can assess whether it is appropriate to their tasks and requirements.

If Advanced Information Systems are to evolve, they must be able to adapt to changing data requirements, workflow patterns, constraints and quality-of-service requirements.

The agent-based approach provides an appropriate framework for building such information systems.



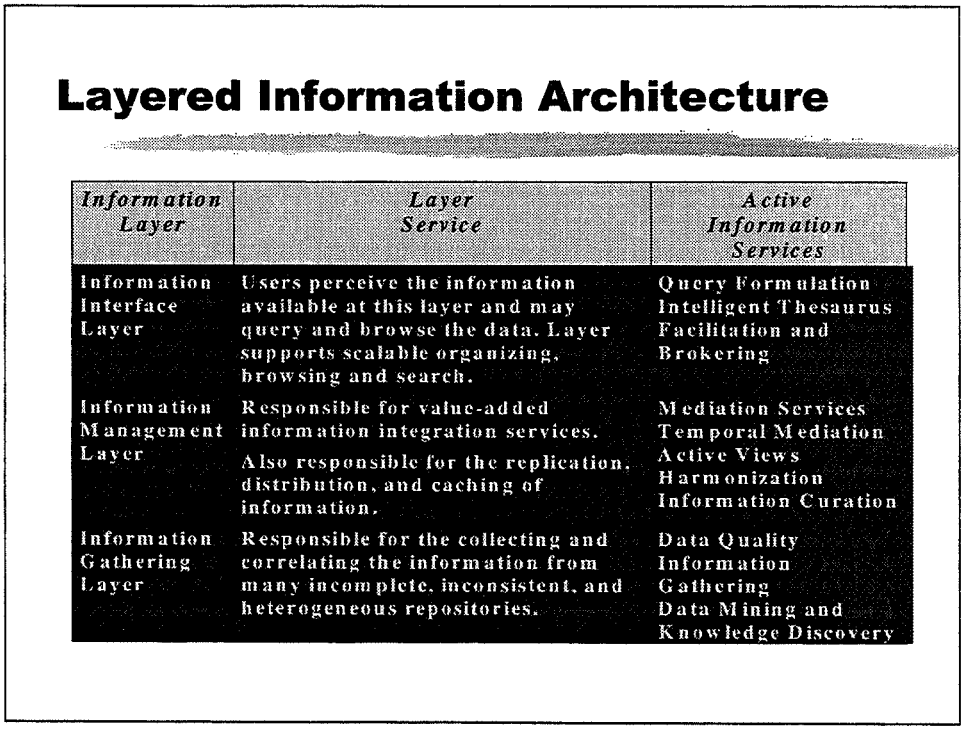
Layered Information Architecture

The layered information architecture incorporates the three information layers consisting of: 1) the Information Interface Layer, the Information Management Layer, and 3) the Information Gathering Layer.

The Active Information Services are those that have been developed by GMU (see references at the end of this presentation).

The GMU research group has developed a federated service architecture that provides Federation Interface Managers (FIMs) as active wrappers for information sources, an intelligent thesaurus, temporal mediation services, active views, and harmonization and inconsistency management services .

Data mining and knowledge discovery techniques are applicable to the study of data quality, user usage patterns, automatic classification, schema evolution, and system evolution.



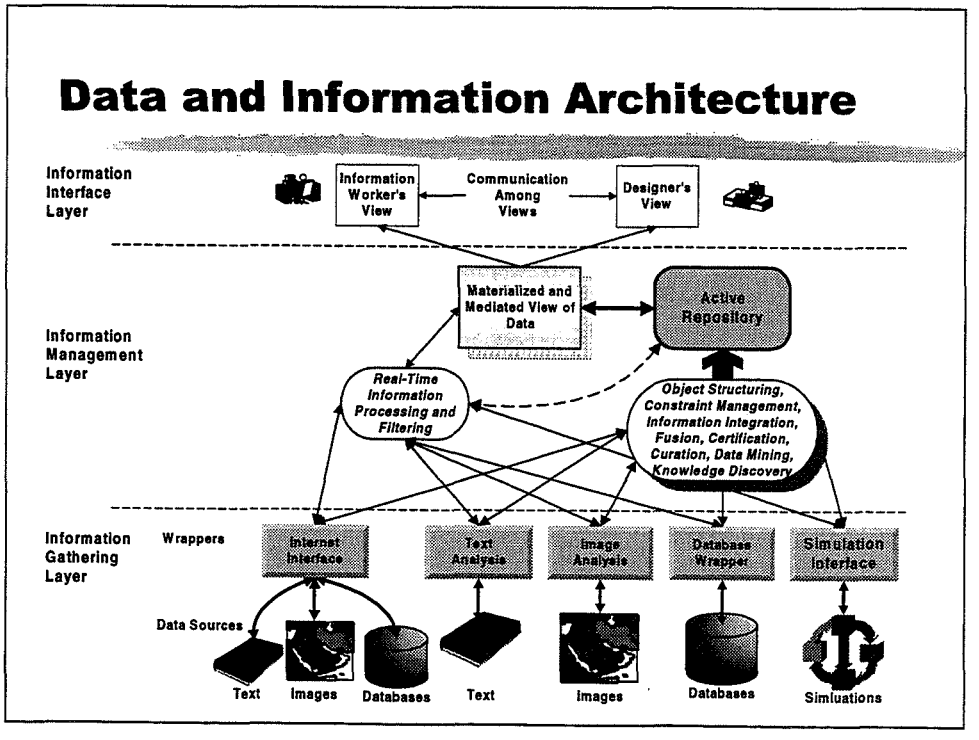
Data and Information Architecture

The data and information architecture incorporates three information layers consisting of:

1) *Information interface layer* where users access the system, formulate queries, collaborate in problem-solving activities, initiate pull scenarios and receive information from push scenarios. Users have access to their local databases and work through local views. We assume that collaboration mechanisms and tools exist at this layer.

2) *Information management layer* where objects, mediated active views, and information in an Information Repository are integrated, managed, replicated, and updated. This layer mediates between the information interface layer and the information gathering layer, allowing users to perceive an *integrated information space*, when in reality, data resides in multiple heterogeneous databases and information sources. A mediated view of data is provided at this layer and user views are materialized from the mediated view.

3) *Information gathering layer* where data from diverse, heterogeneous inter-networked information sources are accessed. Special rover agents are used to perform the mediated access to local as well as internet resources.

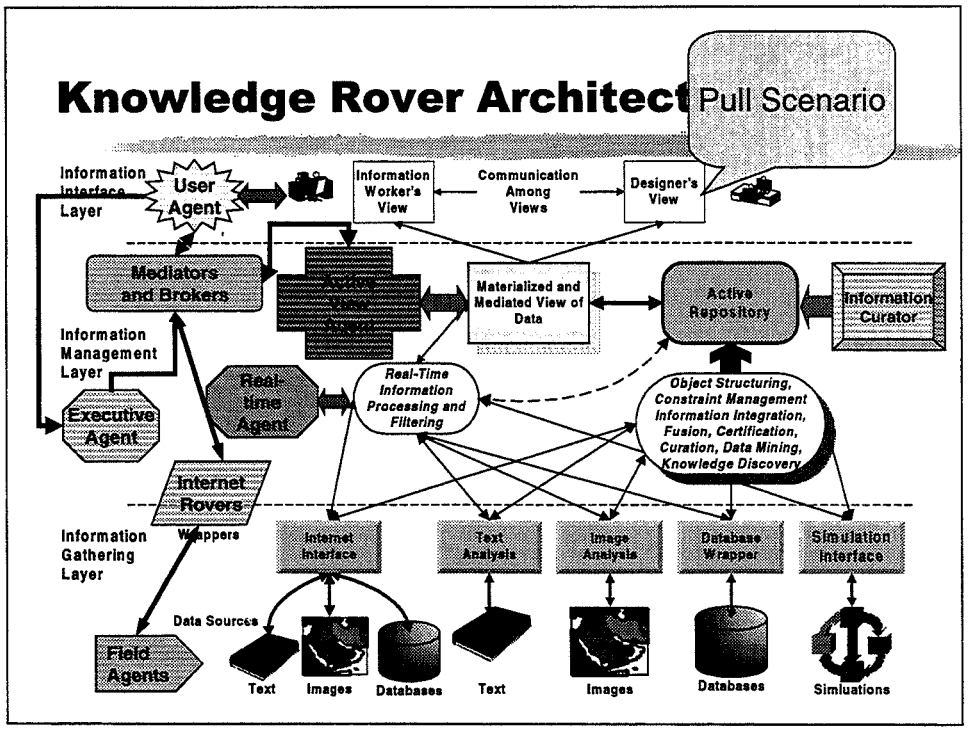


Knowledge Rover Architecture

This slide depicts a "pull scenario" in which a "decision-maker" works cooperatively with an "information worker" to pose a query to the User Agent. The User Agent consults with the Executive Agent to schedule the query. The Executive in turn coordinates with the other agents (Mediators and Brokers, Active View Agent, Internet Rovers and Field Agents).

Note that the Real-Time Agent works continuously to monitor for events and conditions of interest, while the Information Curator Agent manages the ongoing process of information integration, for the information that will be stored in the Information Repository.

In addition to Pull Scenarios, we have Push Scenarios where data is being provided by the heterogeneous data sources, by field agents and by Internet rovers. The real-time agent monitors important events and provides such information to Active View Agents which may update mediated data views or inform humans of such events. The data feeds also are processed by the information curator agent.



Knowledge Rover Family

The next few slides provide a definition of the services provided by the various agents in the Knowledge Rover family.

Knowledge Rover Family

- ***Executive Agent***— is a coordinator for a group of agents. It is informed of significant events. A significant event can lead to the activation of new agents. For example, if the system is notified of a request, then the executive agent would coordinate with other agents in implementing the *scenario.*
- ***User Agent***— acts on behalf of a user, and is responsible for assisting users in: 1) browsing catalogs and information holdings such as the information repository, 2) the intelligent formulation of queries, and 3) the planning of tasks within a mission-specific scenario such as modifying the design of a spacecraft's power supply.

Knowledge Rover Family

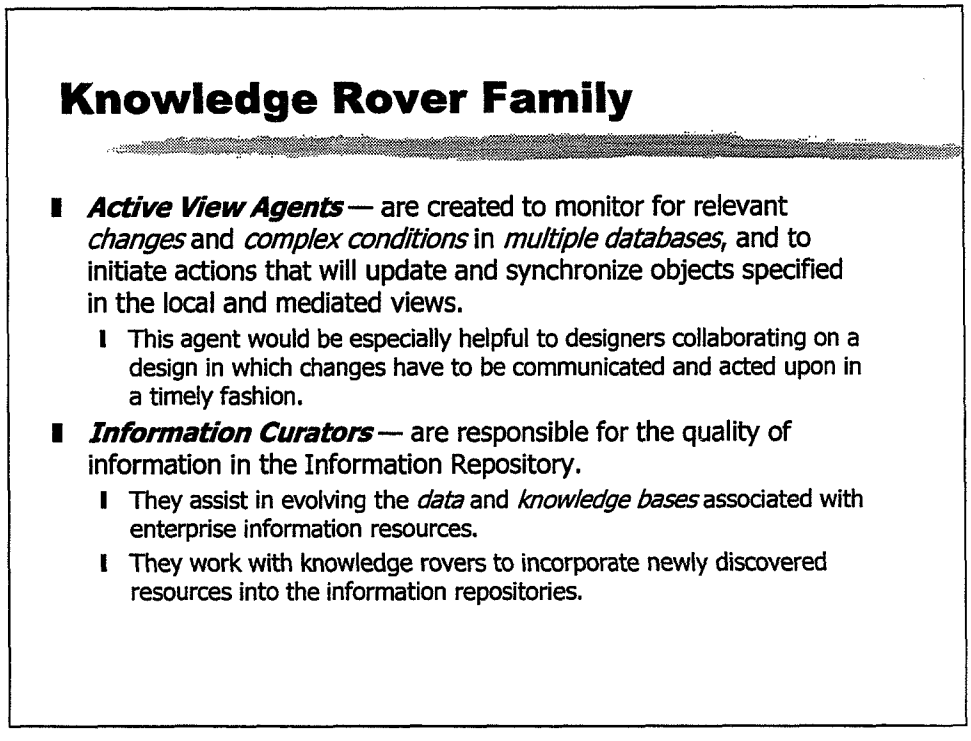
The next few slides provide a definition of the services provided by the various agents in the Knowledge Rover family.

Knowledge Rover Family

- ***Real-time Agent*** — is mission-specific, defined and configured to process incoming data, and update the appropriate database or notify the appropriate users. They monitor the external environment, and signal relevant agents when an event is detected.
- ***Broker Agent*** — provides intelligent dictionary, yellow pages, object location and brokerage services.
- ***Mediation Agents*** — are configured to assist in the integration of information from multiple data and information sources, having diverse data formats, different meanings, differing time units, and providing differing levels of information quality.

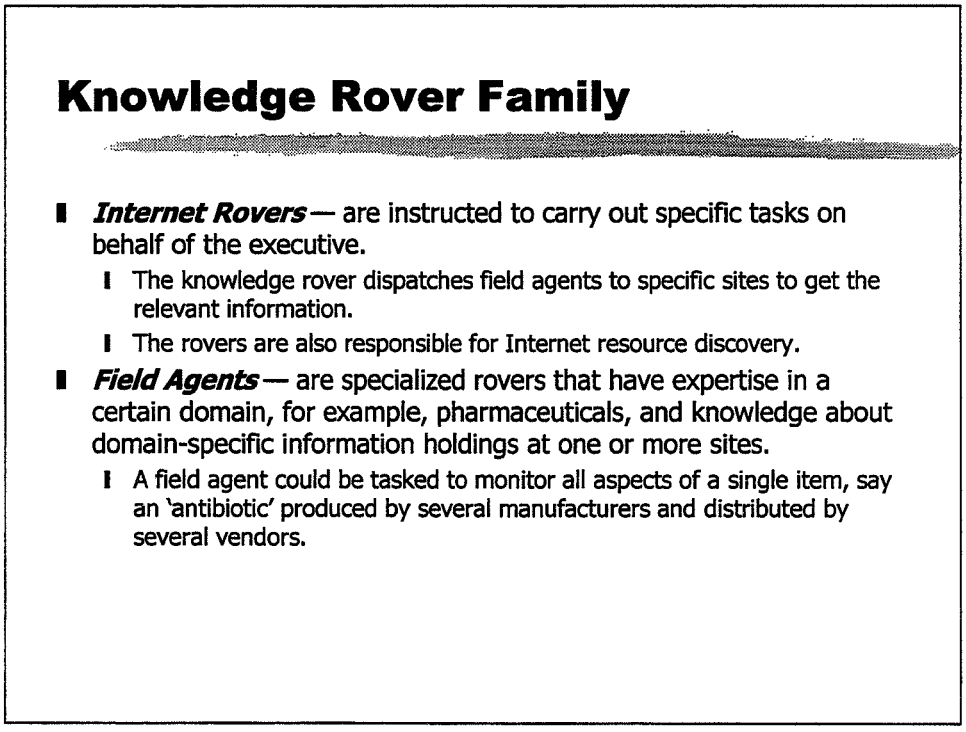
Knowledge Rover Family

The next few slides provide a definition of the services provided by the various agents in the Knowledge Rover family.



Knowledge Rover Family

The next few slides provide a definition of the services provided by the various agents in the Knowledge Rover family.



Active View Agent

Active View Agents (AVA) — are created to support user views specified over multiple, autonomous and heterogeneous data sources. In many cases, users do not have to be notified of all event occurrences; rather, users and their associated active views are notified whenever *critical* events and object states are signaled. The active view agent is initially specified as a view materialized from multiple data sources. In addition, integrity constraints, called *staleness conditions*, are specified, and the AVA then transforms and distributes the constraints to local data sources. The AVAs and Real Time Agents cooperate in assessing when the significant events arise or staleness conditions are violated. The views are then materialized and appropriate triggers are invoked and pre-specified actions are taken.

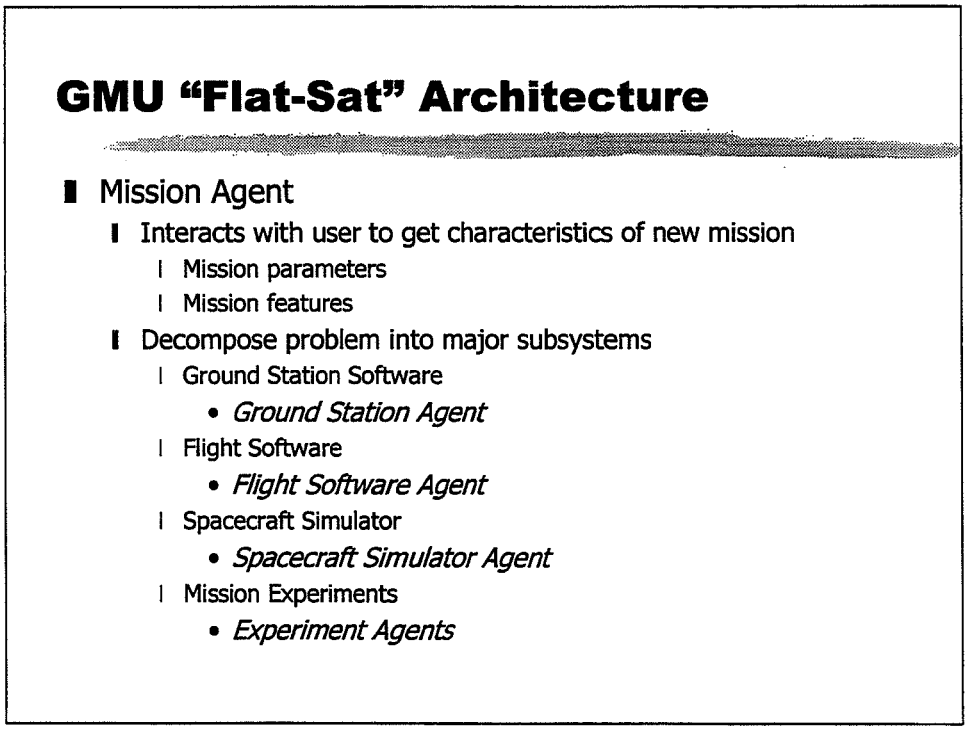
Active View Agent

- Intelligent view and cache management
  - Materialized views + staleness conditions,
  - Refresh the cache only with important changes,
  - Reduces pull scenarios that poll database periodically, and
  - Mediates between data push versus user data pull needs
- Provides *active monitoring* of significant events and conditions.
- Automatically generates Internet rovers and field agents with associated rules, triggers, event conditions and object subscriptions.
- Discussed in doctoral dissertation of Samuel Varas.

GMU "Flat-Sat" Architecture

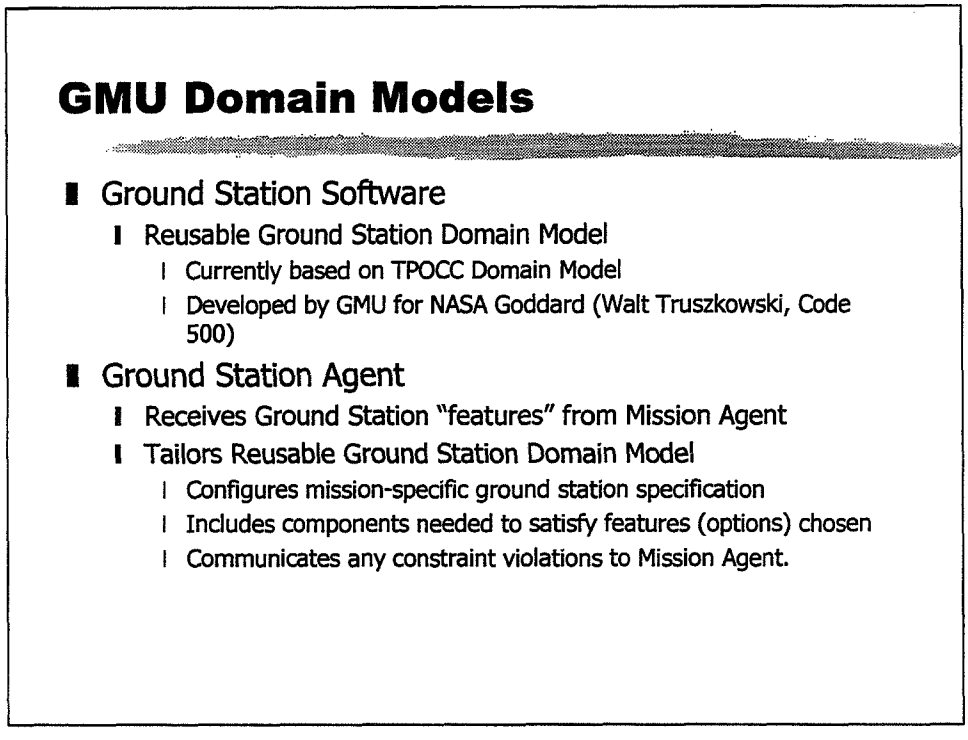
GMU (Gomaa and Kerschberg) is working with NASA Goddard's IMDC (Integrated Mission Development Center) to develop the "Flat-Sat" concept of configuring a mission from hardware, software and simulated components. The metaphor is that the components are laid out on a table and an agent-based configuration tool would assist in the configuration of the mission.

Our approach is to formulate domain models for the major subsystems involved in a mission, and to have a Mission Agent coordinate with Subsystem Agents to arrive at a solution to the configuration of a mission.



GMU Domain Models

We have constructed a domain model for ground station software and are working to construct other domain models such as flight simulation software.



Conclusions

- Agents-based architectures promote *active advanced information systems.*
- Knowledge rover family of cooperating agents provides *active services* supporting the intelligent enterprise's information architecture.
  - Agents such as Active View, Real-Time, Curators, and Rovers are *middle-ware agents* that mediate user requests against multiple heterogeneous information sources.
  - These agents may also be used to monitor relevant data/knowledge for *active* data management, constraint management, data mining, knowledge discovery and evolution.
- Agents and reusable "product line" architectures provide a framework for future design and synthesis environments.

Selected References

1. Gomaa, H., Kerschberg, L. et al., "A Knowledge-Based Software Environment for Reusable Software Requirements and Architectures," *Journal of Automated Software Engineering,* Vol. 3, Nos. 3/4, 1996, pp. 285-307.
2. Kerschberg, L., "Knowledge Rovers: Cooperative Intelligent Agent Support for Enterprise Information Architectures," in *Lecture Notes in Computer Science*, Peter Kandzia and Matthias Klusch (eds.), Springer-Verlag, Heidelberg, Vol. 1202, 1997, pp. 79-100.
3. Kerschberg, L., "The Role of Intelligent Software Agents in Advanced Information Systems," in *Advances in Databases*, Carol Small, Paul Douglas, Roger Johnson, Peter King and Nigel Martin (eds.), Springer-Verlag, London, Vol. 1271, 1997, pp. 1-22.
4. Kerschberg, L., Gomaa, H., Menascé, D. A. and Yoon, J. P., "Data and Information Architectures for Large-Scale Distributed Data Intensive Information Systems," in *Proc. of the Eighth IEEE International Conference on Scientific and Statistical Database Management*, Stockholm, Sweden, 1996.
5. Menascé, D. A., Gomaa, H. and Kerschberg, L., "A Performance-Oriented Design Methodology for Large-Scale Distributed Data Intensive Information Systems," *First IEEE International Conference on Engineering of Complex Computer Systems*, Florida (Outstanding Paper Award), 1995.
6. Seligman, L. and Kerschberg, L., "A Mediator for Approximate Consistency: Supporting 'Good Enough' Materialized Views," *Journal of Intelligent Information Systems*, Vol. 8, No. 3, 1997, pp. 203-225.

53-61
032517
370165
24P

"Conscious" Software Agents

Stanley Franklin
The Institute for Intelligent Systems
University of Memphis
Memphis, TN

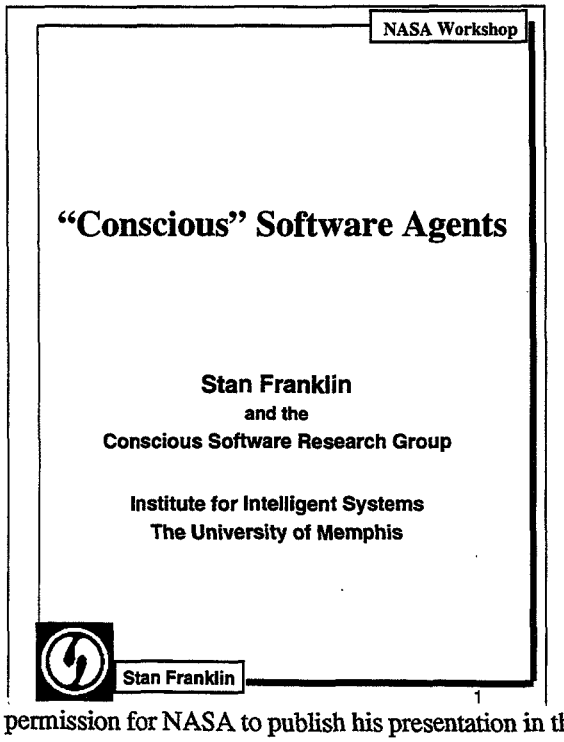
Page intentionally left blank

Conscious Software Agents

Stan Franklin
Conscious Software Research Group (CSRG)
The Institute for Intelligent Systems (IIS)
University of Memphis, Memphis, TN

The Conscious Software Research Group currently includes Stan Franklin, Art Graesser, Sri Satish Ambati, Myles Bogner, Derek Harter, Arpad Kelemen, Irina Makkaveeva, Lee McCauley, Aregahegn Negatu, Fergus Nolan, Uma Ramamurthy and Zhaohua Zhang.

The major mission of the Institute for Intelligent Systems (IIS) is to explore intelligent systems in humans, animals, computers, and abstract information technologies. It is widely recognized that there are substantial limitations with the conventional systems that have attempted to provide solutions to problems in computer science, telecommunications, business, management and science. Most of the conventional systems are static, linear, brittle, inflexible, slow, and/or not adaptive to changes in the world. Scientists, engineers and scholars throughout the world have therefore been developing intelligent systems that are considerably more powerful. They have recognized that some of the most intelligent systems already exist in biology and the human mind. Therefore, the intelligent systems of tomorrow will be hybrids of the intelligence in machines, biology and human cognition.



Dr. Stanley Franklin gives permission for NASA to publish his presentation in the *Workshop Proceedings*.

Definition of An Autonomous Agent

Note that neither "autonomous" nor "agent" is being defined, but rather the technical term "autonomous agent."

This definition comes from the paper:

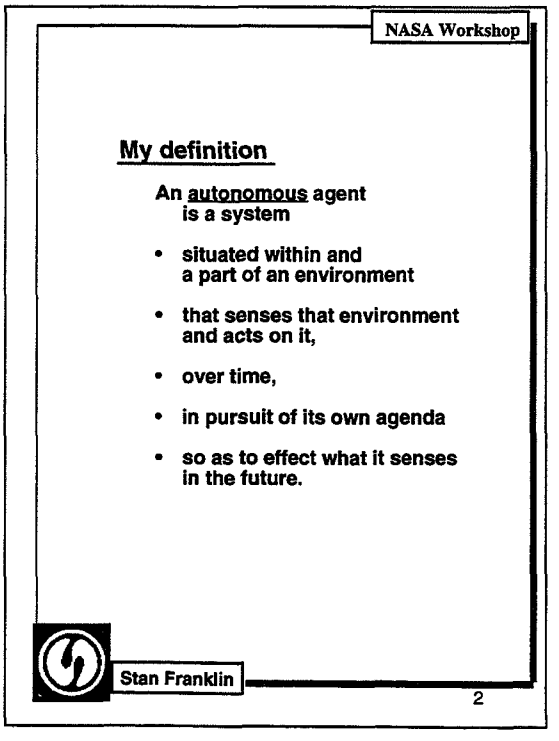
"Is it An Agent, or Just a Program?: A Taxonomy for Autonomous Agents," Stan Franklin and Art Graesser, Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, in *Intelligent Agents III*, Springer-Verlag, 1997, pp. 21-35.

It can be found on the web at:

http://www.msci.memphis.edu/~franklin/AgentProg.html.

In humans, the agent's agenda derives from evolved in-drives, in software agents, from drives or goals built in by the designer.

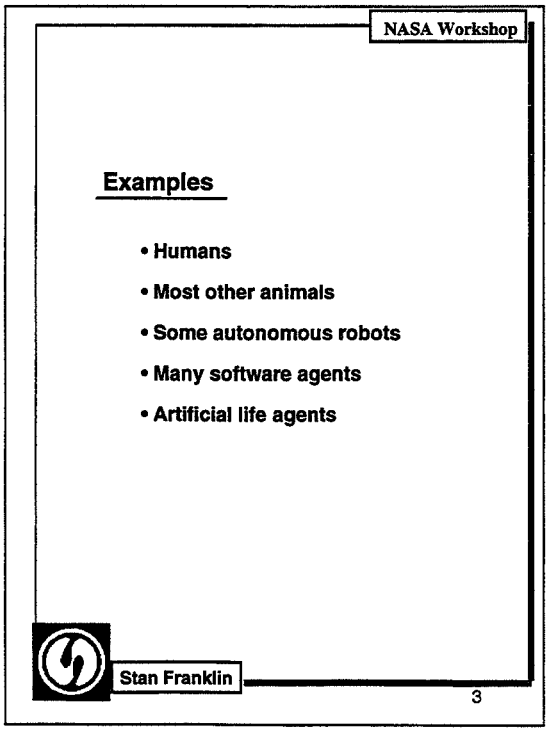
The last requirement can be rephrased to say that the agent must be structurally coupled to its environment in the sense of Maturana and Varela.



Examples of Autonomous Agents

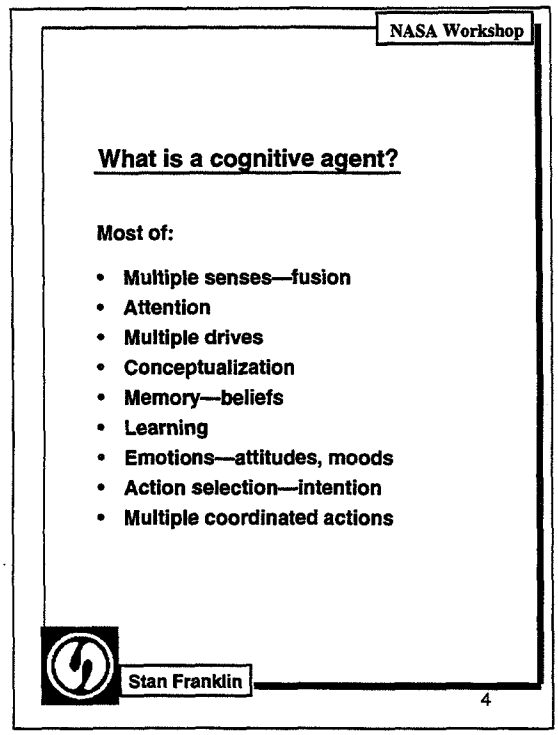
One way of clarifying the boundaries of this definition is by looking at extreme cases. Humans and some animals are at the high end of being an agent, with multiple, conflicting drives, multiple senses, multiple possible actions, and complex, sophisticated control structures. At the low end, with one or two senses, a single action, and an absurdly simple control structure we find a thermostat.

Software agents are to be distinguished from artificial life agents. The latter "live" in artificial computational environments created just for them, while the former "live" in real computational environments such as file systems, databases or networks.



A Cognitive Agent

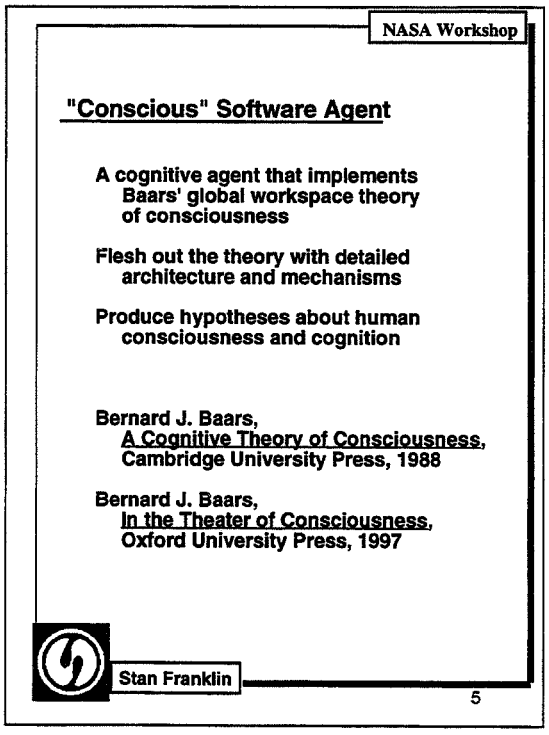
This notion comes from, "Autonomous Agents as Embodied AI," Stan Franklin, *Cybernetics and Systems*, Vol. 28, No. 6, 1997, pp. 499-520 (special issue on Epistomological Issues in Embedded AI). A general architecture for a cognitive agent is outlined and discussed. The paper is also available on the web at: http://www.msci.memphis.edu/~franklin/AAEI.html



Conscious Software Agents

Global workspace theory is, as a good psychology theory should be, relatively abstract, with a high level architecture and its functionality specified. A conscious software agent must flesh out the theory with lower level architecture and mechanisms.

Hypotheses about human consciousness and cognition are produced by assuming that humans do it like the agent does. Thus, every design decision becomes an hypothesis.



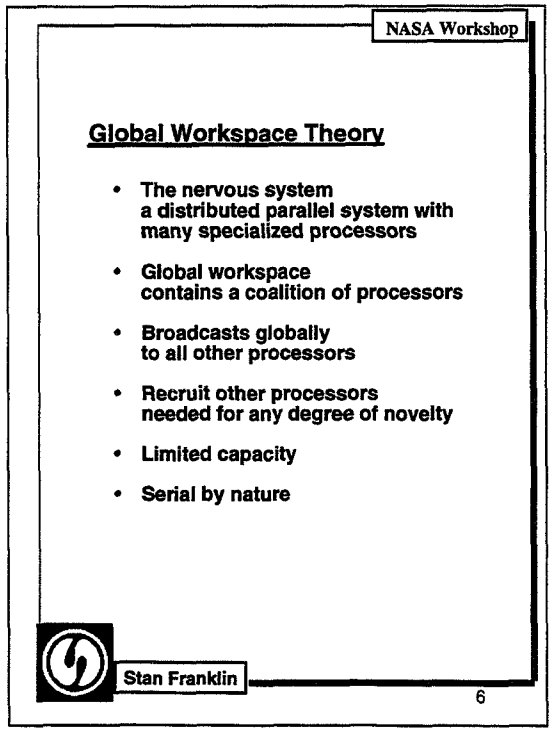
Global Workspace Theory

For a current, quite readable account of the theory see the second book listed on the previous slide. For a much more scholarly approach that specifies the empirical evidence on which the theory is based, see the first book listed on that slide.

The single most important contribution of global workspace theory is the idea of broadcasting the contents of consciousness in order to recruit relevant processors to help deal with whatever is new or problematic in the current situation. Anything else can be dealt with subconsciousnessly.

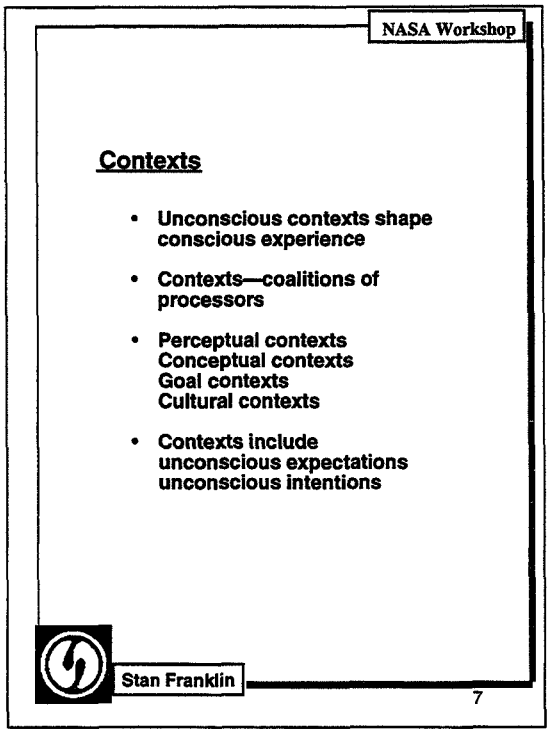
Of course capacity must be limited. Large messages cannot be easily understood.

The same argument shows why consciousness is serial. Messages arriving in parallel are not likely to be understood.



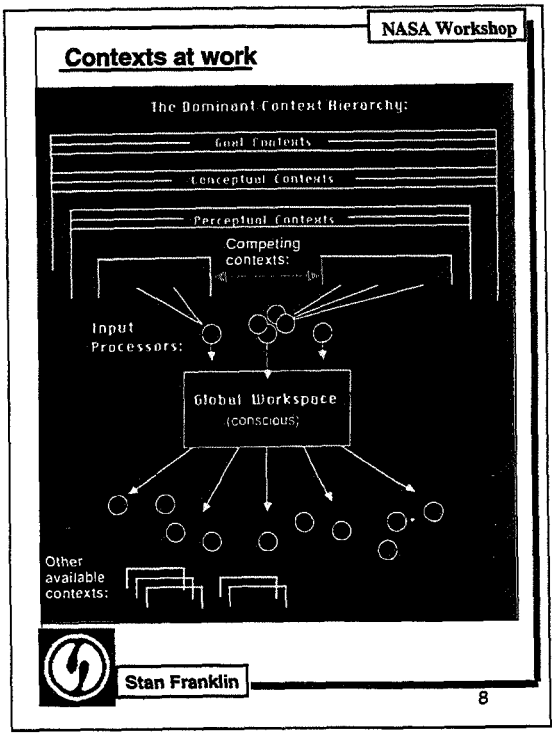
Contexts

Thinking of contexts as coalitions of processors is one of the major contributions of global workspace theory.



Contexts at Work

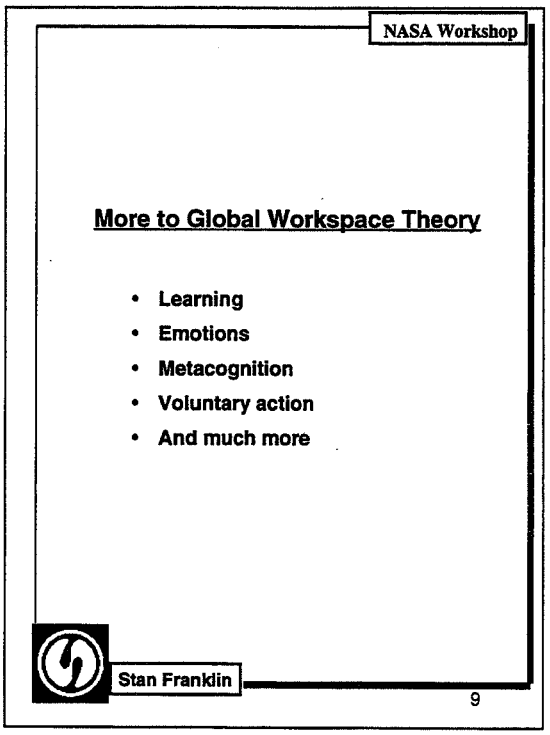
This figure was taken from Baars' 1988 book.



I hereby give permission for Stan Franklin's republication from my 1988 book, **A Cognitive Theory of Consciousness**, for which I now own copyright.
Bernard Baars, The Wright Institute, Berkeley, CA

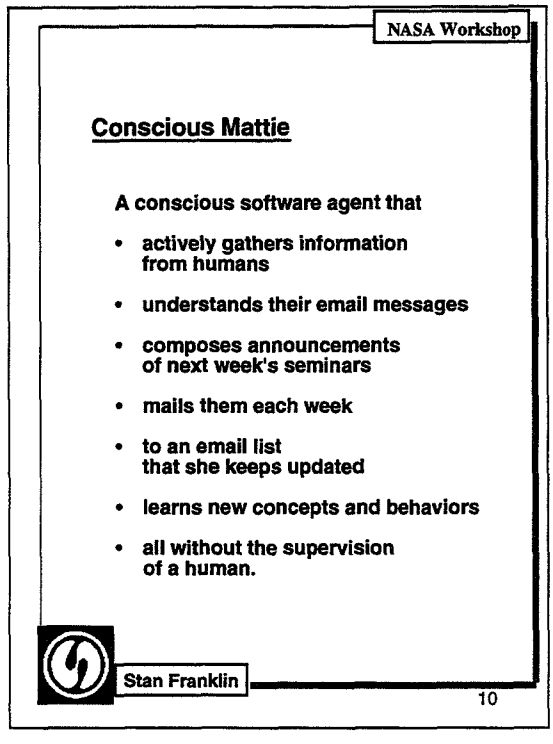
More to Global Workspace Theory

Global workspace theory is sufficiently broad as to serve as a general theory of cognition. It even deals with the issue of the self.



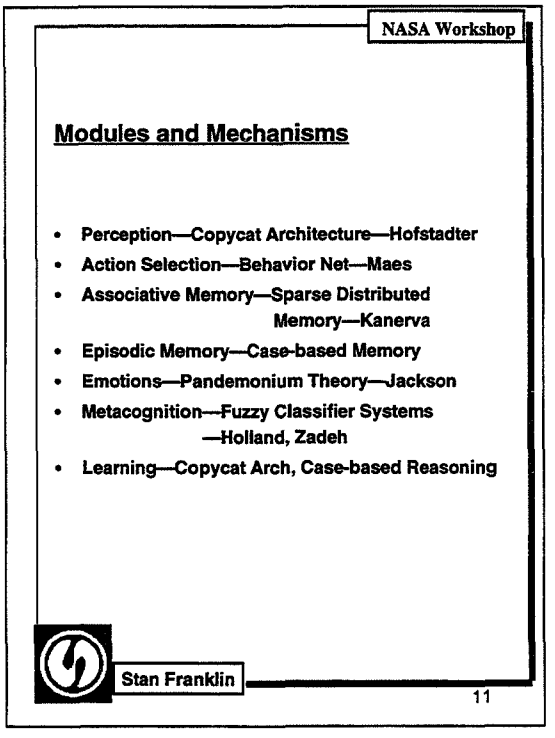
Conscious Mattie

Conscious Mattie (CMattie) is our first implementation of global workspace theory, our first conscious software agent. She will "live" in a UNIX system, carrying out her tasks autonomously, and corresponding in natural language with seminar organizers and participants via email.



Modules and Mechanisms

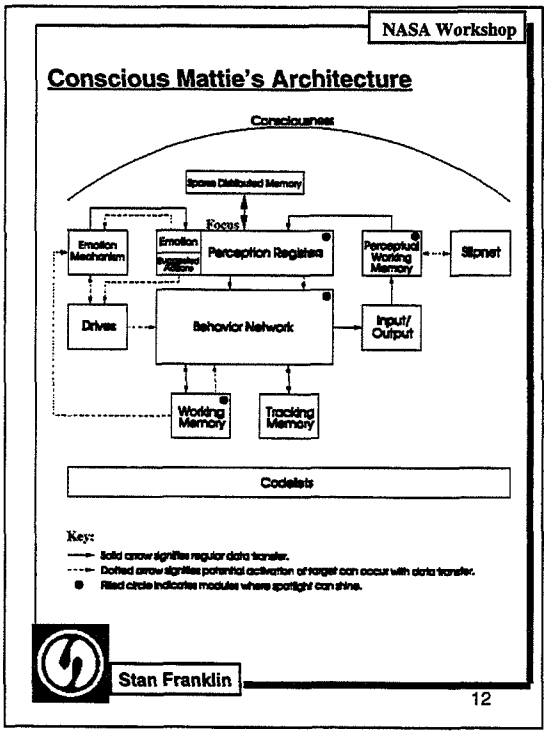
CMattie's various modules are, for the most part, implemented with mechanisms taken from the "new AI." Readable introductions to almost all of them can be found in Stan Franklin's *Artificial Minds* (MIT Press, 1995).



CMattie's Architecture

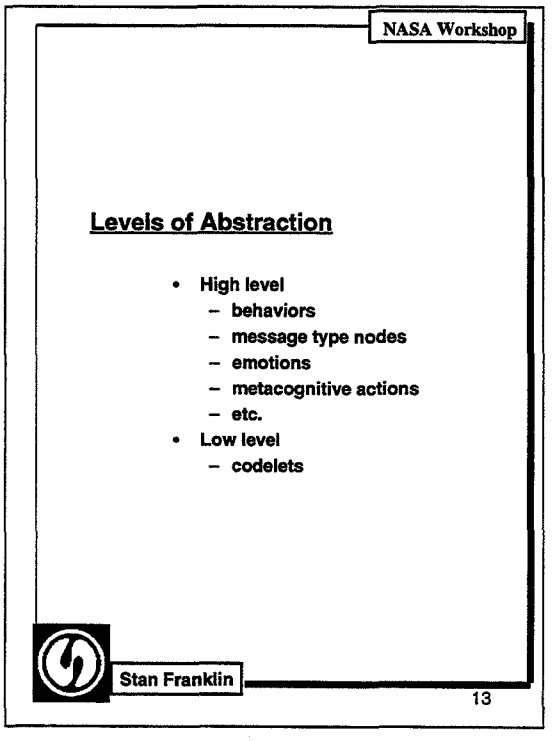
Though modular by design, CMattie's architecture is highly interconnected.

The diagram below is both incomplete and out of date. Metacognition and learning are missing, as is episodic memory.



Levels of Abstraction

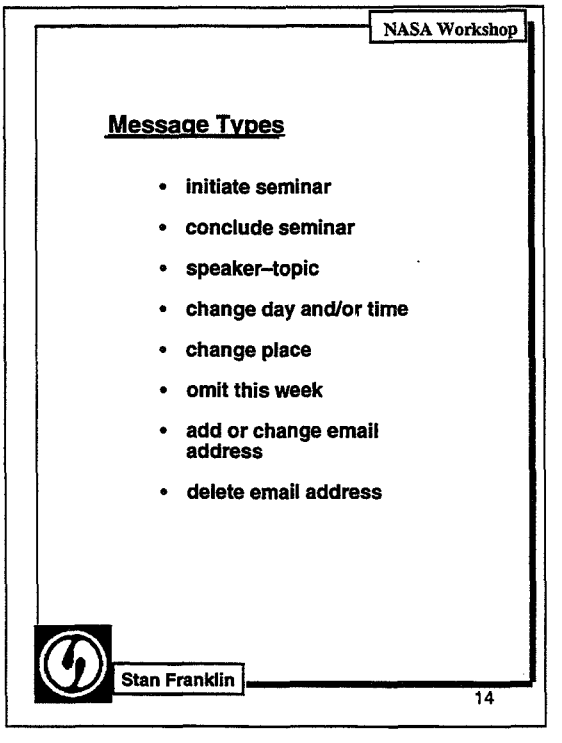
CMattie's architecture is conveniently categorized into high and low level pieces. The high level modules include all those needed to make her a cognitive agent. At the low level are the codelets that actually do all the work. She's a multi-agent system.



Natural Language Processing

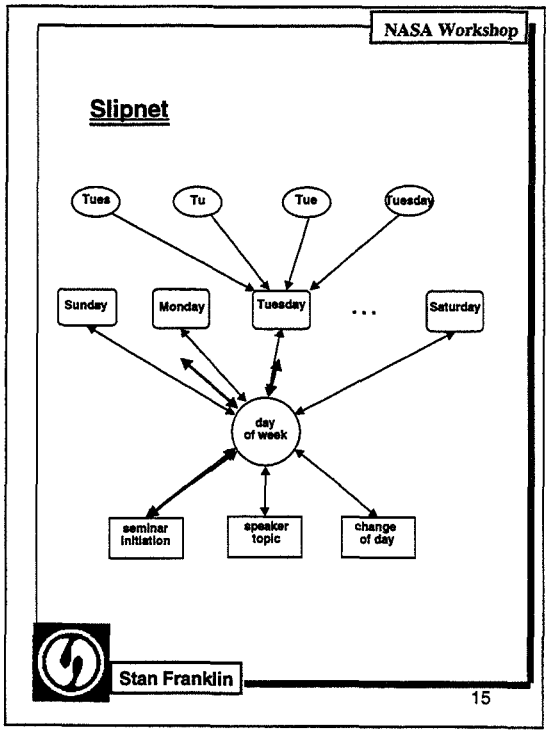
CMattie corresponds with seminar organizers and participants via email in natural language. This is possible because of her quite narrow domain. There are only a few things people want to talk to her about. The key to understanding a message is to ferret out its message type, of which there are only about a dozen. This allows natural language understanding via surface features.

See "Natural Language Sensing for Autonomous Agents," Zhaohua Zhang, Brent Olde, Stan Franklin Art Graesser and Yun Wan, INTERNATIONAL IEEE JOINT SYMPOSIA on INTELLIGENCE and SYSTEMS, 1998.



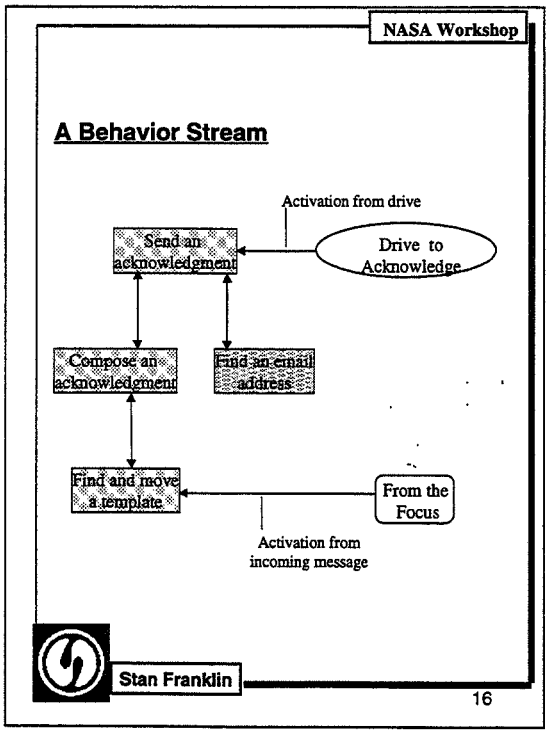
Slipnet

CMattie's slipnet contains the knowledge needed for understanding incoming messages. Here is a tiny portion of it. Codelets might identify one of the many forms of Tuesday, activate the Tuesday node which spreads activity to the day-of-the-week node, which in turn helps activate the message types that are expected to contain a day of the week.



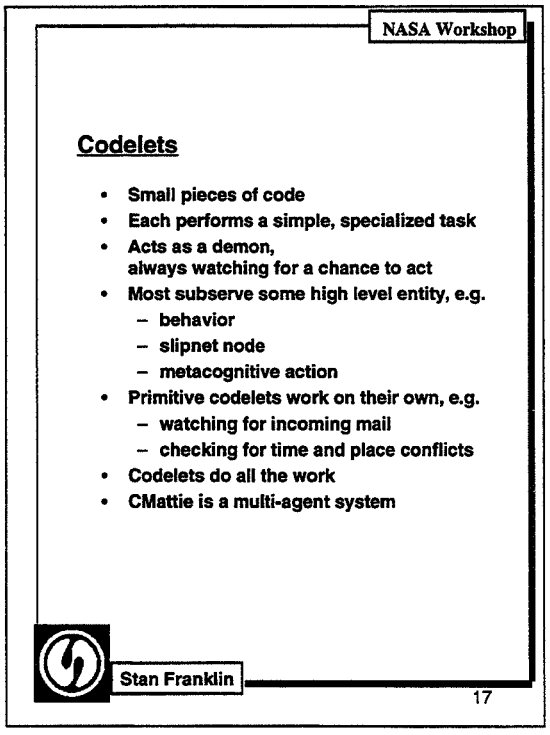
Behavior Net

CMattie selects her next behavior (goal context in global workspace theory) by means of an expanded version of Maes' behavior net. Activation is spread in two directions, one originating with an explicitly represented drive, and the other from the environment. In this case, environmental information comes through perception whose output is to the focus. A behavior is chosen to be active if all it preconditions are satisfied, if its activation is above threshold, and if its activation is the highest such.



Codelets

These small pieces of code do almost all of CMattie's work. They correspond to Baars' processors, Minsky's agents, Jackson's demons, Ornstein's small minds, and to Hofstadter's codelets, from whence comes our name for them.



CMattie Today

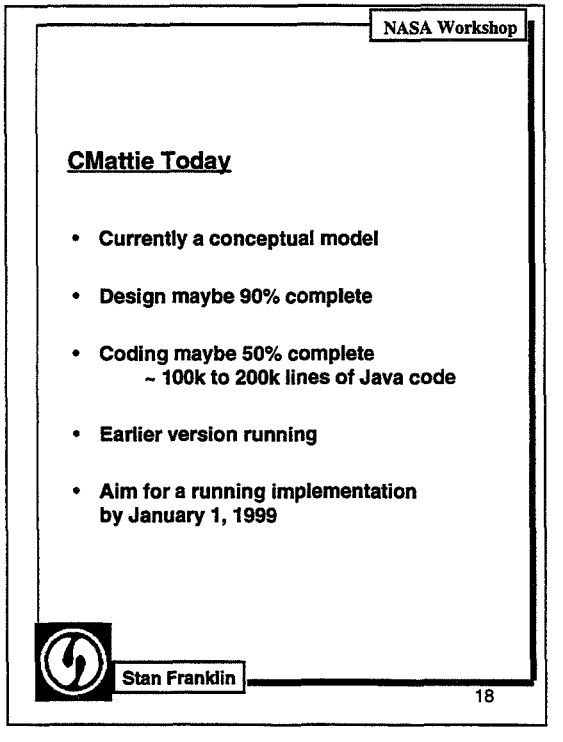
Several papers describe CMattie's architecture, mechanisms and functioning:

"Virtual Mattie—An Intelligent Clerical Agent" (Stan Franklin, Art Graesser, Brent Olde, Hongjun Song, and Aregahegn Negatu), presented to the AAAI Symposium on Embodied Cognition and Action, Cambridge MA, November 1996.

"Learning Concepts in Software Agents," Uma Ramamurthy, Stan Franklin and Aregahegn Negatu, Fifth International Conference of The Society for Adaptive Behavior 98, Zurich, 1998.

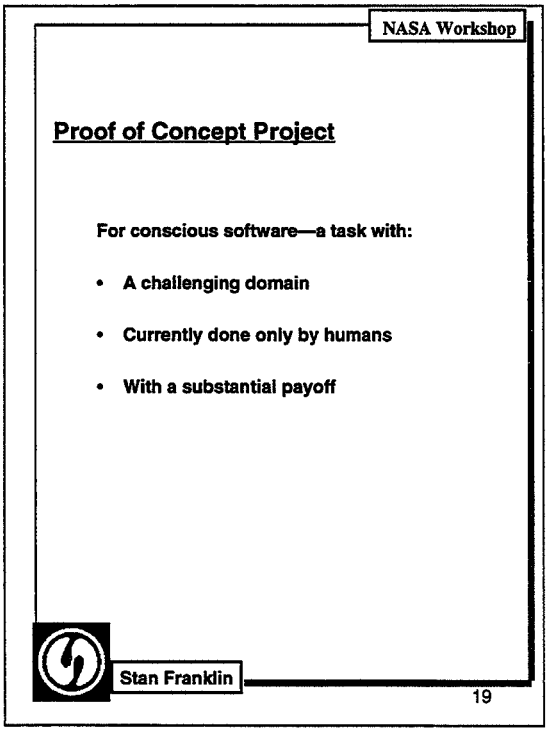
"Metacognition in Software Agents Using Classifier Systems," Zhaohua Zhang, Stan Franklin and Dipankar Dasgupta, Fifteenth National Conference on Artificial Intelligence, Madison, Wisconsin, 1998.

Several more papers are in draft form and will appear.



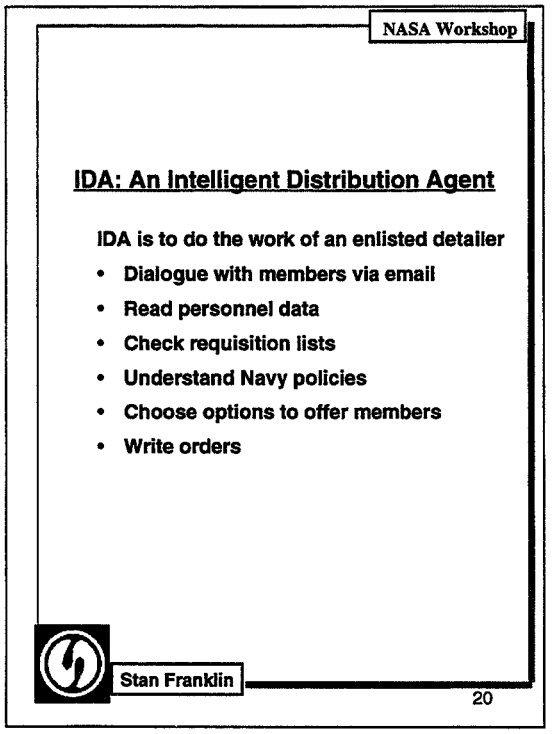
Proof of Concept Project

If conscious software is to prove itself as a technology, it must do so in a more challenging domain than that of CMattie. I can imagine building a system using classical AI techniques that would perform CMattie's tasks. For a proof of concept project for conscious software, we need one that no one would think to implement that way, in fact, one that no one would think to implement in software at all.



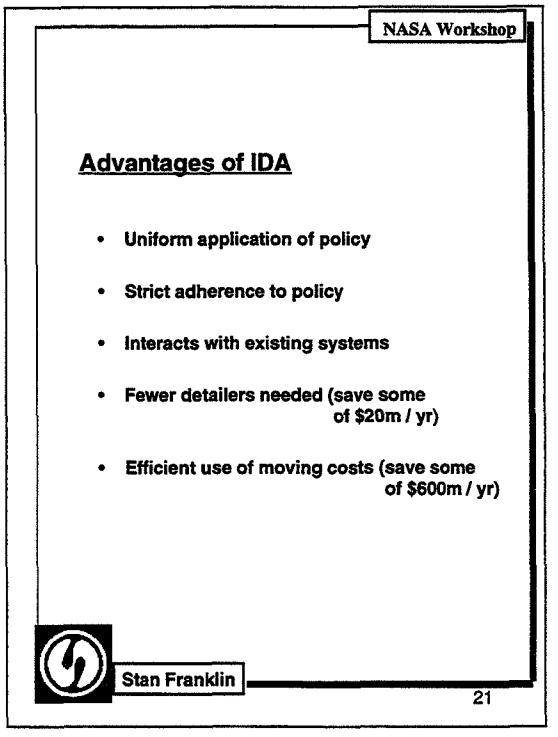
IDA

IDA promises to be a proof of concept project. She will assign sailors to new billets at the end of their current tour. Two-hundred and eighty humans, called detailers by the Navy, and no software systems, currently do this job. The Navy has previously supported software projects aimed at assisting detailers but none aimed at replacing them.



Advantages

IDA shows sufficient promise that the Navy is currently funding her development and enthusiastically cooperating with it.



Page intentionally left blank

54-81
032518
37067

Decentralized Decision Making in Concurrent Engineering

William P. Birmingham
University of Michigan
Ann Arbor, MI

Page intentionally left blank

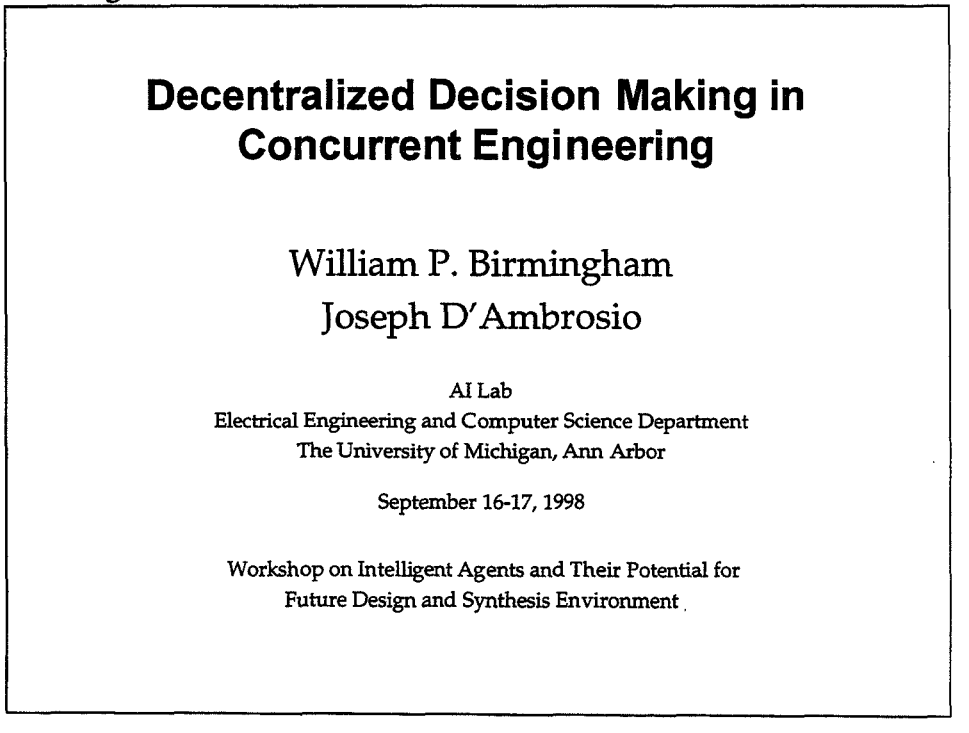
Decentralized Decision Making in Concurrent Engineering

William P. Birmingham
Joseph D'Ambrosio
AI Lab
Electrical Engineering and Computer Science Department
The University of Michigan, Ann Arbor

This talk describes a view of concurrent engineering (CE) as a coordinated decision process [1,2]. This view assumes that engineering design in general, and CE in particular, are fundamentally decentralized processes. Thus, it is important to find ways to coordinate decision making of all participants in any CE activity, while striking a balance between concurrency and coherent action.

The main ideas of this talk are:

- Concurrent engineering and design in general are naturally distributed and decentralized activities.
- Designers act as decision makers, choosing among design alternatives and other activities, eventually resulting in their taking some action.
- Preferences are central to this activity. Hierarchical preferences exist and can be exploited to increase concurrency (i.e., decrease design time) and design quality.
- Design processes result from "preference-guided" actions taken by designers.

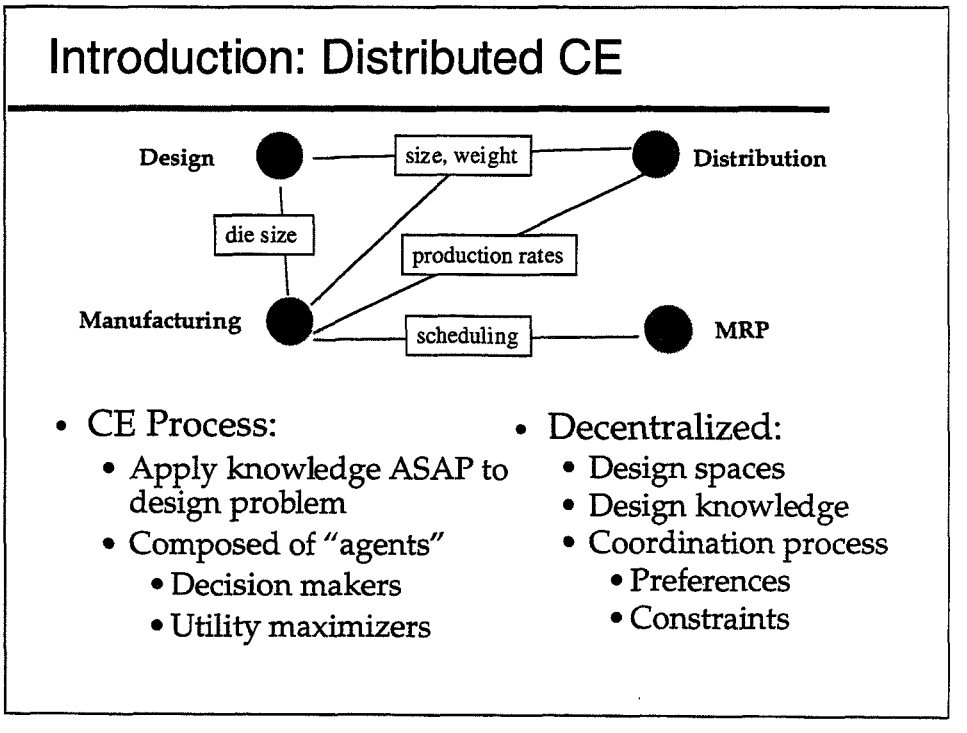


Introduction: Distributed CE

A CE process consists of a variety of agents (who may be human or artificial), that contain local knowledge, preferences and data. The agents each maintain a private view of the emerging design: this is a manifestation of the distributed, decentralized nature of any design or CE process that contains multiple agents. This view of the design -- the design space -- is represented in our work by a (possibly large) set of attributes that describe important features of the artifact being designed.

While the CE process is decentralized, there is still need for agents to communicate. The CE *network* we propose is based on constraints and attributes. Agents are linked via constraints, which are defined over attributes. So, if an agent has an attributed named "weight" and there is a constraint defined over weight, such as "max weight < 90 lbs," that agent will communicate its weight attribute to all other agents via this constraint.

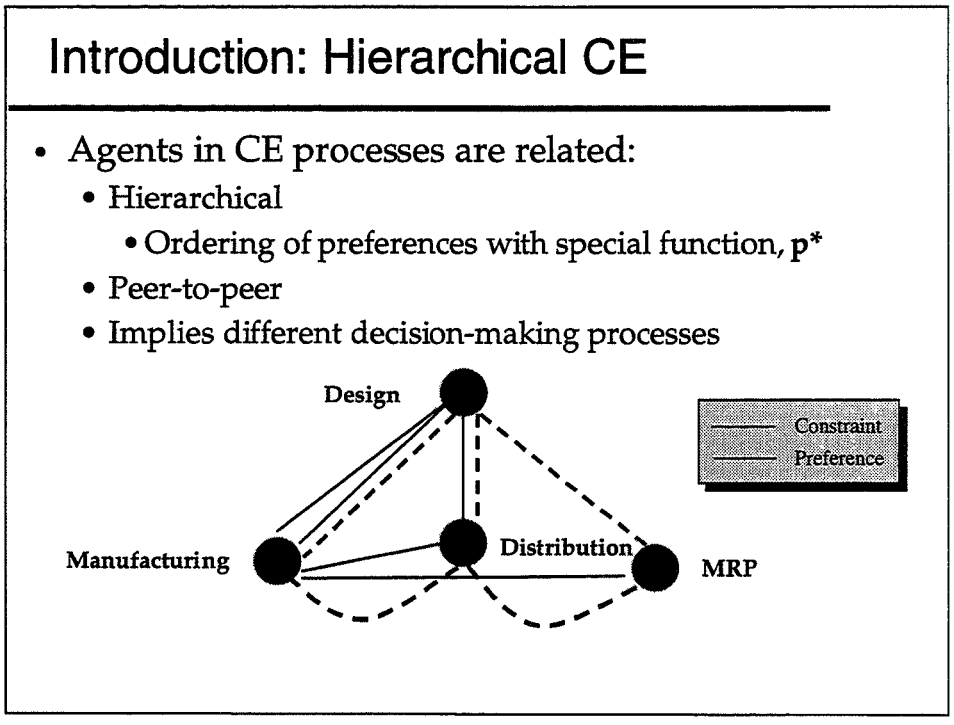
We view agents thus as decision makers: they try to choose assignments to the set of attributes they maintain so as to maximize their preferences. Since agents may have conflicting preferences, a coordination process is needed. The assignment of attribute values is conditioned on both constraints and preferences.



Introduction: Hierarchical CE

To control decision making in a large organization, designers (agents in our CE model) are organized hierarchically. Some agents are responsible for some decisions, and it is usually expected that the results of these decisions are used by agents lower in the hierarchy. We have adopted a general model of hierarchy, where the preferences of *supervisor* agents (those higher in hierarchy) have higher priority over those lower in the hierarchy. We define:

- p*: the function that defines priority
- Simple version: p* is a lexicographical order on preference application for agents.
- Among peers (agents at the same level in the hierarchy), the preference priority function cannot exist. So, these agents use group decision-making processes (e.g., Nash or voting).

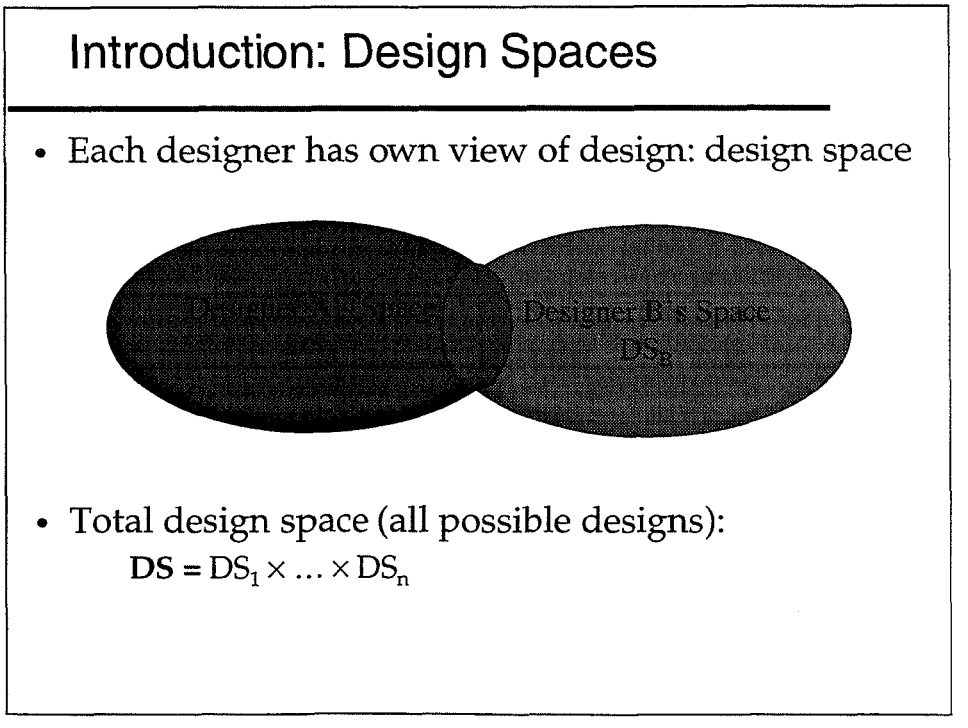


Introduction: Design Spaces

Important note about representation: As mentioned earlier, we assume a distributed view of design:

- No single shared representation (e.g., no blackboard)
- Communication among agents occurs only for those things that need to be shared for decision making. Communication is directed by constraints.

The action is in the interface: that is where agents need to coordinate their decision making. This is because attributes (as reflected in either the constraints or the preferences of individual agents) are shared, and thus are possibly in contention.



Introduction: Distributed CE Advantages

The view of CE, and design in general, as a distributed, decentralized process has many advantages.

Introduction: Distributed CE Advantages

- Better representation of problem
  - Recognizes distributed knowledge, control
  - Exploits inherent parallelism
    - Reduced design time
- Robust against changes in organization
  - Minimal shared information (information hiding)
  - Participates come/go as needed without undue impact
- Optimization
  - Achieve "optimal" results (where possible)
  - Key: shared preference structure, decision processes

Modeling: Decision Makers

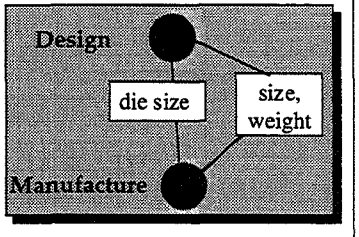
As shown in the next few slides, designers can be modeled as decision makers. They are primarily concerned with assigning values to attributes, typically by exploring various design alternatives (e.g., exploring the design space).

Designers solve an optimal-choice problem: choose the best assignment of attribute values, which is the optimal (or satisfying) design. Sometimes this is possible, often it is not: in this case, the agent acts on the best information it has to choose the best design alternative possible. Designers effect the design state, however, by performing some action. As described later, we are ultimately concerned with the action selected by a designer to make a change in the design state.

It is important to note that *assigning values to attributes* is an abstraction that includes a wide variety of design activity, from selecting a component in a catalog to doing creative exploration of the design space.

Modeling: Decision Makers

- Designers: decision makers
  - Autonomous: make decision over own design space (DS_i)
    - *Exploring the design space*
  - Task: choose best design from DS_i via action(s)
    - Implies preferences & rational decision making
    - May have to coordinate decision making with other designers



Modeling: Decision Makers

In formulating the decision problem, we model design possibilities (generally, the feasible alternatives) as "outcomes," over which the preference function is defined. We further assume that designers -- agents -- have regular preferences that can be represented as a utility function.

Modeling: Decision Makers

- Outcomes: possible designs
  - Created/explored via "know how" or algorithms
    - Denoted by: θ_i
  - Influenced by: constraints, preference structure
- Decision-theoretic model
  - Preferences have structure: utility function
    $U_i: DS_1 \times \ldots \times DS_n \rightarrow \mathbf{R}$
  - Can optimize w.r.t. U_i
    - "Globally" (under certain conditions)
    - Locally

Modeling: Decision Makers

The outcome of a decision is an action that an agent takes. Thus, we are ultimately concerned with choosing the best action that results in the best design. The range of actions an agent may take is very large: it can choose a component, run an analysis tool, or simply wait to hear from other agents about what decisions they made.

The notion of*designer-as-agent* is summarized by the function given in the slide. This function describes the information used by the agent in its decision making and shows the result as a choice of action.

Modeling: Decision Makers

- Actions: designer must take some action
  - Forces convergence to a solution
    - Examples:
      - Remove a design from design space
      - Choose a design from design space
  - Assume: designer has set of actions
    - Consistent with "design knowledge"
    - Denoted by: Act_i

$A = designer_i(\mathbf{DS}, U_i, \theta_i, Act_i)$

Where $A \in Act_i$

Modeling: Mediators

Decision making is not the only activity that occurs in a CE process. There are any number of tools that are used to analyze designs, run simulations, and so forth. The key difference between these tools and agents is that the tools make no decisions about what actions to take, i.e., how to modify the design space.

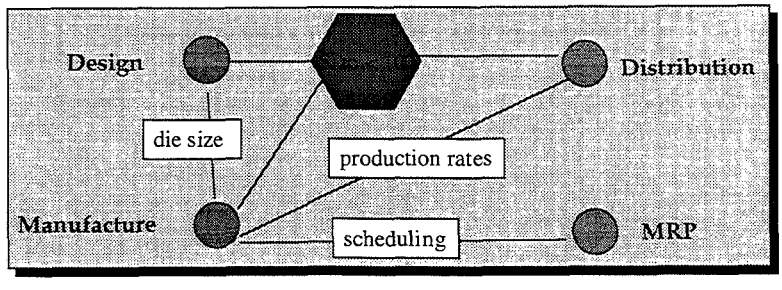
Similarly to the designer model, we define tools as *mediators*. Note that mediators are part of the network.

Modeling: Mediators

- Other activities in CE process: analyze, evaluate, ...
- Mediator: similar to decision makers, but
  - Take no *actions*

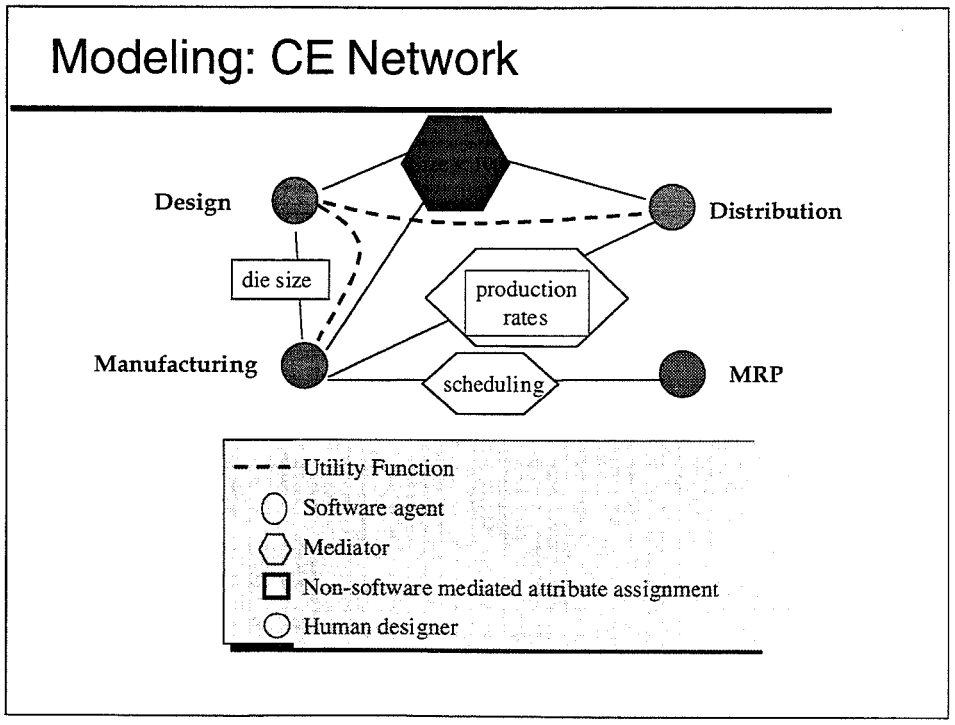
$DS_j = \text{mediator}_j(\mathbf{DS}, \theta_j)$

Where $DS_j \subseteq \mathbf{DS}$



Modeling: CE Network

The slide below casts the CE organization shown earlier in this presentation as a set of mediators, designer (agents), constraints and preferences.



Software Agents: Definition

As we are interested in providing as much computational support for the CE process as possible, we now define designer agents and mediators as software agents. (We do not prohibit humans as designer agents.)

A model of software agents that we use is based on the notion of rational decision maker. The software agent can be viewed as having mental states, which represents the information needed to make a decision.

We have intentionally made parallels between software agents described here and the model of designers in previous slides: we are establishing that the agent model is appropriate for CE tasks, and thus, software agents are a natural outcome of this modeling perspective.

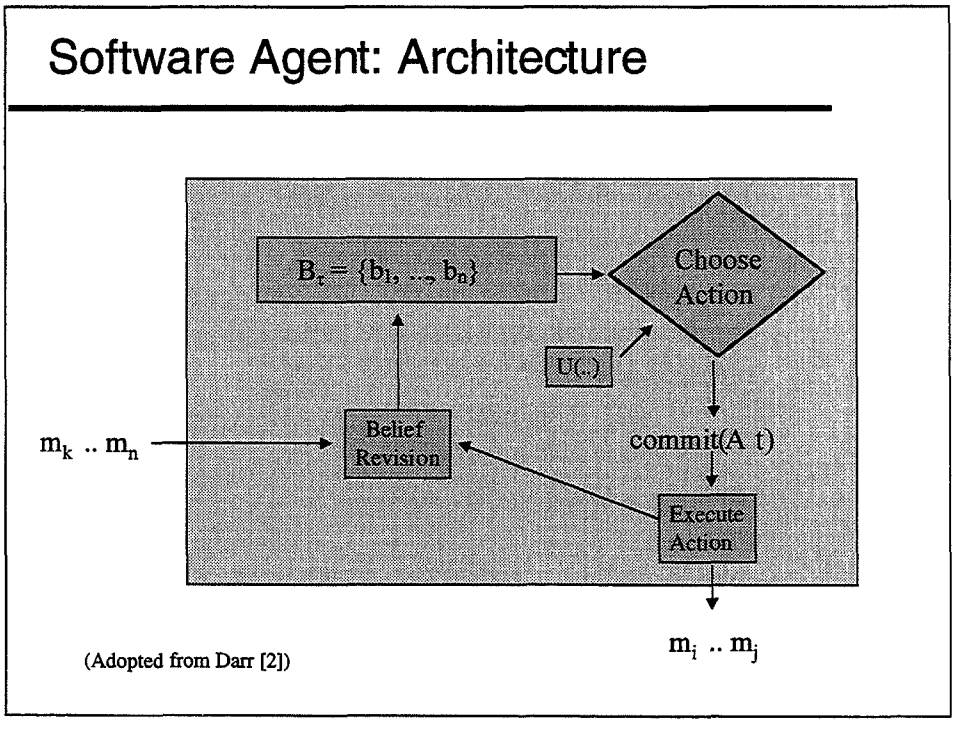
Software Agents: Definition

- Autonomous rational **decision maker**
  - Utility maximizing
  - Optimal choice problem: $A \in Act_i$
    - "Best design" choice
- Mental states
  - Preferences over designs, actions
  - Beliefs
    - Itself (U_i, θ_i, Act_i)
    - Other agents & mediators
- Interacts with "environment"

Software Agent: Architecture

A schematic of the agent's decision making process is given in this slide: this is a schematic representation of how the designer function could be implemented. In addition, the schematic shows how the agent interfaces with the network. The designer sends and receives messages (denoted by m_x) that correspond to attribute values and preference statements.

The utility function used in the schematic (U(..)) takes as input preference functions of the designer agent's supervisor, and is required to obey any restrictions imposed by p*. Thus, we can ensure that the behavior of the designer agents is consistent, as they will value different outcomes (designs) in consistent ways. For example, if a designer agent decides to eliminate some portion of the design space, it can be sure that this decision is consistent with the preferences of its supervisor.



Design Process: Overview

So far, we have shown how individual designer agents make decisions. Most design processes, however, consist of more than a single decision, implying a design process where decisions are made over time. Our model currently does not support explicit reasoning over time, yet we are able to construct design processes by using the principles outlined thus far. We show this in a simple example in the upcoming slides.

In our design processes, we attempt to exploit concurrency by maximizing concurrent decision making by agents. We also assume that the designer and mediator agent organization are hierarchically arranged (although this is not necessary).

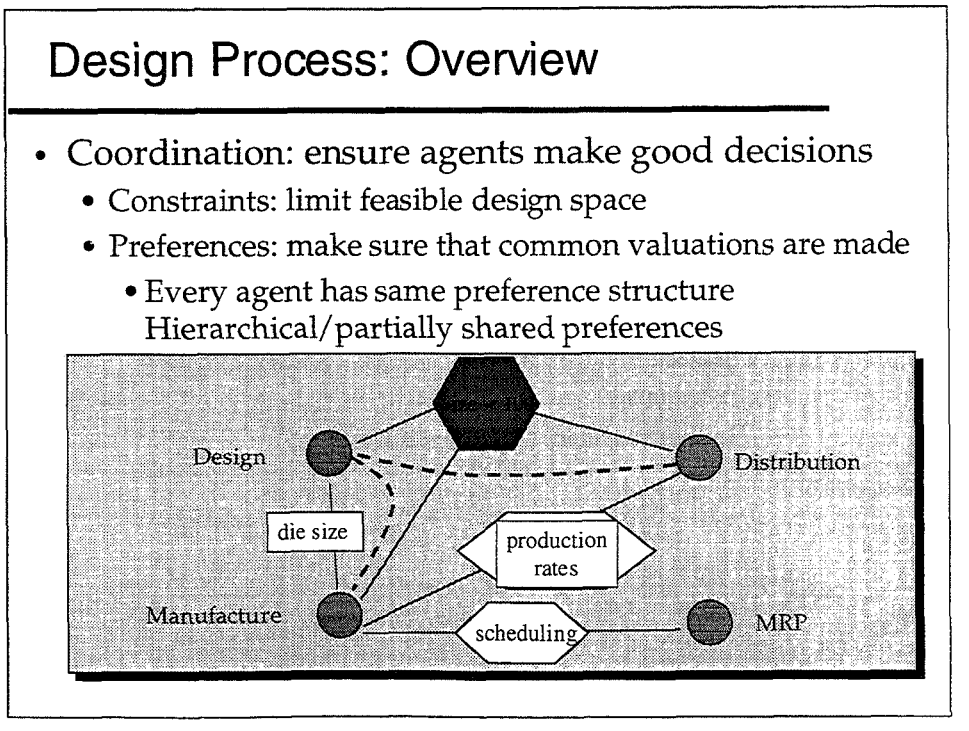
Design Process: Overview

- Context: Decentralized decision making
  - Agents and mediators operating concurrently
  - Need to ensure that agents are making "good" decisions
  - Need to achieve various scoped objectives
    - System
    - Subsystem
    - Individual agent (e.g., wants to do well)

Design Process: Overview

We have found that preferences are a powerful way to coordinate multiple agents, leading to design processes. The basis for design processes using our CE model is the following:

- Establish the network by setting up constraints and distributing preference structures.
- Agents then send messages to determine feasible design spaces.
- Once feasibility is established, the agents then attempt to find the best design they can.



Design Process: Example

The next two slides present an example of a design process, albeit on a very simple problem. The design objective is to select two components that are compatible, as expressed by the constraints, with the best possible utility (from the possible choices). For this simple example, we assume that the utility functions are the same for both agents.

An interesting point in this example is that we have defined utility functions over various design states. So those design states that are "consistent" (each constraint has at least one solution) are preferred to "decomposable" states (where all remaining parts are guaranteed to be in at least one feasible solution). Through these very general statements about *design states*, rather than specific design steps, we can induce an effective design process.

The design process, then, is the following:

1. Make the constraint network consistent.
2. Make the network decomposable.
3. Pick a solution.

Design Process: Example

- Task: constraint problem w/utility maximizing
- Design organization:
  - Two agents: Battery, Starter
    - Actions: select a component, eliminate a component
  - Mediator: Power_Balance
- Design process:
  - Reach consistency, then decomposability
    - U(Consistency (**DS**)) > U(Decomposable(**DS**)) > (U(¬Decomposable(**DS**)) or U(¬Consistency(**DS**))
  - Choose solution

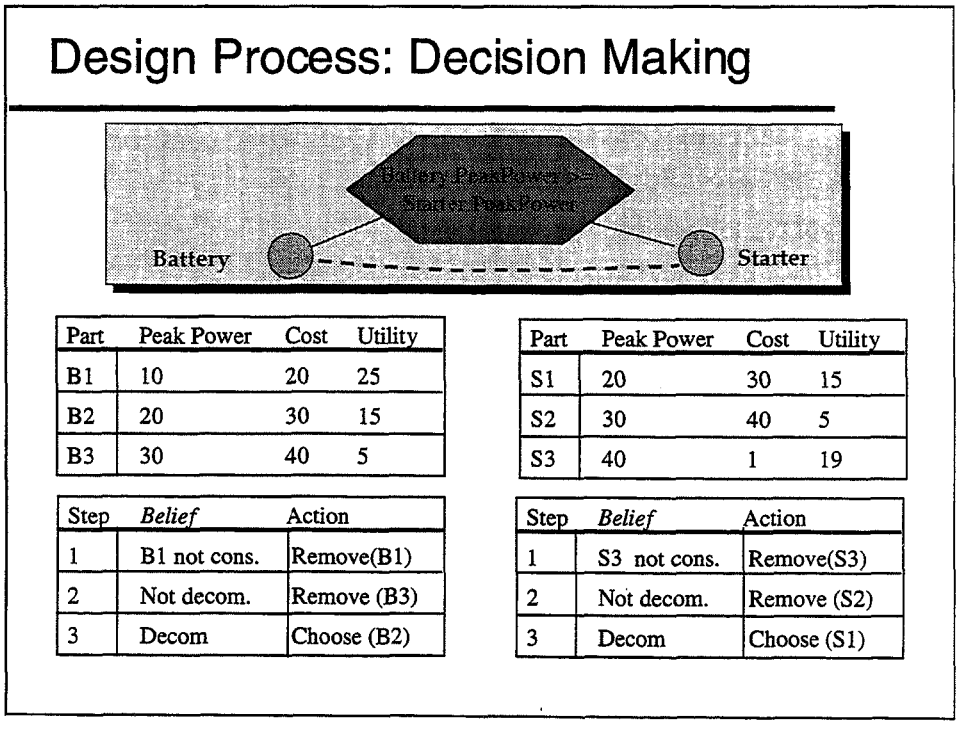
Design Process: Decision Making

The steps used to solve the problem are shown below. The initial components for each designer are listed in the top two tables. Components are removed in a search for a solution. Note that "belief" means the agent, based on its knowledge and communication with other agents, believes the listed proposition to be true.

Here is what happens in each step:

1. Each designer agent concurrently removes parts that are infeasible. The agent makes a determination by sending its attribute value assignments to the constraint, which report feasibility statements.
2. Each designer agent concurrently removes parts that are non-decomposable. If there is a tie, the agent prefers to throw away components with lower utility. The agent makes a determination by sending its attribute value assignments to the constraint, which report feasibility statements.
3. Each designer agent concurrently chooses its highest utility part and returns it as an element of the solution to the design problem.

It is important to note that this design process is heuristic, based on attempting to find a decomposable network and preferring to discard lower utility components. It is possible that backtracking will be needed for some design problems.



Part	Peak Power	Cost	Utility
B1	10	20	25
B2	20	30	15
B3	30	40	5

Part	Peak Power	Cost	Utility
S1	20	30	15
S2	30	40	5
S3	40	1	19

Step	*Belief*	Action
1	B1 not cons.	Remove(B1)
2	Not decom.	Remove (B3)
3	Decom	Choose (B2)

Step	*Belief*	Action
1	S3 not cons.	Remove(S3)
2	Not decom.	Remove (S2)
3	Decom	Choose (S1)

Summary

We summarize this talk by recapitulating the major points:

- Concurrent engineering and design in general are naturally distributed and decentralized activities.
- Designers act as decision makers, choosing among design alternatives and other activities, eventually resulting in their taking some action.
- Preferences are central to this activity, and hierarchical preferences exist and can be exploited to increase concurrency (i.e., decrease design time) and design quality.
- Design processes result from "preference-guided" actions taken by designers.

Additional points are given in the slide below.

Summary

- CE Design Processes:
  - Decentralized decision makers (designers)
    - Design spaces and actions
  - *Coordinating decision making necessary*
- "Designing" Agents and Mediators:
  - Rational, autonomous decision makers
  - Fits CE process "naturally"
- Benefits
  - Potentially faster design processes
  - Scalable
  - Uniform framework for human and software agents

References

1. D'Ambrosio, J., Darr, T. P. and Birmingham, W. P., "Hierarchical Concurrent Engineering in a Multiagent Framework," *Concurrent Engineering: Research and Applications*,, Vol. 4, No. 1, May 1996, pp.

2. D'Ambrosio, J., Darr, T. P. and Birmingham, W. P., "A Constraint Satisfaction Approach for Multi-Attribute Design Problems," ASME DTM, Sept., 1997.

3. Darr, T. P., "A Constraint-Satisfaction Problem Computational Model for Distributed Part Selection," Ph.D. Dissertation, EECS Department, The University of Michigan, 1997.

MultiAgent Systems, WWW, and Networked Scientific Computing

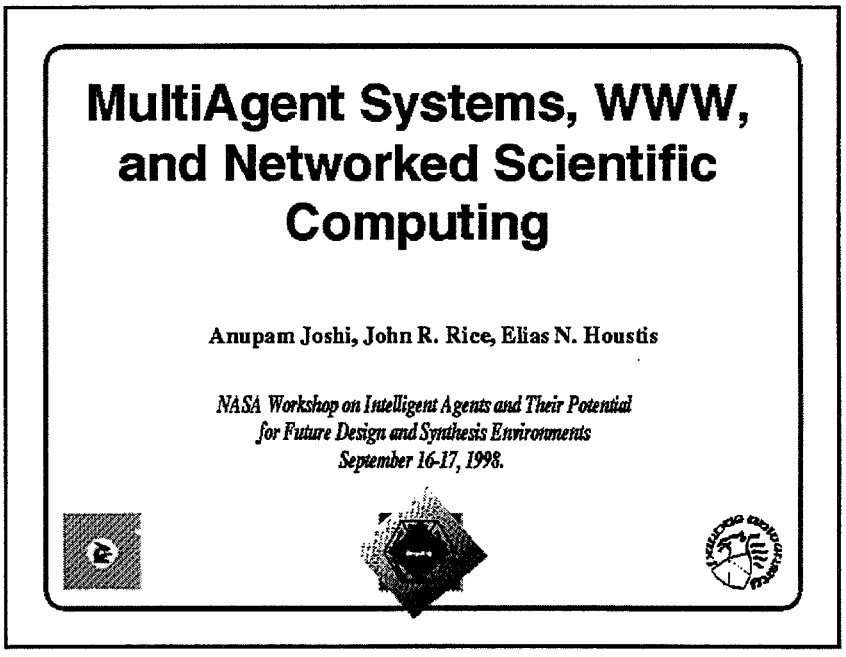
55-62
03257
370168
22p

Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250

John R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

and

Elias N. Houstis
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907



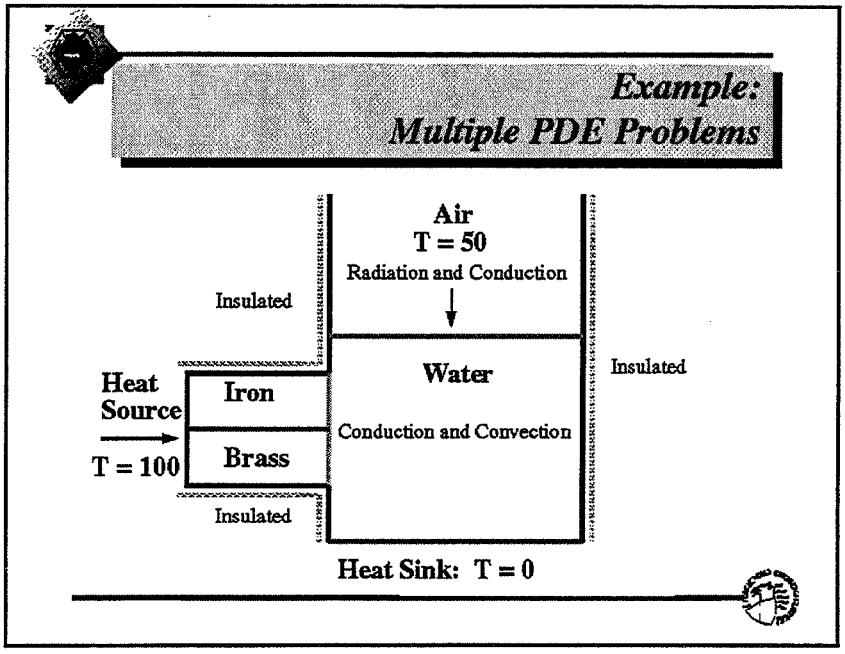
Overview

- The Multidisciplinary Problem Solving Environments Idea
- Enabling Technologies for Building MPSEs
- Background for Building MPSEs in Scientific Computing
- An Agent-Based MPSE Architecture
- *SciAgents* - an MPSE for Solving Composite PDE Models

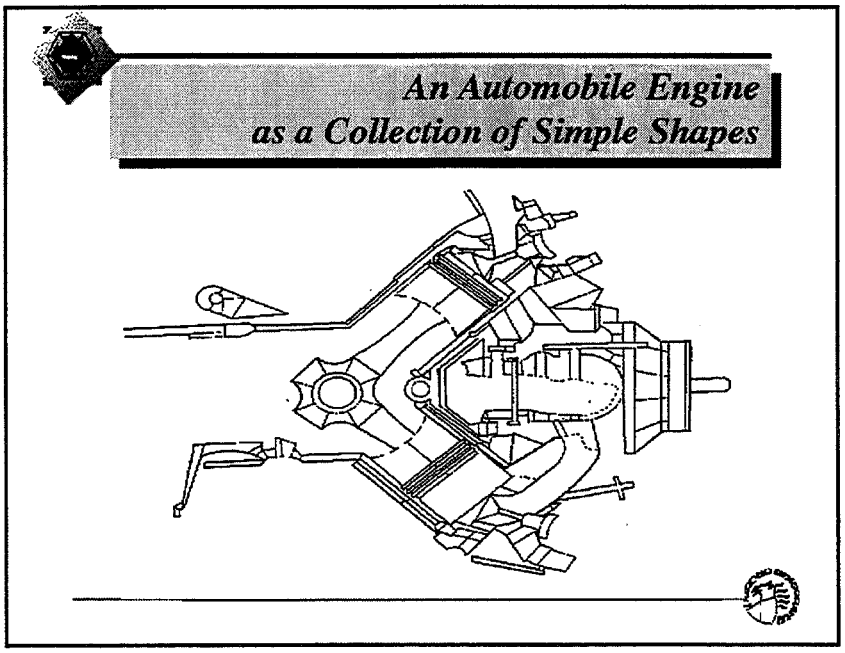
Problem Solving Environments Definition and Examples

- A PSE is a computer system that provides all computational facilities needed to solve a *target class* of problems.
  Facilities include solution methods, selection among them, presentation tools, programming-in-the-large, tools for hardware selection and for parallel and distributed computing (if needed).
- Examples
  //ELLPACK
  Matlab
  Framemaker
  MS Excel

The diagram illustrates a heat flow problem on a system with many materials and boundary/interface conditions. The overall system is quite complex, but can be viewed as a set of interacting entities.



An automobile engine. Its behaviour can be viewed as emerging from the interaction of multiple simple(r) parts, which provides a mechanism to use distributed agent based simulation.



Statements

- Many problems from the physical world are very complicated
- No universal solvers for multiple-domain PDE problems can be built
- Building a PDE solver for a complex problem is very expensive, time consuming, and even unrealistic
- Users cannot wait

How Does the Real World Work?

- A complex physical phenomenon consists of a collection of simple parts
- Each part obeys a single physical law locally
- Different parts work together by adjusting interface conditions with neighbors
- The real world evolves in this way without a "global" controller

Networks of cooperating PDE solvers can imitate this physical situation

Multidisciplinary Problem Solving Environments

- **Definition**: An MPSE is a framework and a software kernel for combining PSEs for tailored, multidisciplinary applications.
- **Another definition**: An MPSE is a software system solving a problem, viewed by the user as a complex, composite one, consisting of a number of smaller, distinct subproblems, each of which is solved by a relatively independent software bearing the features of a PSE.
- The subproblems should be viewed by the user as distinct.
- The subproblems should be related but independent, not simply different stages of the solution of a single problem.

MPSEs Defining Features

- The PSEs solving the subproblems interact with each other (through the MPSE kernel) and the interaction is required to solve most of the subproblems.
- There is an object ("global solution" for the user) obtained as a result of the work and the interaction of the individual PSEs. The user perceives it as the ultimate goal of this MPSE and it cannot be achieved by any subset of the PSE components.
- The MPSE kernel provides a (global) user interface which does not depend on the subproblems and the PSEs participating in the MPSE.

MPSEs
Why Do We Need Them?

- To be able to build software for solving **complex**, **multidisciplinary** (heterogeneous), real-life problems for reasonable time and cost which uses all available variety of computational and communication resources.
- To provide the users with cooperative environment in which they interact and solve (sub)problems through their specific domain of expertise.
- To make possible the reuse and the evolution of the available high-quality applications (PSEs)

MPSEs
Research Issues

- Technologies for building the MPSE kernels

Methods for interaction between the PSEs that are general enough and also allow for dialogs containing specific data pertinent to the problem at hand.
Methods for "upgrading" the existing PSEs in order to be able to interact.
Specific mediation schemes between the subproblems.
Methods for locating the available resources (hardware, software, and others).
Methods for controlling, recognizing, and representing the global solution.

MPSEs Research Issues

- User interface issues

  Abstracting the underlying hardware and communication structure
  Abstracting the multidisciplinary aspect of the problem solving
  Allowing the problem definition and the presentation of the solution in **user's terms**
  Allowing control of the solution process, if desired
  Facilitating the users' cooperation and collaboration

Overview

- The Multidisciplinary Problem Solving Environments Idea
- Enabling Technologies for Building MPSEs
- Background for Building MPSEs in Scientific Computing
- An Agent-Based MPSE Architecture
- *SciAgents* - an MPSE for Solving Composite PDE Models

Review of Prospective Technologies for Building MPSEs

- Distributed Computing and Problem Solving
  Essential for building MPSEs
- Parallel Computing
- Mobile Computing
- Virtual Computing
  Computing on WWW
- Object-Oriented Technology

Agent-Based Computing Defining Properties of an Agent

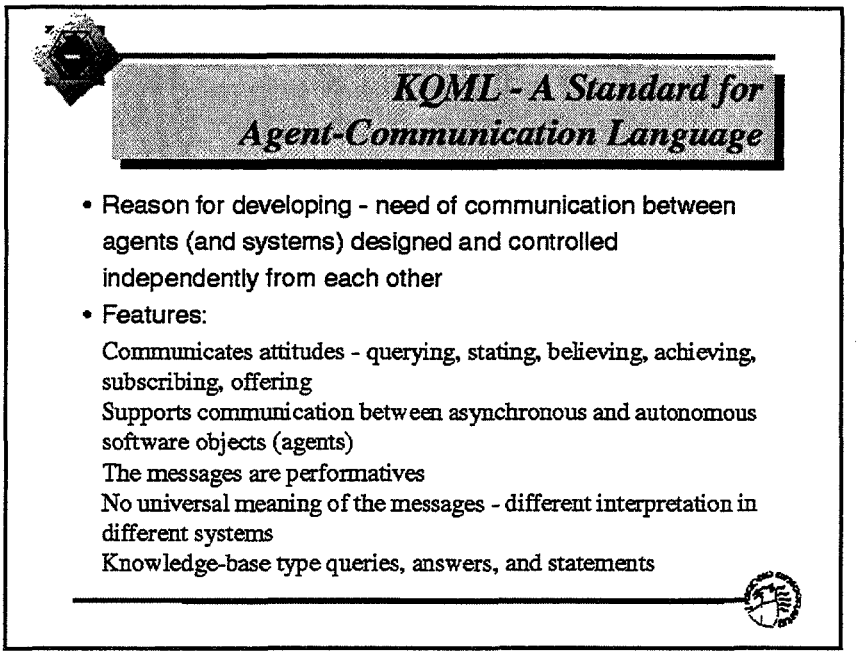
- Autonomy - operating without the direct intervention of humans or others and having some control over the agent's own actions and internal state
- Social ability - interacting with other agents via agent-communication language
- Reactivity - perceiving the environment and responding to changes in it
- Pro-activeness - exhibiting a goal-directed behavior by taking the initiative

Agent-based Architectures: Features

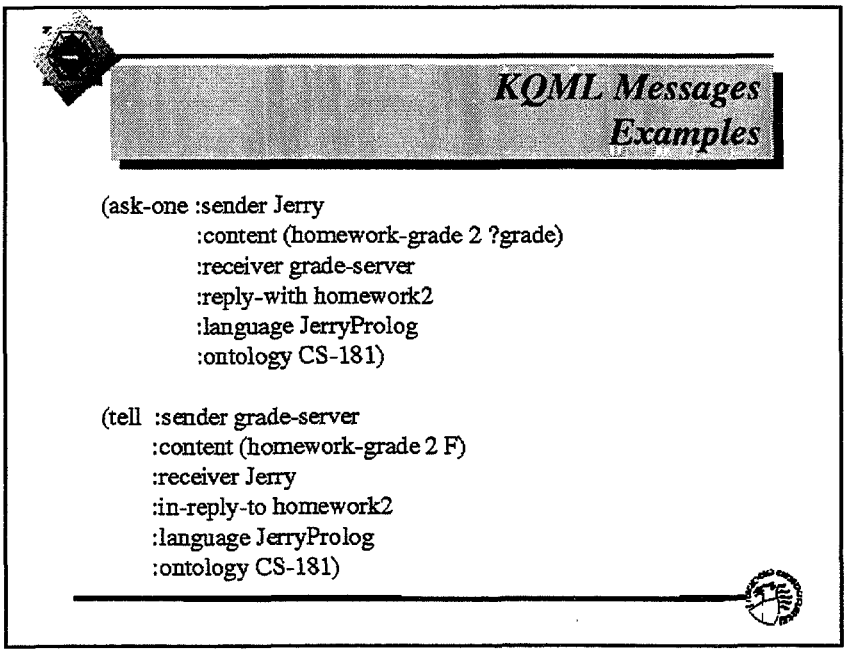
- A step beyond the object-oriented paradigm
- Encapsulation - accessing an agent only through sending and receiving messages
- Visual representation of the agents
- Modularity and flexibility - add/remove agents, move them around
- Extensive use of concurrency
- *Agent Architecture* = methodology for building agents + communication system for agents + rules for placing and executing agents
- Decomposing large tasks into self-contained modules

Multi-Agent Systems Examples and Features

- None known in use in Scientific Computing and Simulation
- Most deal with information systems and services; locating and searching for specific information; exchanging data
- Emphasize the AI aspects of the agent technology - beliefs, thrust, intentions; also security, authentication
- Interesting example: M. Girard's Ph.D. thesis on building multiagent systems



Sample KQML (Knowledge Query and Manipulation Language) messages exchanged between a client and a server. The example illustrates a client querying a server about a person's grades.



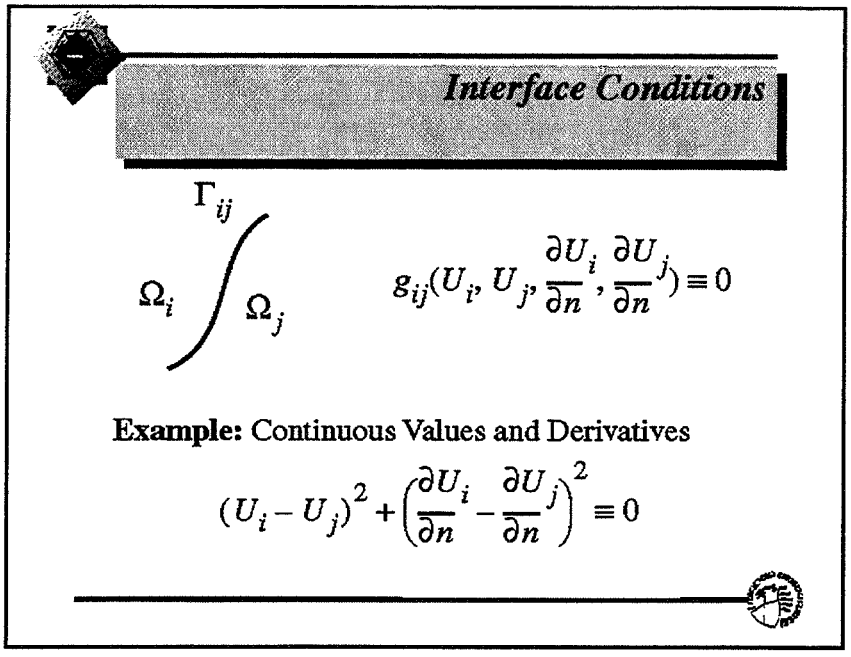
Overview

- The Multidisciplinary Problem Solving Environments Idea
- Enabling Technologies for Building MPSEs
- Background for Building MPSEs in Scientific Computing
- An Agent-Based MPSE Architecture
- *SciAgents* - an MPSE for Solving Composite PDE Models

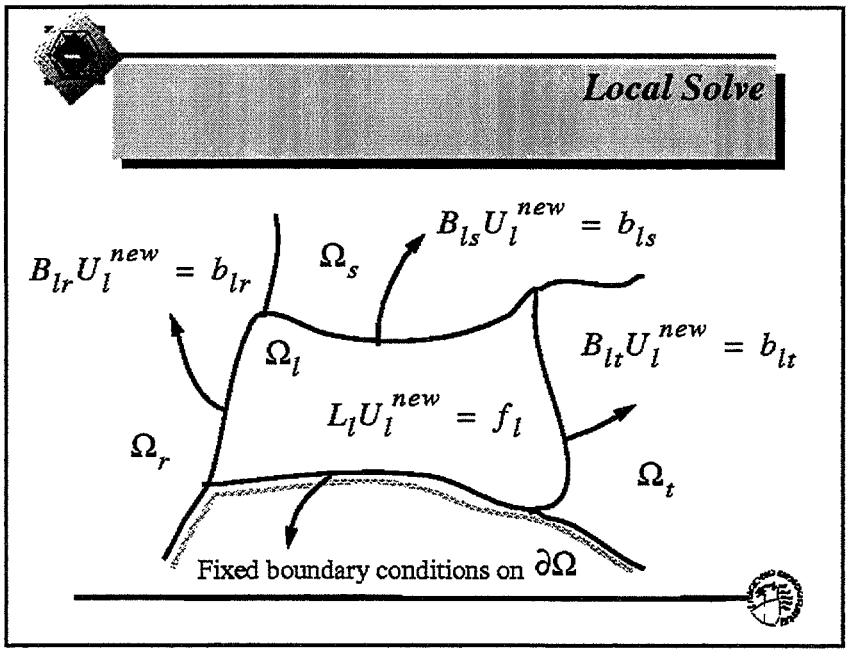
What Did We Have and What Did We Need?

- Available Modules:
  - Simple shapes with one PDE or ODE system
  - Ordinary boundary conditions
  - Reasonably simple forms for the DEs
- Needed Modules and Code
  - Interfaces between DE systems
  - More general approximation schemes along the interfaces
- Must always write code for special, nonstandard situations

Equations describing the interface conditions between two components. The specific example provided shows a case where both the values and the derivatives are continuous across the interface.



This diagram illustrates the "local" nature of each solution, which proceeds with interface values from neighboring solvers from some prior iteration.



The diagram illustrates in an abstract manner the computation process that takes place in interface relaxation.

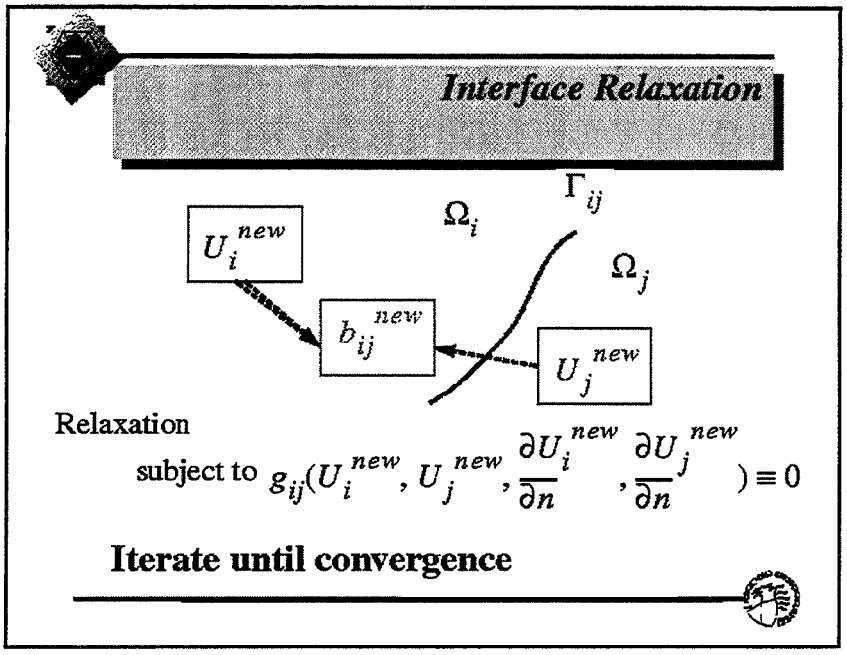
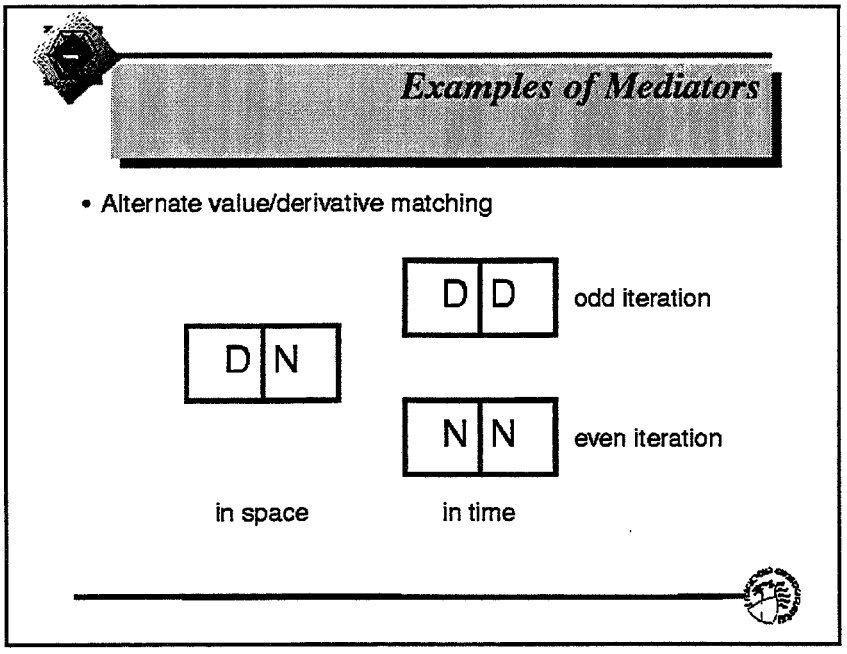
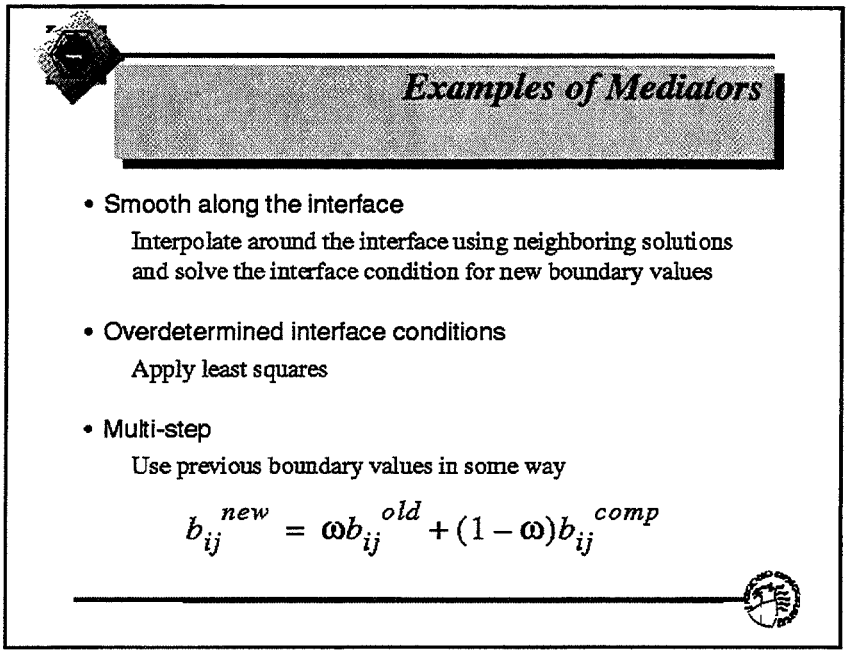
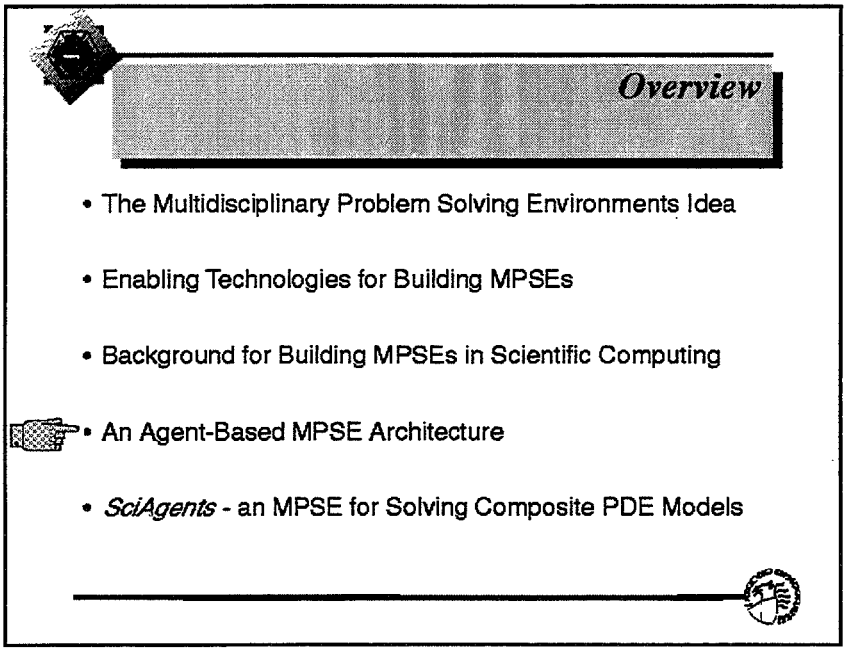


Illustration of a mediation process, where the values and derivatives are matched across the interface on alternate iterations.



Illustrations of some other interface relaxation methods.





Agent-Based Approach to Building MPSE Systems

- A natural idea: Build the MPSE as a multi-agent system
- Have the MPSE kernel provide all necessary functions of interagent communication medium
- Each PSE working on a subproblem behaves as an independent agent, called ***solver agent.***

  In order to achieve this behavior, a piece of software called ***agent wrapper*** is provided, which takes care of the interaction with the other agents and the other aspects of emulating agent behavior. The wrapper encapsulates the original PSE and is responsible for running it and for the necessary interpretation of the results.

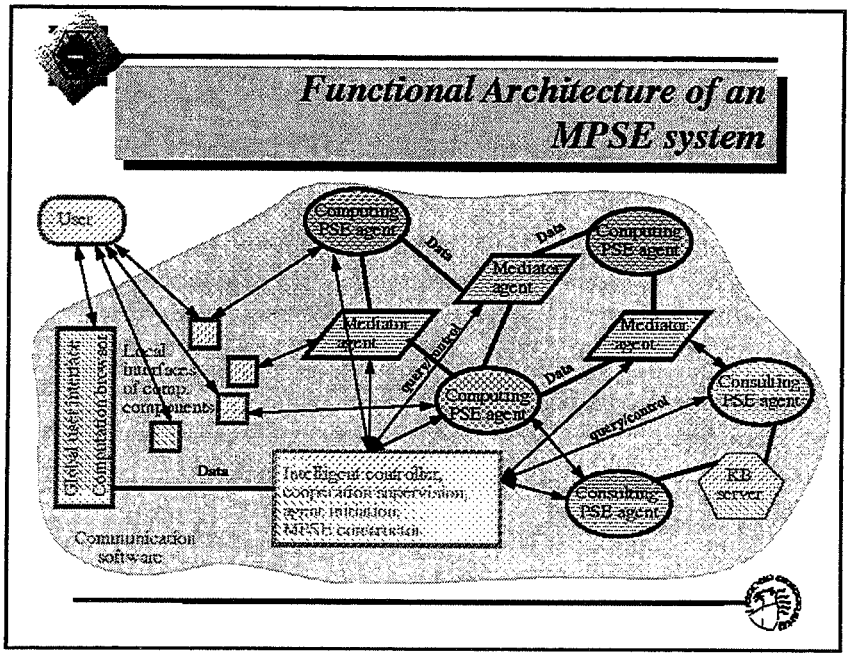
Agent-Based Approach to Building MPSE Systems

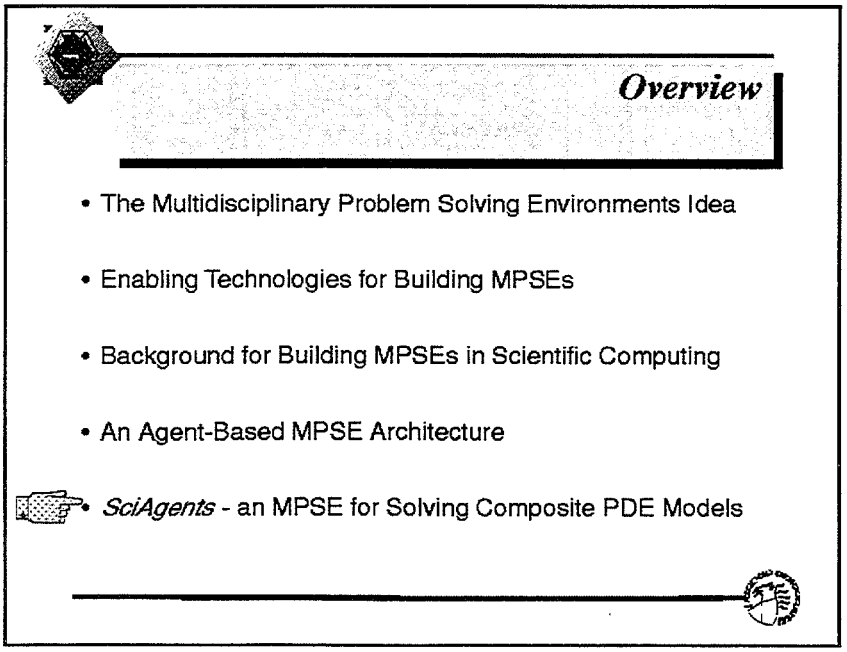
- Any interaction (mediation) between any two solver agents during the solution process goes through ***mediator agents***

  The mediators process the data exchange between the solvers in order to enforce some global or local solution strategy, recognize (locally) that some goal has been achieved, and provide information to the global controller
- Advantages of this approach

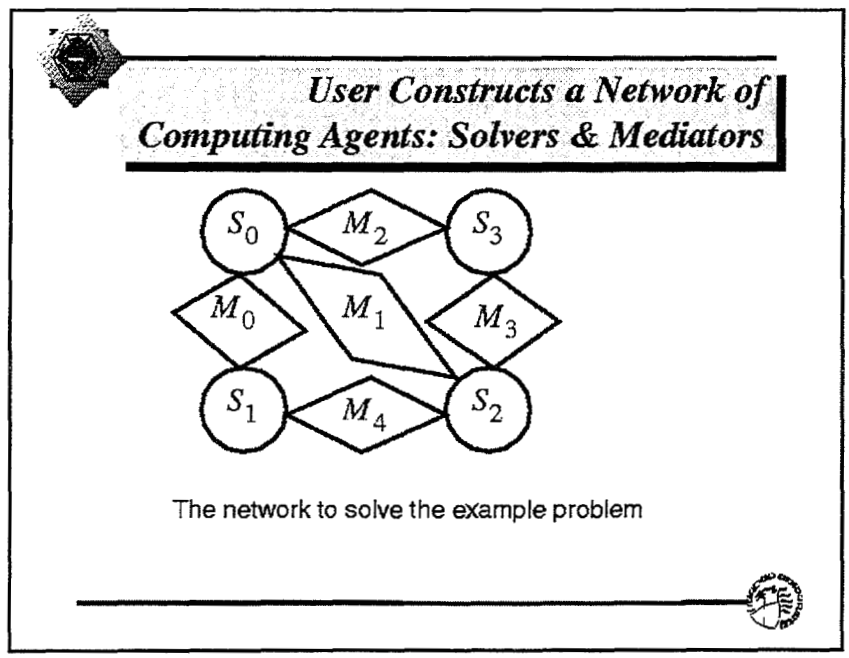
  Handles well the independence and the interaction of the PSEs
  Allows for easy software reuse and evolution of the PSEs
  Allows for "seamless" computing of the global solution
  Allows for the use of techniques for interagent communication

An illustration of the functional architecture of the SciAgent system. We can see multiple solvers, mediators and recommenders interacting.

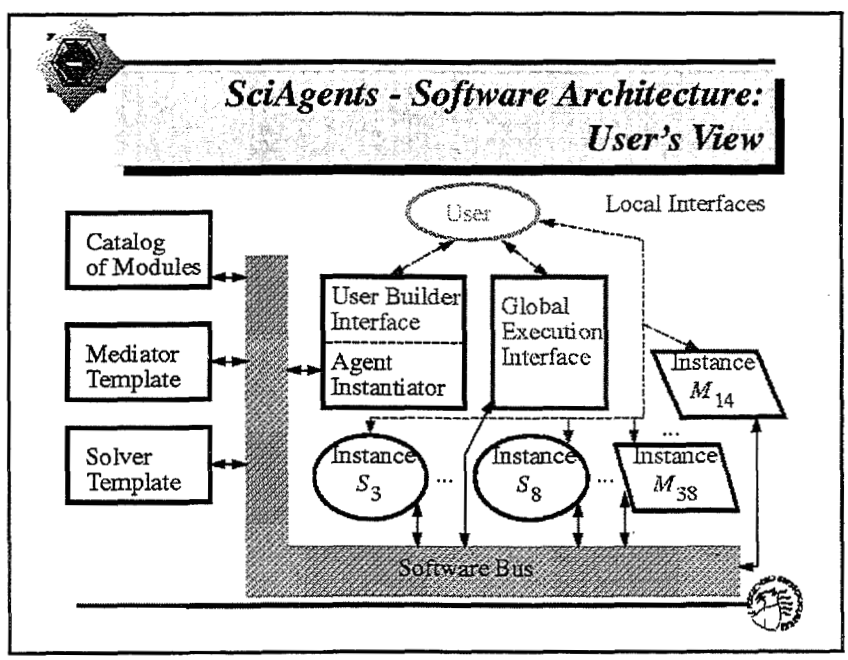




The user's view of the SciAgents system. The domain expert visualizes the system as a collection of solvers and mediators and specifies the interaction between them.



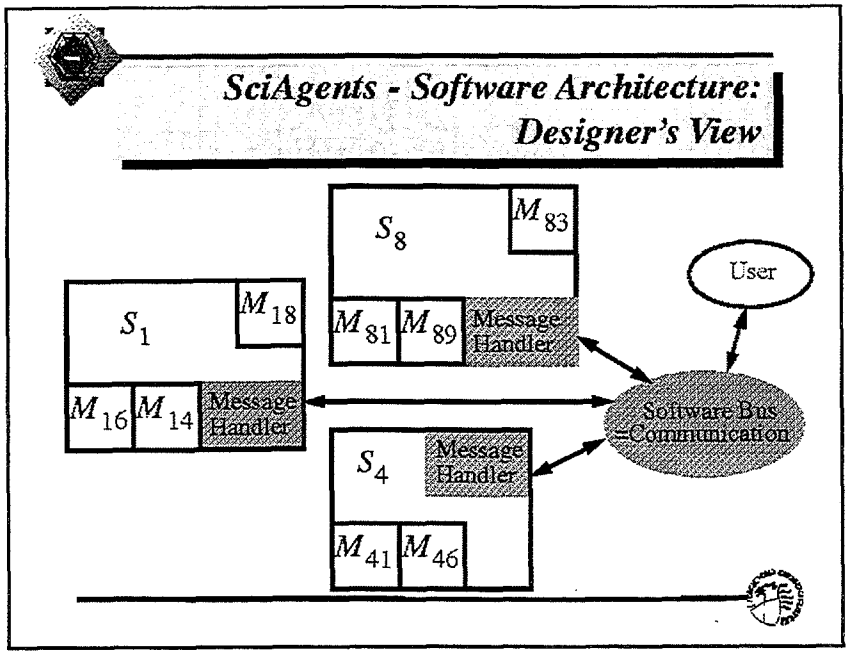
The user's view of the SciAgents architecture. S/he interacts with both the global interface, as well as the interfaces (if any) of the solvers and mediators. The components are connected via a software bus like mechanism.



SciAgents - Software Architecture: Major Features

- There may be many users using a single network of agents. A user might build only part of the network
- Only the instances of the agents compute, the user builds a network by instantiating them
- The agents communicate sending and receiving messages using the software bus as a global message handler
- The agents are aware of their environment. If a necessary computing agent has not been instantiated, the computation will be suspended (locally)
- The agents use their interfaces to communicate with the user

The abstraction of SciAgents as seen by a designer. Every physical component has a solver and some mediators defining its interface conditions with other components. There is also a message handler, which acts as a stub for the software bus functionality.



Interagent Coordination During the Solution Process

- The mediators govern the computations locally; the global solution "emerges" from the local computations as a result of local independent decision making and communication
- All agents *actively* seek and exchange data through KQML messages
- Interagent coordination

  Initial data exchange
  Exchanging iteration data
  Local suspension and resumption of the computations
  Global control and steering of the process
  Decisions of the global controller

SciAgents Prototype Implementation

- Working, reasonably extensively tested version

  More than 25 problems with many variations each

- Multithreaded agent implementations (wrapper, mediator, and global controller) for Solaris environments

  Communication and computation threads in every agent

- KQML-based communication; used available KQML libraries; name service by an HTTP-server
- //ELLPACK is the only available solver repository

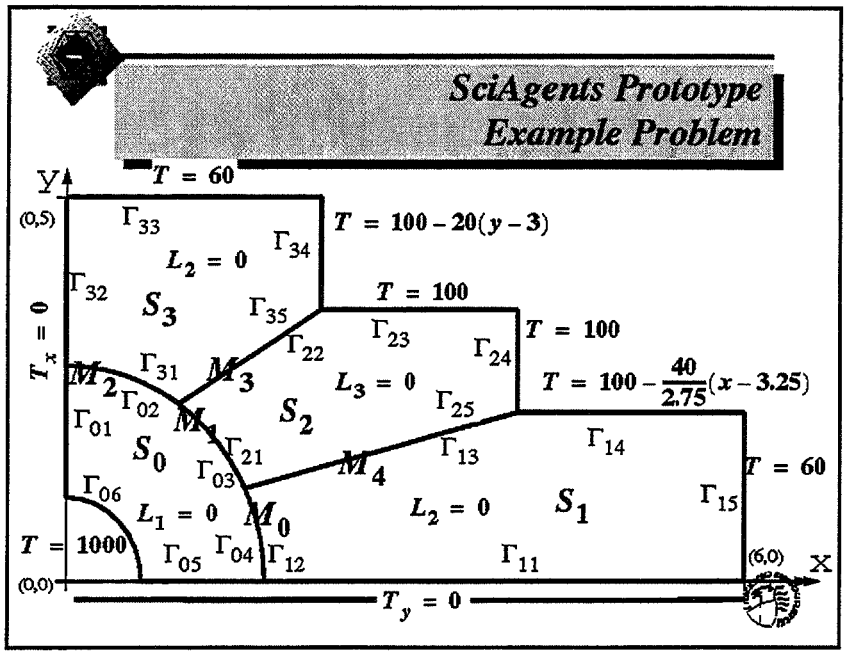
SciAgents
Prototype Implementation

- Limited number of mediator templates
  Adding a few currently
- In search of the "right" tool and design of the global interface - currently input from a script
- Very minimal changes and additions to //ELLPACK - less than 1000 lines of code
- Running on a cluster of workstations - RPC-based remote instantiation

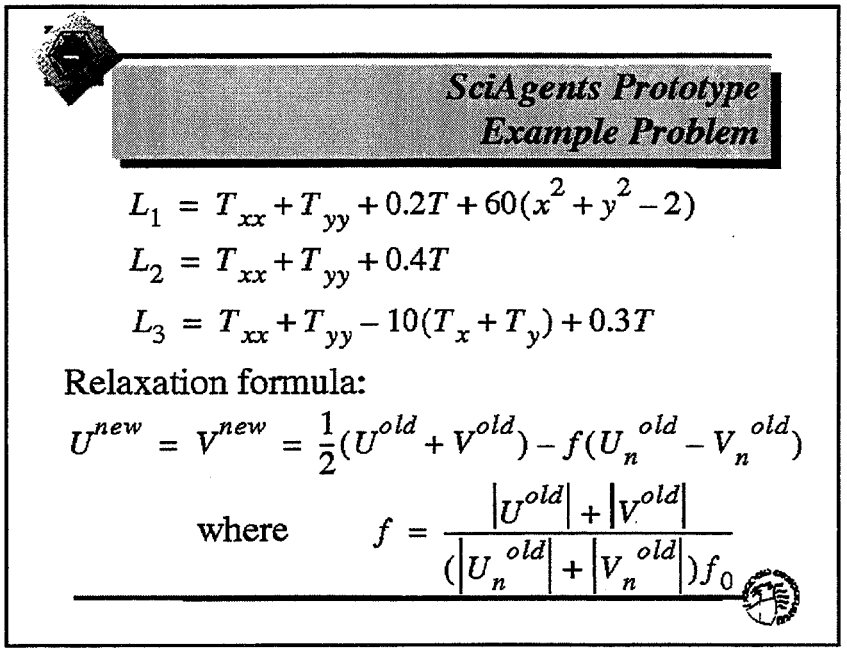
SciAgents
Prototype Implementation

- Asynchronous global control and steering of the computations; "stop-and-go" at any time
- Demonstrated locality of the decisions, automatic local stopping and restarting the computations depending on the local convergence
- Demonstrated basic agent characteristics - encapsulation, pro-activeness, goal orientation, complex interactive behavior
- Easily extendable with new solvers and mediator schemes
- Demonstrated applicability of the interface relaxation technique

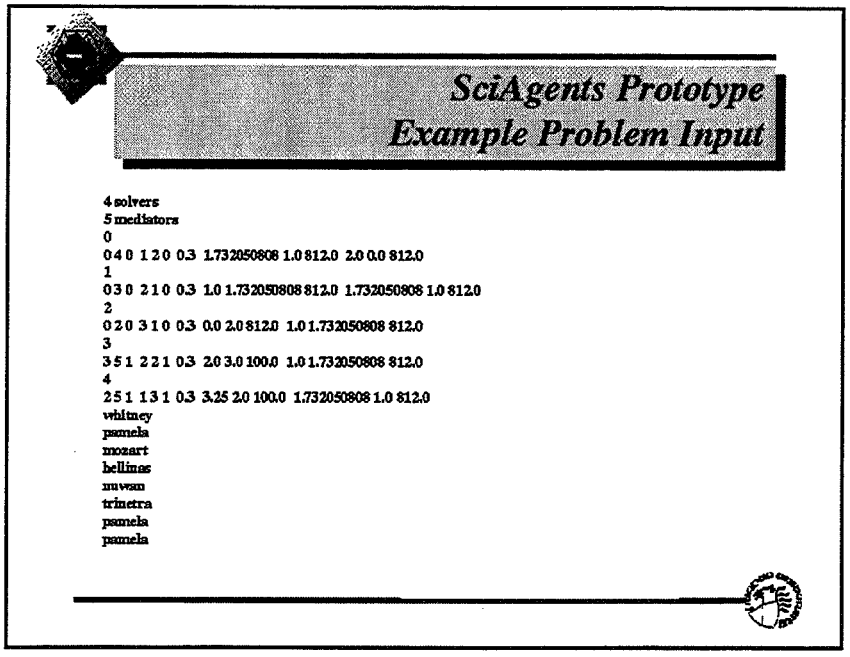
An example problem, illustrating heat dissipation from a heat source which is centered at (0,0). It is surrounded by two layers of materials of different geometries and conductive properties, which are in turn exposed to the atmosphere. The boundary conditions are specified. Each piece shows its interface components and the mediators(M) and solvers(S).



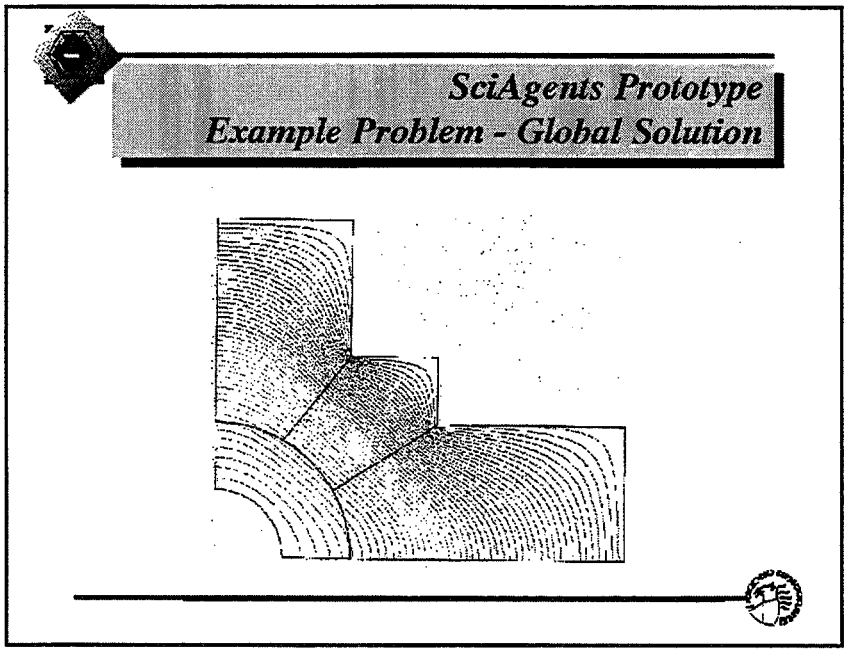
Specifies the system of differential equations arising from the example in the previous slide. We also describe the relaxation technique used by the mediators.



An example input file to the SciAgents system. We specify the number of solvers and mediators. For each mediator, we specify the interface components it connects across various solvers. Finally, we specify the various machines on which these solvers and mediators will run.



The solution to the system. Note that the solution obeys the continuity conditions imposed by the simulation.



36-61
032520
37070
24P

Some Standards Activity in Agent-Based Learning and Virtual Consultation for Manufacturing

Manuel Aparicio IV
IBM Knowledge Management and
Intelligent Agent Services
Research Triangle Park, NC

Page intentionally left blank

Some Standards Activity in Agent-Based Learning and Virtual Consultation for Manufacturing

Manuel Aparicio IV
Chief Scientist, IBM Knowledge Management and
Intelligent Agent Services, Research Triangle Park, NC

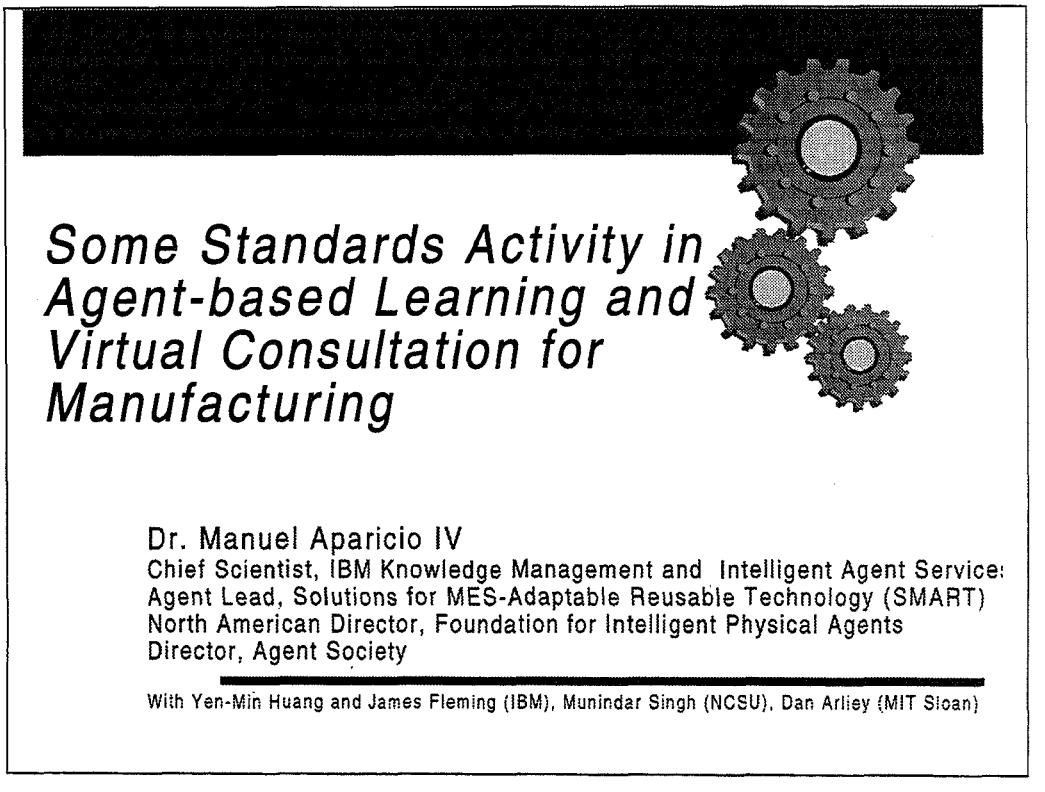
This presentation represents several interests in intelligent agents.

First, it will introduce one of IBM's primary commercial initiatives in intelligent agents. The market is still emerging and is a hard market, but agent technology is providing real value. IBM provides such technology through our Knowledge Management and Intelligent Agent Services.

Second, we are providing the agent technology within the SMART consortium, focused on advanced manufacturing, particularly for lower MES and the make-side of supply-chains.

Third, we are active in several international, multi-organizational agent organizations. FIPA is becoming the de facto and de jur standards body for agents, while the Agent Society is a trade organization and excellent Web site for more information.

Thanks to Yen-Min and Jim for our joint services and SMART work. In the latter, Munindar has been an enormous source of ideas and research pre-work of what we have developed, particularly about supply-chain commitments. Also, Dan has helped us understand human decision making and how agents should help.



Agenda

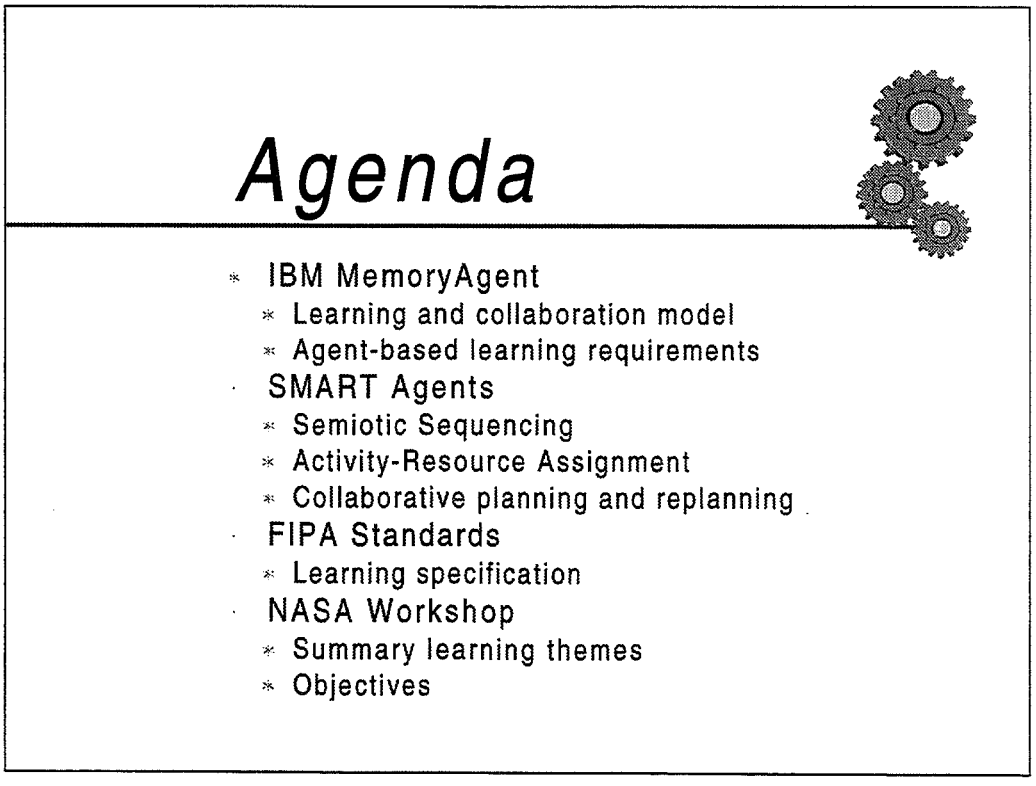
Following these interests, the presentation agenda will introduce each. Agent-based learning will be a primary theme, not only because of our work, but because all the other presentations so far have mentioned agent learning, which I would like to emphasize.

First, our services work this year has concentrated on MemoryAgent, which includes the core learning technology as well as a collaborative model for organizational knowledge management. At a deeper technical level, I will also introduce a list of requirements for agent-based learning.

In the SMART consortium, we have applied MemoryAgent to semiotic sequences and activity-resource assignment, and are now looking at collaborative planning.

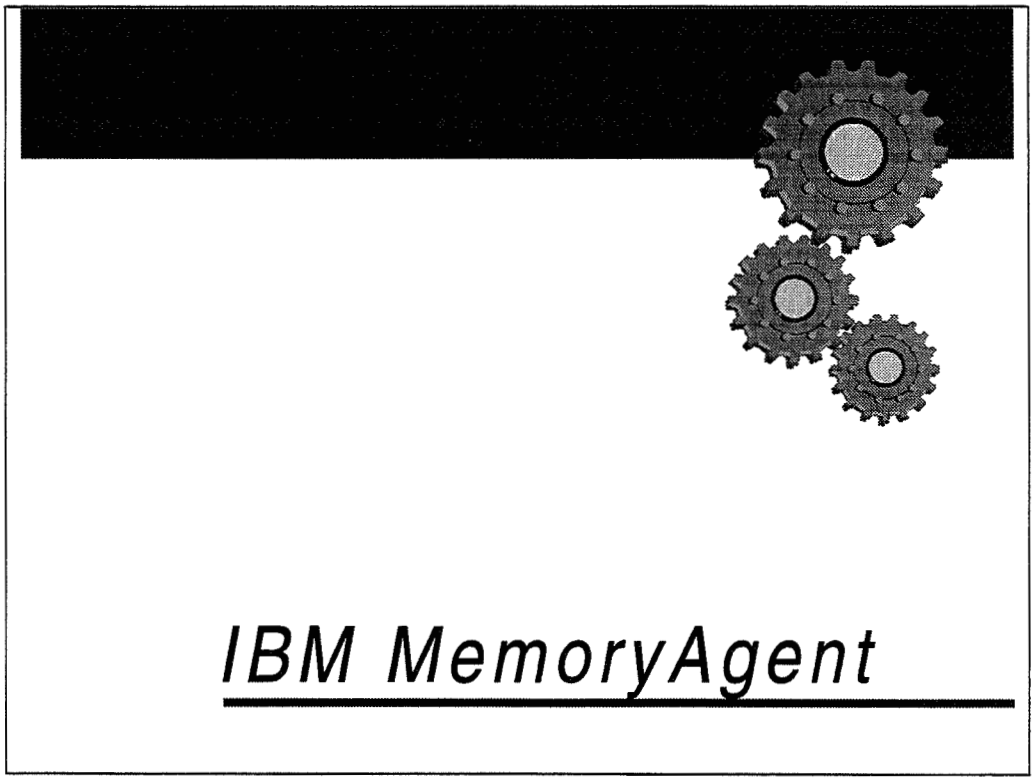
For interoperation of MemoryAgent with other agents, we are working with other members of FIPA on human-interaction, user personalization, and learning services.

I would then like to summarize the theme of agent-based learning running through this workshop and make some final comments toward the workshop's objectives.



MemoryAgent

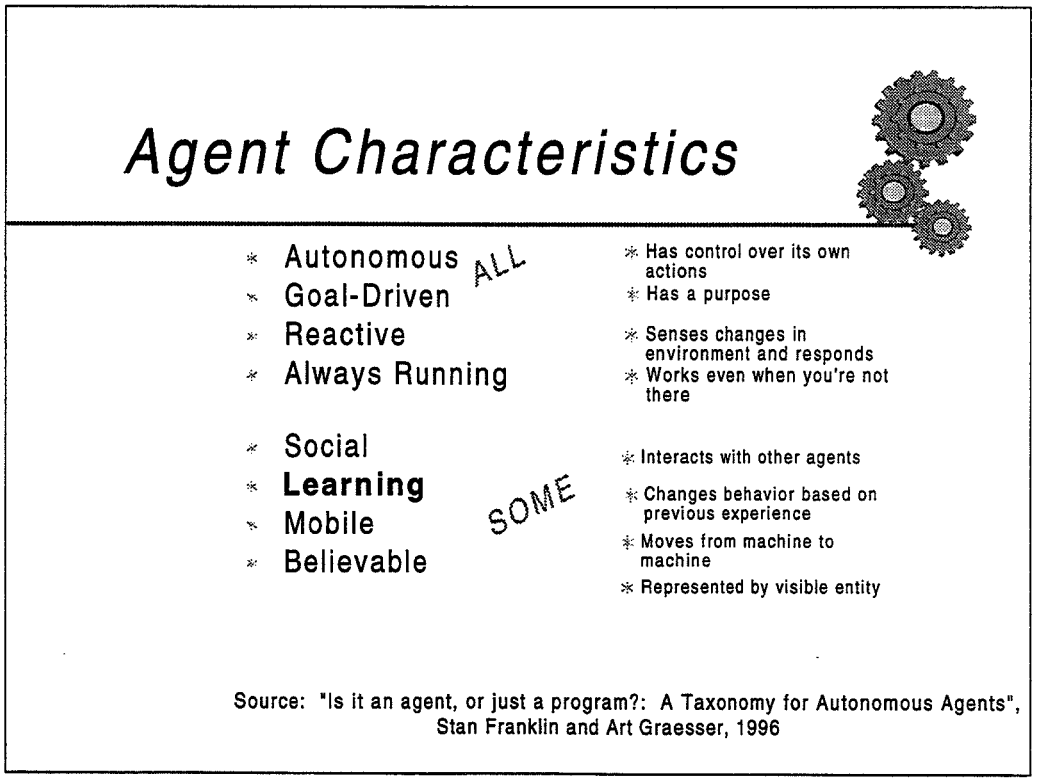
First, MemoryAgent.



Agent Characteristics

Franklin and Graesser established a very well known characterization of intelligent agents. We use this list in IBM as part of our education efforts and in SMART to track our progress along the different dimensions of agency.

While they listed learning as an optional characteristic, we noticed this starting to become the strongest customer requirement at the end of last year. Agent learning is commonly understood as a requirement by the common person, "I want an intelligent assistant that watches me and learns how to help me." Of course, a more rigorous definition is needed to distinguish learning from simple customization and adjustments on the one hand and from pure magic on the other. The point now is simply to focus on learning as a key attribute of increasing importance in agents.



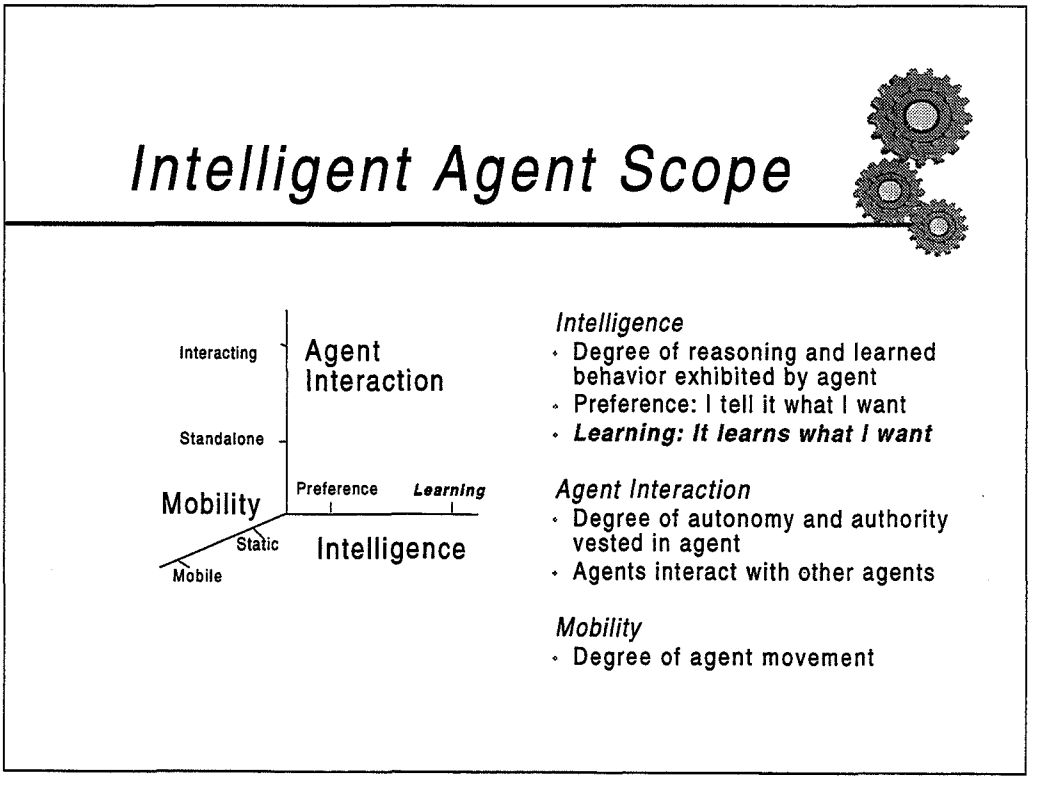
Intelligent Agent Scope

Another way to introduce and understand agents is to consider their power along three dimensions.

I will not be discussing mobile agents such as IBM Aglets (agent applets), although all three dimensions can and have been combined in the more sophisticated applications.

The other two dimensions are the focus of this presentation. Intelligence moves from simple facts and profiles to rule-based inferencing for example, but learning should be considered as an ultimate addition. Learning does not totally replace other forms of intelligence but is often hybridized with them.

The interaction dimension will be secondary to learning in this presentation, but the collaborative model of agents sharing experience with each other will be included. This dimension also implies agent interaction with users -- a most critical aspect, which will also be addressed.

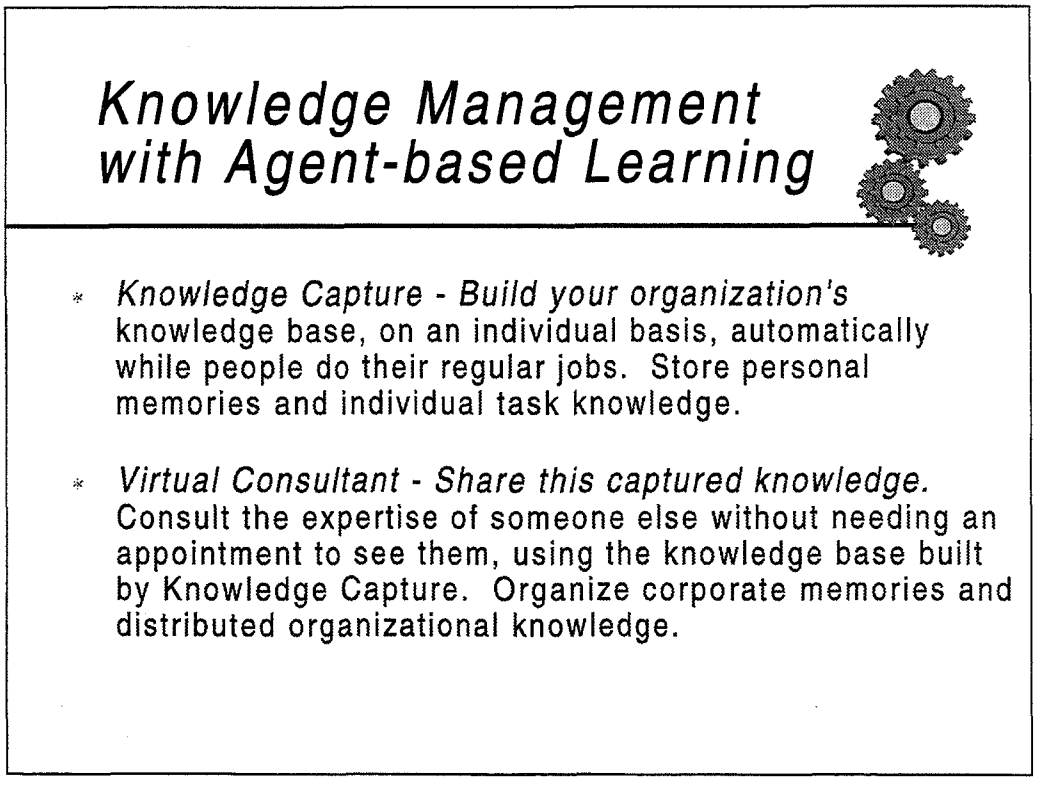


Knowledge Management

We see agent-based learning as simplified into these two aspects, especially for the paradigm of what is being called knowledge management.

First, expertise is what experts do. Knowledge management is largely focused on text analysis and indexing for document-based information and best practices. However, we add that much of knowledge is task-based and recordable by the agent while the user works.

Second, once this behavioral-based expertise is captured, the organization can share it among all its experts to support each other. It can also be used to teach novices in the task or to provide this expertise to other more general populations.



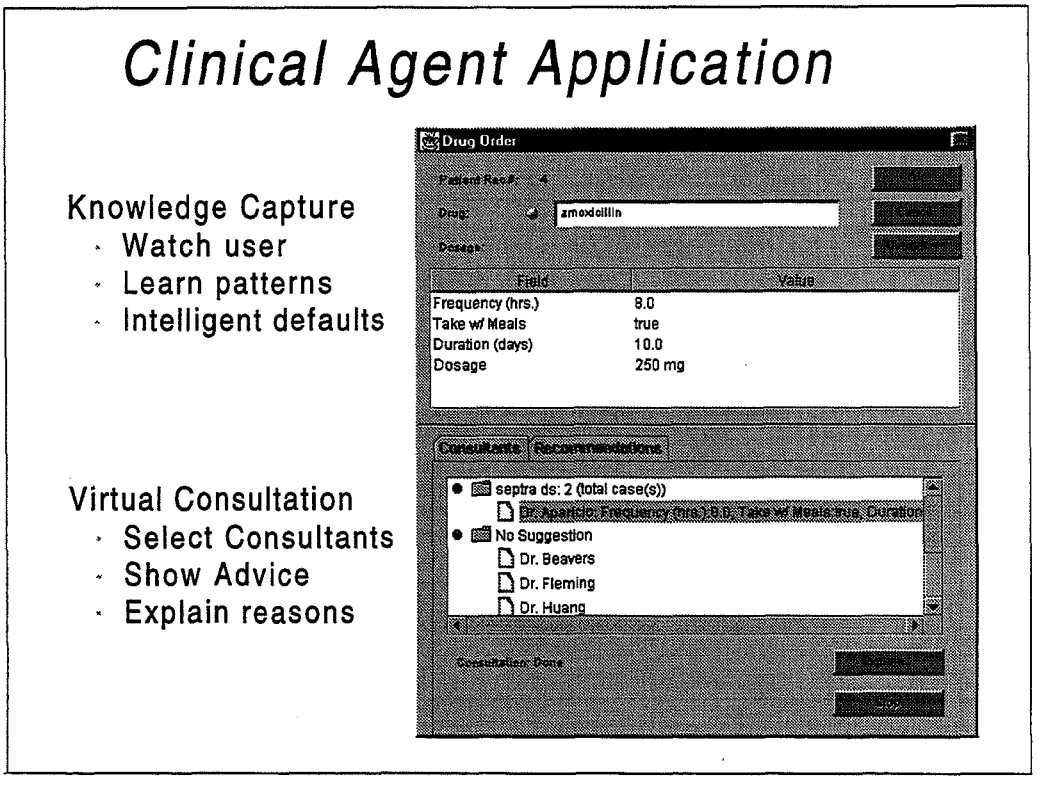
Clinical Agent Application

For example, we tend to use clinical decision support because it highlights a number of key issues and everybody understands this domain to some degree.

Imagine a physician presented with a case and in ordering a prescription online (such as intelligent wireless devices). To provide the individual with some value proposition, especially to encourage lead physicians to use the system, the agent provides an intelligent default. Based on past practice patterns, the agent suggests a drug and shows its confidence in the suggestion, as well as intelligent defaults on the parameters of the order.

If the physician asks for a consultation, then the probable practice patterns of others are gathered and displayed. This is not an expert system of abstracted, engineered knowledge presented by an impersonal machine. This is a community of experts helping each other, mediated by learning agents.

This same approach is applicable to financial advising, and vendor selection as other examples. The latter is of value to manufacturing supply-chains with a community of buyers sharing reputation and quality of service predictions.



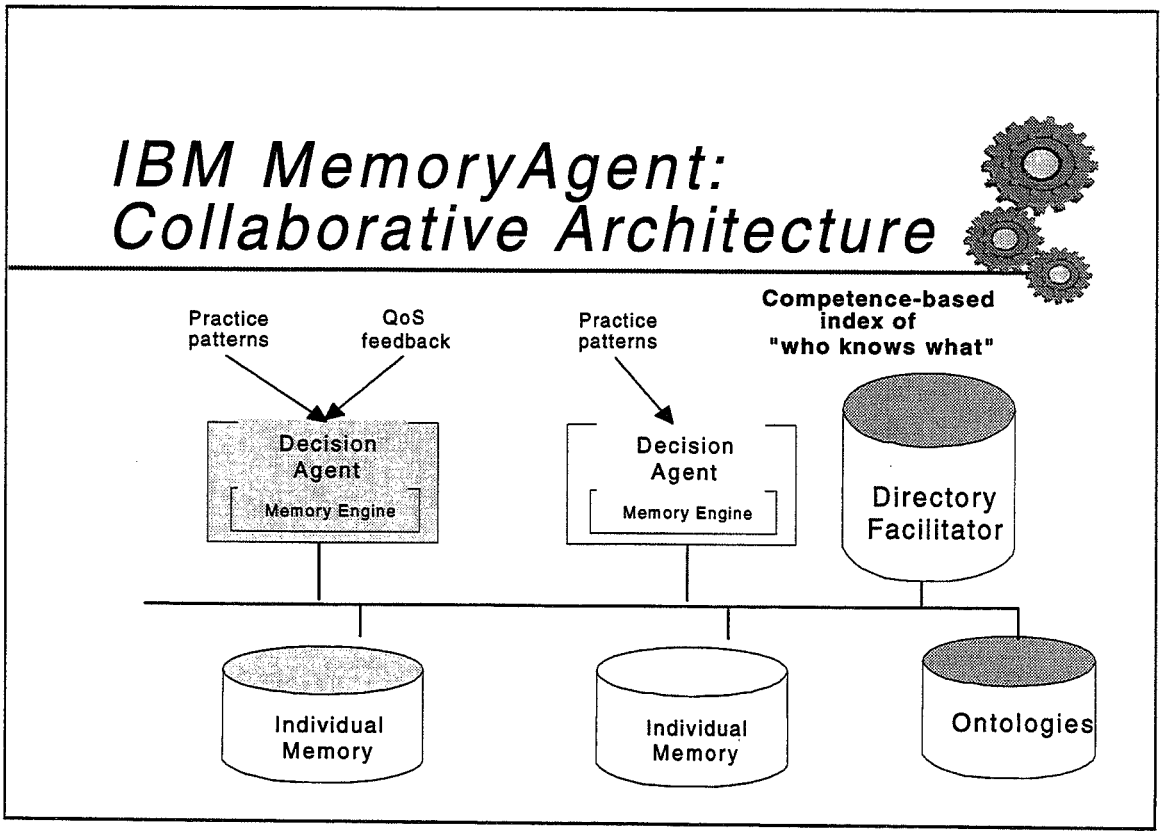
Collaborative Architecture

The collaborative architecture is seen to require the services of other standard agent types.

For instance, the expertise of each agent must be registered with a directory facilitator so that other agents can search for it. This registration can be as simple as keyword (drug name, vendor name, action type, etc.) indexing, but the competence of the agent in regard to the subject can also be registered. This notion of competence will be mentioned later but represents the degree of confidence to which the agent and its user are able to answer a question about the subject.

Ontology services are obviously required to make such a search less brittle, by using taxonomy, synonym, or more complex relations to understand the search query.

Note how this is different than collaborative filter, which is usually based on clustering techniques. The search for expertise is based on the task at hand, not the similarity of end-users. The task at hand is about real work and behaviors, not merely preferences, although intelligent default within the individual agents does provide for such task-based preference.

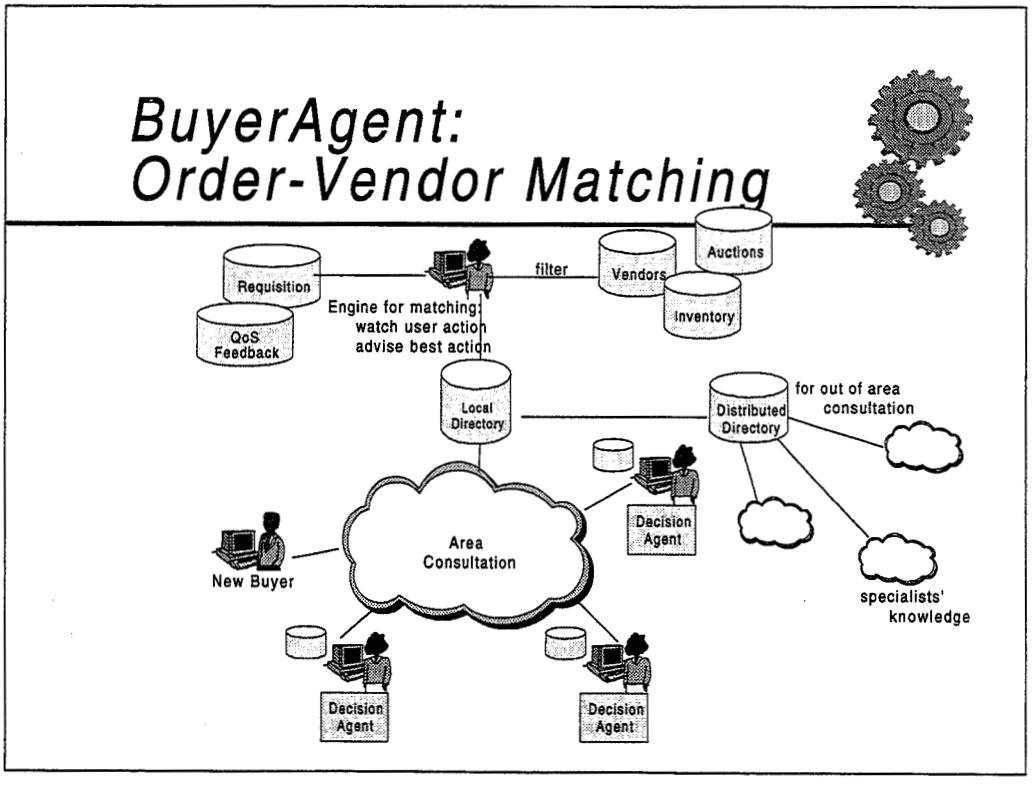


BuyerAgent

This architecture is demonstrated as an open distributed system of MemoryAgent and directories. Given a source of purchase requisitions, for instance, an expert buyer can select from a list of vendors, auction sites, or look at internal inventory. As this and other buyers make such selections under different conditions in the requisition (material, volume, due date, quality of service needs), the agents learn and share such knowledge across the organization, including new users still learning the job.

Note that if quality of service is also available, the agent can learn not just the expert's actions, but the probable consequences. For instance, under certain order attributes such as large volumes, a vendor might often make partial delivery with a back order.

Generally, no action is good or bad in an absolute sense. No expert knows everything. The agents learn under what conditions different actions are best indicated and which other agent/users to consult.



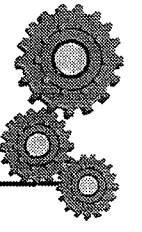
Agent-Based Learning

In order to perform such learning scenarios, agent-based learning is defined to include special, difficult attributes.

For instance, this learning is assumed to happen in "real-time," when the users make their selections and options. Learning should not be a batch-mode, off-line process. Unlike most learning techniques more generally defined, the agents should simply see and learn what they see and manage themselves. They cannot have black-art parameters and predefinition of the problem space, for example.

These requirements are best met by case-based or memory-based techniques. While neural network and case-based techniques have become very successful in the last several years, these requirements drive toward more advanced, second-generation techniques. It is these newer and next methods that will provide such agent-based value.

Agent-based Learning: Requirements

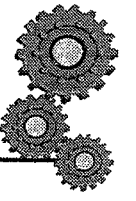


- *True Learning must be*
  - *Incremental - Immediate value, grows with each case*
  - *Parameter-free - No black-arts, "knob tweaking"*
  - *Positive-instance only - Might not get negative feedback*
  - *Explainable - Boolean rules or fuzzy membership functions*
  - *Malleable - Adjusts to new attributes as they are seen*
  - *Bounded - Not exponential (Learning is NP-complete)*
  - *Controllable generalizing - Measurable distance/risk depending on context*
  - *Self-pruning - Learn to ignore what is irrelevant*
  - *Self-competent - Know self-maturity and what is not known*
- Solution:
  - *Associative-memory and case-based techniques*
  - *Neural Networks are 1 of 9 fundamental emerging technologies (Red Herring)*
  - *IBM MemoryAgent does all of the above*

IBM MemoryAgent

Further description of these agent-based learning requirements, along with demonstrations and APIs are available in an evaluation package, which is also the basis for FIPA's learning specification. An FIPA compliant, openly available agent service will soon be available for qualified experimentation with other agents.

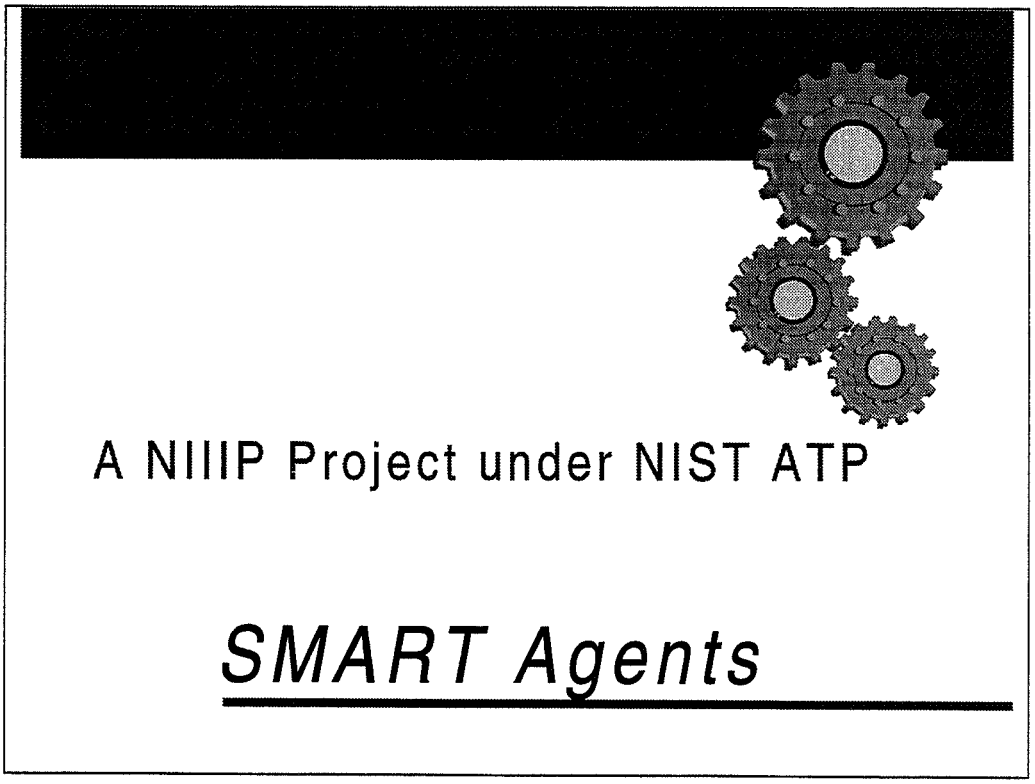
IBM MemoryAgent: Evaluation Package



- Request from http://www.networking.ibm.com/iag/iaghome.html
- General Documentation
  - Intro and application scenarios
- DecisionAgent Shell
  - Set of demos for medical, financial, and purchasing
  - Documentation to build new demo
- MemoryAgent Core
  - 80K Java LEARNING.JAR (times out)
  - Java Docs (with example code)
- Also look at FIPA Personalization/Learning standard
  - http://drogo.cselt.it/fipa (www.fipa.org will be soon)
  - MemoryAgent leads User Personalization, Learning Service spec
  - Now building IIOP-available (FIPA spec) MemoryAgent Service

An NIIIP Project Under NIST ATP

More toward NASA's interests in the workshop, our use of MemoryAgent within the SMART manufacturing consortium will now be reviewed.

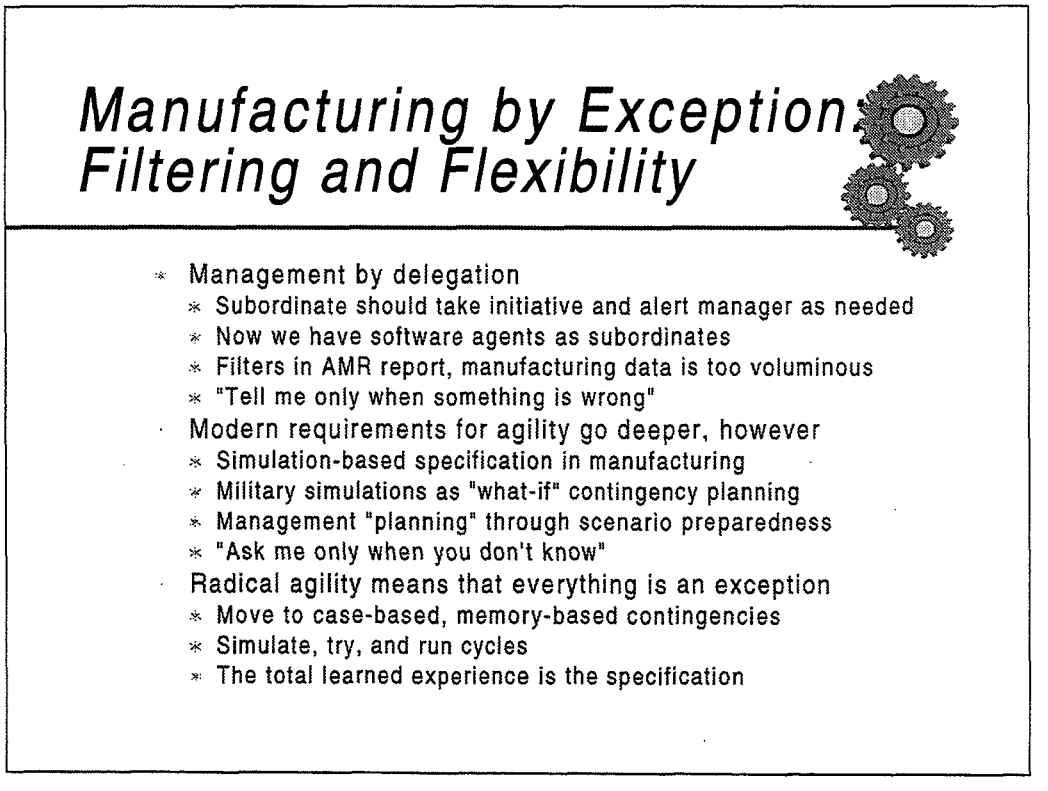


Manufacturing by Exception

SMART agents provide flexibility to manufacturing through a manufacturing-by-exception philosophy, espoused by AMR in its reports. The idea of agent-based filtering is clearly applicable; an agent can be delegated to watch for engineering changes, production quality, or any other changes or transients, notifying its user when values are out of bounds or some other exception-condition occurs.

We have taken this idea further with agent learning by including simulation-based specification. Agility is developed through simulation of contingencies or being actually faced with different conditions and learning how to best respond, depending on conditions. Explicit definition of all such processes is impractical; the specification of even one process is a secondary task to the process itself.

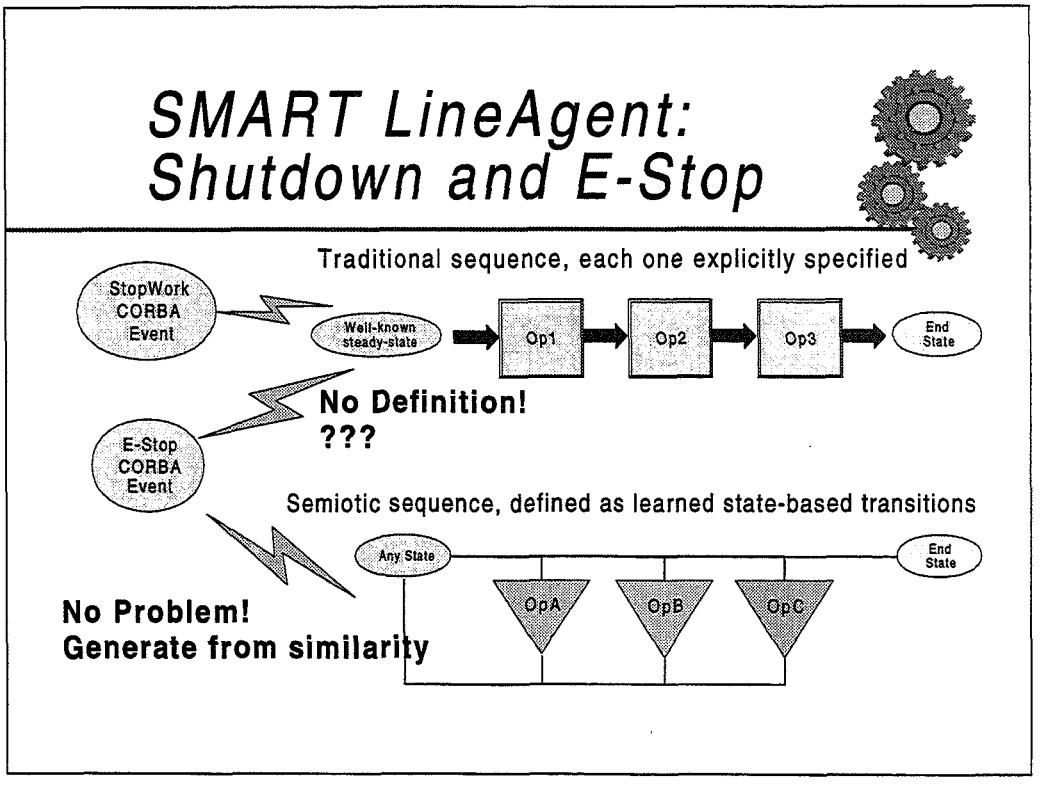
Therefore, by watching the processes in simulation or real action, agent-based learning can become the specification by suggesting the best action and processes. Through generalization such as in semiotic sequence learning, novel but appropriate processes can be generated even if never explicitly trained. This is a radical form of agility, but the truth is that a form-freedom balance is most advisable as will be described.



Shutdown and E-Stop

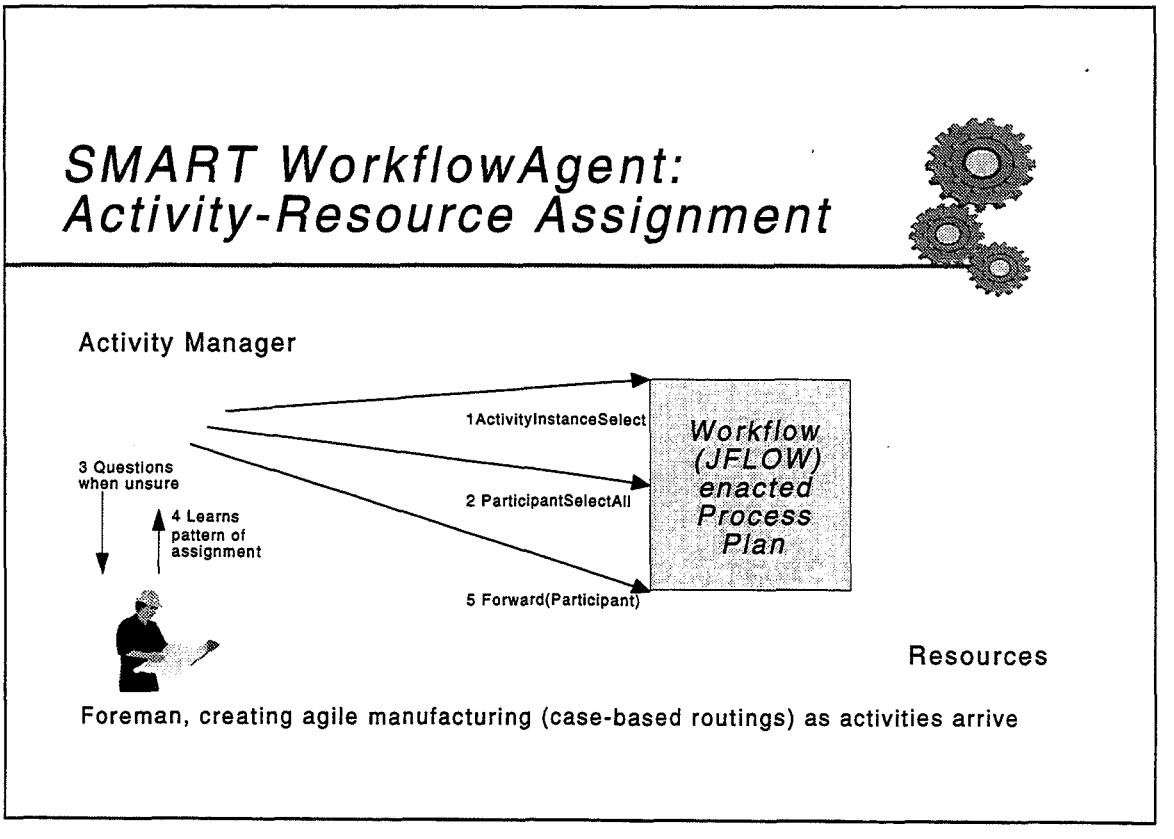
To demonstrate the radical agility of learning agents, we built a LineAgent that listened for CORBA events, from plant control to individual machines and could also query work-in-progress. This agent could watch and learn shutdown sequences from a manufacturing engineer (from a simulator or actual line events), so that it could suggest the sequence to forepersons and operators whenever it received a shutdown event from plant control. Moreover, because the representation was semiotic (similarity-based), LineAgent could generate and suggest novel but appropriate sequences, based on similarity of state to known past states. For instance, LineAgent could receive an E-stop event from one of the machines, and even though no emergency stop procedure was explicitly defined, the goal of efficiently and safely stopping the line was common to other known procedures.

Even though such simulation-based or programming-by-example "procedures" are deterministic and replicable and in many ways more than equivalent to hand-coded procedures, the social acceptance issues of learning and generating such critical operation sequences also needs to be addressed. In fact, in subsequent work we focused on less emotional tasks and worked more on trust and control.



Activity-Resource Assignment

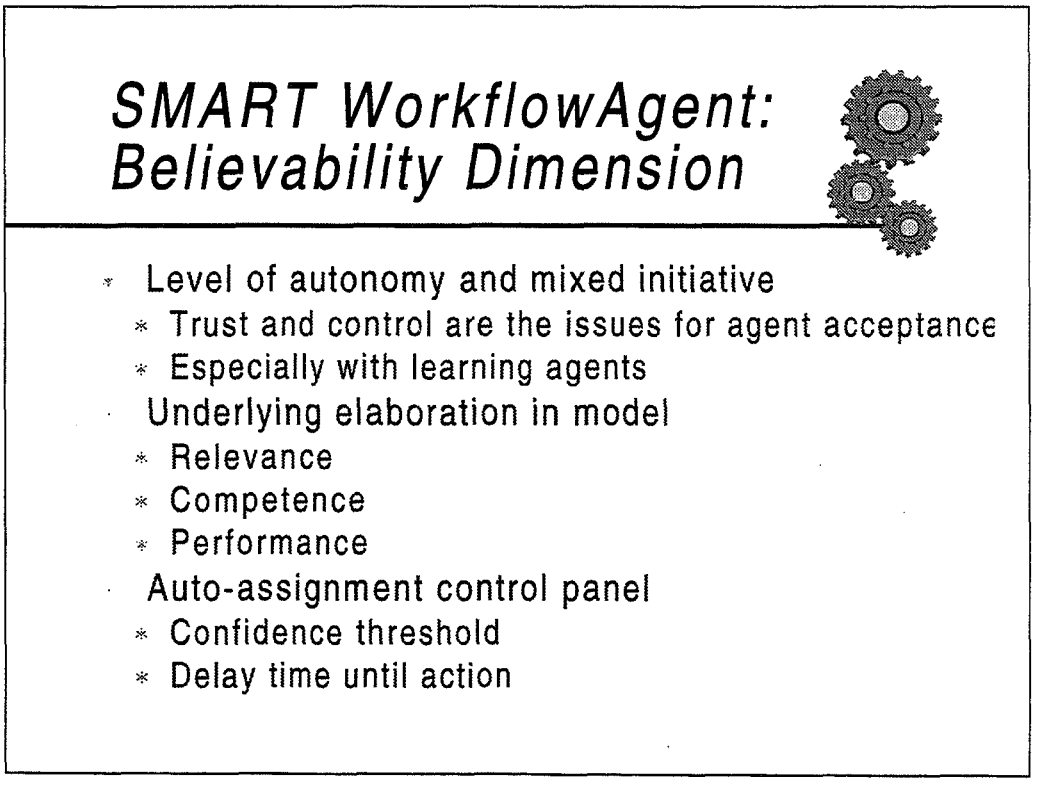
Working with the domain experts in SMART to find a valuable but less radical application, we applied MemoryAgent to activity-resource assignment, more specifically, to the enact process routings from process plans. SMART technologies also include workflow systems, which were used to send process operations as JFLOW (OMG workflow standard) activities to a WorkflowAgent. Assuming that one such agent was responsible for an agile manufacturing line, these activity requests would be received by the agent and displayed to the foreperson or manufacturing engineer along with a list of possible routings (machines, other lines/agents, or human/manualshops listed as JFLOW participants). The user would make the appropriate assignment and the agent would learn this. As same or similar activity descriptions arrived, the agent would begin to suggest such routings as a form of intelligent default. To the degree that the user became comfortable with the agent's performance in suggesting, he/she could adjust a level-of-autonomy control, a confidence threshold above which the agent would autoassign. The user could also specify a time delay before such action, allowing the user to see and change if needed. In short, what the agent did not know to autoassign was thrown an "exception" to the user, who would show the assignment, making the agent smarter to later assignments, etc.



Believability Dimension

This issue of trust of control is critical to acceptance of agents in manufacturing. Aside from the technical issues which seem rather solvable, the human and social dimensions can inhibit deployment. The introduction of learning agents makes trust and control issues even more critical.

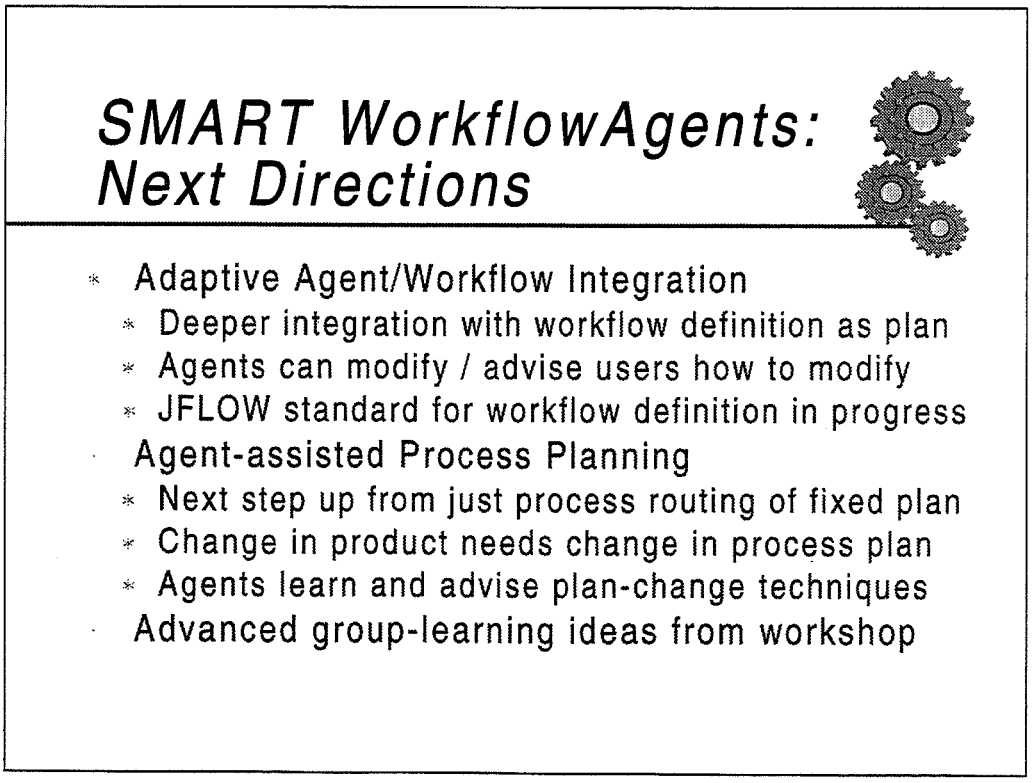
Our approach has been to develop level-of-autonomy controls in the human-computer interface itself and to more fully elaborate the underlying model of learning. "Confidence" is really a more complex variable, which we have split into relevance and competence components. Relevance is the degree of association or membership of a given case to a group of already observed cases. Purely, it represents a distance of the case in some memory space (such as in a sparse distributed memory). Competence represents the statistical power and significance of all the observations. A naïve agent might report high relevance between similar cases (which is true), but should also know and report its level and clarity of experience. These dimensions can be variously used by different applications, depending on the application and its decision criteria. Actual performance of predictions is obviously the final measure, but we did not include it in this particular application (naïve learning by observation).



Next Directions

Starting with WorkflowAgent as described but including process planning as well (somewhat of a return to sequence learning), a form of memory-based planning and replanning can be developed. Of particular interest to other speakers in this workshop would be the inclusion of multi-user planning and replanning. Problem decomposition, hierarchical organization, and peer negotiation would be required additions.

Other standards must also mature for such work to most benefit in a continued relationship to workflow. For one, a standard process definition would be required. The runtime interoperability standards of WfMC and JFLOW, for instance, are adequate for activity-resource assignment (the routing), but more interoperability between workflow and agents would require standard definition of workflow plans themselves. For another, better standards for organizational structure is desirable. Even for activity-resource assignment using workflow runtime interfaces, better definitions of organizational structure and roles would be helpful.

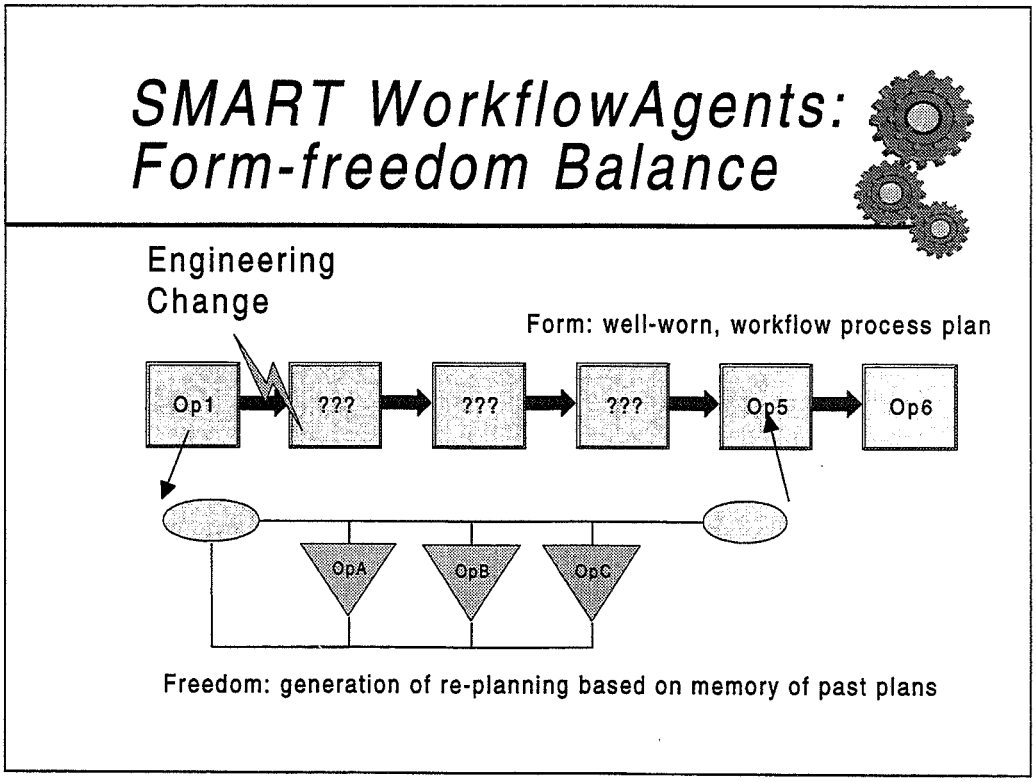


Form-Freedom Balance

While agent-based learning provides adaptability along with its representation of specifications, hybridization with other representations is less radical. A balance of well-known forms such as explicit procedure definitions combined with the freedom of learning is indicated.

For instance, a workflow or process plan can represent well-known or "hardened" processes. Learning agents can watch and represent real procedures as found in end-user behaviors. As these actual procedures are observed and repeated, they can be promoted to explicit procedures.

However, all procedures are still faced with too many exception conditions and the plan must often change as real-world conditions change. Specifying all such exception conditions and contingent procedures is impractical and leads to spagetti-looking process definitions, not the clear-cut standard procedure a workflow or process plan is intended to provide. Such exception handling should be left to agent learning and its ability to store and generate procedures based on similarity to past experience. This form and freedom can work together when standard procedure needs to change but can be re-specified by recalling past procedures, again based on similarity measures.



FIPA Standard

The Foundation for Intelligent Physical Agents provides a set of agent standards. A learning service interface is among them. This learning interface is part of the Human Interaction specification. Of course, learning is a much more general application technology, but as presented here, agent-based learning is strongly associated with human interaction. Our philosophy with MemoryAgent and its effect on this standard are to focus on learning by observation of expert end-users.

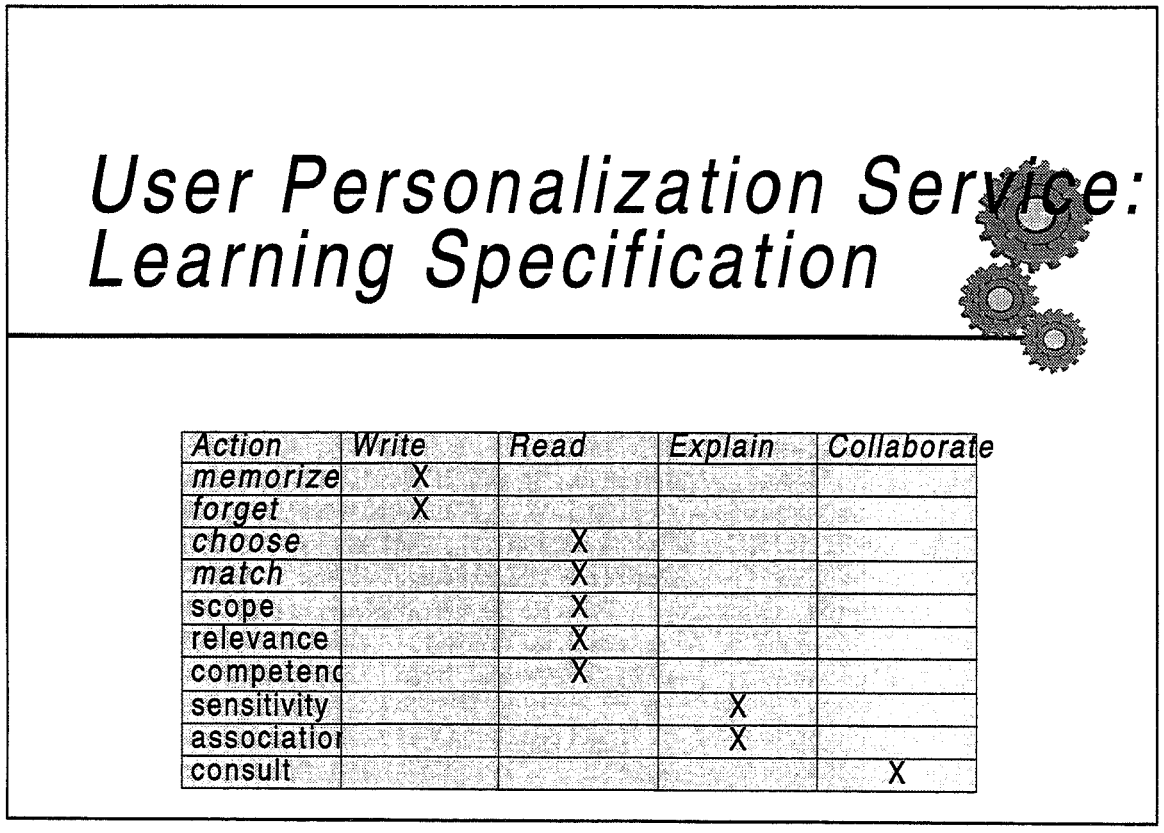
Learning Specification

The FIPA specification for learning can be outlined as a set of actions. Memorize and forget provide the core actions for storing or removing observations.

Choose and match provide the core actions for using a memory of such observations. These actions reflect the two primary types of decisions that humans make; given a situation, we can choose one or more options from a set of selections; given a particular selection we can measure or match the attributes to each other, such as when setting a good price on a selected product or predicting the quality of service from a selected vendor. Scope is a more refined action, similar to choose.

Relevance and competence allow the client to ask for measures of similarity and statistical confidence. Relevance provides a measure of "membership" for a new observation to the set of prior observations. For instance, how closely does a new operation belong to the operations typically routed to a specific machine. Competence indicates the maturity of the agent and its clarity of observations to make such a recommendation.

Sensitivity and association provide linear and nonlinear forms of explanation about the recommendation, while consult provides a model of collaboration between learning agents.

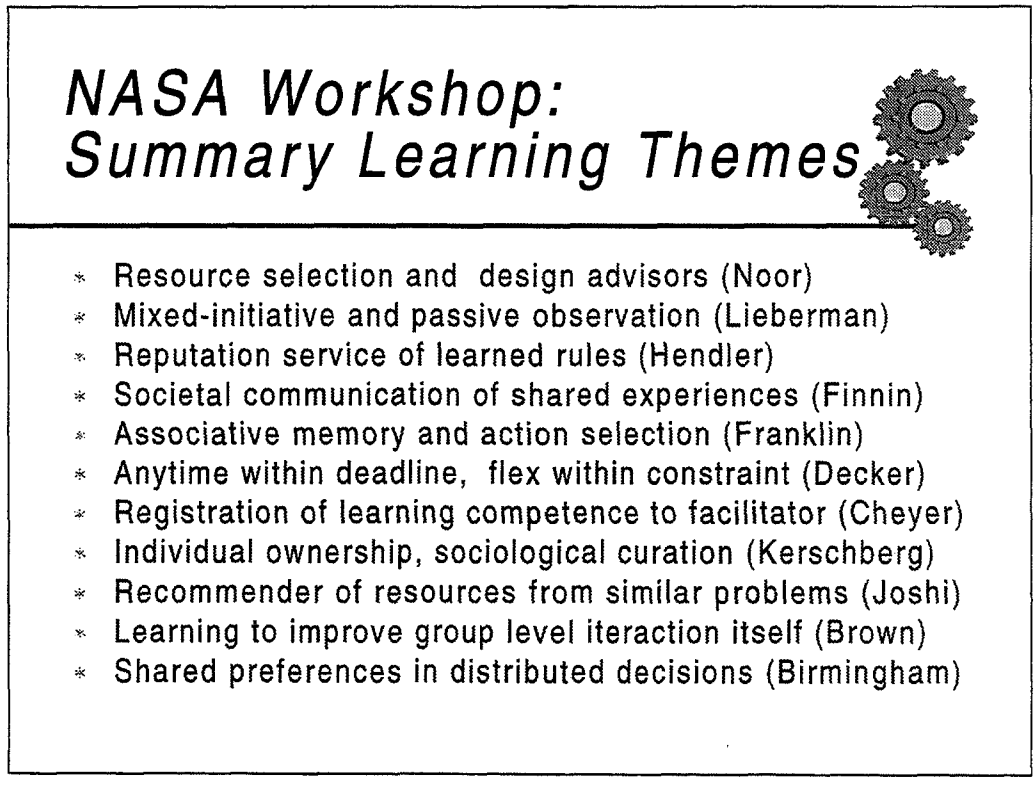


Learning Theme

All of the previous speakers have mentioned learning, which I would like to list here as a theme of this workshop. (Mike Huhn is next and last to speak so is not included here but has contributed significant work on multi-agent learning systems as well.)

Some of the other speakers have provided very clear examples of agents that learn and recommend from observation, which is very similar to what is presented in this talk. Of most direct use in agent-based simulation, resource and parameter selection seems like an ideal problem for such preference-watching agents. Of course, simulation and design decisions are more than mere individual "preferences." This is a matter of organizational expertise which should also be shared in a collaborative community of users and their agents.

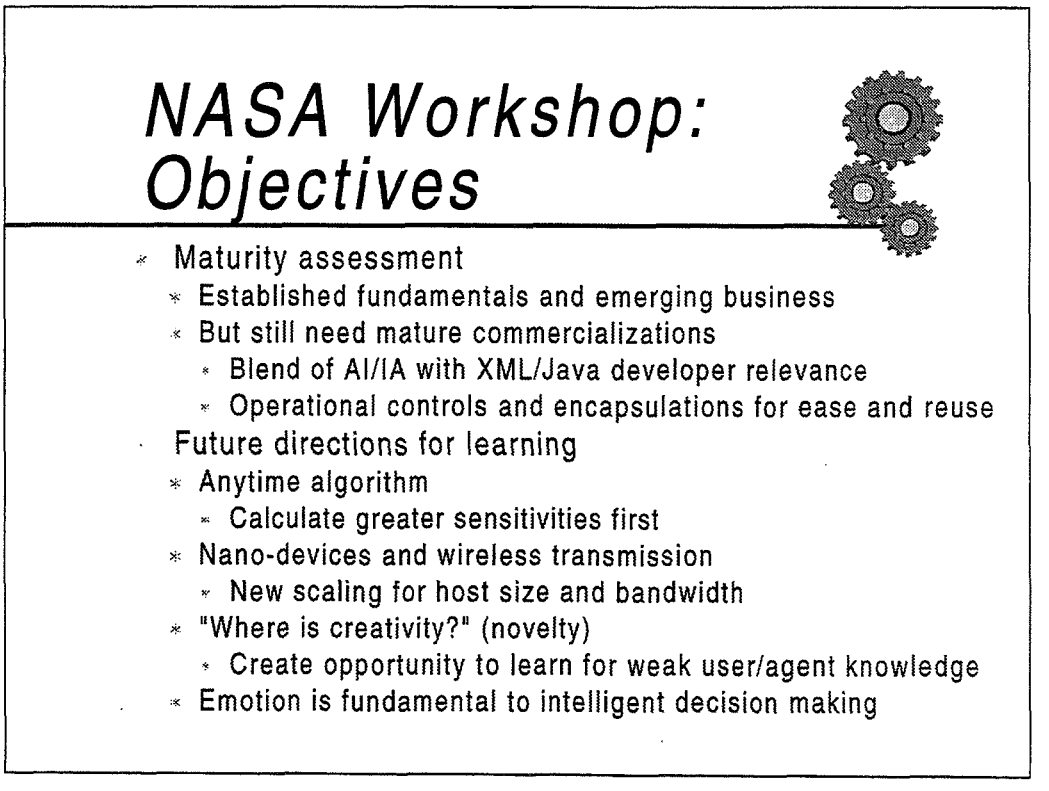
Similar to MemoryAgent's collaborative model presented in this talk, some of the other speakers have mentioned reputation services, agent-based sharing of experience, and facilitated search for agents that are most competent to perform a particular task (or most competent to advise). For instance, recommender agents can register their competence in a resource, making a facilitator into the hub of a system for asking other agents about their choice experience - the reputation of the resource.



Workshop Objectives

The workshop objectives are to evaluate the market as well as the technical maturity. For our experience, the market is still emerging and some aspects of agent-based learning and collaboration still need research. However, the basic technologies and collaboration models are commercially available. The only difficulty is in mapping the raw technology to some particular applications, representations, and legacy systems. As this commercialization quickly matures, however, the complexity and expertise in learning and agents per se will tend to be encapsulated. For instance, a Recommendation System shell can allow the client system to focus on the decision attributes, what kinds of choices are possible, and effecting those choices, rather than the technology itself.

For personalization agents, we are working to give the user a better variety of choices, providing novelty, and giving both the user the best opportunity to learn about the space and the agent to learn about the user (and space). In addition, agent-based learning is generally understood as critical to ubitiquous computing, from personal communicators to nano-satellites that learn to effectively coordinate with each other. Otherwise, emotional intelligence has been virtually ignored in commercial learning systems, but will later emerge as a critical dimension in situated agent systems.



59-61
032521
370171
10P

Intelligent Agents for Design and Synthesis Environments: My Summary

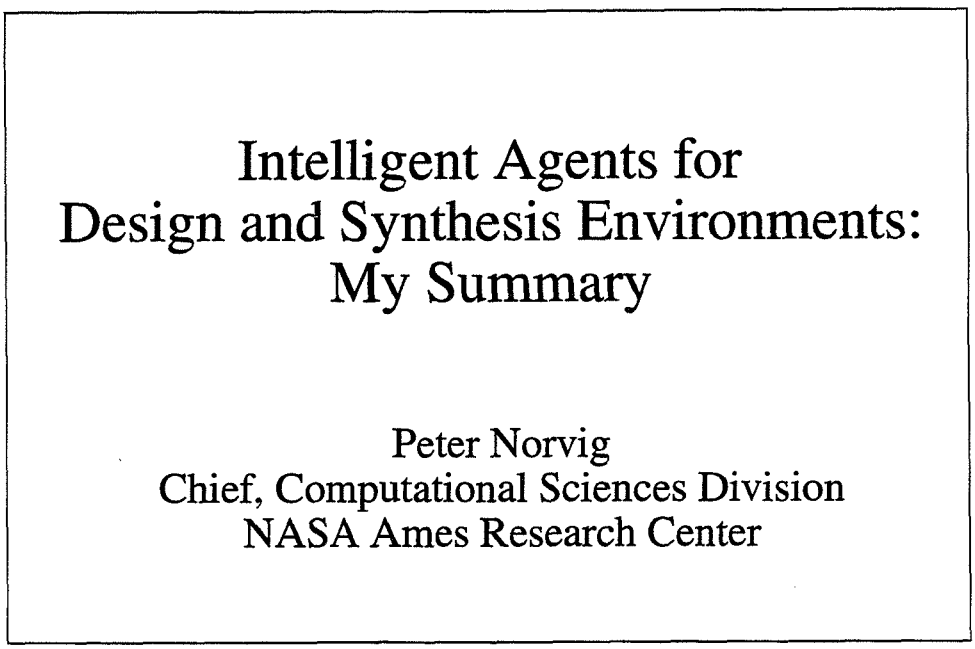
Peter Norvig
Computational Sciences Division
NASA Ames Research Center
Moffett Field, CA

Page intentionally left blank

Intelligent Agents for Design and Synthesis Environments: My Summary

Peter Norvig
Chief, Computational Sciences Division
NASA Ames Research Center
Moffett Field, CA

This presentation gives a summary of intelligent agents for design synthesis environments, from my own personal point of view, and from what I have seen of the participants' presentations.

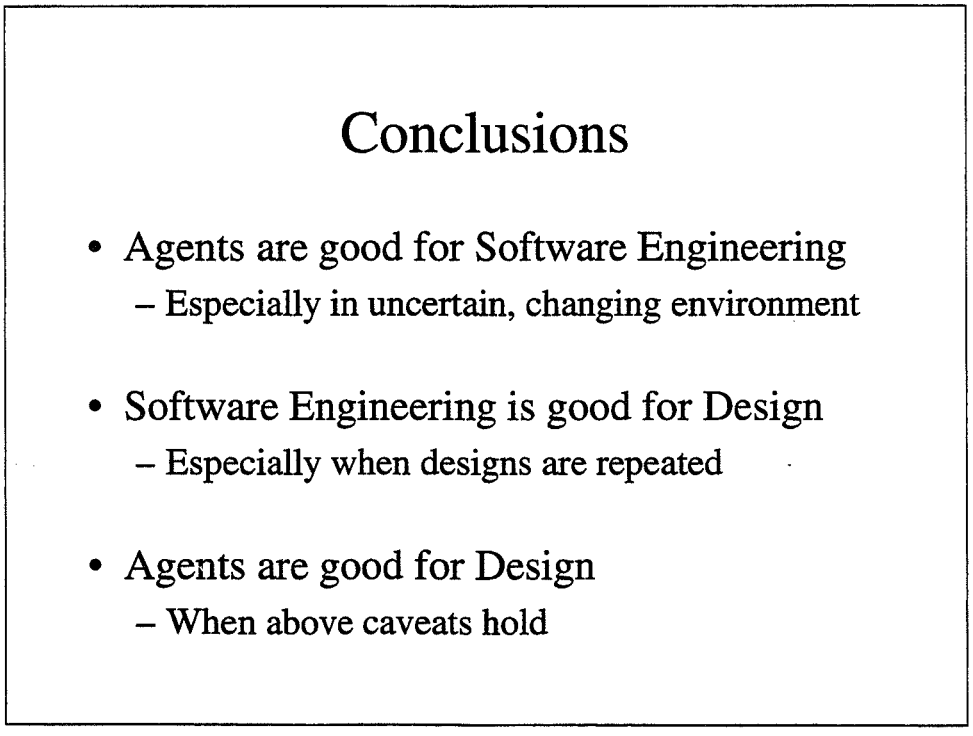


Conclusions

We'll start with the conclusions, and work backwards to justify them. First, an important assumption is that agents (whatever they are) are good for software engineering. This is especially true for software that operates in an uncertain, changing environment. The "real world" of physical artifacts is like that: uncertain in what we can measure, changing in that things are always breaking down, and we must interact with non-software entities.

The second point is that software engineering techniques can contribute to good design. There may have been a time when we wanted to build simple artifacts containing little or no software. But modern aircraft and spacecraft are complex, and rely on a great deal of software. So better software engineering leads to better designed artifacts, especially when we are designing a series of related artifacts and can amortize the costs of software development.

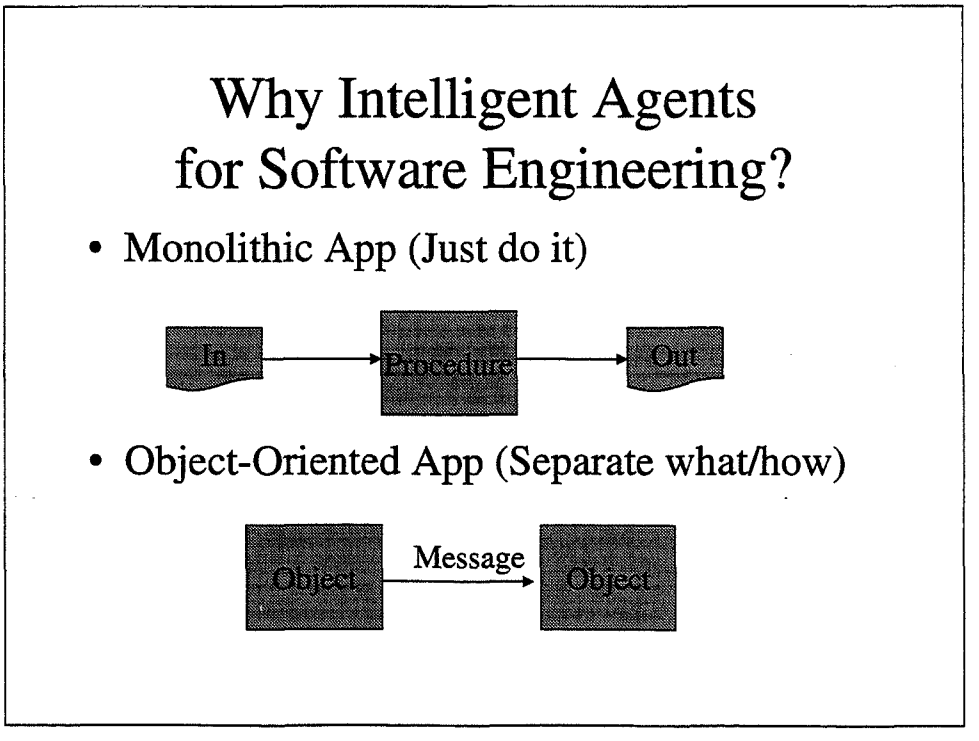
The third point is that agents are *especially* useful for design tasks, above and beyond their general usefulness for software engineering, and the usefulness of software engineering to design.



Why Intelligent Agents?

To see why intelligent agents are important for software engineering, we need to look at some history. Up through the 1970's, software was mostly built in terms of monolithic applications. They were designed and built in terms of input/output behavior. Like a mathematical function, if you provide them with a certain input, they are supposed to respond with a certain output.

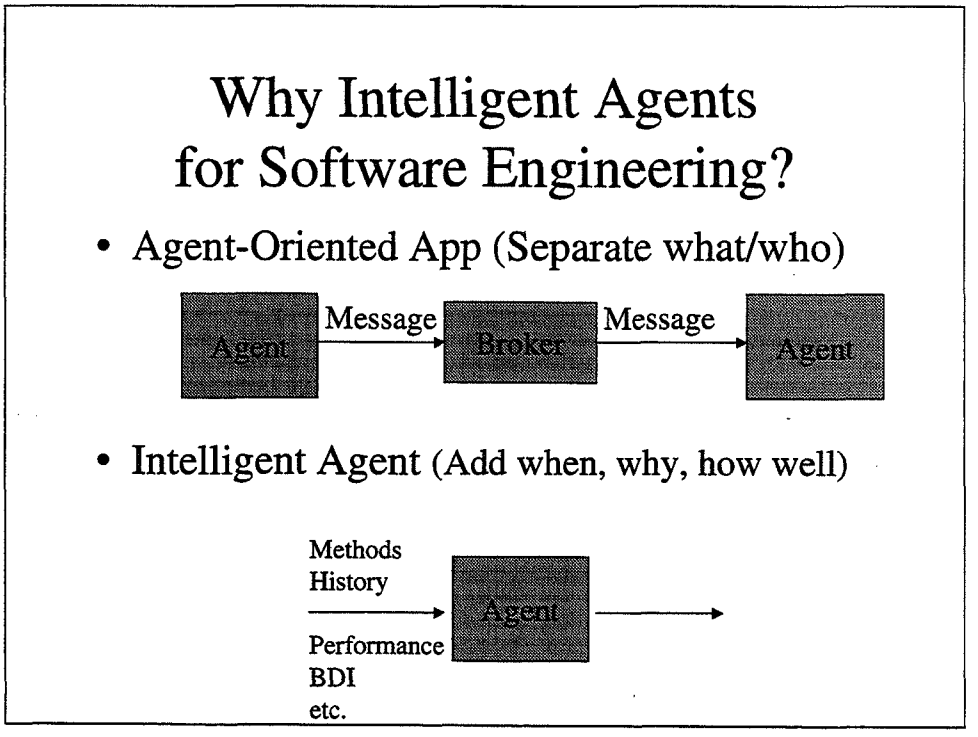
In the 1980's we see a movement towards object-oriented applications. There are two main innovations. First is to concentrate more on *objects* rather than *procedures* and their input/output behavior. An *object* contains both *state* information (data) and *behavior* specification (a set of things the object can do). The second innovation is to separate *what* the object can do from *how* it is done. In the monolithic application, a procedure is simultaneously a mathematical specification (what) and a particular implementation (how). In the object-oriented approach, the *message* says what we want done, but that can be accomplished by one of several possible *methods*, and we are always free to add new methods. We increase modularity by separating *what* from *how*.



Why Intelligent Agents?

In the 1990's, we begin to see agent-oriented applications. Again there are two innovations. First, certain objects are thought of as *agents*. That means that they can initiate actions rather than just responding to messages. It often means that the agents persist for long periods of time, and that they serve for the benefit of some other person or software entity. Second, agents do not need to know all the other agents. Rather than having to know who to send a message to, they can broadcast the message to a broker, who relays it to an appropriate receiving agent. We are always free to add or subtract agents. That means that we increase modularity by separating *what* from *who*.

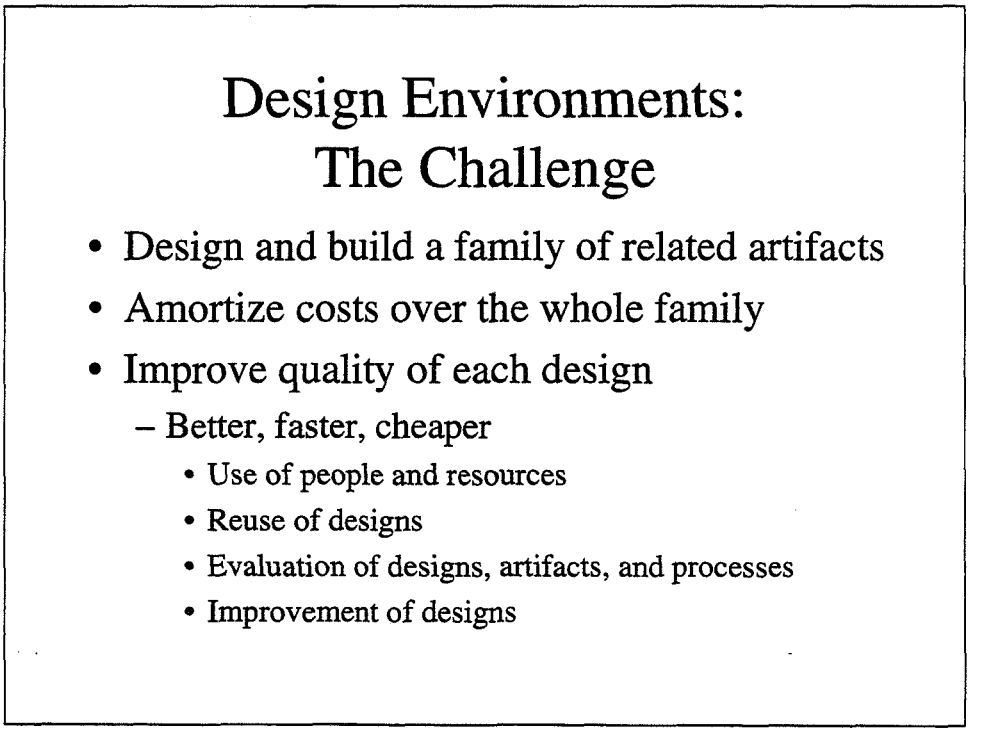
In the late 1990's, we see intelligent agents, which can reason about and improve their performance. An intelligent agent has a set of *base* methods that it can perform (like a regular agent), but it also has *meta* methods. You can ask it what it can do, how well it can do it, what resources is it likely to need to do it, etc. A set of agents, communicating along these meta-method channels, can optimize its use of resources, finding the best subset of agents and the best methods to accomplish a task. Regular agents (or regular object-oriented or monolithic applications) cannot even be asked these questions, let alone optimize a solution.



Design Environments: The Challenge

The challenge we have is to come up with a good design environment (or design process) that lets us build a family of related artifacts (such as a sequence of spacecraft for a related set of missions). By *good* I mean that we want to end up with an artifact that works better, is more reliable, is easier and cheaper to build, and is faster to develop.

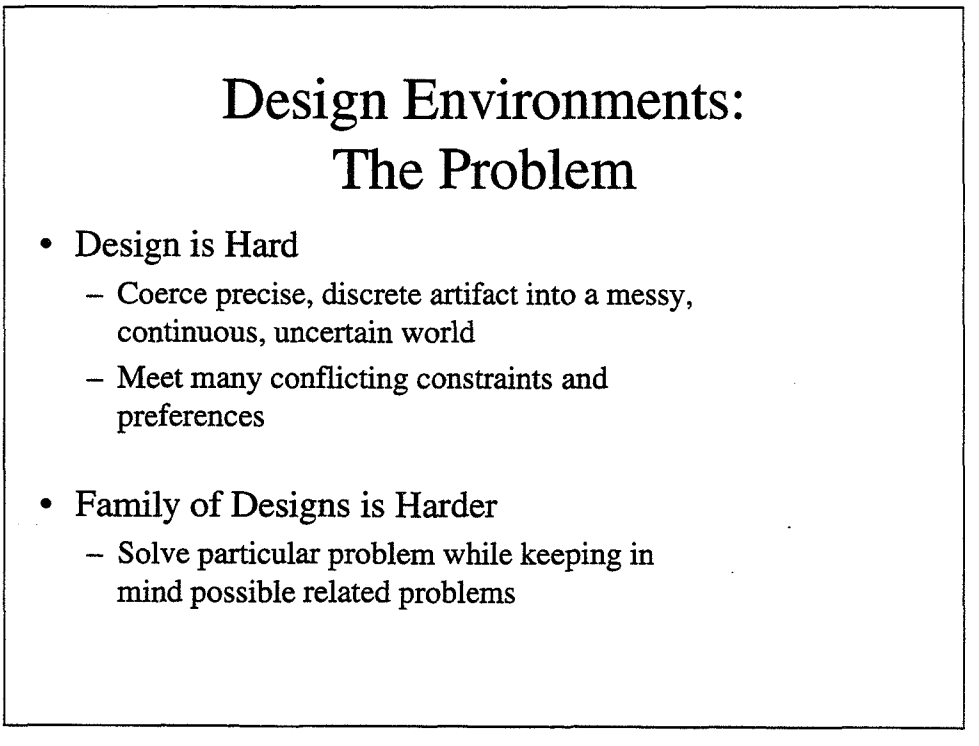
An important point is that we are amortizing the overhead of the design process over the whole sequence of designs. Some types of bookkeeping that would be wasteful overhead on a single design project end up being big time-savers over a sequence of designs. Sometimes the design environment may not seem to help on the first design, but it starts to show up when subsequent designs draw on the lessons that were learned and documented the first time around.



Design Environments: The Problem

The real world is a messy, unpredictable place. Design is hard because we must build an artifact based on a huge set of assumptions, and then unleash it into the messy world where many of those assumptions may fail. Design is also hard because there are always trade-offs between strength and weight, cost and reliability, speed and carrying capacity, etc. The designer can use help in weighing these trade-offs against one another.

Building a family of designs is harder, because we must remember all the assumptions, and the reasons behind them. Suppose the second vehicle we build only needs to carry half the weight of the first. What does that mean about the required strength, type of materials, and size? What assumptions that lead to the original design were dependent on carrying capacity? How do we know we have the right answers, when some of the original team members are no longer on the project? A design environment must provide a way for us to record the rationale behind design decisions, and it should help us sort out the ramifications of a change in requirements.



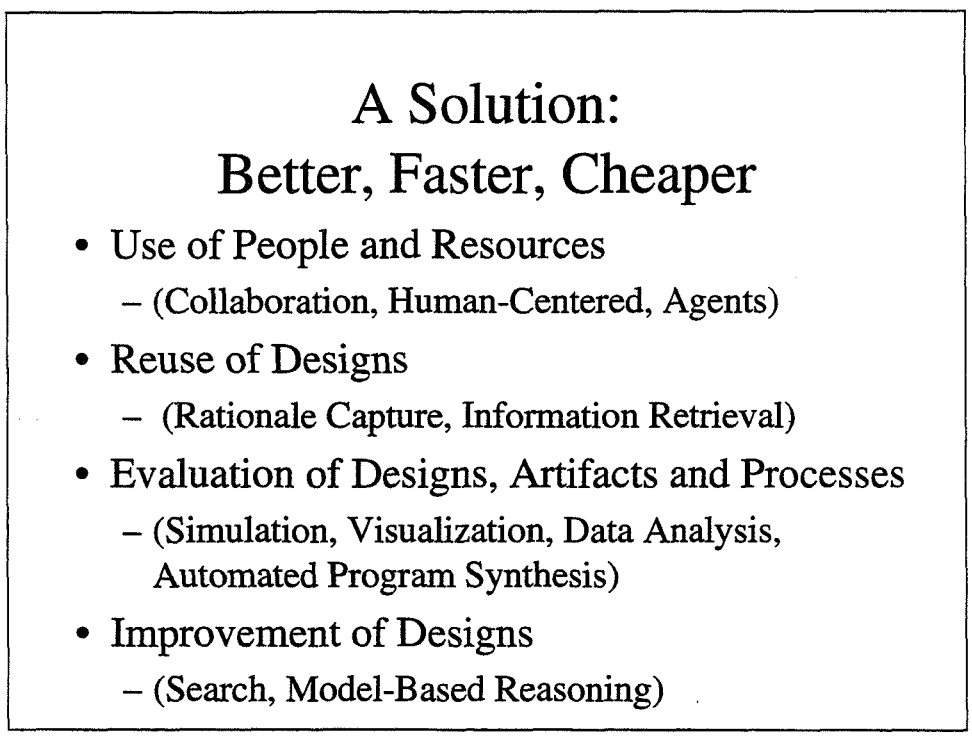
A Solution: Better, Faster, Cheaper

Four ways to make the design process better are described here. First, we can make the people on the team work better together. Collaboration software and hardware (such as videocameras) can improve communication. Principles of human-centered computing can make the interaction with machines more productive. Software agents can free the team members from routine tasks.

Second, we can make it easier to reuse designs. Good software engineering practices (including agent-oriented design) helps. Recording the rationale behind design decisions is essential, and information retrieval (of documents, simulation runs, recorded videoconferences, etc.) helps team members understand the context of a decision.

Third, given a proposed design, we can provide tools that evaluate how well the design meets the goals. We can generate code from specifications, run simulations, and analyze the resulting data.

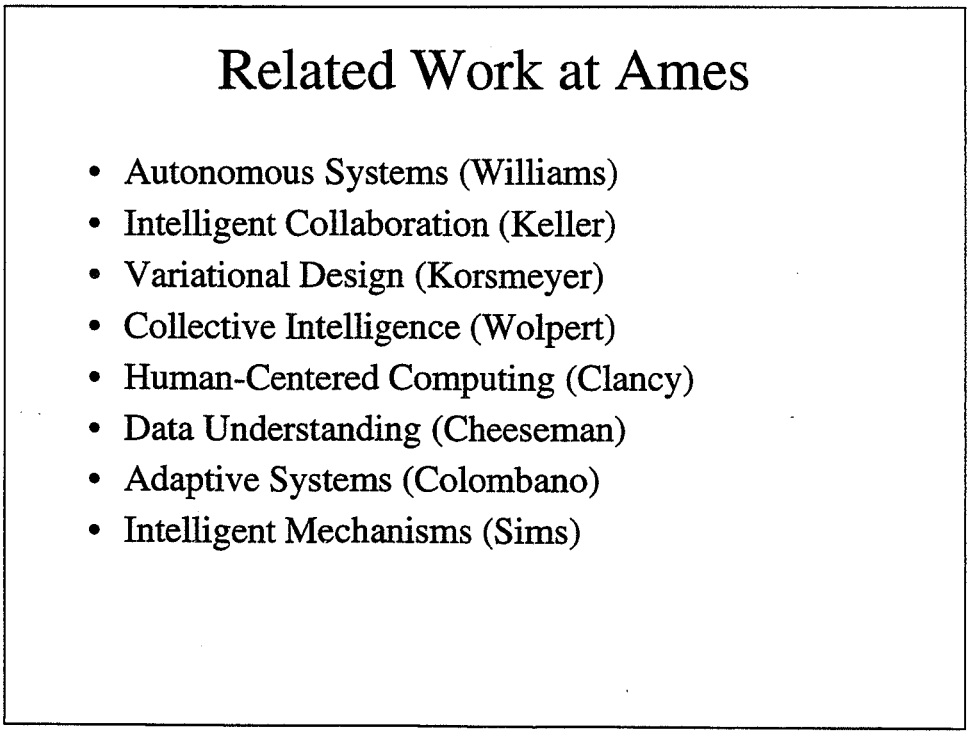
Fourth, we can search for new designs that incrementally improve on previously proposed designs. Search techniques from AI and OR can help here, as can model-based reasoning techniques that suggest what components should change.



Related Work at Ames

Much of the work in information technology at Ames is directly applicable to design and synthesis environments. It is mentioned here to give you an idea of what kind of technology to expect today, before even doing any research and development specifically for design and synthesis environments.

We can start with the Autonomous Systems group. Their intelligent agent, planning, scheduling, and model-based reasoning technology can be part of an intelligent artifact, as in their *remote agent* technology which is flying on DS-1, or it can be used to analyze artifacts. The Intelligent Collaboration group helps dispersed teams work together, and the Variational Design group applies similar techniques to specific problems such as working with wind tunnel test data. The Collective Intelligence project studies how to optimize a system of agents by learning a good collection of utility functions. The Human-Centered Computing group provides tools for designing systems that humans use. The Data Understanding and Adaptive Systems groups analyze, optimize, and categorize data of all kinds, and track changes in the data. The Intelligent Mechanisms group uses photo-realistic virtual reality to drive tele-operated robots. This work could be applied to other sorts of visualization problems.



58-61
032522
370172
309

Improving Design with Agents, or, Improving Agents by Design

David C. Brown
AI in Design Group
Worcester Polytechnic Institute
Worcester, MA

Page intentionally left blank

Improving Design with Agents, *or*, Improving Agents by Design

David C. Brown
AI in Design Group
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

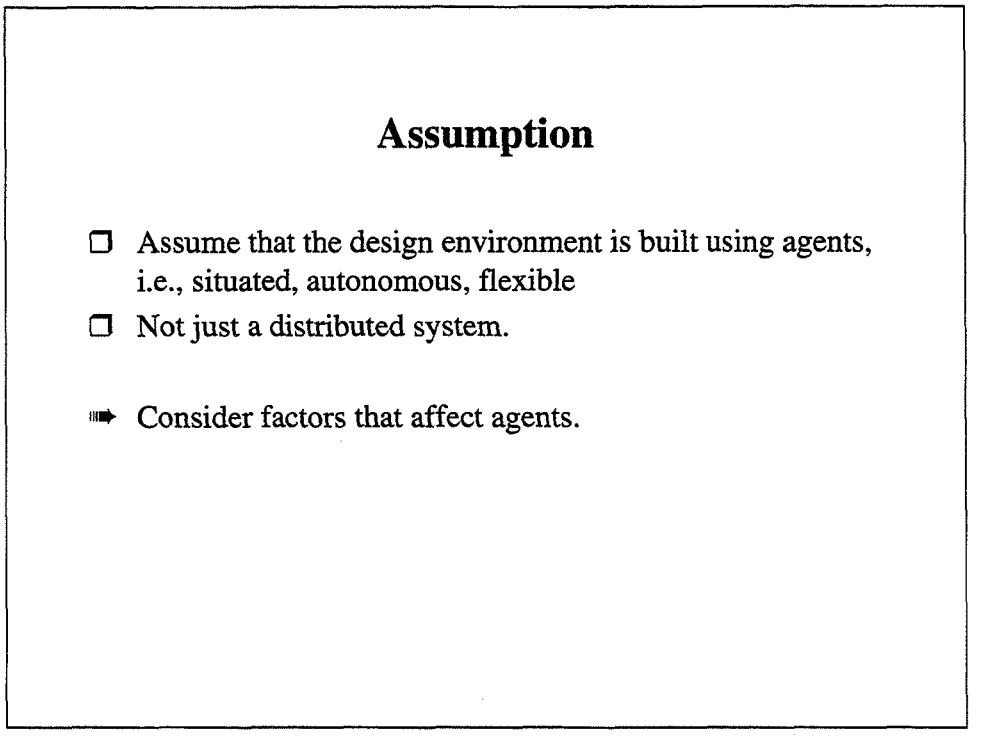
WPI
Improving Design with Agents,
or,
Improving Agents by Design

David C. Brown
AI in Design Group
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

Dr. D.C. Brown hereby gives NASA permission to publish his paper, *Improving Design with Agents*, or, *Improving Agents by Design* in the *Workshop Proceedings*.

Assumption

We start by assuming that NASA's future design and synthesis environment will be built as a *real* multi-agent system. In what follows, we will first look at the task that the environment will need to support, and then examine the consequences of using agents for this environment.



Design Problem Requirements

Like all good designers, we examine the requirements for such an environment. It is immediately clear that most of the design decisions will be critical, and that the activity will be non-routine with creativity involved.

Design Problem Requirements

- ❒ Use in space ➠ hazardous environment
- ❒ High speed ➠ stresses, fast reacting
- ❒ Human users ➠ safety, reliability

i.e., critical design decisions

- ❒ New, very unusual and difficult problems
  - ➠ non-routine design,
    creativity

Other Aspects

Other requirements on the synthesis environment, due to the designs to be generated, will need to be handled using a distributed, concurrent and integrated approach. Consequently the environment will be very complex.

Other Aspects

- Repairability, etc. ➠ DFX
  (life cycle issues)
- Complexity ➠ Decomposition
- ➠
  Concurrent engineering,
  teams,
  distributed designers,
  parallel activity,
  integration

i.e., a very complex system.

Reliability

Highly reliable designs need to be generated. Design reuse and simulation are the two software solutions.

Reliability

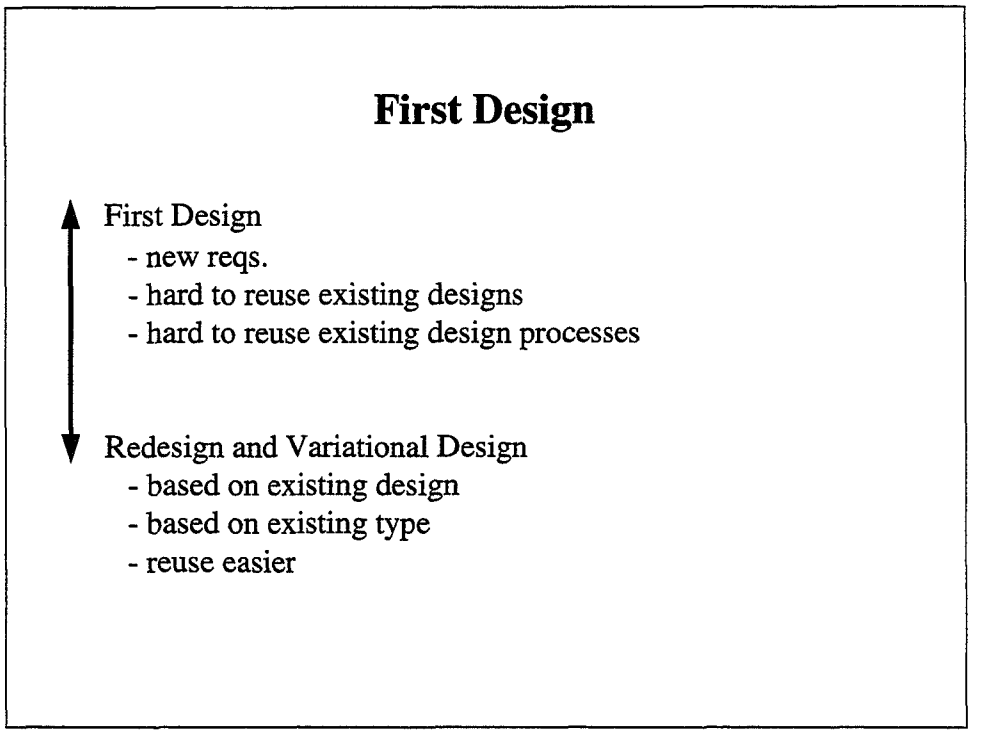
- ❒ Highly reliable design needed

Usual methods:

- ❒ Reuse known reliable designs
  - ◆ less able to do this here
- ❒ Build and test
  - ◆ expensive and slow
- ❒ Simulate
  - ◆ Virtual build and test
  - ◆ Virtual Reality
  - ◆ Simulation based design

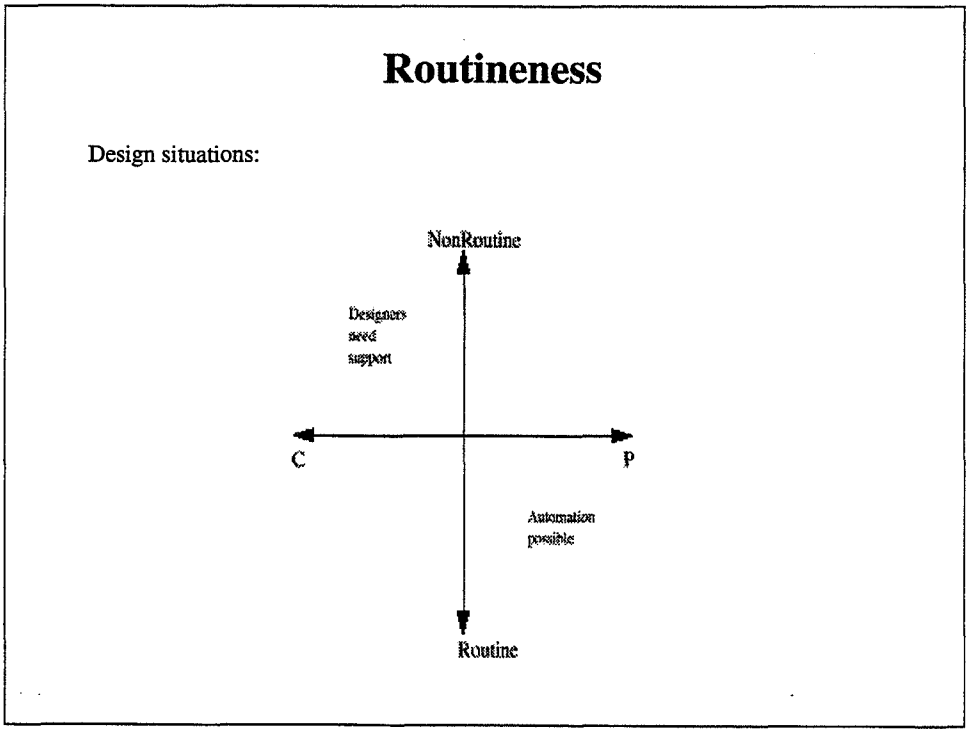
First Design

On the dimension that goes from "common" to "uncommon," it is clear that most of the design problems to be tackled using the environment will be quite unusual, with requirements that have not been seen before. This makes both design reuse and design process reuse difficult.



Routineness

Design situations vary during the design process depending on the knowledge available and the experience of the designer(s). A Routine situation is recognizable and both the methods and the knowledge can be immediately retrieved for that situation. In a Non-Routine situation this is not the case. The space of design situations is multi-dimensional, but here we just concentrate on the abstractness of what needs to be decided, the Conceptual to Parametric dimension. Moving in the non-routine conceptual direction requires the designers to be provided with support. Routine parametric situations can be automated.



Creativity

If the need for creativity is perceived then it can act as a goal to the designer, producing different behavior. As creativity is determined relative to a standard, designers will attempt to produce non-standard designs or use non-standard design processes. The unusual nature of the design requirements in this synthesis environment will already be forcing the designers towards creativity.

Creativity

- Creativity is determined by comparison with *a standard.*
- The comparison applies to the:
  - Design process
  - The design product
- The standard can be set by the past performance of the:
  - Individual (designer or design team)
  - Community
- Creativity arises in situations where the designer is forced to produce unusual designs or processes.
- Creativity can act as a "goal," changing activity.

Decomposition

Another issue to consider is how the choice of agents might be made. There are several ways to decompose a system into agents. In a complex system several of these would be competing as candidates. There probably isn't any single correct way.

Decomposition

- How to decompose into agents?
  - there are many pressures
- by design process
  - tasks and subtasks
  - by reasoning type
    - selection, evaluation, estimation, etc.
- by design product structure
  - systems and subsystems
  - components and subcomponents
- by knowledge available
  - by discipline (e.g., kinematics)
- by convention/historical
  - organizational structure
  - legacy systems (e.g., CAD)

Types of Design

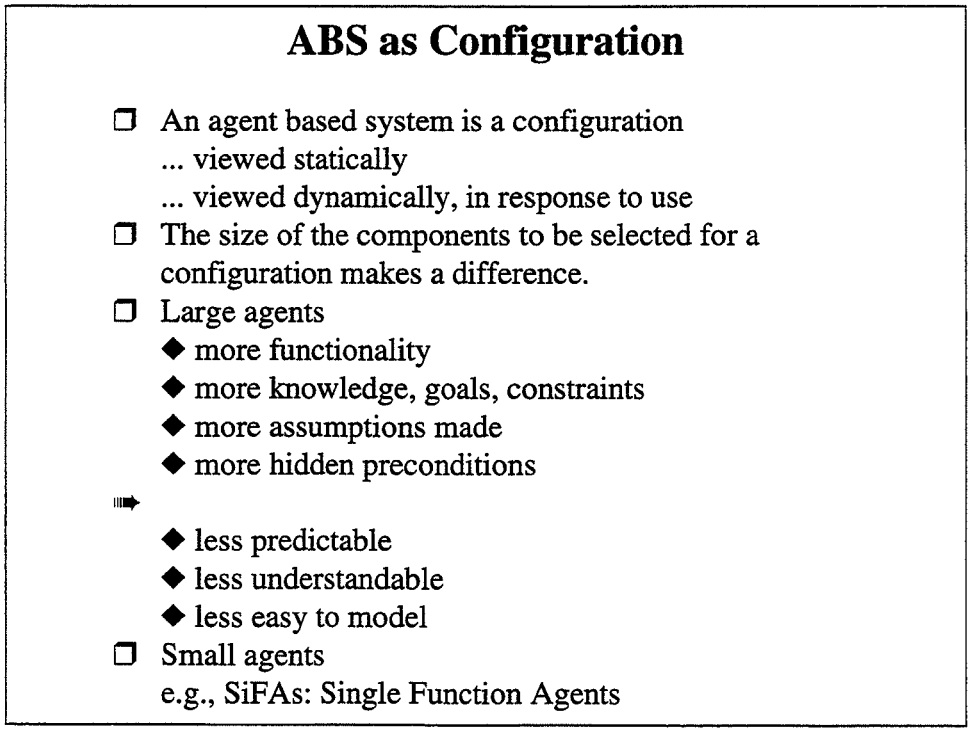
There are many categories of design that appear in the literature. As one moves further away from Parametric, fewer methods and software tools are currently available. Conceptual design, much needed for unusual design tasks, is the hardest to support.

Types of Design

- Parametric
  - High automation,
    many methods.
    - KBS, CSP, CAD, Optimization, etc.
- Configuration
  - Medium automation,
    possible.
    - KBS, constraint techniques, GAs
- Conceptual
  - Low automation,
    much harder at present
    - representation difficult
    - procedures less known
    - rarely routine

ABS as Configuration

Agent-based systems can be seen as a configuration of agents, both in the static sense with agents put together to build a system, but also in the dynamic sense, with interacting agents forming configurations in response to the shared task Large agents, which are quite common and may already exist as legacy systems, have both advantages and disadvantages. Small agents remove many of those disadvantages, but add communication overhead. They would need to be custom built.



The Consequences

From what has already been presented, it appears that the environment will need to be used for unusual, creative, conceptual, non-routine designs. This has many unfortunate consequences for the design of a multi-agent version of the environment.

The Consequences

The more first-time, 1-off, creative, conceptual, non-routine the design is,

the less...

... we can predict the design process.

... we can predict the result.

... we can predict the necessary agents of the MADS.

... we can predict an appropriate organization for the MADS.

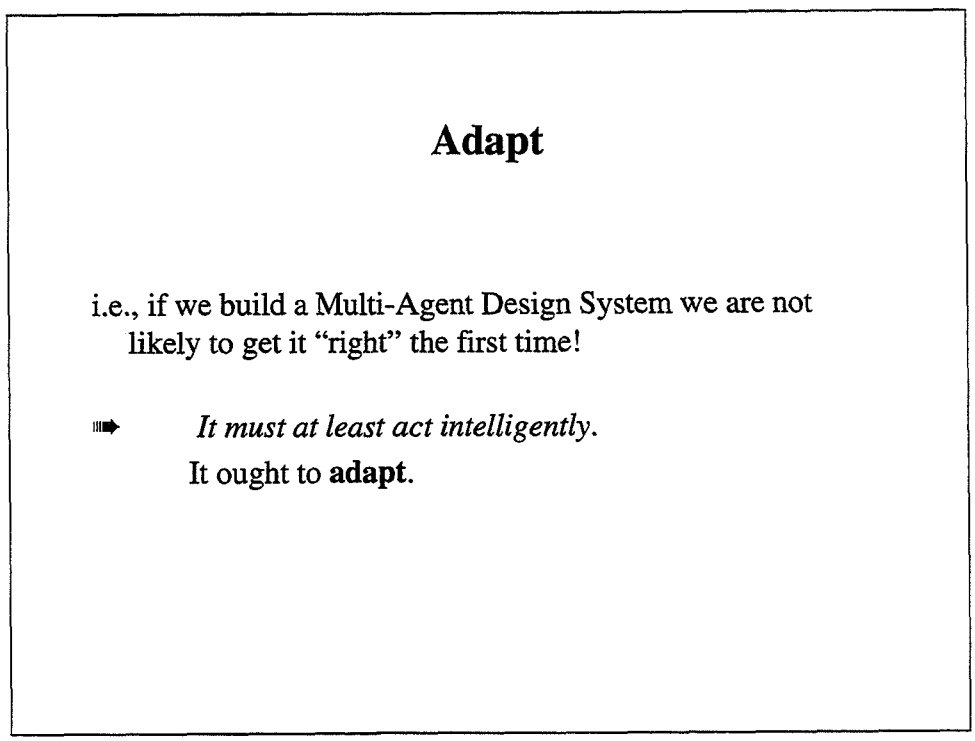
... we can predict the necessary ingredients of the agents.

... we can predict the agent-agent interactions.

... existing software systems (including AI in Design) can help.

Adapt

If we build a multi-agent design system for NASA's design and synthesis environment, we are not likely to get it "right" the first time. In order to compensate for this the system must at least act intelligently. A better response is for it to adapt, and consequently, to compensate for its inadequacies.



The Cure?

The use of learning in multi-agent design systems is quite a new area. Learning might play a part in both the support and automation roles of the environment. There are rich opportunities for learning in MADS.

The Cure?

- ❒ Learning
  i.e., ML in MADS

Rest of the talk:

Support & Automation

Rich Opportunities:

Dimensions of ML in D

Learning needs models

Evaluation of ML in MADS

MADS Research Examples

Conclusions

Support

In a support situation most of the environment's intelligence will be added as it gets used, as it will not be possible to anticipate everything *a priori*. There are many possible things to learn, including learning about the user, the design product, etc.

Support

Conceptual, non-routine, creative, synthesis

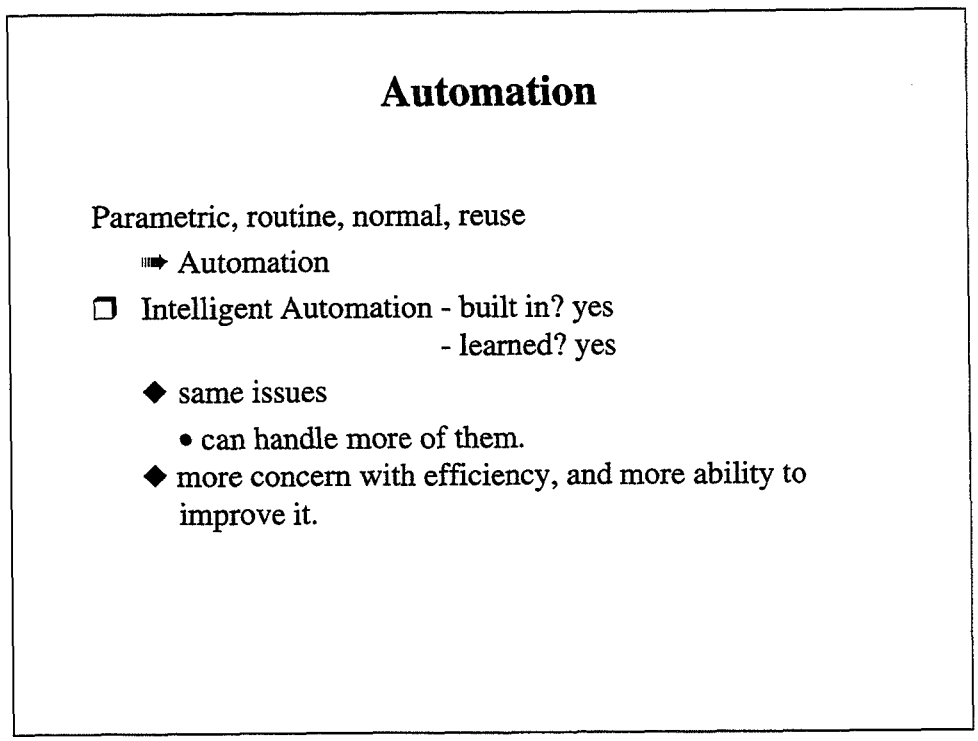
➠ Support

❒ Intelligent Support - built in? no
- learned? yes

- ◆ know the user
  - • learn
- ◆ know the design product
  - • learn
- ◆ know the design process
  - • learn
- ◆ know the architecture
  - • learn
- ◆ know the agents
  - • learn
  - • capabilities, limitations, assumptions, ...
  - • preferences, knowledge, goals, plans, ...
- ◆ know the interactions
  - • learn

Automation

In an automation situation, much more of the intelligence can be built in from the start.



An Agent's Models

In order to learn, agents need to have models. Updating these models constitutes the learning.

An Agent's Model

Agents need models to learn:

- ❒ Model of Agent(s)
  e.g., own abilities;
  beliefs of others.
- ❒ Organizational Model
  e.g., a hierarchy
- ❒ Cooperation Model
  e.g., delegation
- ❒ Communication Model
  e.g., who to send to

[Based on ideas of S. Labidi, 1997]

Variations in MLinD

There are man variations on learning in design systems. The seven dimensions developed by Grecu & Brown provide a large space of learning activities, and suggest new opportunities.

Variations in MLinD

1. What can trigger learning?
   e.g., expectation violations.
2. What elements support learning?
   e.g., sequences of design decisions;
   post-design feedback.
3. What might be learned?
   e.g., design preferences.
4. Availability of knowledge for learning
   e.g., via direct communication.
5. Methods of learning
   e.g., case-based and analogical learning.
6. Local vs. global learning
   e.g., learning between design team agents.
7. Consequences of learning
   e.g., design improvement;
   process improvement;
   {organization improvement}.

[Grecu & Brown 1998c]

ML in MADS Examples

Some of the research at the AI in Design Group at WPI is concerned with learning in design. Next we will provide three examples.

ML in MADS Examples

1 Learning Multidisciplinary Design Methodologies
- ◆ to improve integration

2 Adjusting an Agent's Design Preferences
- ◆ from agent interactions

3 Learning Key Features
- ◆ from expectation violations

❒ Other ML in MA(D)S Work
- ◆ Deng & Sycara 1997
- ◆ Nagendra Prasad, Lesser & Lander 1997
- ◆ plus other ML in MAS work

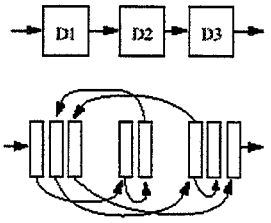
http://dis.cs.umass.edu/research/agents-learn.html

Discipline Problems

This work uses an agent-based system to generate design traces that are turned into design methodologies for multi-disciplinary designs. Agents are built by cutting large blocks of discipline-based knowledge (e.g., D1, below) into smaller pieces. Each piece becomes an agent. The system is exercised with many design problems, generating many traces. Traces are patterns of design methods. These traces are clustered and generalized into methodologies that are appropriate for many design problems. Hence, methodologies are learned from system behavior.

Discipline Problems

- ❒ Multidisciplinary design problems.
- ❒ Knowledge: large discipline-based chunks.
- ❒ Leads to less integrated design process.
- ❒ Break chunks into smaller methods.
- ❒ Encode as agents.
- ❒ Experiment with resulting ABS.
- ❒ Traces ➠ learned Methodologies.

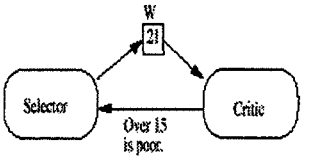


[Shakeri, Brown & Noori 1998]

Design Preferences

This work uses a conflict between agents to trigger learning. The Selector sets 21 as the value of W, but the Critic provides a critique indicating that values over 15 are poor. The Selector learns to avoid situations such as this. Learning significantly improved the number of interaction that occurred due to conflict.

Design Preferences



- Learning is triggered by conflict
  e.g., Selector-Critic; Selector-Selector
- Use concept formation to learn responses of agents in particular situations.
- Depending on experiences, interactions reduced by 9% - 55%

[Grecu & Brown 1996]

Key Features

In this work, what triggers learning is expectation violations. The agent reasons out what features might have contributed to the violation, and then uses some learning experiments to determine the key features, I.e., those that are most predictive of the violation.

Key Features

- Agents have expectations
  - e.g., values,
    response time of an agent,
    quality of agent's response, ...
- Agent detects expectation violations
- Knowledge used to produce list of features that might have produced this violation.
- Determine key feature(s) using inductive learning experiments.
- Incorporate learned relationship into knowledge.

[Grecu & Brown 1998a]

ML in MADS Evaluation

A complex multi-agent design system requires very careful and comprehensive evaluation, as there are many possible effects that might alter its performance.

ML in MADS Evaluation

- What to consider when evaluating distributed learning in design systems.

For example:

- The response to objectives
  e.g., low cost
- Learning processes shared by multiple objectives.
- Interference of learning processes.
- Cross-talk resulting from training on several classes of design problems.

[Grecu & Brown 1998b]

Conclusions

The multi-agent implementation of the design and synthesis environment will have faults built into it. It will need to learn in order to compensate for and correct these problems. The area of learning in multi-agent design systems is an important and exciting new challenge that will have significant payoffs.

Conclusions

- The nature of the design problem affects the use of agents for constructing an environment for the design of future aerospace systems.
- Any MADS built will be inefficient and ineffective relative to the task.
- It will need to compensate for these weaknesses by learning.
- ML in design research is flourishing.
- ML in MAS research is flourishing.
- ML in MADS research is newer but is an area in which major opportunities exist for significant advances.

References

This is a very small selection of the references about agents, learning in design, learning in multi-agent systems, and learning in multi-agent design systems.

References

D. L. Grecu and D. C. Brown, "Learning by Single Function Agents During Spring Design," *Artificial Intelligence in Design'96,* J. S. Gero and F. Sudweeks (eds.) , Kluwer Academic Publishers, 1996, pp. 409-428.

D. L. Grecu and D. C. Brown, "Guiding Agent Learning in Design," *Proc. KIC3: Third Workshop on Knowledge Integrated CAD,* Tokyo, Japan, 1998a.

D. L. Grecu and D. C. Brown, "Evaluating the Impact of Distributed Learning in Real-World Design Problems," *AID98 Workshop on Machine Learning in Design,* Lisbon, Portugal, 1998b (preprints).

References (Cont'd.)

D. L. Grecu and D. C. Brown, "Dimensions of Machine Learning in Design," *AI EDAM,* special issue on Machine Learning in Design, A. H. B. Duffy, D. C. Brown and A. K. Goel (eds.) , Cambridge University Press, 1998c, pp. 117-121.

S. Labidi, "Managing Multi-Expertise in Design of Effective Cooperative Knowledge-Based Systems," *Proc. IEEE KDEX'97, Knowledge and Data Engineering Exchange,* Newport Beach, CA, 1997.

M. V. Nagendra Prasad, V. R. Lesser and S. E. Lander, "Learning Organizational Roles for Negotiated Search in a Multi-Agent System," in special issue on Evolution and Learning in Multi-Agent Systems, *International Journal of Human-Computer Studies*, 1997.

References (Cont'd.)

C. Shakeri, D. C. Brown and M. N. Noori, "Discovering Methodologies for Integrated Product Design," *Proc. Artificial Intelligence and Manufacturing: Second Bi-annual AI & Mfg. Workshop,* Albuquerque, NM, 1998.

D. Zeng and K. Sycara, "Benefits of Learning in Negotiation," *Proc. AAAI-97.*

Page intentionally left blank

39-61
032523
370173

Developing CORBA-Based Information Agents

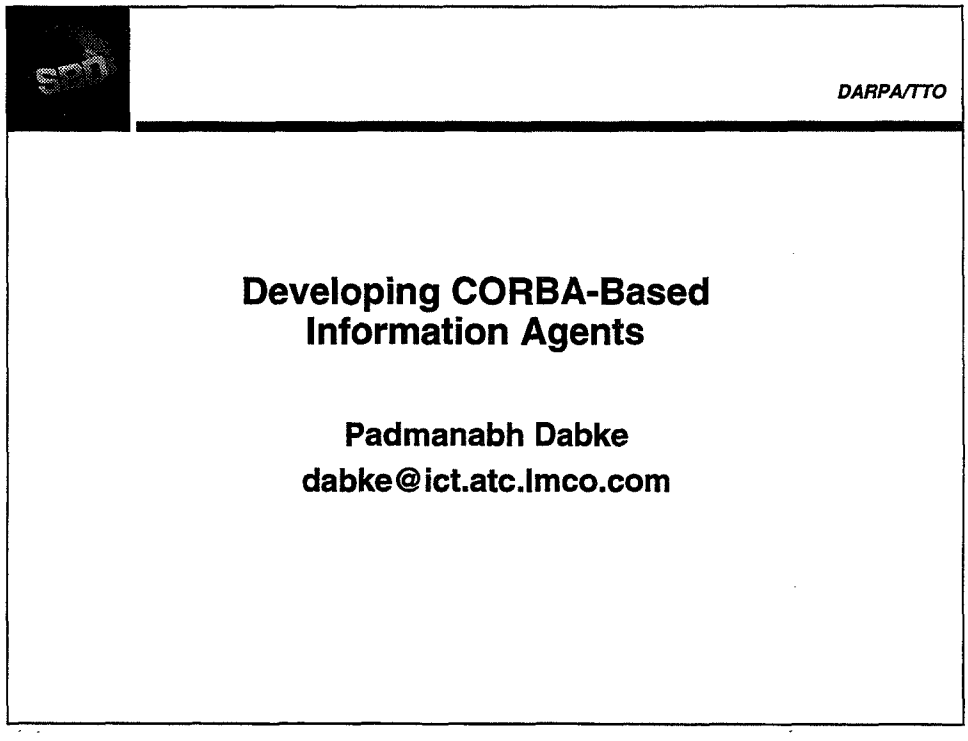
209

Padmanabh Dabke
Lockheed Martin Missiles & Space
Palo Alto, CA

Page intentionally left blank

Developing CORBA-Based Information Agents

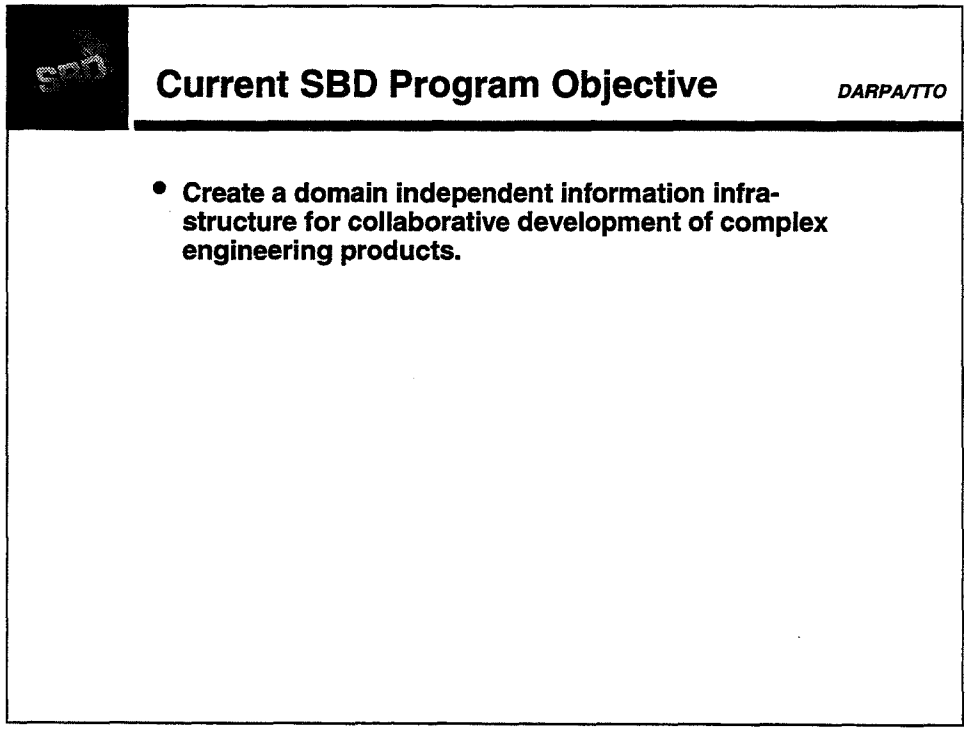
Padmanabh Dabke
Lockheed Martin Missiles & Space
Orgn H1-43, Bldg. 255
3251 Hanover Street
Palo Alto, CA 94304-1191
dabke@ict.atc.lmco.com



The work described in this presentation was supported under the Defense Advanced Project Agency (DARPA) contract MDA 972-95-D-0003

SBD Objective

The primary goal of the current phase of the Simulation Based Design(SBD) [1] program is to create a software infrastructure for managing distributed collaborative product development projects. The SBD infrastructure is not tied to any application domain, rather it provides domain-independent enterprise integration middleware for achieving Integrated Product and Process Development (IPPD). The infrastructure assists in capturing important product characteristics as well as business processes within an enterprise. Once the product and process related information is captured, SBD infrastructure facilitates effective management of an IPPD environment by maintaining product coherency and automating enterprise processes.

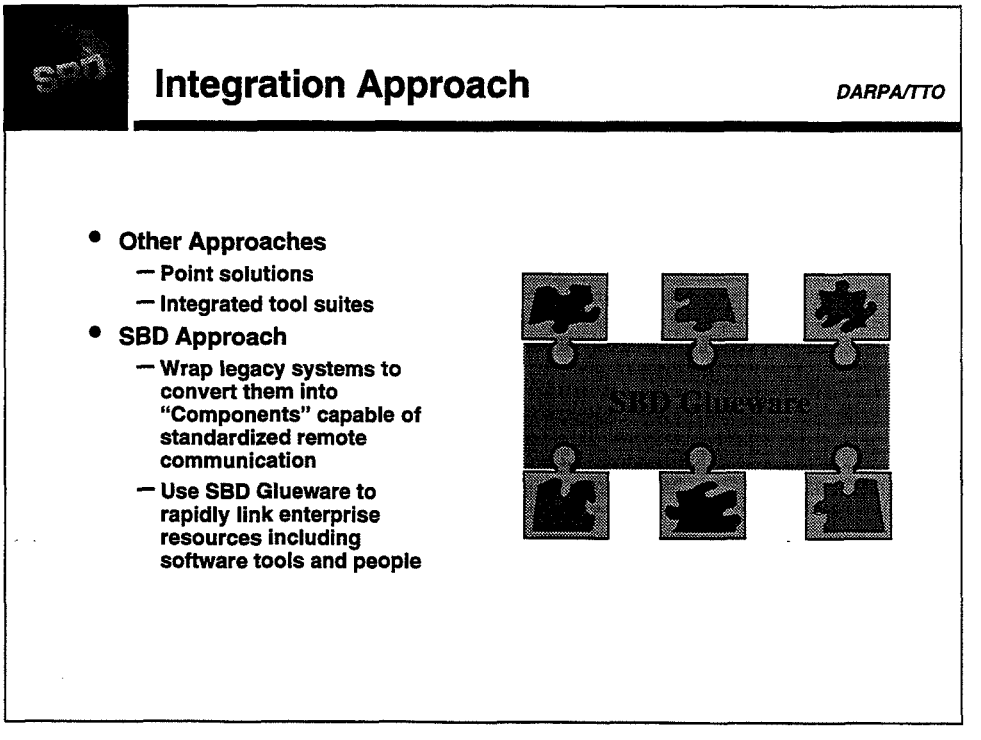


Integration Approach

Traditional approaches to enterprise integration can be broadly classified into two types. In the first approach, a set of software tools are linked together in an ad-hoc manner via a set of scripts, hard-coded programs, etc. The second approach, called the "gateway" approach, involves committing to a single product or an integrated family of products and using "gateways" provided by the vendor to integrate other tools.

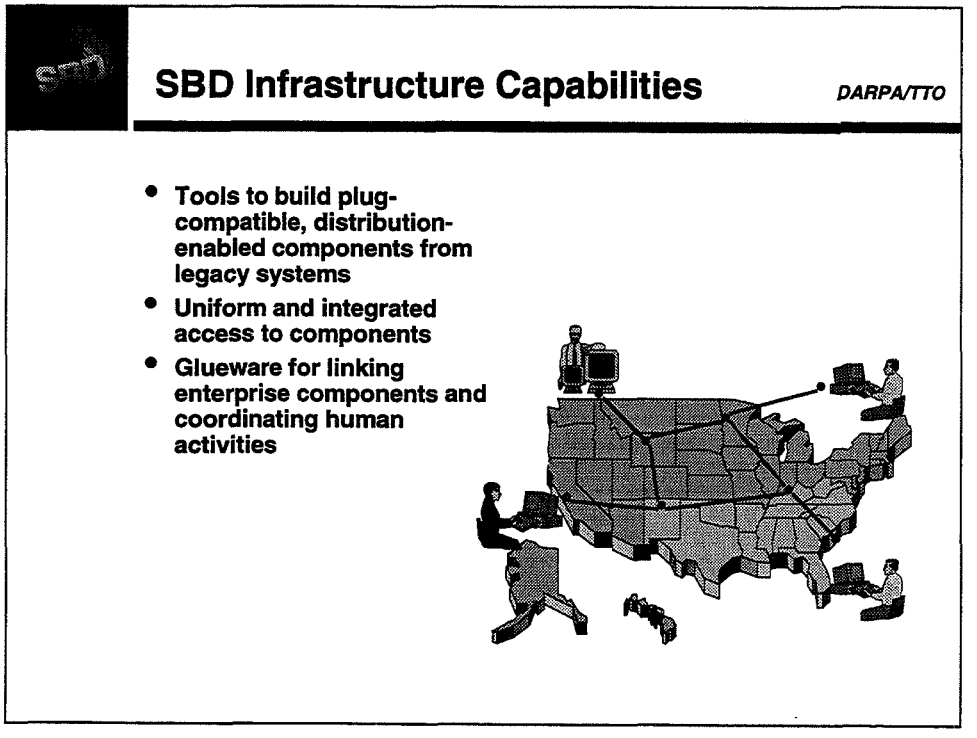
The first approach works well for static environments involving a few co-located teams of people. The approach lacks the maintainability and reusability needed in more dynamic environments. The second approach is both efficient and powerful within a single discipline employing a narrow set of technologies; however, it is inadequate for more diverse multi-disciplinary environments.

SBD adopts a "component-based" approach towards enterprise integration. In this approach the legacy systems are encapsulated as "components" which are software objects with well-defined interfaces for remote communication. SBD then provides a variety of "glueware" services for integrating these components.



SBD Infrastructure Capabilities

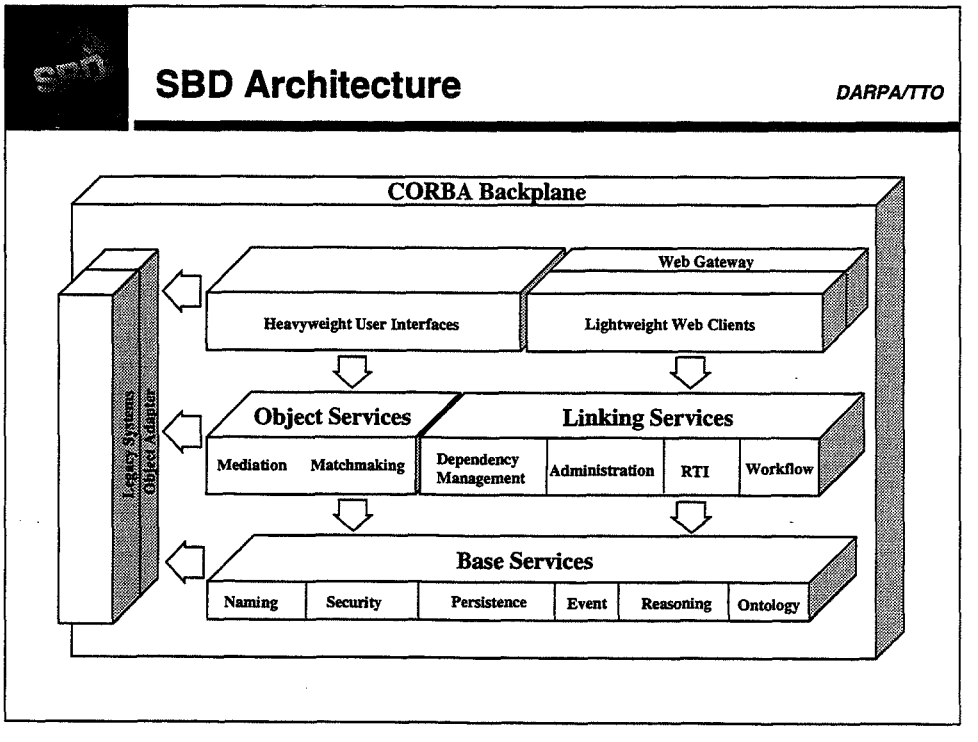
SBD infrastructure provides three broad capabilities to support its component-based integration approach. It provides tools that assist users in "wrapping" legacy systems to create Common Object Request Broker Architecture (CORBA) [2] based components. Once built, these components can be accessed in a uniform and integrated manner via SBD's cataloging service. Finally, SBD middleware provides "glueware" services in the form of workflow automation, event management, etc., that provide mechanisms for maintaining a coherent product model, and automating business processes.



SBD Architecture

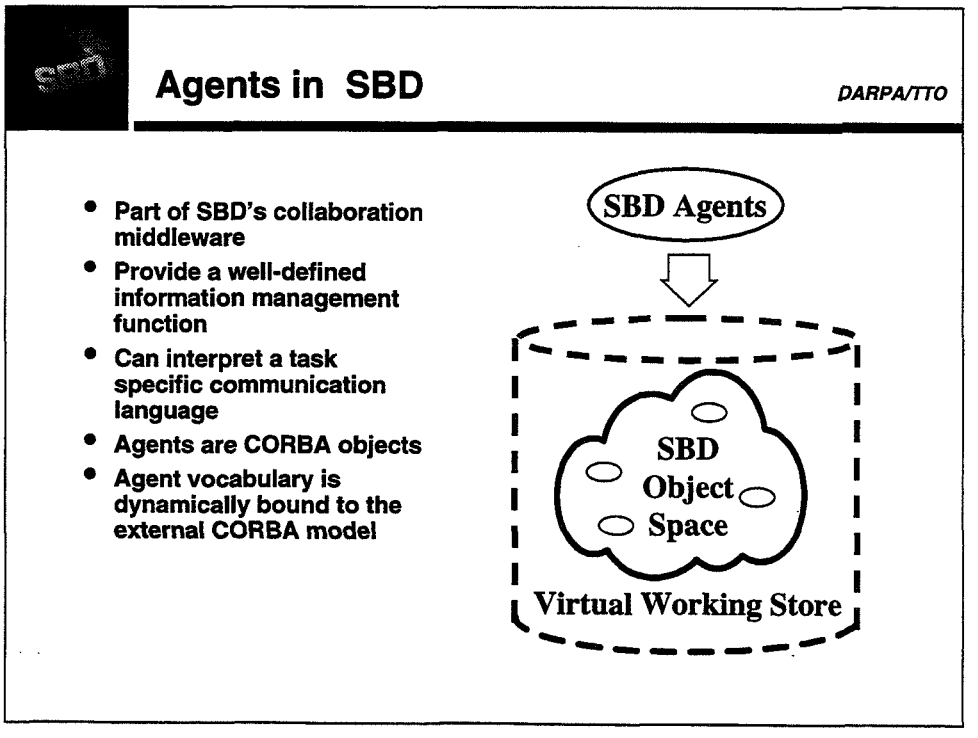
SBD's integration approach is implemented as a set of CORBA-based services collectively known as the "Core Processing System" (CPS). The CPS consists of three layers, each plugged into the CORBA backplane. The bottom layer consists of a collection of "Base Services" such as Naming (which provides a global namespace for all SBD objects), security, persistence (e.g. databases), and ontology (CORBA interface repository being an example of a simple ontology server).

The middle layer in the architecture consists of collaboration services that facilitate location and integration of CORBA-based components. Services in this layer can be further classified into "Object Services" and "Linking Services." Object Services enable integrated and uniform discovery of enterprise resources. Linking services assist enterprise management by automating workflows, change-notifications, event management, etc. Finally, the top layer of the SBD system consists of the user interfaces to the underlying system. Heavyweight user interfaces can be built directly on the CORBA backplane. Alternatively, very thin Web-based interfaces can be developed via a "Web Gateway" that bridges CORBA and HTTP-based communication.



Agents in SBD

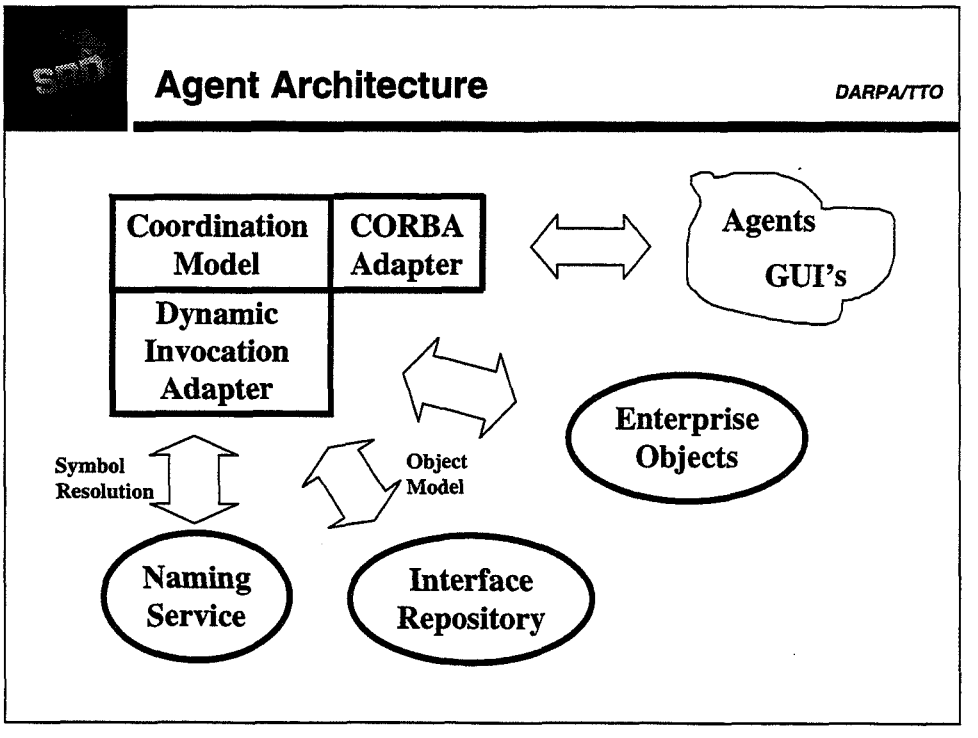
SBD's Information Agents are part of the middle layer of the CPS. Each agent in SBD has four distinguishing characteristics. An agent is a "component" in the SBD environment and therefore a CORBA object. Further, each agent implements a reasoning scheme that automates a focused information management paradigm such as mediation and dependency management. Each agent understands a paradigm-specific control language that allows human users and other agents to effectively communicate with the agent. Finally, the most important feature of SBD agents is that their vocabulary is dynamically bound to the external CORBA objects registered with the CPS. This feature allows SBD agents to use the external object space as a virtual working store (as opposed to maintaining a huge database of objects internally). This feature allows SBD agents to have a small footprint and yet quickly adapt to the application domain through external application CORBA objects.



Agent Architecture

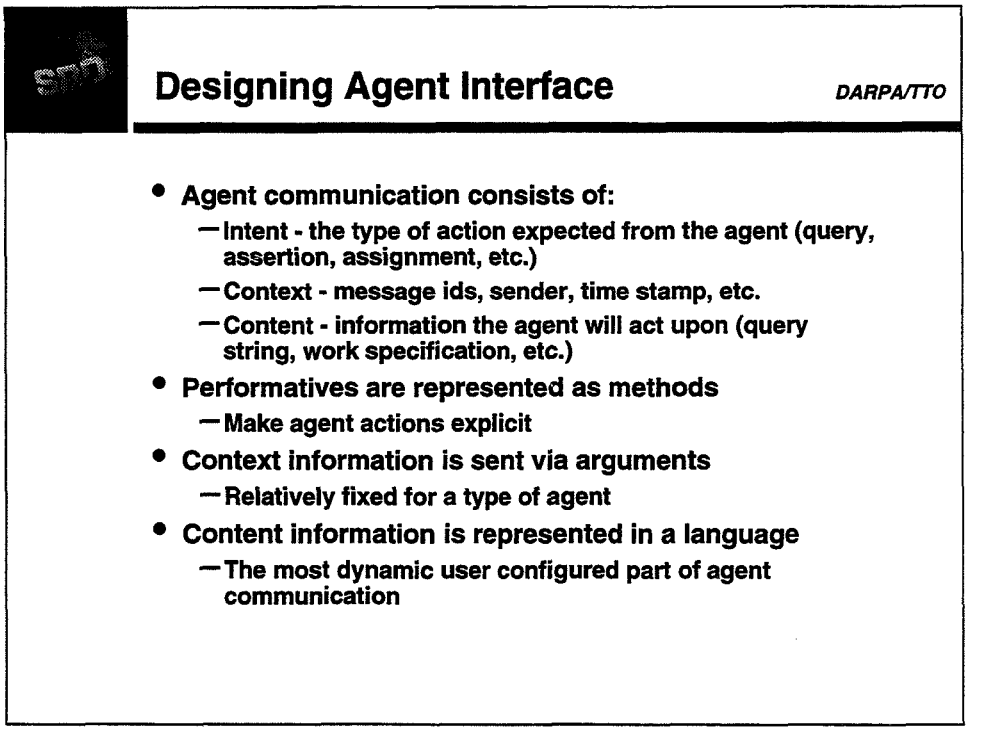
Each SBD agent is composed of three components: an inference engine implementing the Agent Coordination Model, a CORBA adapter, and a Dynamic Invocation Adapter (DIA). The first two components are specific to the agent being implemented. The DIA is a reusable component that is part of all SBD agents.

An SBD agent communicates with other agents and client user interfaces through the CORBA adapter. An agent operates on enterprise CORBA objects through the DIA. DIA in turn relies on CORBA Naming Service and Interface Repository to enable dynamic communication with external CORBA objects.



Designing Agent Interface

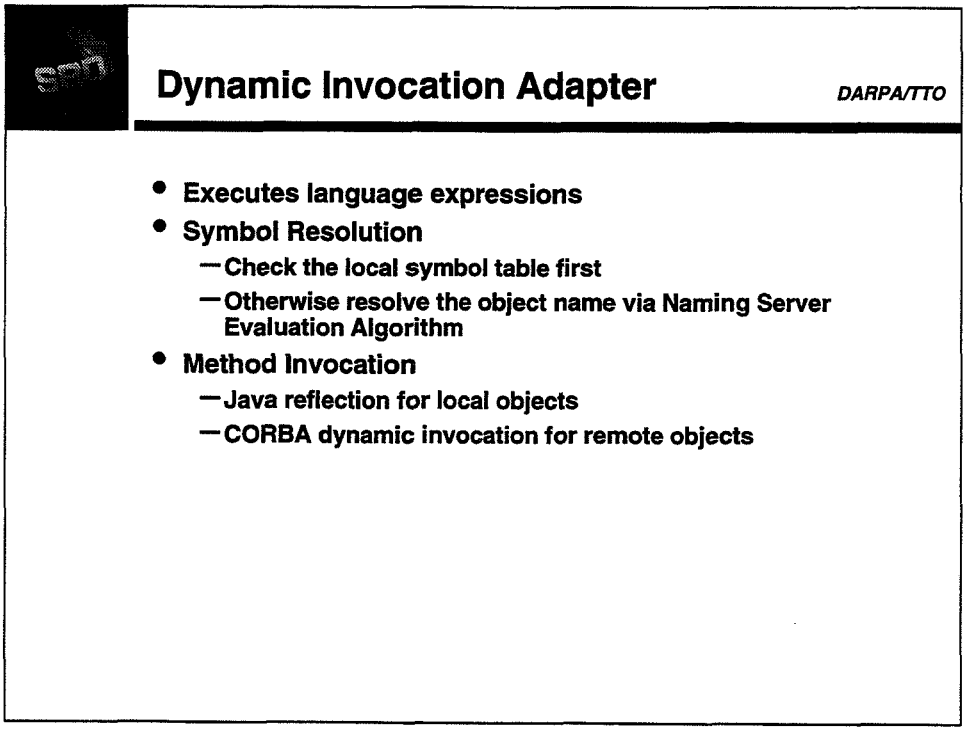
Communicating with an SBD agent typically requires a specification of the intent (whether it is a query, assertion, task to be performed by the agent, etc.), content (query string, subscription, etc.), and a context for the message (sender name, message ID, etc.). The CORBA interface for each agent is designed by mapping these three elements in a standard manner. "Performatives" that capture the intent are mapped into methods with the same name (e.g., the Query Agent has a method called "evaluate"). The content portion is passed as an argument and is expressed in an agent-specific language. The contextual information is also passed as arguments of appropriate data type.



Dynamic Invocation Adapter

The Dynamic Invocation Adapter (DIA) is used by SBD agents to evaluate expressions in agents' content language. Evaluating language expressions requires resolving symbols to the corresponding internal or external objects, as well as invoking methods on the resolved objects.

The DIA exploits the introspection capability built in Java as well as the CORBA framework. Local symbols are resolved by looking up the internal symbol table. External objects are looked up in the CORBA Naming Service. The method invocation in local objects is supported through the Java reflection API. The remote method invocation is supported via CORBA's Interface Repository and the Dynamic Invocation Interface (DII).



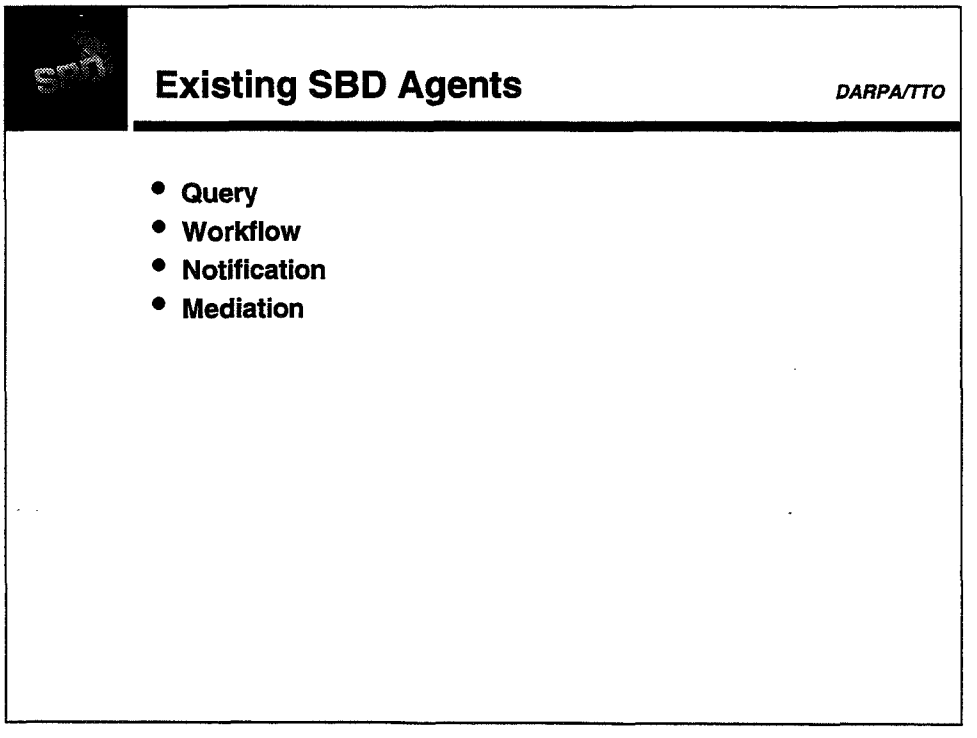
Existing SBD Agents

At the time of this presentation, SBD's Core Processing System includes four information agents. The Query Agent, also referred to as the Object Server, serves as an object integrator and supports distributed queries specified using a subset of Object Query Language (OQL) defined by Object Data Management Group (ODMG) [3].

The Workflow Agent understands a tasking language. The fundamental unit in this language is a task that has inputs, outputs, a goal, and a plan for achieving the goal. The language includes constructs for exchanging data, loops, switches, spawning activities in series or parallel, etc.

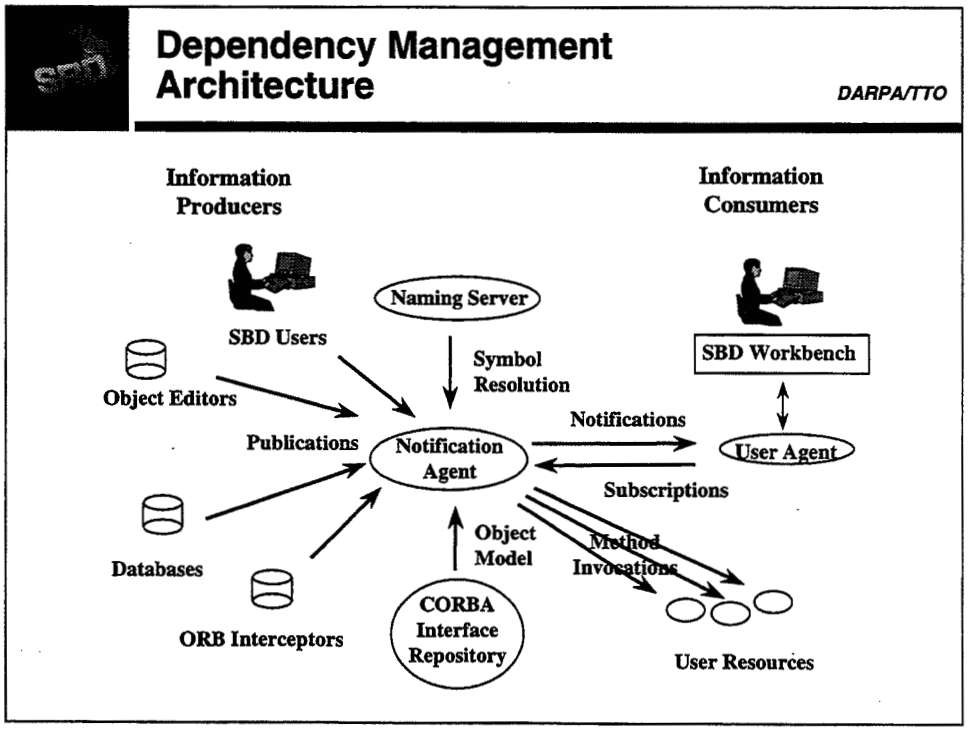
The mediation agent provides an engine and a graphical user interface for constructing data transformations. Users can define transforms between two object models in Java, or as tasks in the tasking language. These transforms can then be interactively composed to provide the mapping in the selected object models.

The Notification Agent provides a distributed dependency management service based on the publish-subscribe model. Notification Agent is described in more detail in the next part of this presentation to illustrate the design and deployment of SBD's information agents.



Dependency Management Architecture

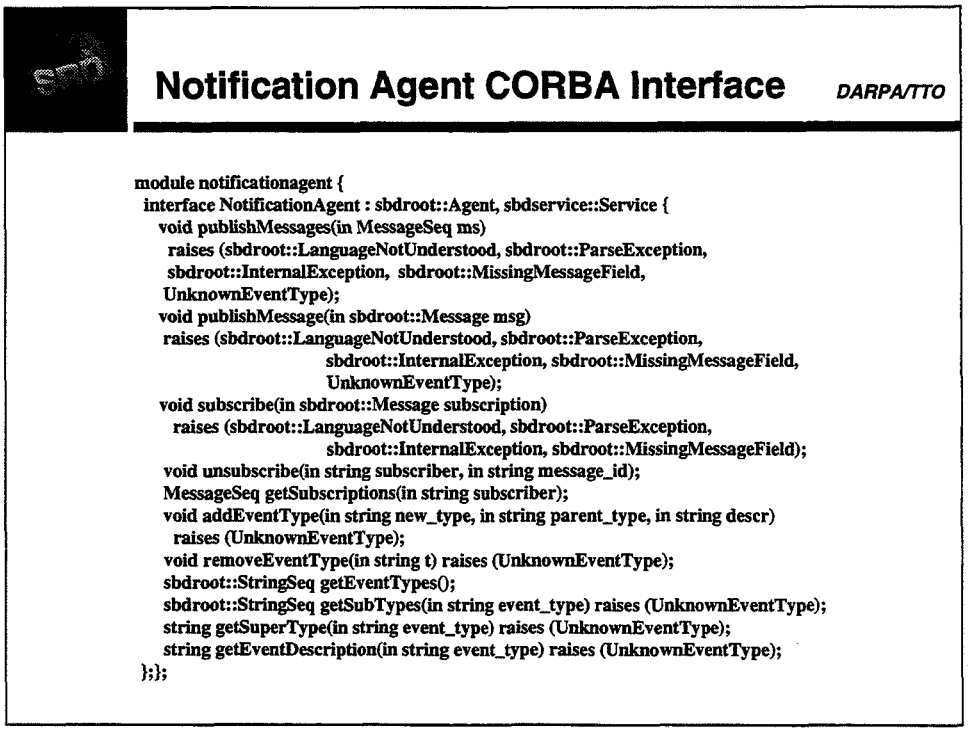
Notification Agent is deployed in situations where information producers do not know the consumers interested in their data, and information consumers are not aware of information suppliers but can characterize potentially relevant information. Information producers (e.g., databases, report writers, etc.) publish shared information with the Notification Agent with an associated event-type. Information consumers post subscriptions to the Notification Agent. The subscriptions characterize potentially interesting information and specify actions on the SBD's object space. The Notification Agent acts as an information broker in this publish-subscribe model. When the agent receives a publication, it matches the associated data with each subscription. If the published data matches a subscriber's interest, the agent operates on the existing enterprise objects (e.g., sending notifications to User Agents representing human participants, running analysis tools with appropriate data sets, etc.) as specified in the subscription.



Notification Agent CORBA Interface

Notification Agent supports three performatives, namely, "publish," "subscribe" and "unsubscribe." The agent's CORBA interface provides corresponding "subscribe" and "unsubscribe" methods. For efficiency, two publish methods (one for publishing a single message, and another for publishing a set of messages simultaneously) are provided.

The agent also provides additional methods for creation, deletion and querying event type information. These methods allow the users to create a hierarchy of event types. Each publication is associated with exactly one event type. The event type offers a simple and widely used means of event filtering.

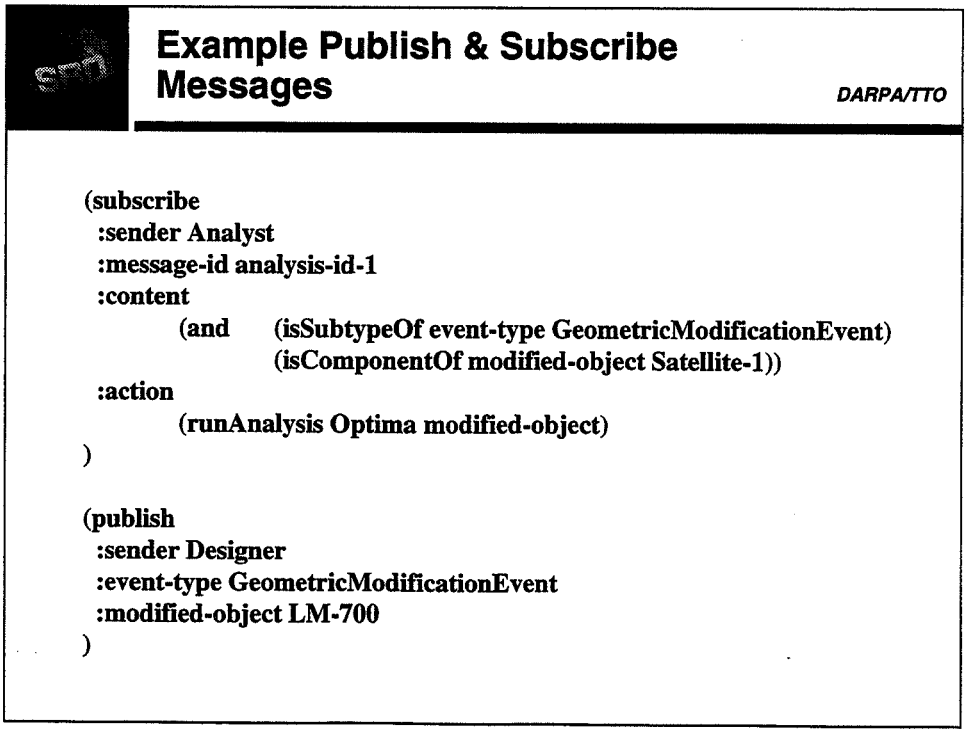


Example Publish and Subscribe Messages

This viewgraph shows examples of a publish and subscribe message. The message is shown in Knowledge Query and Manipulation Language (KQML) [4] only for the purpose of illustration. The actual communication is achieved via the agent's CORBA interface described in the previous viewgraph.

The publish message specifies the event type to be "GeometricModificationEvent." The publication has two data fields: "sender" of the message is a User Agent called "Designer," and "modified-object" is a satellite bus called "LM-700."

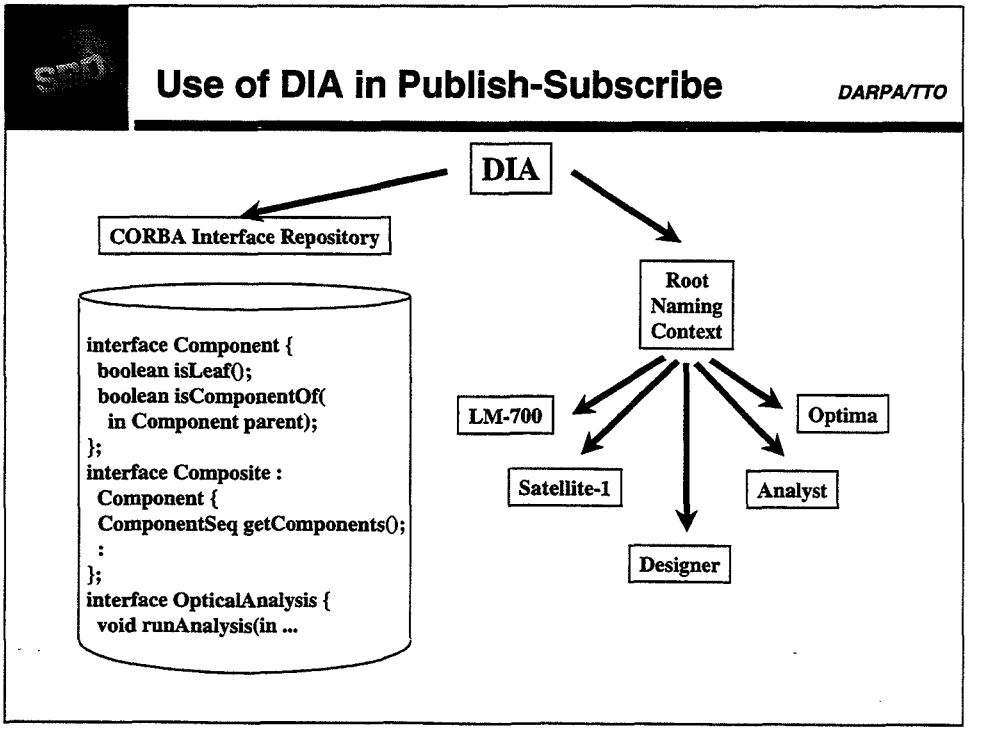
The subscription message indicates interest in events of type "GeometricModificationEvent." The subscriber is interested in the event only if the object in the "modified-object" field is a component of "Satellite-1." If a match is made, the subscriber wants the Notification Agent to run optical analysis on the component using a tool called "Optima."



Use of DIA in Publish-Subscribe

The publish/subscribe messages presented in the previous viewgraph are interpreted by the Notification Agent via Dynamic Invocation Adapter (DIA). The interpretation of the messages depends on five objects (Designer, LM-700, Satellite-1, Optima, Analyst) being registered with the Naming Service. LM-700 and Satellite-1 are of type "Composite" and their interface definition is registered with CORBA Interface Repository.

Note that the Notification Agent itself is completely unaware of the domain-specific part-subpart relationship between LM-700 and Satellite-1. Effective management of information within any domain requires use of such knowledge. However, maintaining all domain specific knowledge internally would significantly increase the size of the Notification Agent and create a potential bottleneck. Delegating part of the reasoning to external enterprise objects allows the agent to remain small, quickly incorporate the domain knowledge, and distribute the computation for better performance.



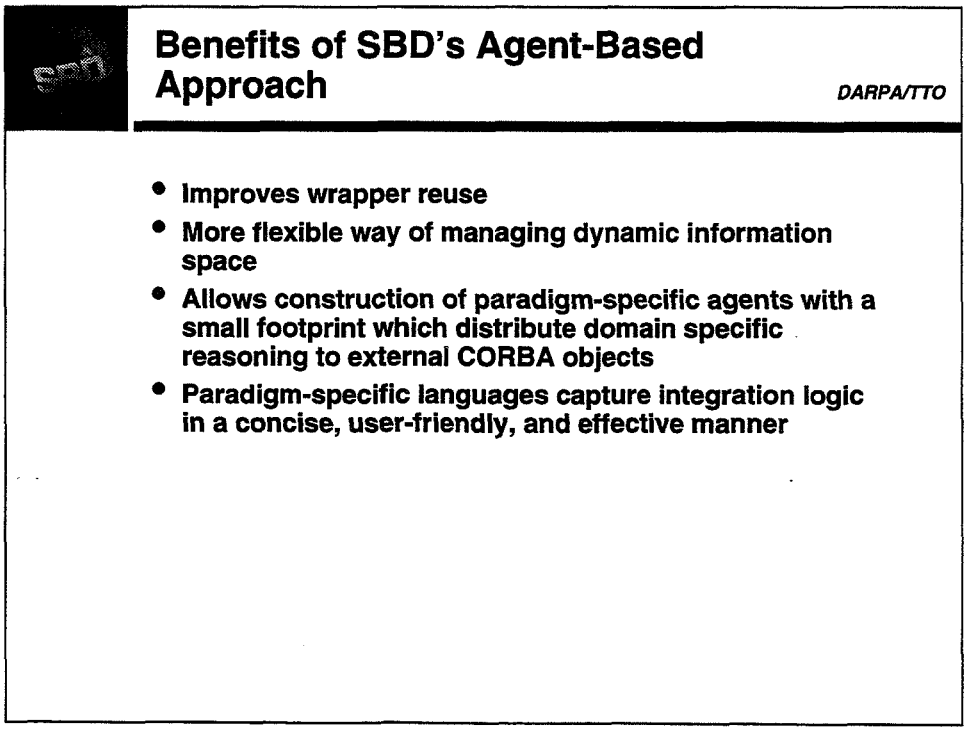
Benefits of SBD'S Agent-Based Approach

SBD's information agents provide an effective mechanism for establishing dynamic information links between enterprise components. Agents make it possible to cleanly separate wrapping of legacy systems from linking them. This allows wrapper builders to focus on exposing the legacy capability without linking consideration, thus improving wrapper reusability across multiple integration schemes.

As business processes evolve, the changes can be implemented by simply sending localized messages to responsible information agents that capture the revised plan of propagating information, as opposed to global recompilation in a monolithic, hard-coded integration environment.

As illustrated by the publish-subscribe example, SBD's agents delegate much of the reasoning to external enterprise objects. This implementation approach allows SBD agents to have a small footprint, and at the same time, quickly adapt to the application domain.

Each SBD agent understands a paradigm-specific language that provides a concise notation for expressing business logic in the coordination model implemented by the agent. Efforts are underway to provide visual communication interfaces for further improving the usability of the agents.



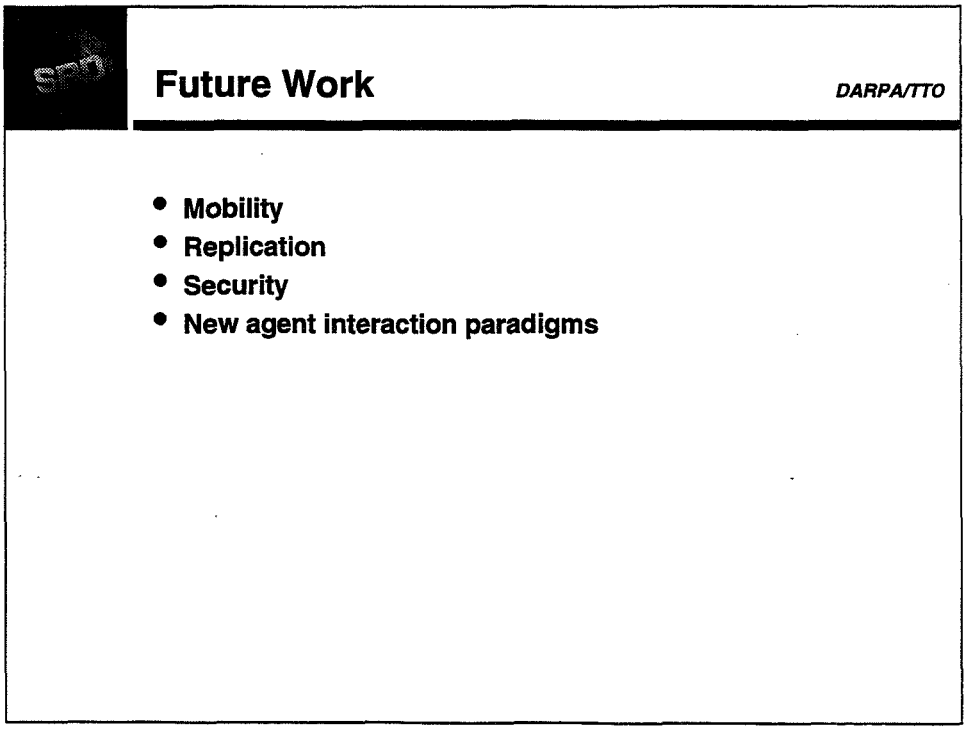
Future Work

One of the key concerns in deploying the CPS is it's performance in a client/server environment and scalability. Server and Client platforms come in all shapes and sizes and there is no universal solution for dividing computation between clients and servers. One of the future areas of research is to use a mobile agent model to create adaptive client/server systems that optimize the client and server computational load.

The issue of scalability can be addressed by creating a federation of information agents to distribute information brokering. The SBD team is currently formulating mechanisms to determine "when" and "how" to replicate agents, and how to keep multiple agent instances in a consistent state.

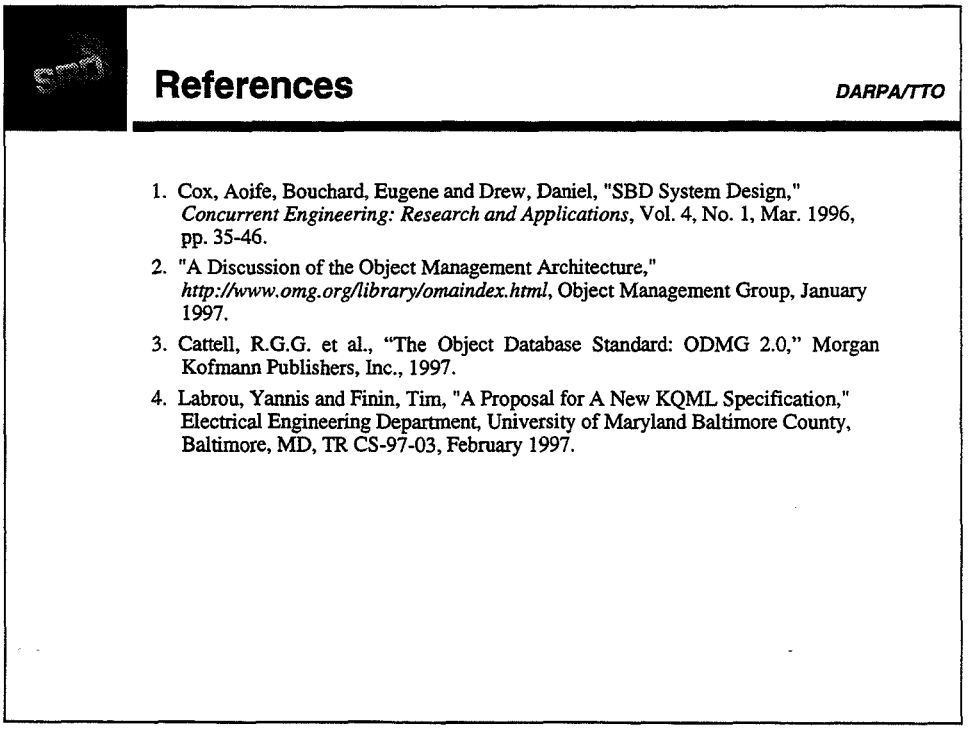
Security, including authentication, authorization, data encryption, etc., is a major concern in deploying the CPS. SBD's approach towards incorporating these features will most likely be based on an enterprise container model. In this model SBD agents will operate as components within a container and delegate the security (as well as transaction management, persistence, etc.) implementation to the container.

Finally, the SBD team is currently examining new agent interaction and coordination paradigms (such as market-based control strategies, adaptive agents, etc.) and evaluating their applicability to SBD's pilot applications.



References

1. Cox, Aoife, Bouchard, Eugene, and Drew, Daniel, "SBD System Design," *Concurrent Engineering: Research and Applications*, Vol. 4, No. 1, Mar. 1996, pp. 35-46.
2. "A Discussion of the Object Management Architecture," *http://www.omg.org/library/omaindex.html*, Object Management Group, January 1997.
3. Cattell, R.G.G. et al., "The Object Database Standard: ODMG 2.0," Morgan Kofmann Publishers, Inc., 1997.
4. Labrou, Yannis, Finin, Tim, "A Proposal for A New KQML Specification," Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD, TR CS-97-03, February 1997.



Page intentionally left blank

510-61
032524
57017
24

Coordinating Intelligent Agents

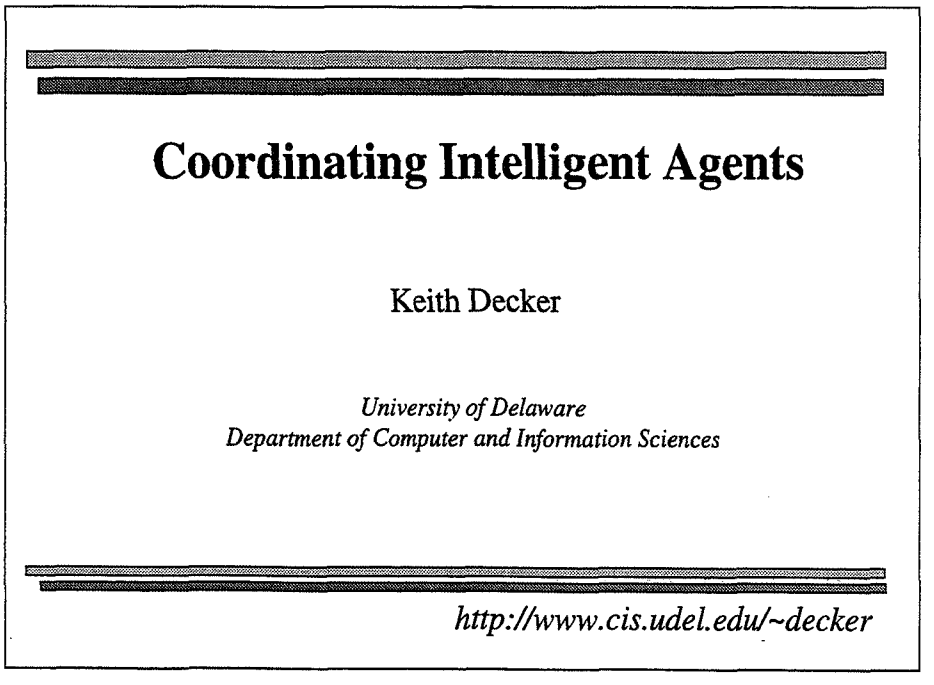
Keith S. Decker
University of Delaware
Newark, DE

Page intentionally left blank

Coordinating Intelligent Agents

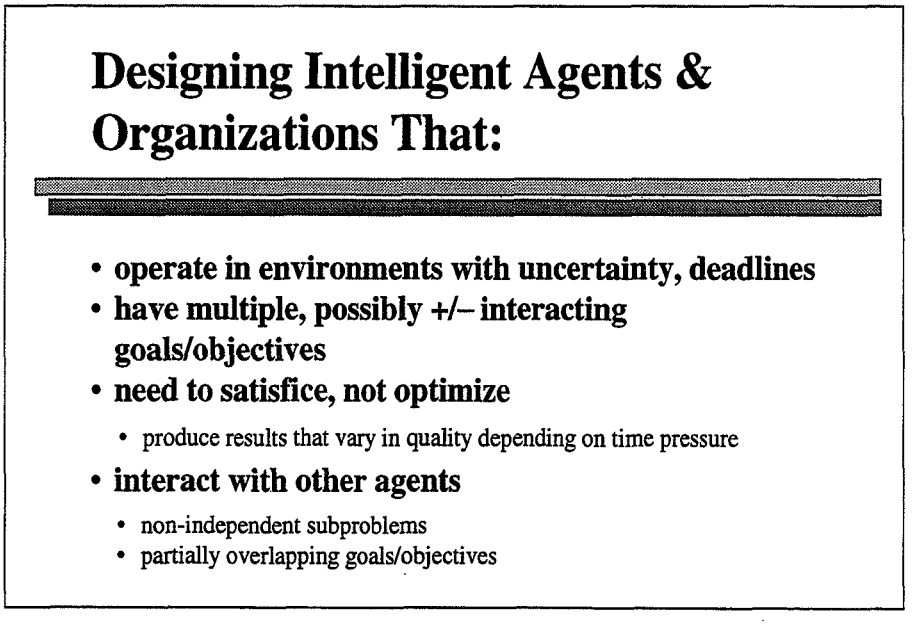
Dr. Keith S. Decker
Department of Computer and Information Sciences
103 Smith Hall
University of Delaware
Newark, DE 19716

Phone: (302) 831-1959
Fax: (302) 831-4091
Email: decker@cis.udel.edu
WWW: http://www.cis.udel.edu/~decker/



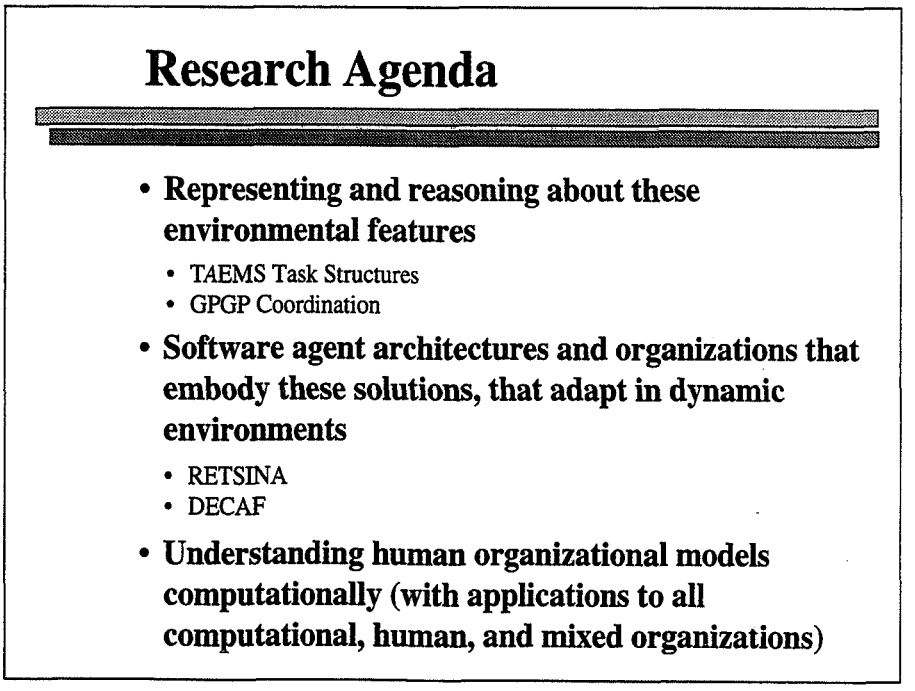
Designing Intelligent Agents and Organizations

Our research program is involved in developing intelligent software agents (large, persistent, autonomous, communicative, goal- and data-driven computer programs) and *organizations* of these agents (including sometimes humans) that can operate in environments where there is a lot of uncertainty about what is happening and where there may be time pressures or deadlines. The agents will in general have many goals, some partially overlapping or conflicting. We cannot realistically look for optimal solutions, but instead must satisfice—try to find a solution that is "good enough," in the time and resources that are available. No agent can work completely alone (regular distributed systems research in CS tends to deal with distributed execution of independent processes).



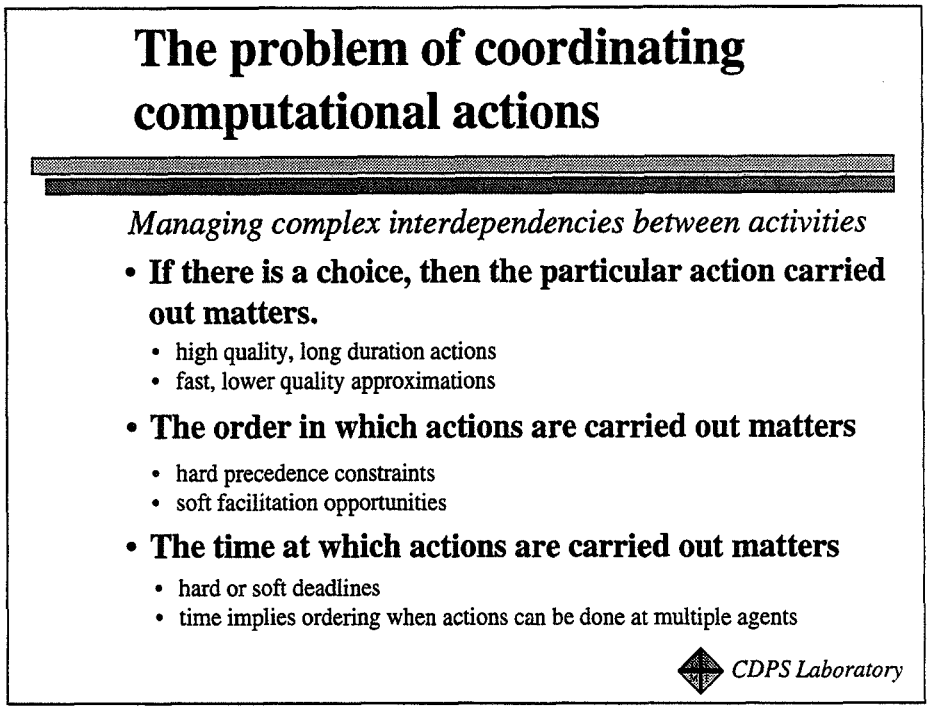
Research Agenda

Our research program can be divided into three areas. First, how to formally represent and reason about these sorts of problems, both as a software engineer and internally as a software agent. To this end we developed the TAEMS task structure description language (representing what we think are the important concepts) and the GPGP approach to coordination (a way to reason about TAEMS descriptions within each software agent so that a team of them acts coherently together). Secondly, we actually build software and tools for building actual software agents. This includes the RETSINA project that I started with Dr. Katia Sycara at CMU, and the DECAF project which is a Java version here at the University of Delaware that combines features of RETSINA and my work on coordination at UMass. Finally, we are also interested in understanding, modeling, and even imitating human organizational structures in the context of software agents (both organizations of ALL software agents, and mixed human/software agent hybrid organizations). This is very important both because complex problems often need more than trivial organizational solutions, and because most real systems are embedding in existing human organizations (so they must respect the boundaries of those organizations and the roles of the people with whom they interact).



The Problem of Coordinating Computational Actions

The problem of coordinating activities (at the level of scheduling action) mostly falls into three general areas: choosing among alternatives, ordering, and locating actions in time with respect to the ordering.



This problem is made worse by the fact that no single agent will have a complete view of the problem being solved. Even if the agents communicated enough to develop such a global view, in many real problems it would soon be out of date, as the world is dynamically changing around the agents. Finally, even if the agents developed a global view and the world stood still while they thought about it, there is still the problem of action outcome uncertainty.

Some of the ways that people deal with the coordination problem are to create schedules, plans, appointments, and so on (commitments to certain actions at certain times and places). At a higher level, people create laws, rules, or social norms that allow us to "know" what others will do without actually communicating with them in every situation (the obvious example is traffic laws that say, for example, what side of the road to drive on and what to do at intersections. Finally, human organizations (and the roles within) allow coordination via general, long-term commitment to certain classes of actions.

The problem of coordinating computational actions

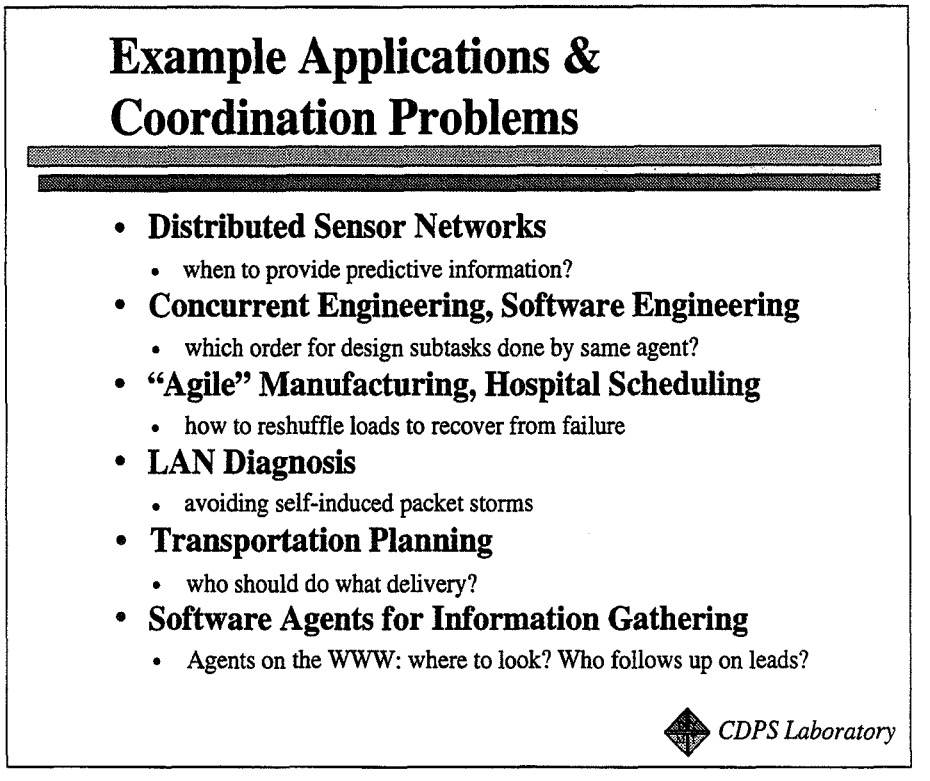
Continued

- **Difficulties in CHOOSING and TEMPORALLY ORDERING actions**
  - Incomplete view of the problem
  - Dynamically changing situation
  - Uncertainty in the outcomes of actions
- **Example Coordination Mechanisms**
  - Schedules, plans, timelines, appointments, commitments
  - Laws, rules, social behavioral norms
  - Organizations, roles, negotiated order

CDPS Laboratory

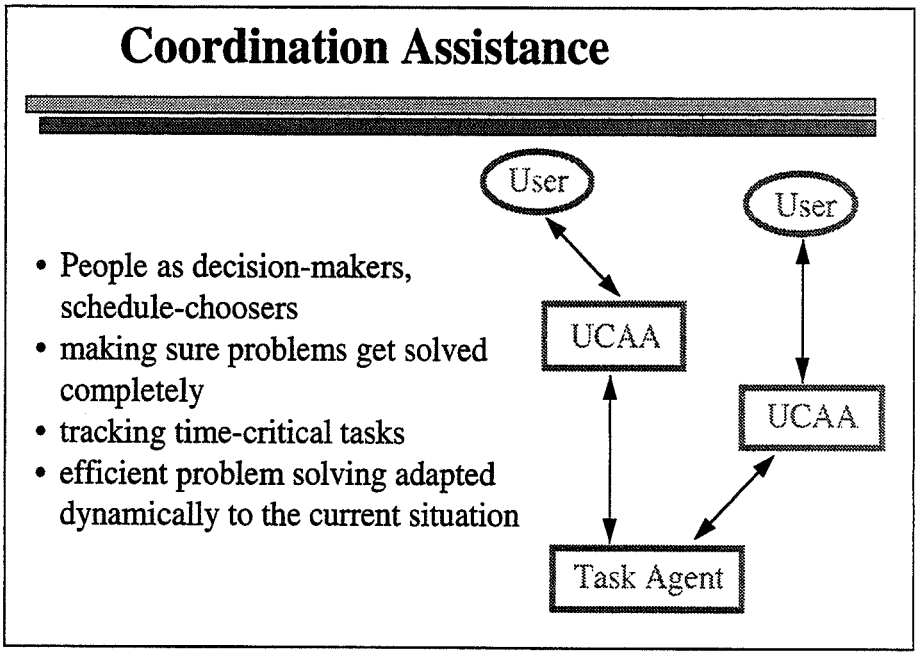
Example Applications and Coordination Problems

This slide briefly mentions several classic example domains and a brief example of one possible coordination problem.



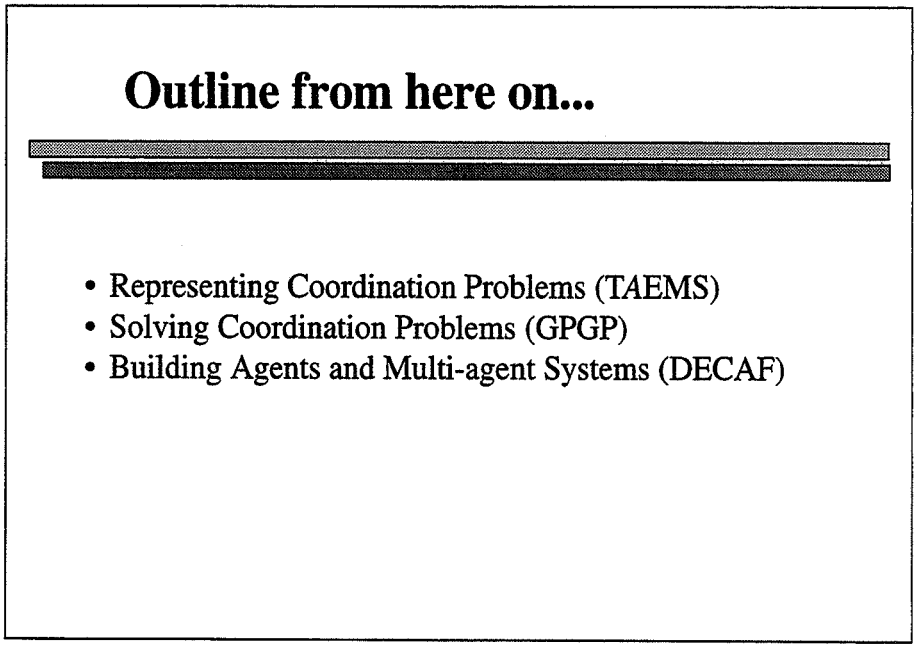
Coordination Assistance

Agents can also be used to assist people in solving coordination problems (as well as needing the coordination of their own, autonomous work). However, because you are dealing with real people, these agents are necessarily limited in what they can and cannot do.



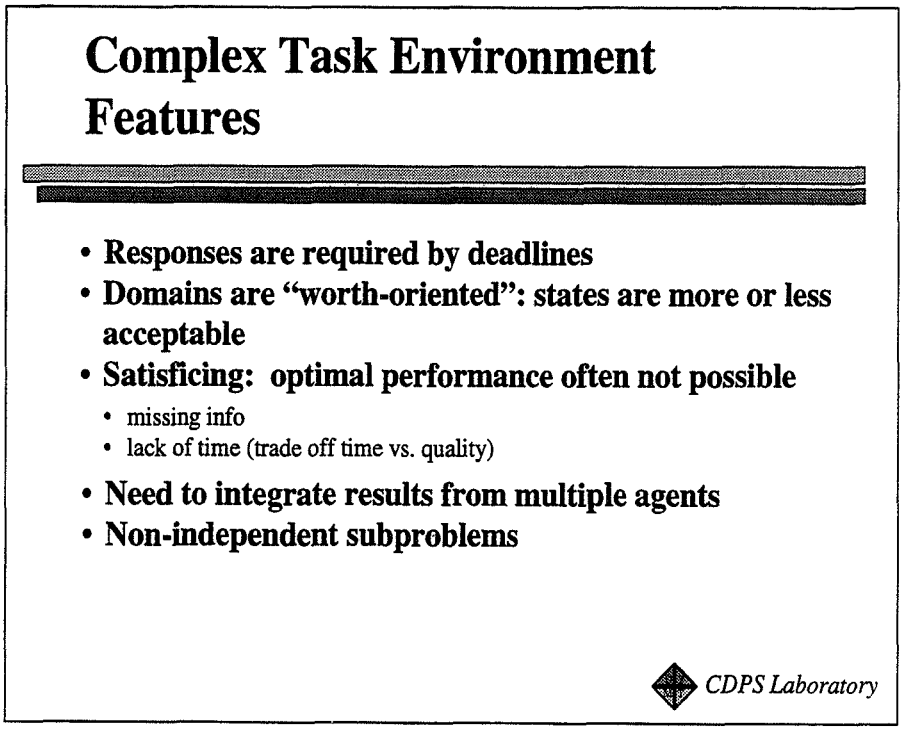
Outline From Here On...

The rest of the talk will discuss these three topics.



Complex Task Environment Features

First, we discuss REPRESENTATION. This slide lists the features of problem solving "task environments" that we wish to be able to represent. Notice that we do not eliminate most of the complexity of real problems, which is a problem with some other approaches.

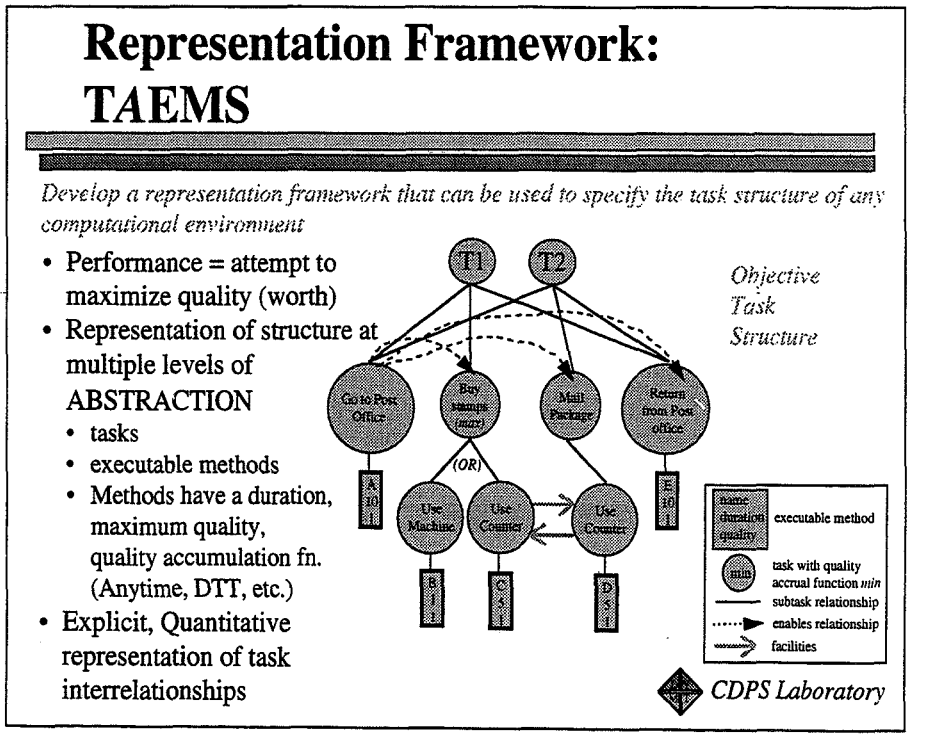


Representation Framework: TAEMS

The TAEMS (Task Analysis and Environment Modeling System) language is used to formally define what a task structure is, what parts are known by what different agents, and what happens when agents execute these parts. TAEMS is often used as an annotation language on top of HTN (Hierarchical Task Network) plans. Pictures such as the one here are based on careful, functional descriptions and an underlying state-based model of computation. Interior nodes in the task structure are abstract tasks, the leaf nodes are specific, instantiated agent actions (for a software agent, these would normally be instantiated executable code).

The basic idea is that each agent is trying to maximize performance, as described by some set of utility characteristics (summarized as "quality" for good characteristics, and "cost" for bad characteristics). Since the time that something gets done often affects these things a lot, we also track the "duration" of various activities. TAEMS task structure annotations describe how the actions of any agent affect the performance of that agent or others (by changing quality, cost or duration). The basic relationship here is the "subtask" relationship; but more important are various hard and soft relationships that might exist between tasks (i.e., "enables" where A must come before B, or "facilitates," where doing A will cause B to be done better, cheaper or quicker). All relationships have a formal, quantitative mathematical definition.

TAEMS agents can reason about these task structures, and even use them as a language for communicating about coordination problems: "Hey Cindy, my task Q3 enables your task P8. I'm letting you know that I will finish task Q3 at 10:45 p.m. today."

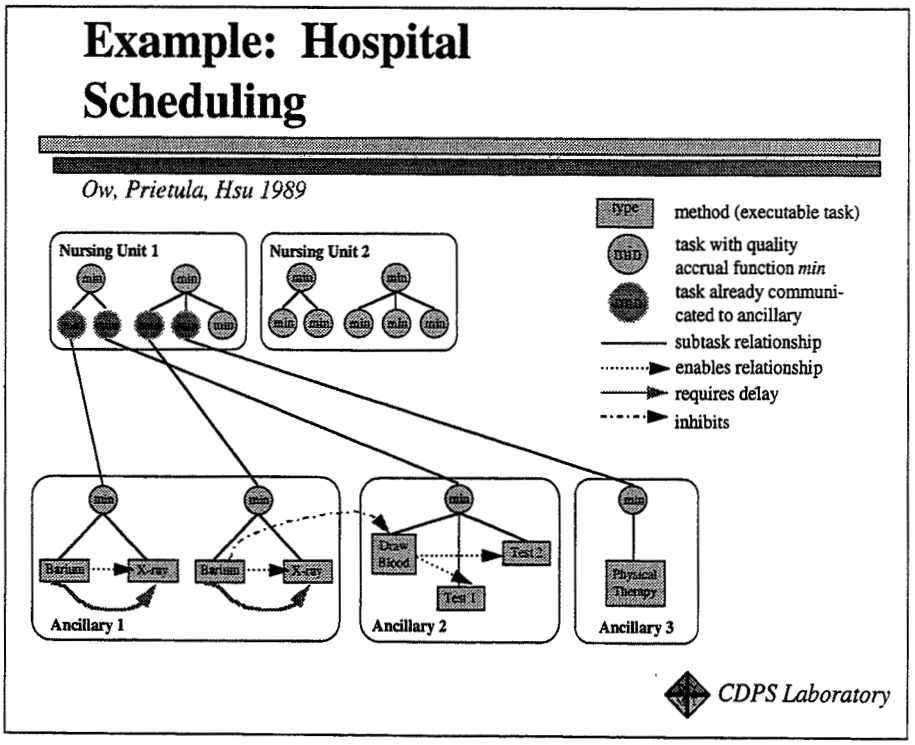


Example: Hospital Scheduling

An example task structure drawn from real case studies. Patients in General Hospital reside in units that are organized by branches of medicine, such as orthopedics or neurosurgery. Each day, physicians request certain tests and/or therapy to be performed as a part of the diagnosis and treatment of a patient. Tests are performed by separate, independent, and distally located ancillary departments} in the hospital. The radiology department, for example, provides x-ray services and may receive requests from a number of different units in the hospital.

Furthermore, each test may interact with other tests in relationships such as "enables," "requires-delay" (a slight variation on "enables"), and "inhibits" (a negative variation of the soft "facilitates" relationship). These task relationships indicate when the execution of one task changes the characteristics (here, primarily duration) of another task.

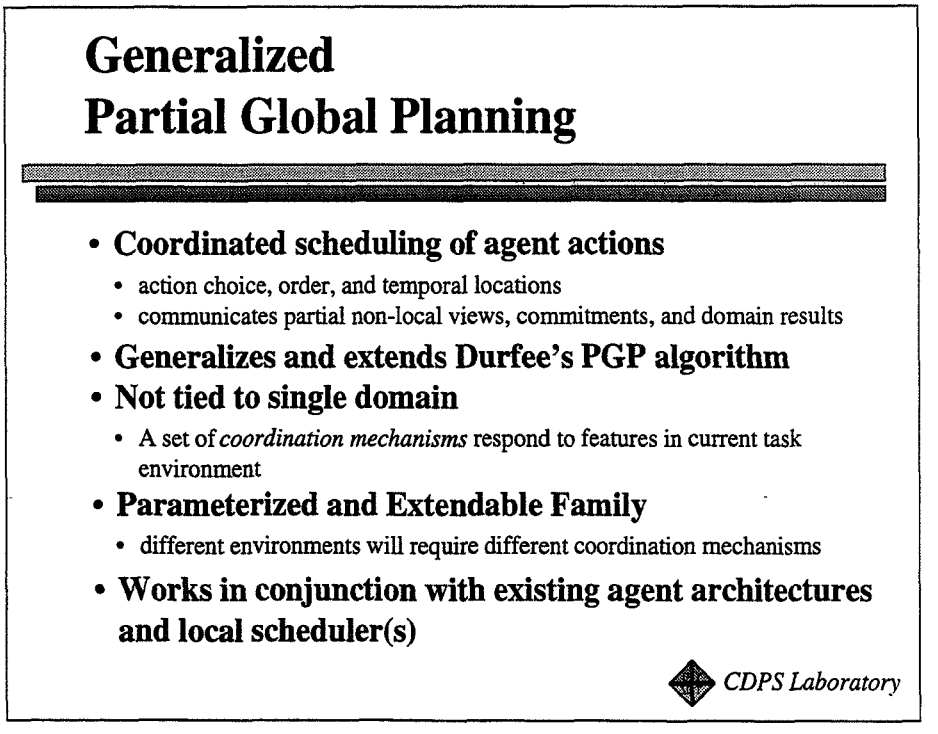
Since different agents may have different goals with respect to each other and with any global goals, the performance of such a system can be worse than that of a centralized system. In many domains such as hospital scheduling (or telescope observation scheduling), we *cannot* centralize scheduling because it would take away the authority of each unit over the day-to-day control of its own activities. A distributed approach matches with the existing human organizational structure. It also allows each unit to attempt to optimize slightly different measures, as may be used by administrators to evaluate human unit-level performance.



Generalized Partial Global Planning

Now that we have talked about REPRESENTATION, let us move onto DESIGNING COORDINATION MECHANISMS (using these representations).

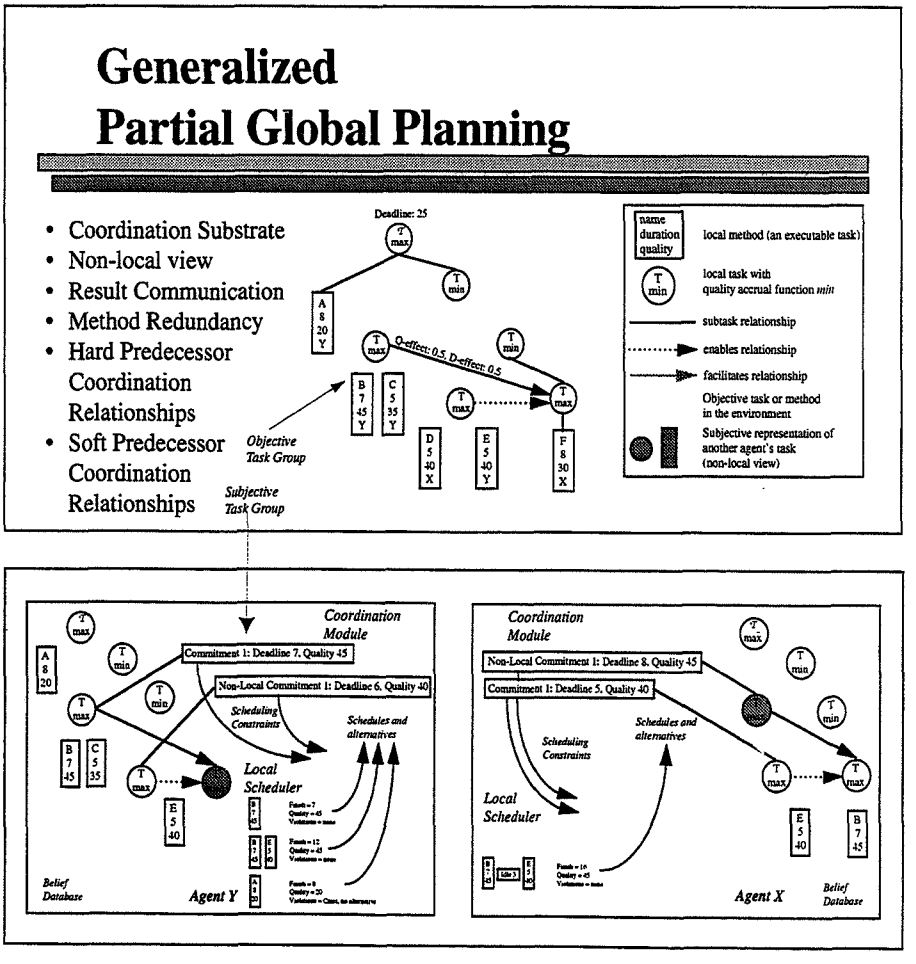
GPGP is a domain independent *scheduling* approach (The term "planning" in the name is historical, from Durfee's PGP. In the modern AI view of a continuum between planning and scheduling, both GPGP and PGP focus on the scheduling side.) The GPGP approach makes several architectural assumptions. Most important of these is that the agent represents its current set of intended tasks using the TAEMS task structure representation language. An agent using the GPGP approach provides a planner or plan retriever to create task structures that attempt to achieve agent goals, and a scheduler that attempts to maximize utility via the choice and temporal location of basic actions in the task structure. Each GPGP mechanism examines the changing task structure for certain situations, such as the appearance of a particular class of task relationship, and responds by making local and non-local *commitments* to tasks, possibly creating new communication actions to transmit commitments or partial task structure information to other agents. The set of coordination mechanisms is extendable, and any subset can be used in response to a particular task environment situation.



Generalized Partial Global Planning

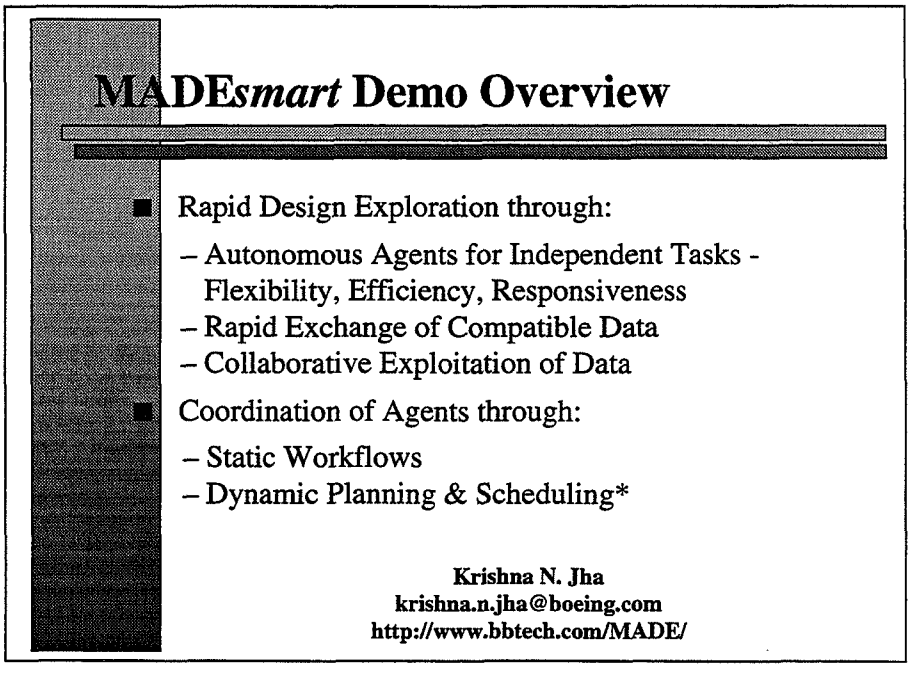
Initially, GPGP defined the following five coordination mechanisms based on Durfee's PGP:

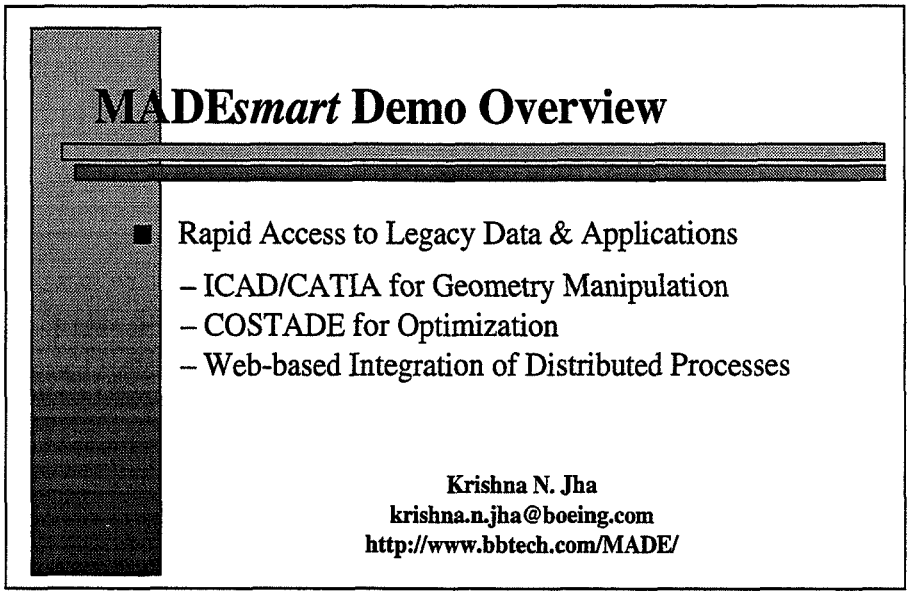
- Updating non-local viewpoints. Each agent detects the possible coordination relationships, then communicates the related task structures.
- Communicate results when they will be used by others.
- Handling simple redundancy. When more than one agent wants to execute a redundant method, one agent is randomly chosen to execute it and send the result to the other interested agents.
- Handling hard relationships (A must come before B) from the predecessor side. A is the "predecessor" task, B is the "successor."
- Handling soft relationships from the predecessor side (if A is executed before B, the execution of B will be perhaps faster or will return better results, but it is not strictly necessary).



MADEsmart Demo Overview

This description is for the slide below and the next slide. We are currently working with a group implementing the UCAA/ACM vision of coordination support as a part of the MADEsmart project at Boeing Helicopters. MADEsmart seeks to partially automate the integrated product teams used to organize design engineers through the use of multi-agent approaches. For example, associated with each human engineer in an integrated product team is a UCAA (User Coordination Assistant Agent) that can interact with that engineer. Other agents, using ACM technology, wrap around existing computationally intensive resources such as composite fiber placement simulations and the COSTADE design cost analysis tool, which uses an existing FORTRAN-based model.

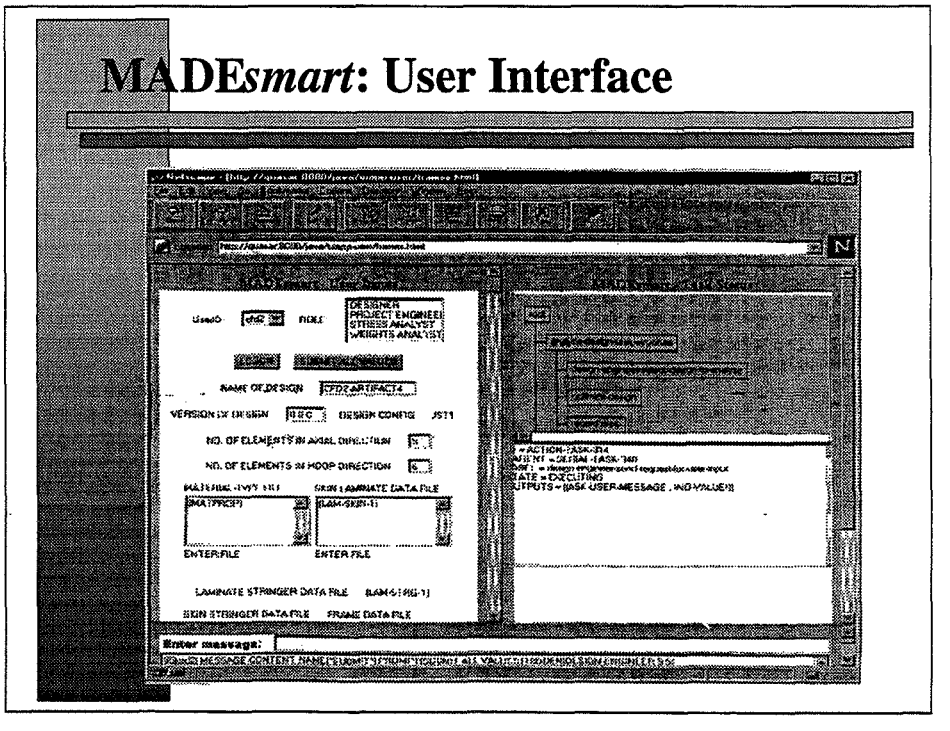



MADEsmart Demo Overview
Rapid Access to Legacy Data & Applications
– ICAD/CATIA for Geometry Manipulation
– COSTADE for Optimization
– Web-based Integration of Distributed Processes
Krishna N. Jha
krishna.n.jha@boeing.com
http://www.bbtech.com/MADE/

MADEsmart - User Interface

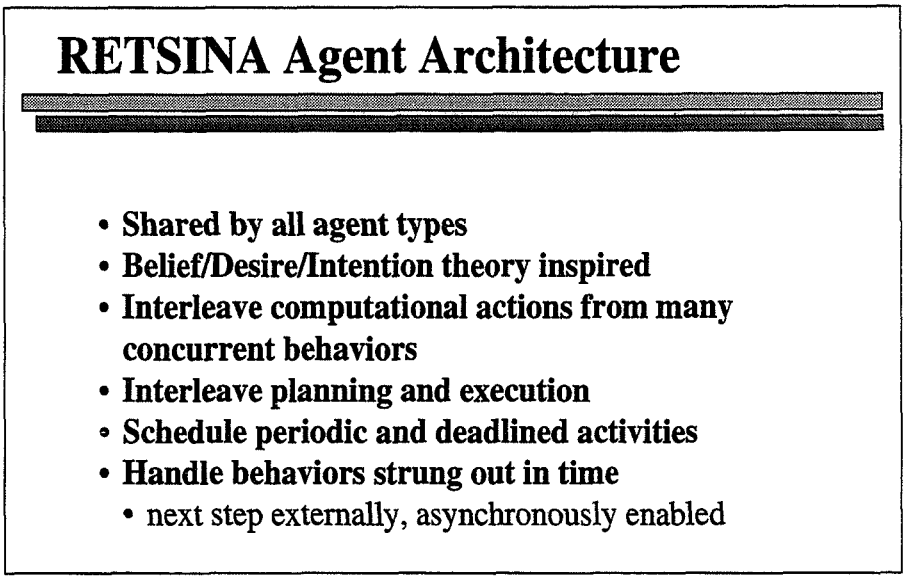
For this project, the core agent architecture components are being integrated using GBB, a commercial blackboard system developed by BBTech. If you look at the upper right of the screen dump, you can see the current task structure. A graphical task structure specification tool allows programmers to create and edit agent-executable task structures (behaviors), including the flows of information between executable methods/basic actions.

In the initial implementation, the UCAA has little scheduling to do, mostly due to the fact that only one project is being worked on, and the initial task structures have been purposely kept quite spartan. We plan to eventually apply our scheduling technologies to intelligent user interfaces (via the Local Schedule Display in the UCAA). The UCAA will help a user to schedule his or her activities at the workstation and display that schedule (using the Local Schedule Display) in a meaningful and expressive form that can be queried and explained. In most cases, the user will have significant freedom in the ordering of his/her activities—the purpose of the Local Schedule Display is to make sure that tasks are not forgotten, that time critical or critical enabling tasks are identified to the user, and that facilitating or other soft-related tasks are also identified.



RETSINA Agent Architecture

Finally, let us turn from REPRESENTATION and COORDINATION MECHANISMS to tools for building software agents. This slide describes some of the features of the RETSINA software agent approach developed with Katia Sycara while I was at CMU. They are continuing to develop toolkit pieces, and we are also doing this at the University of Delaware (under the name DECAF: Distributed, Environment-Centered Agent Framework).



RETSINA Agent Architecture

This slide describes the general structure of a RETSINA agent's internal control loop. In DECAF, the inner threads are executed concurrently.

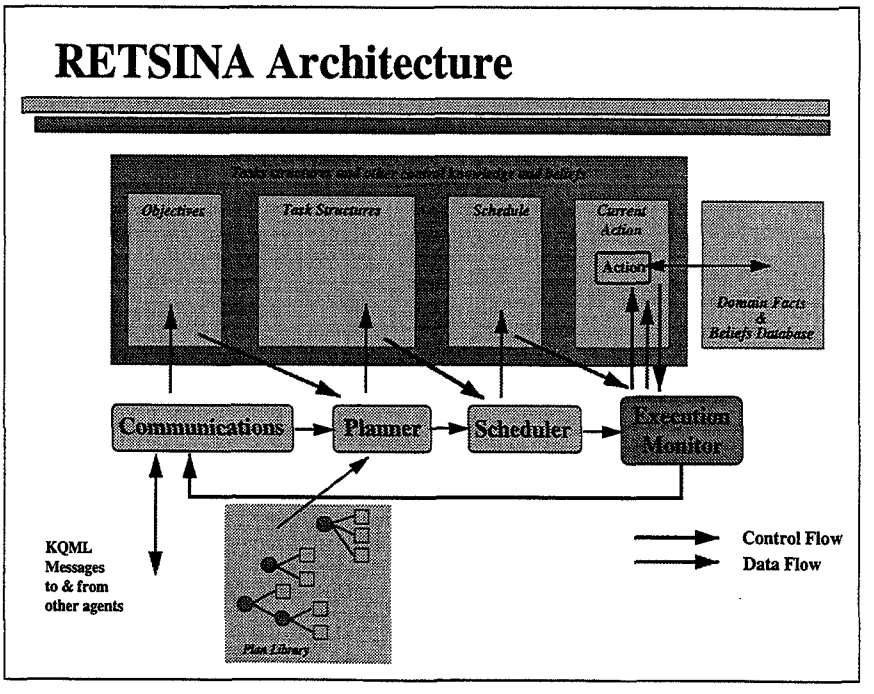
RETSINA Agent Architecture

Meta-Control Loop

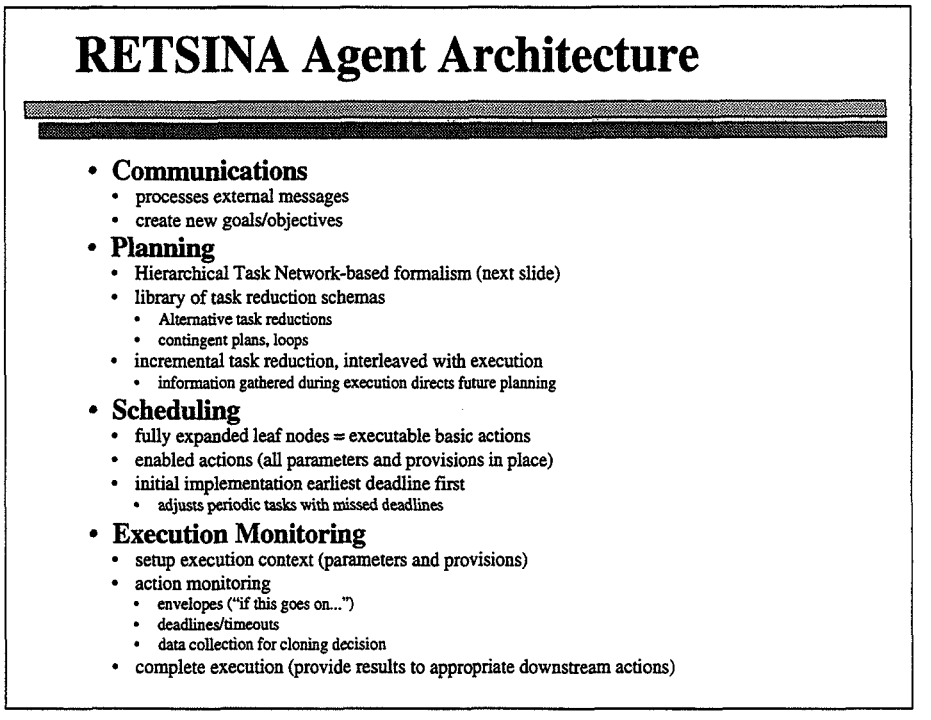
- **Startup**
  - initial goals [e.g. advertise-self]
- **Loop**
  - Communication (& coordination)
  - Planning
  - Scheduling
  - Action Execution & Monitoring
- **Shutdown**
  - cleanup; notify any related agents

RETSINA Architecture

This is a pictorial view of the previous slide. Again, in DECAF the communicating/planning/scheduling/executions are concurrent. The data flow in both systems is that new KQML messages (i.e., ASK) create new *objectives*. The planner creates task structures to achieve the objectives. There are usually many simultaneous plans and possible actions vying for agent resources—the scheduler creates an appropriate agenda of tasks. Finally, the execution monitor actually carries out the agenda. IN DECAF, THESE ARE DONE CONCURRENTLY AND CONSTANTLY. The agent is thus CONSTANTLY re-planning and re-scheduling as the world changes dynamically about it, and in response to uncertain action outcomes that force it to interleave planning and execution.

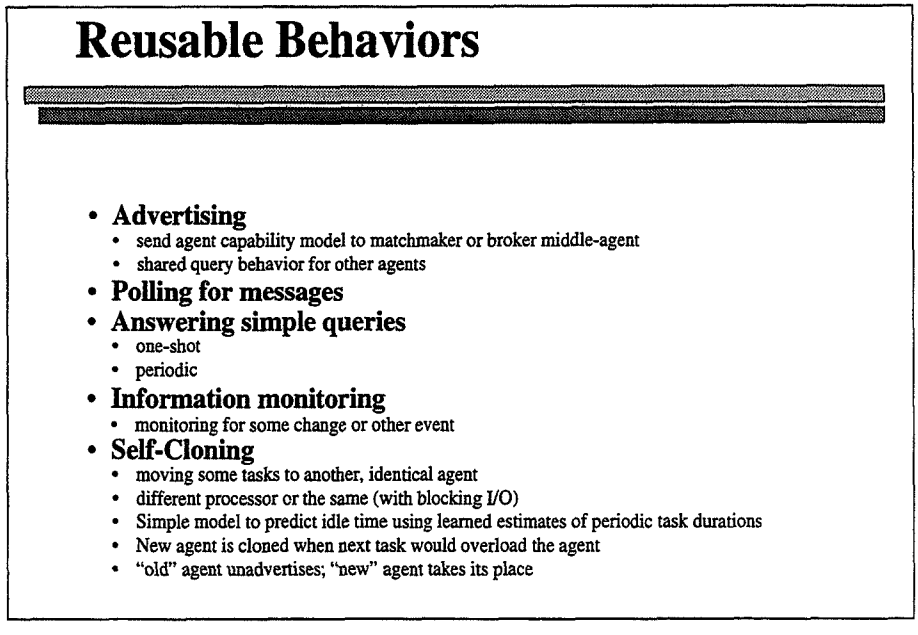


This shows a more detailed breakdown of activities in the four main control threads.



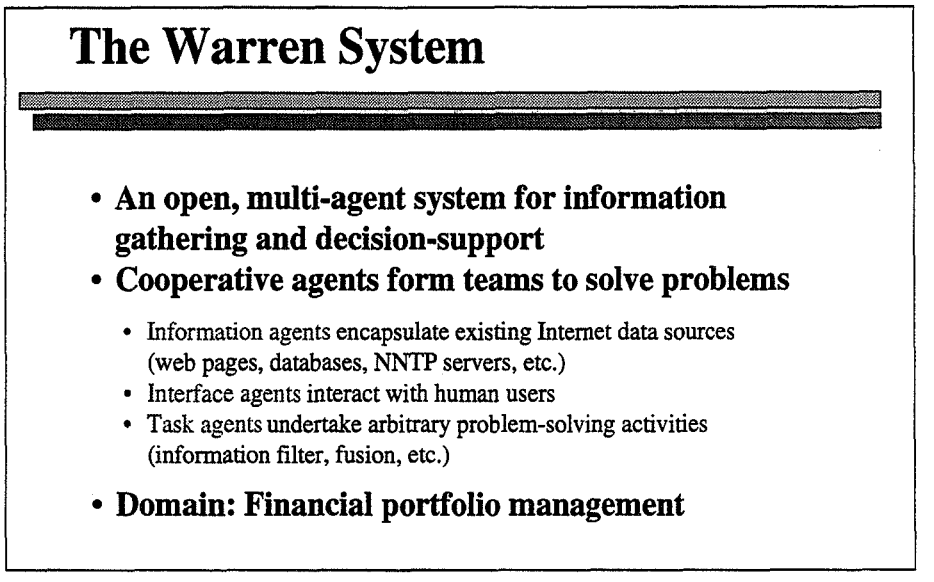
Reusable Behaviors

A very important feature of these architectures is the ability to reuse certain plans (capabilities) over and over in many different agents targeted for many different application domains.



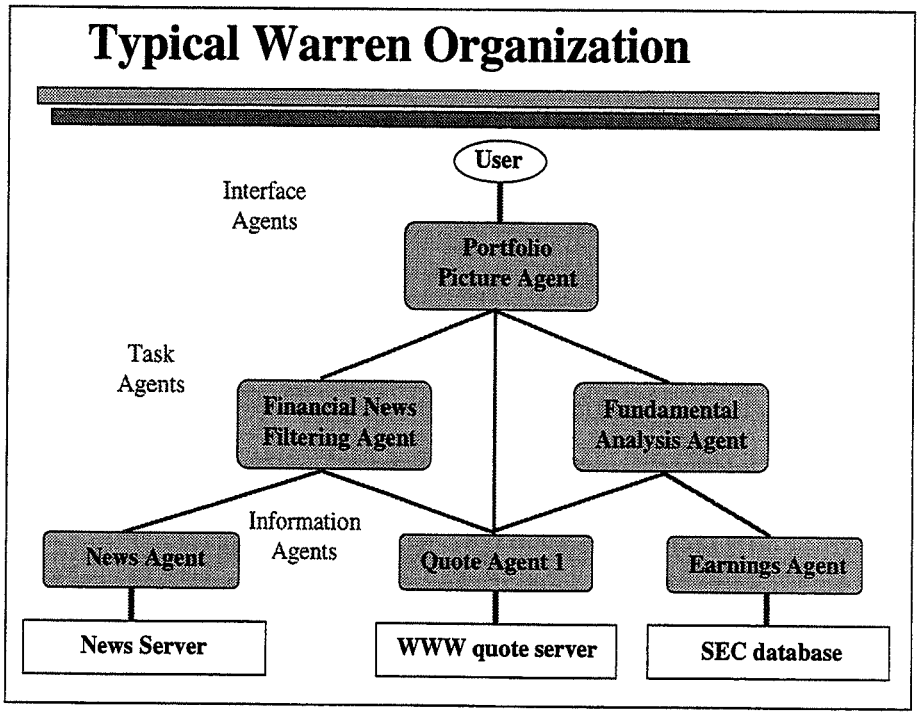
The Warren System

Warren was one RETSINA application for Stock Portfolio Management.



Typical Warren Organization

This is a picture of the different agents that collectively went into the Warren system.



Conclusions

- **Coordination: locally choosing and temporally ordering actions**
- **TAEMS: representing coordination problems**
- **GPGP: mechanisms for dealing with coordination problems**
- **DECAF: agent building toolkit**

- For more information, see http://www.cis.udel.edu/~decker

51-61
032525
370175
24p

Multiagent-Oriented Programming

Michael N. Huhns
Center for Information Technology
University of South Carolina
Columbia, SC

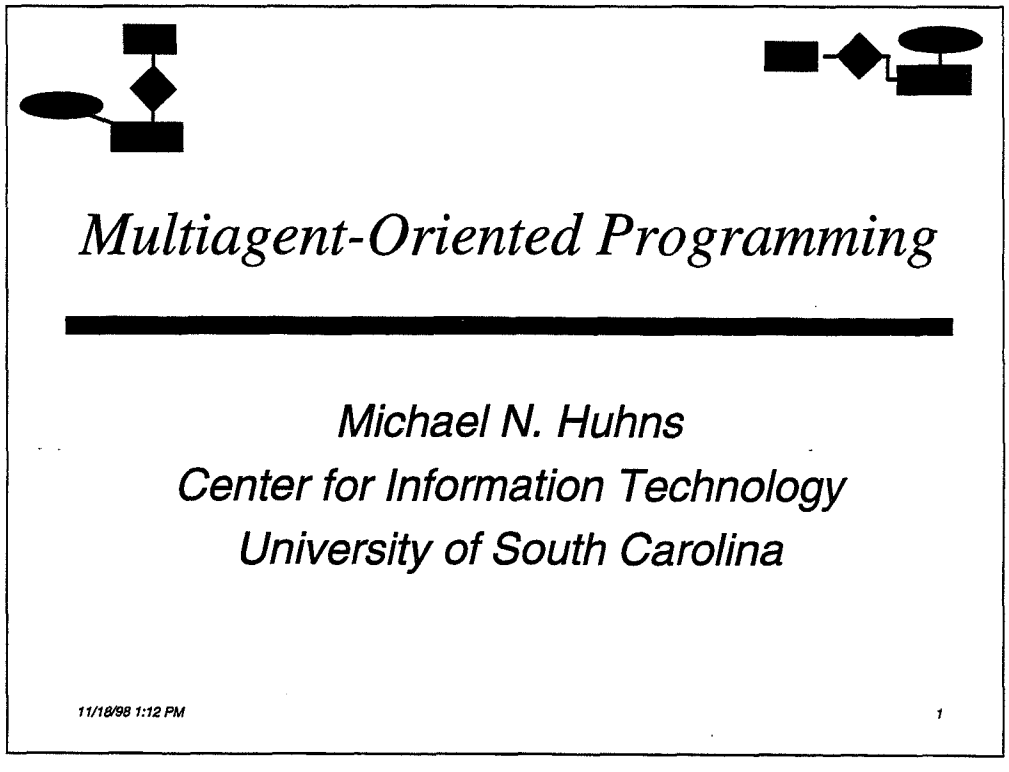
Page intentionally left blank

Multiagent-Oriented Programming

Michael N. Huhns
Center for Information Technology
University of South Carolina
Columbia, SC

http://www.engr.sc.edu/research/CIT/people/faculty/huhns.html

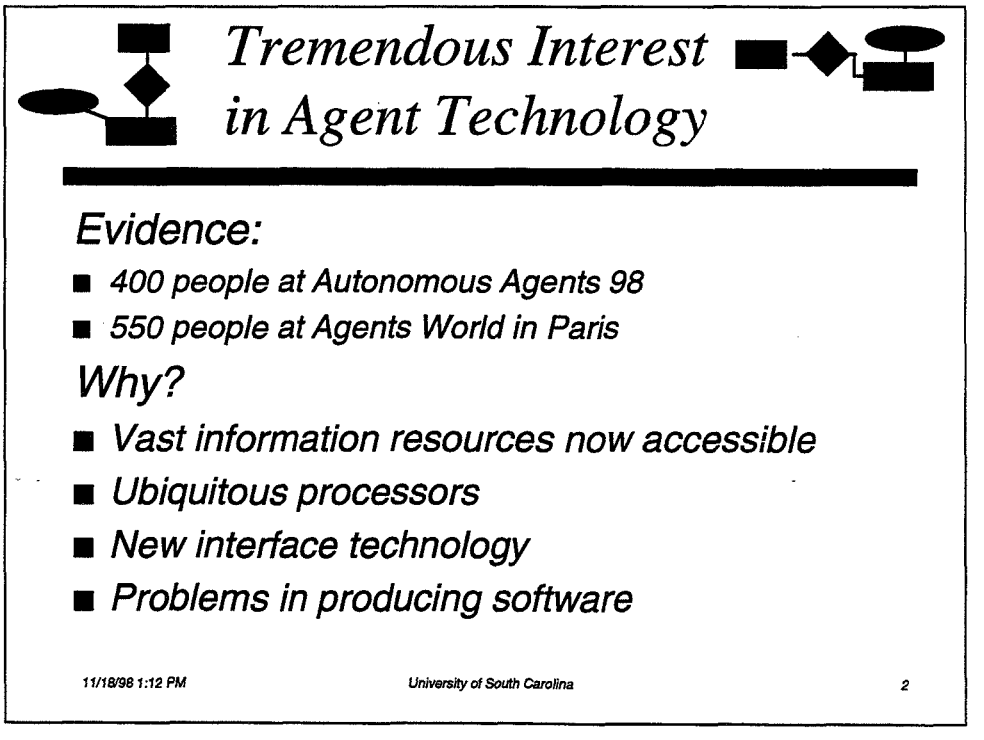
This presentation describes a new approach to the production of robust software. The approach is motivated by explaining why the two major goals of software engineering—*correct* software and *reusable* software—are not being addressed by the current state of software practice. A methodology based on active, cooperative, and persistent software components, i.e., *agents*, and how the methodology enables robust and reusable software to be produced is described. Requirements are derived for the structure and behavior of the agents, and a methodology is described that satisfies the requirements. The presentation concludes with examples of the use of the methodology and ruminations about a new computational paradigm.



Tremendous Interest in Agent Technology

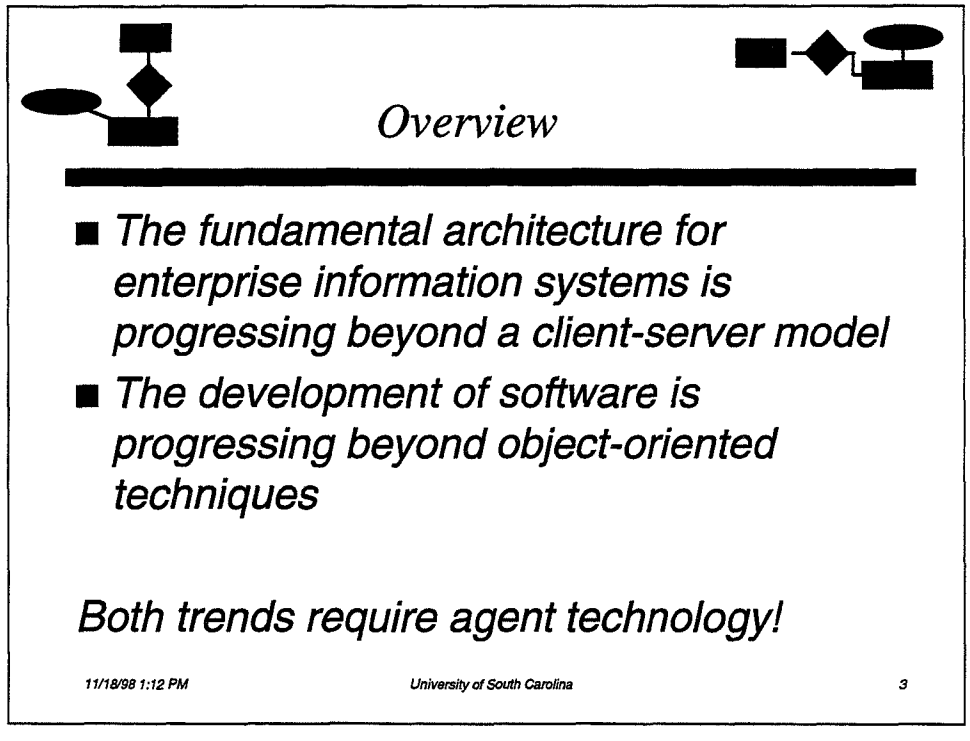
There is great interest in software agent technology currently. As evidence, there were more than a dozen conferences and workshops devoted to agents held around the world during the summer. There are four major reasons for this interest:

1) The Internet has made vast numbers of heterogeneous resources available which software agents are needed to access and manage.
2) Processors are being used to control devices throughout our environment, such as automobiles, appliances, and consumer devices. These devices are much more useful if they can communicate intelligently with users and each other.
3) New speech understanding technology is making it feasible for people to communicate with devices in natural language, and this is more effective if the devices appear to be intelligent agents.
4) Software development continues to be problematic, and multiagent technology can provide a new paradigm.



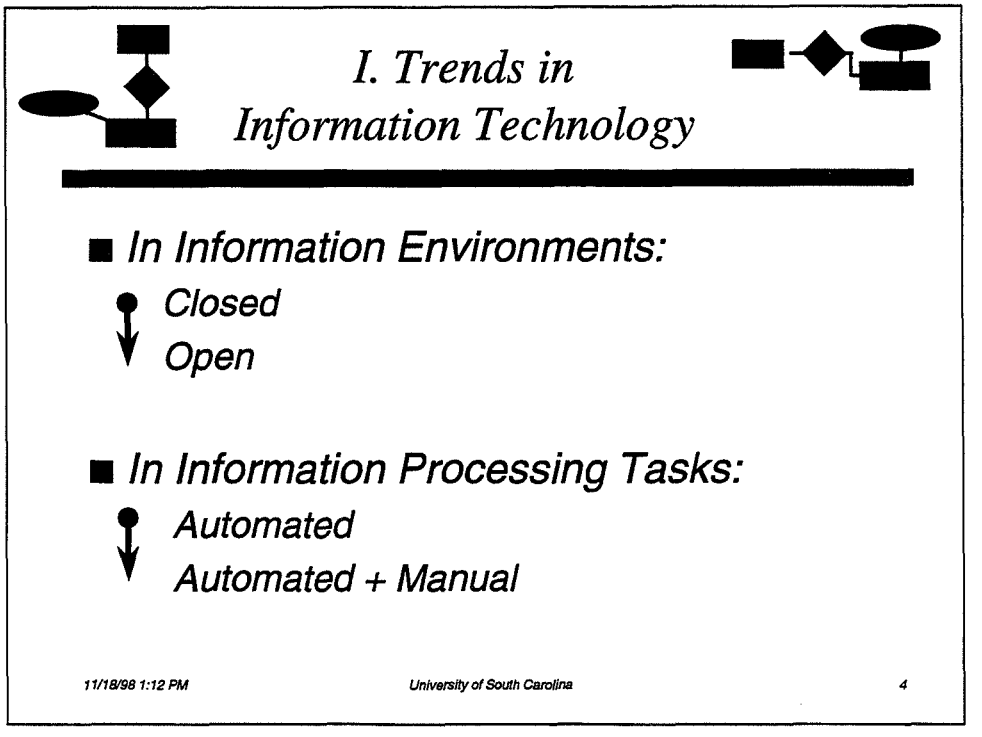
Overview

There are two primary reasons for the rise in popularity of agent technology. Each of these is fundamental to computer science, so that agent technology is likely to be important and viable for the foreseeable future.



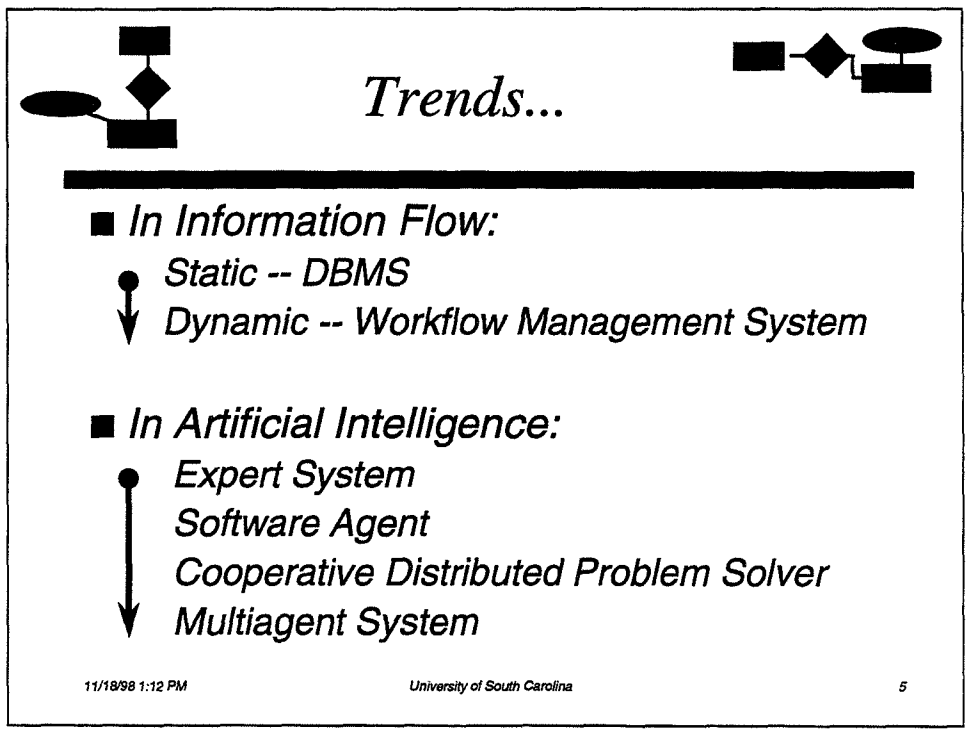
Trends in Information Technology

Information environments have moved beyond the closed corporate environments of the past, and are now open: the resources that are available via networks are dynamic and cannot be predicted or controlled. Also, information processing tasks have moved from batch jobs to applications that combine contributions from humans and computers.



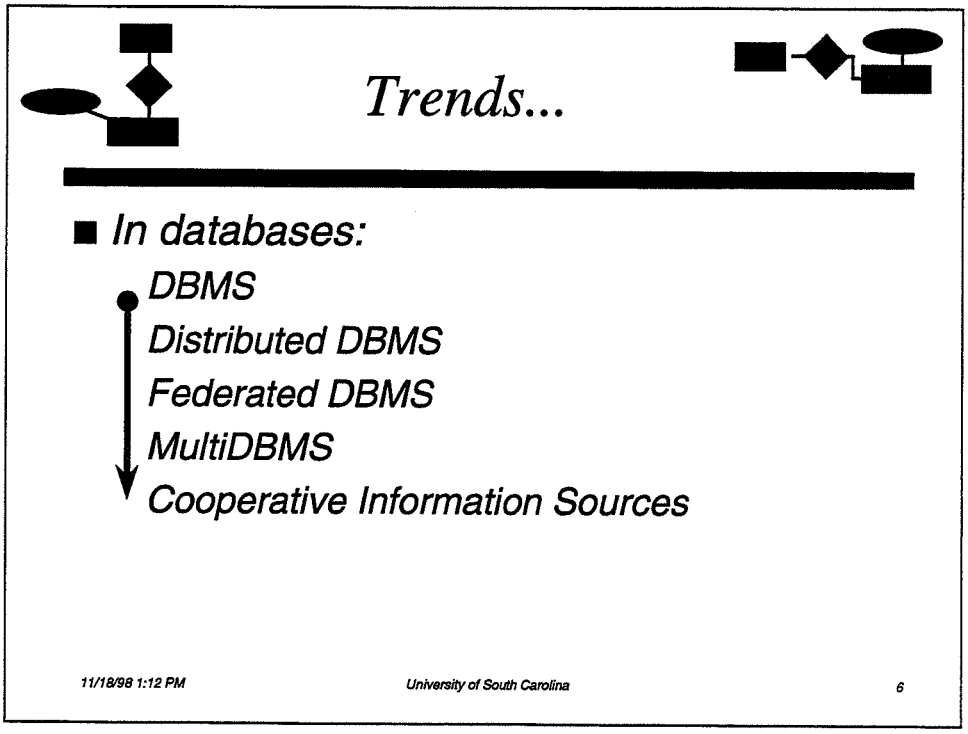
Trends...

Information is no longer treated as just the static data that is found in databases. The dynamic flow of information to and from databases must be considered as well. Artificial intelligence has progressed from expert systems to individual agents to cooperative problem-solving agents to multiagent systems.



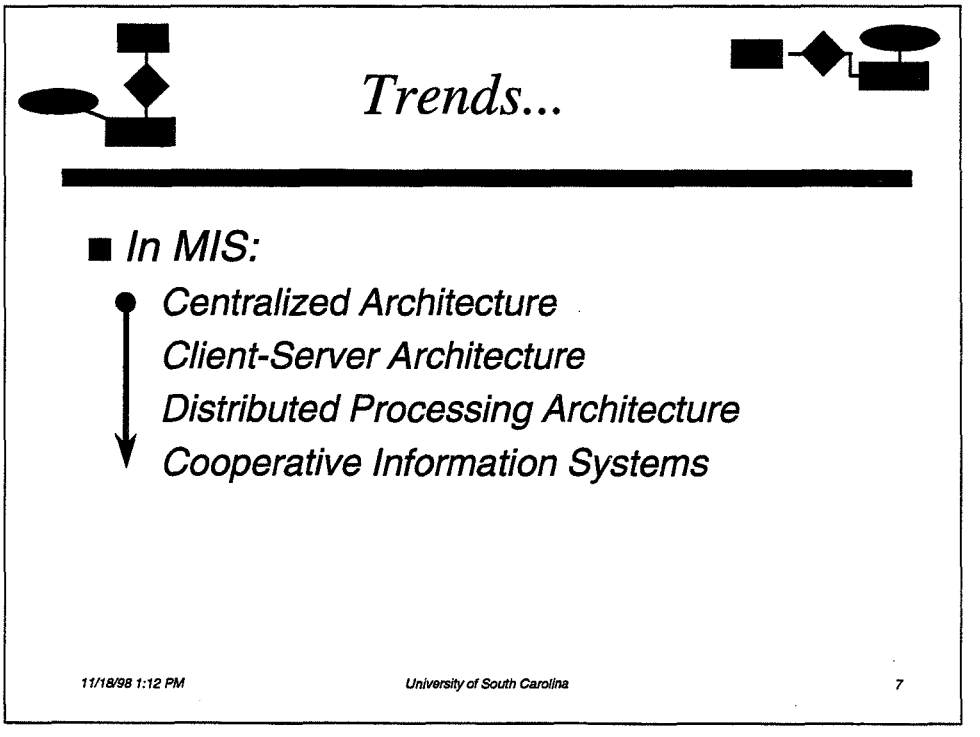
Trends...

Database technology has progressed from individual database management systems to tightly coupled, homogeneous, distributed DBMSs, to federated DBMSs with a single global schema to cooperative DBMSs that are active, autonomous, and heterogeneous.



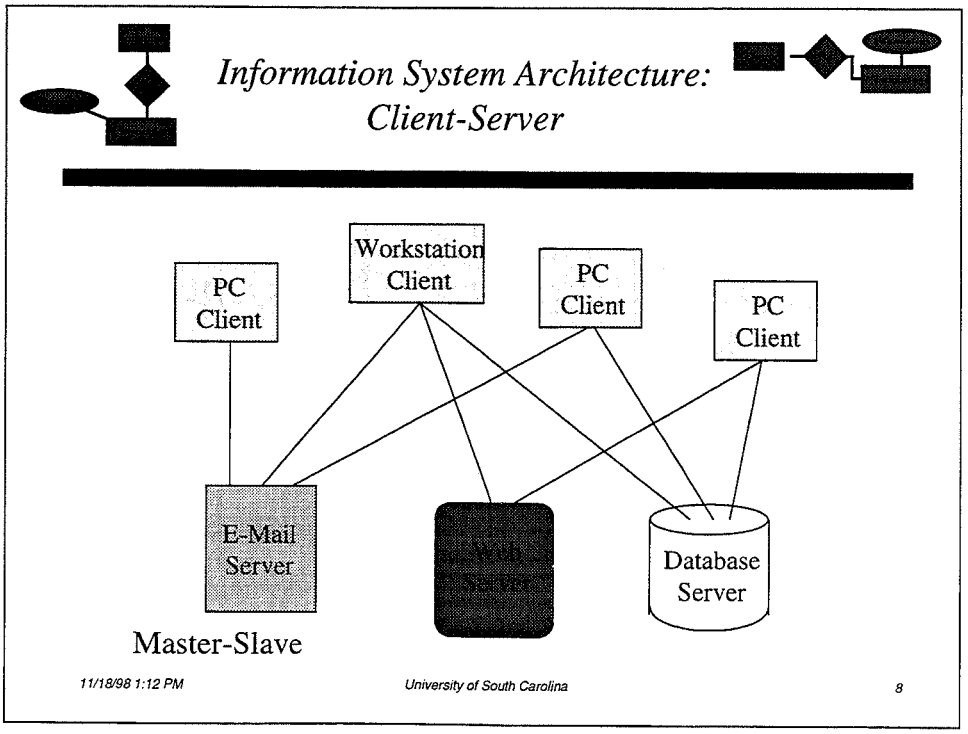
Trends...

Most corporate information systems are being converted from centralized architectures to client-server architectures. However, the trend is to move to distributed information system architectures featuring peer-to-peer interactions and, eventually, to cooperative information system architectures where the peers cooperate in processing information tasks.



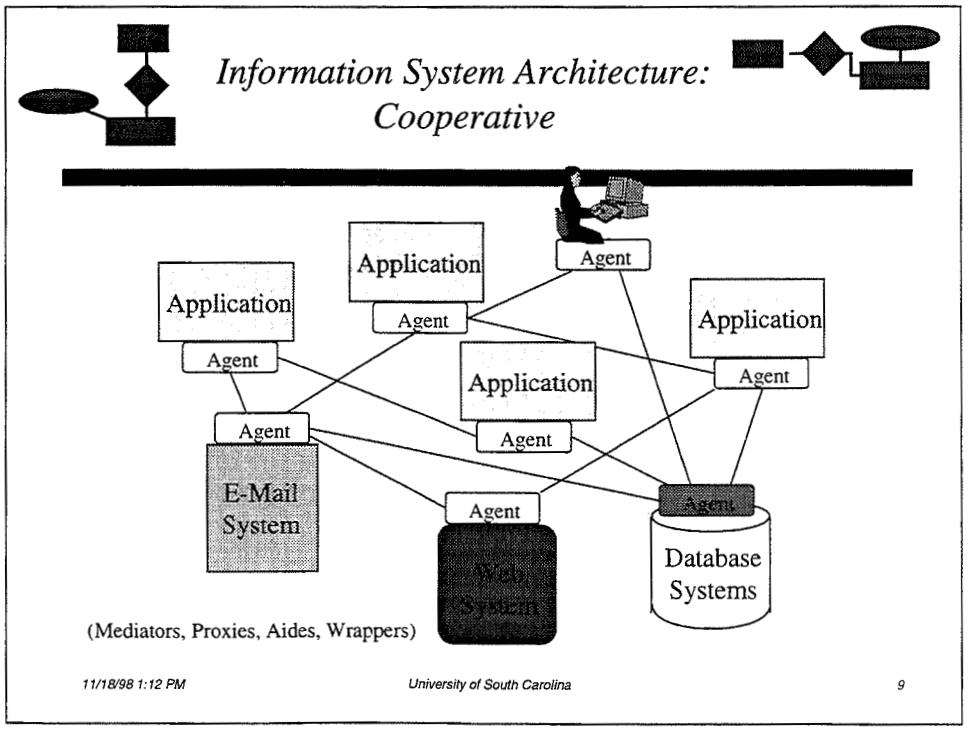
Information System Architecture: Client-Server

A client-server architecture is hierarchical, with no formal interactions among servers or among clients.



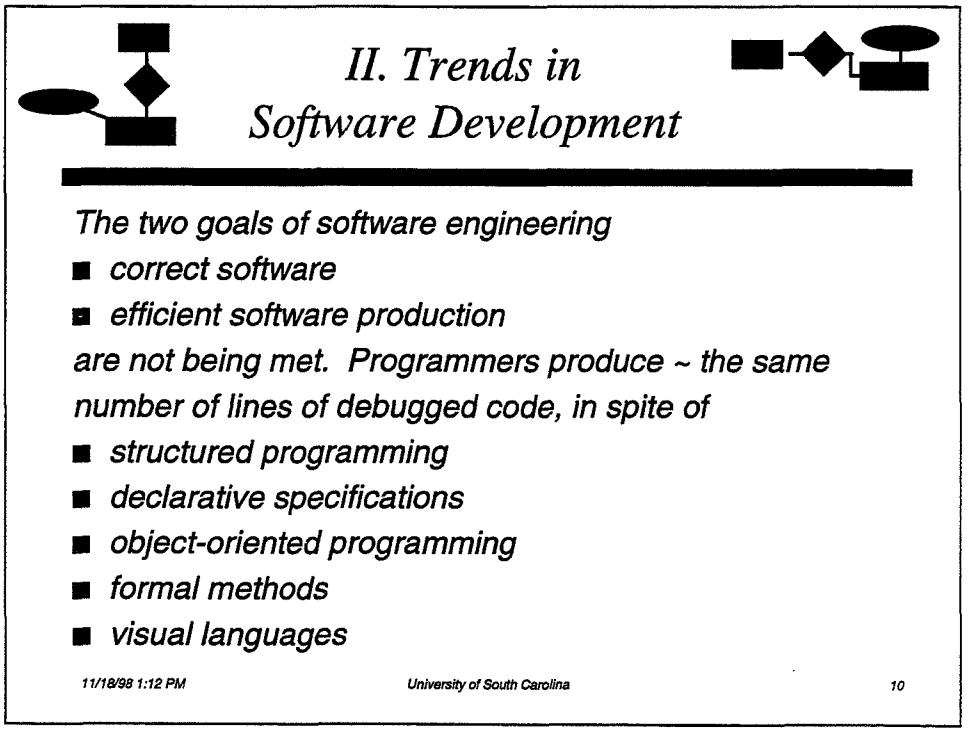
Information System Architecture: Cooperative

A cooperative architecture allows interactions among servers and among clients. The interactions can be cooperative in that the components can assist each other in solving tasks when the tasks are consistent with their own best interests.



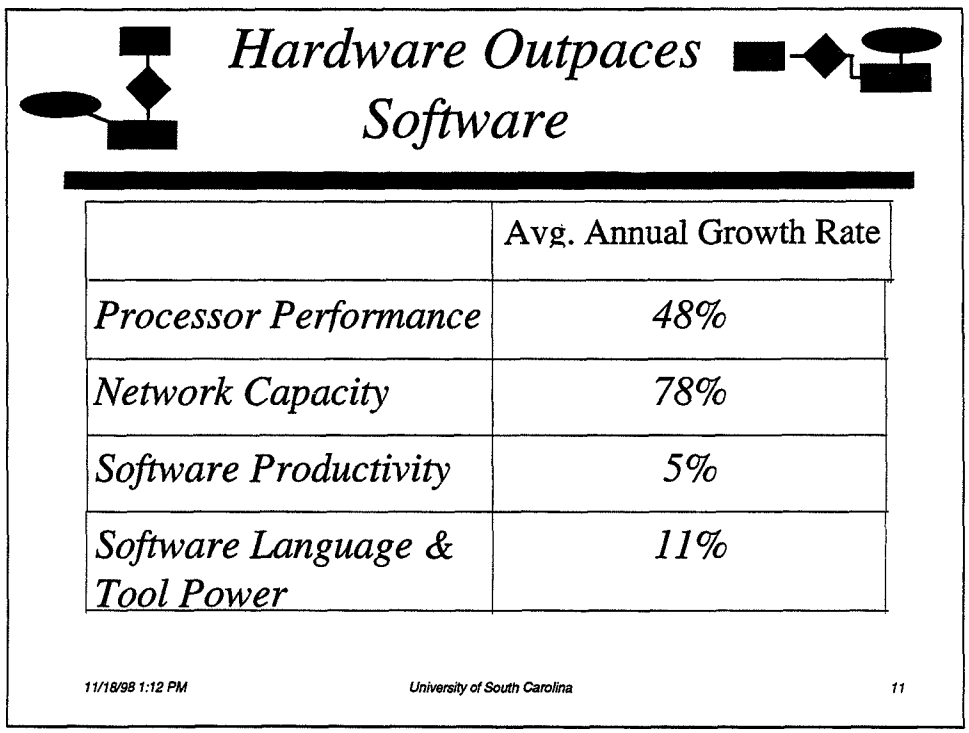
II. Trends in Software Development

The two major goals of software engineering, correct software and efficient production of software, are not being met. Programmers currently produce approximately the same number of lines of debugged code as they ever did, in spite of many developments that were supposed to be "magic bullets," such as structured programming, declarative specifications, object-oriented programming, formal methods, and visual languages.



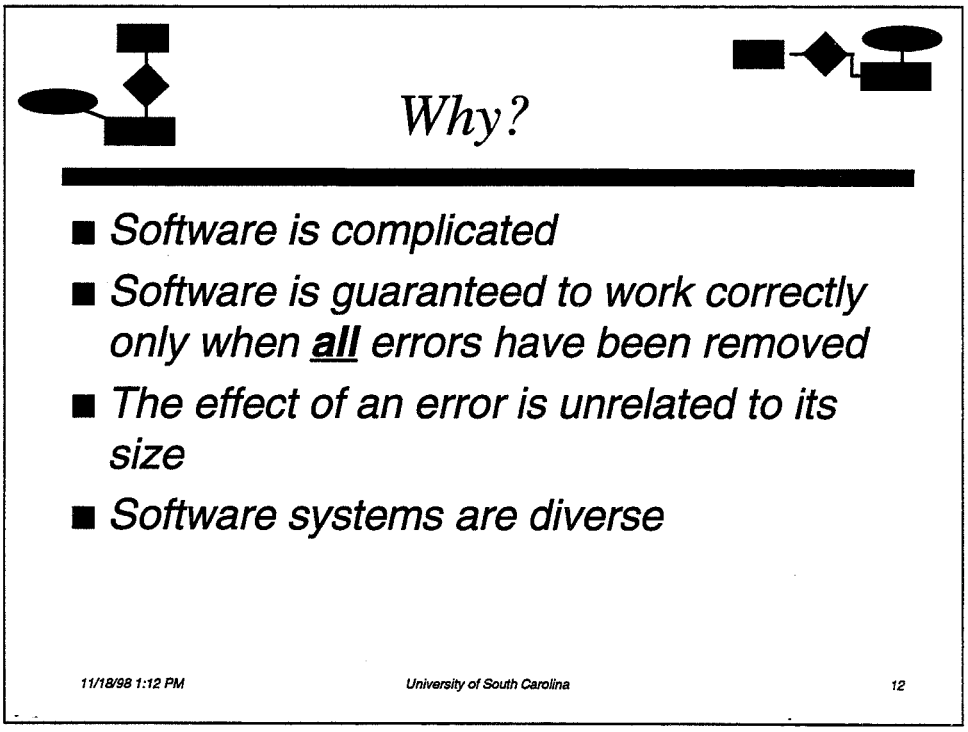
Hardware Outpaces Software

Over the last dozen years, processor performance and memory chip capacity have doubled every two years, in accordance with Moore's "Law." Network capacity has grown even faster, but software productivity has been almost static.



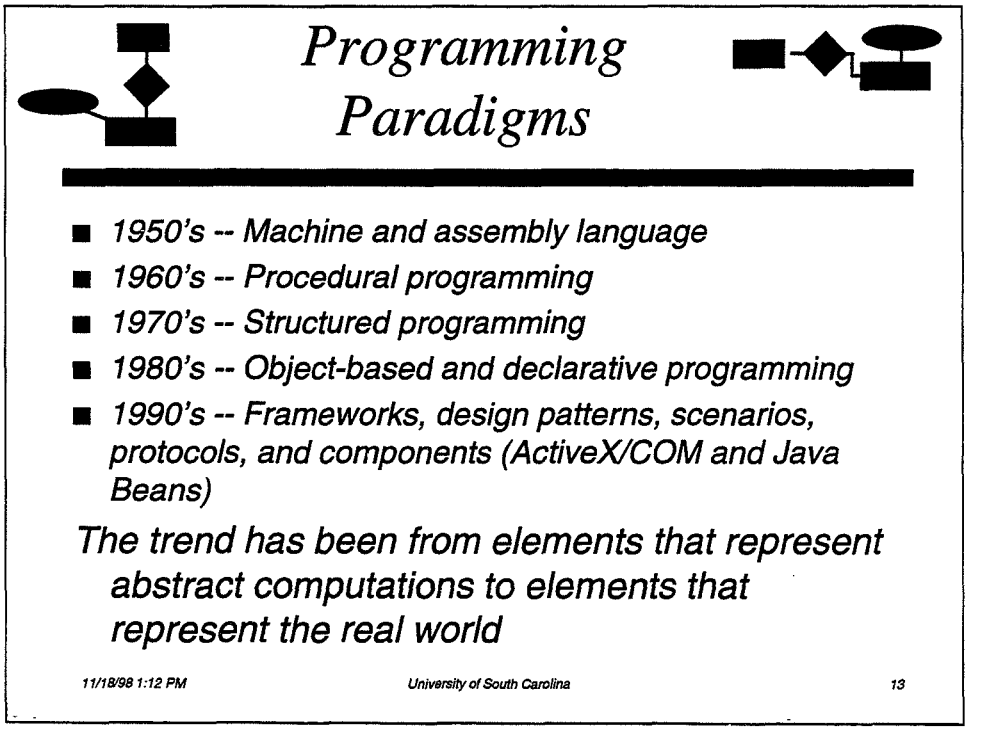
Why?

There are several reasons for the difficulty in improving the process by which software is produced. First, software is complicated, and is typically considered the most complex activity undertaken by humans. Second, software must be *perfect*, and is guaranteed to work correctly only when *all* errors have been removed. Third, the effect of an error is relatively independent of its size, in that the simple omission of a comma can render a million lines of code inoperable. Fourth, software systems are typically diverse and too often crafted afresh for each application.



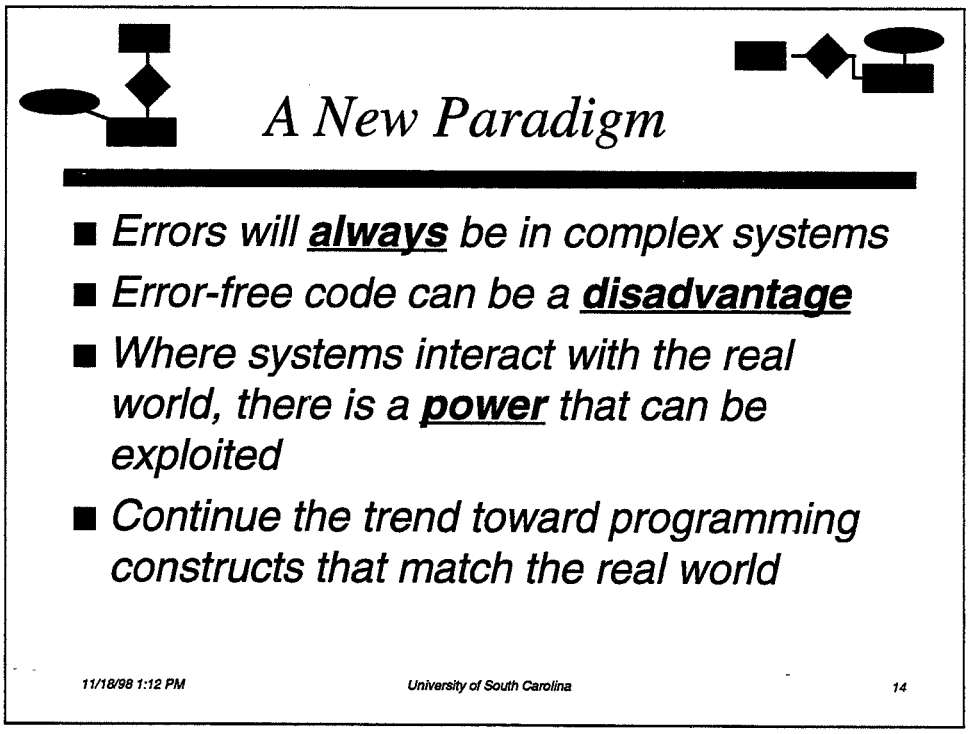
Programming Paradigms

There have been a number of different paradigms for the production of software since the 1950's. These paradigms have moved the basic unit of abstraction from components that model and implement computations to components that model and implement real-world objects. For example, the concept of an "employee" in a relational database is less like a real-world employee than is the class "employee" in an object-oriented database, because the object model includes the *behavior* of objects in the class.



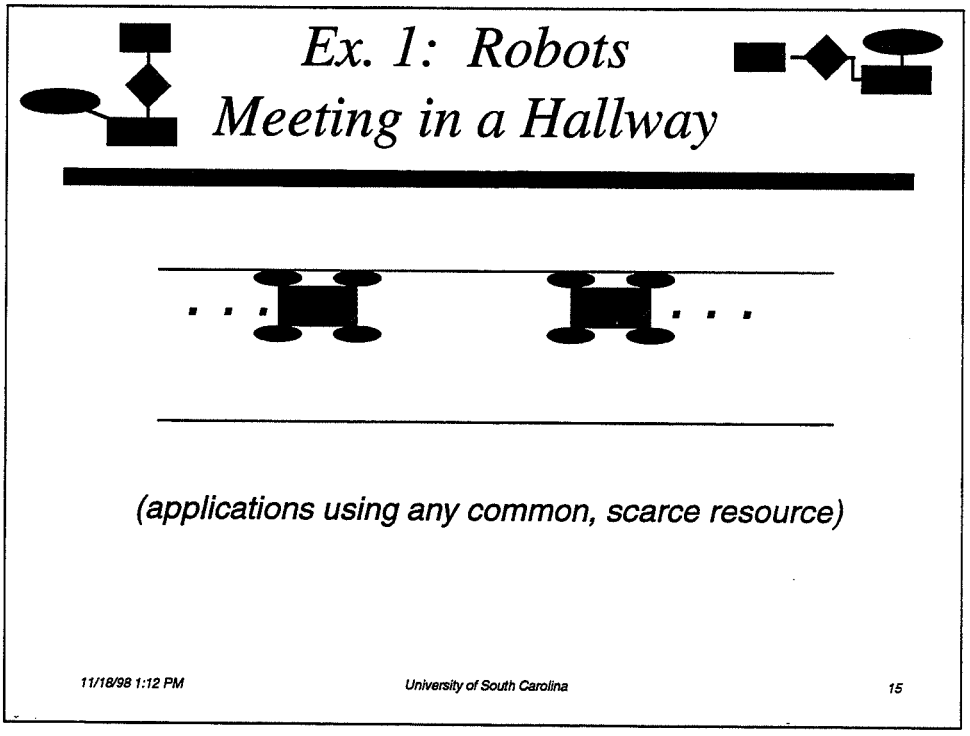
A New Paradigm

It is time to consider a new paradigm for software development, a paradigm that is based on the following premises. First, it is important to recognize that errors will *always* be in complex systems, and that it is necessary to accommodate them. Second, there are circumstances where perfect, error-free code can be a *disadvantage*. Third, systems that interact with the real world can take advantage of the uncertainties inherent in the world to lead to more robust and simpler software. Fourth, the new paradigm should continue the trend toward programming constructs that are more faithful to the real-world components they are meant to model.



Example 1: Robots Meeting in a Hallway

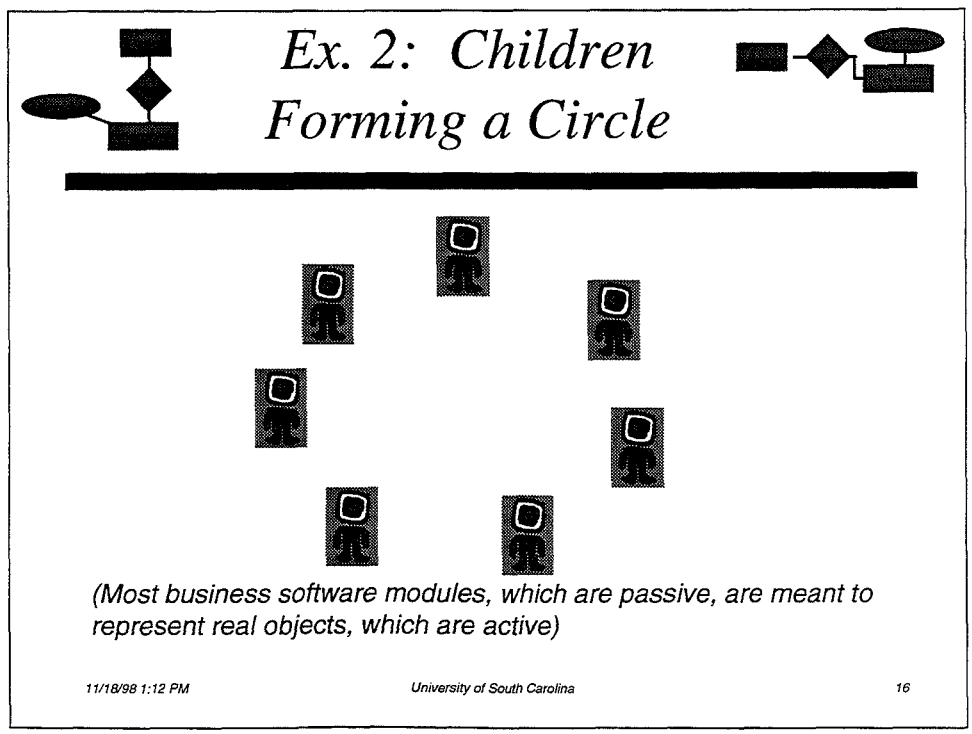
An example of how error-free code can be a disadvantage occurs when two identically and perfectly programmed robots meet in a hallway. They each will move side-to-side in synchrony and will never be able to pass. Now what if one of the robots has an error in its programming? It will then not behave the same as the other, the robots will break synchrony, and they will each be able to pass each other and continue their progress. The overall system of robots is more robust because of the presence of an error. The example is representative of any situation where there is contention for a scarce resource, such as access to a database.



Example 2: Children Forming a Circle

When a teacher tells a group of children to form a circle, they do this very robustly. They can form a circle whether there are 5 or 50 children, whether the children are large or small, and whether or not all of the children are old enough to understand the concept of a circle. Children can be added to or removed from an existing circle, and it will re-form correctly. The children implement a circle-forming algorithm that is distributed and requires no central control.

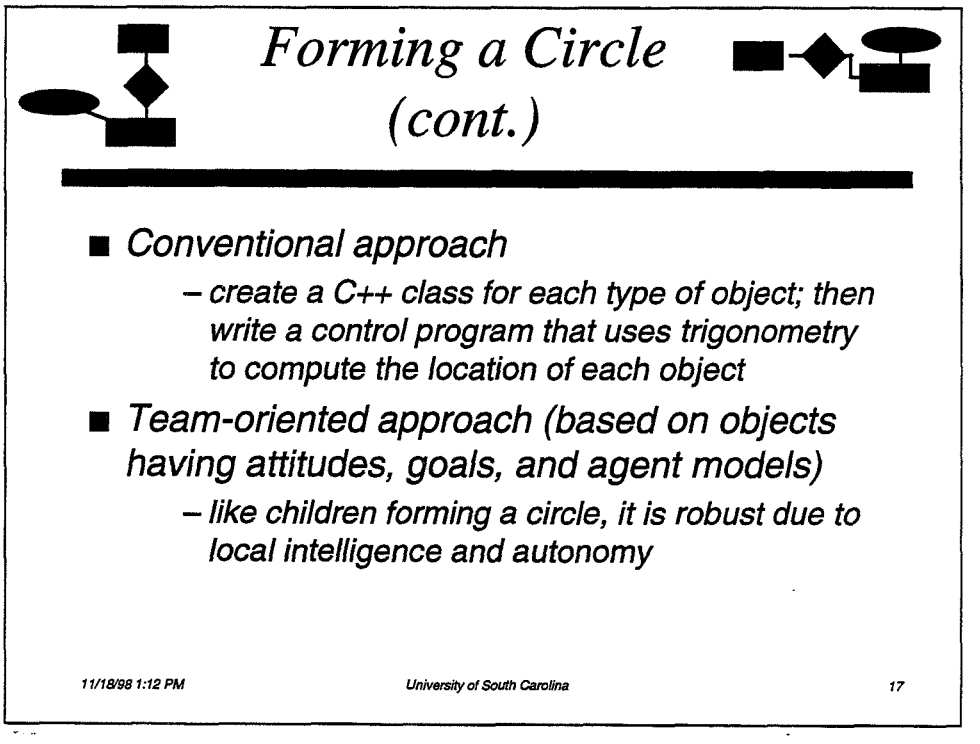
The robustness is due to the knowledge and ability of each child regarding what a circle is and how each child can contribute to its formation.



Forming a Circle

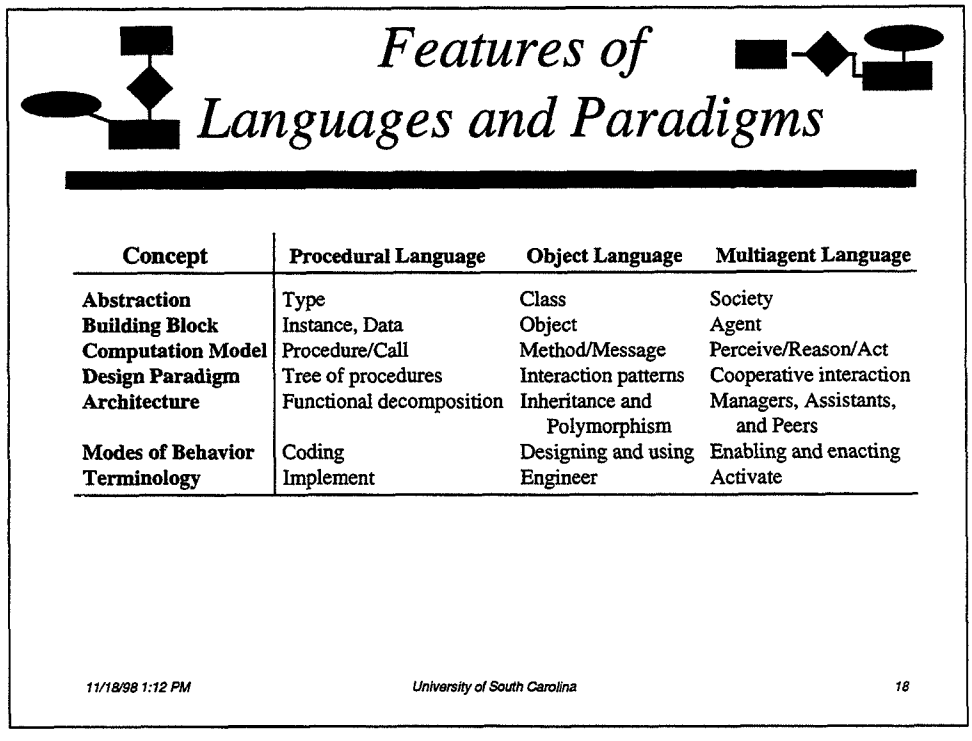
A conventional object-oriented approach to programming a circle algorithm would involve creating a class for each type of object that might be part of the circle, and then writing a control program that would use trigonometry to compute the location of each object. The addition or removal of objects would require recomputing all locations.

A multiagent or team-oriented approach would represent each child by an agent, and would give each agent the knowledge of what a circle is and the ability to position itself to be part of a circle.



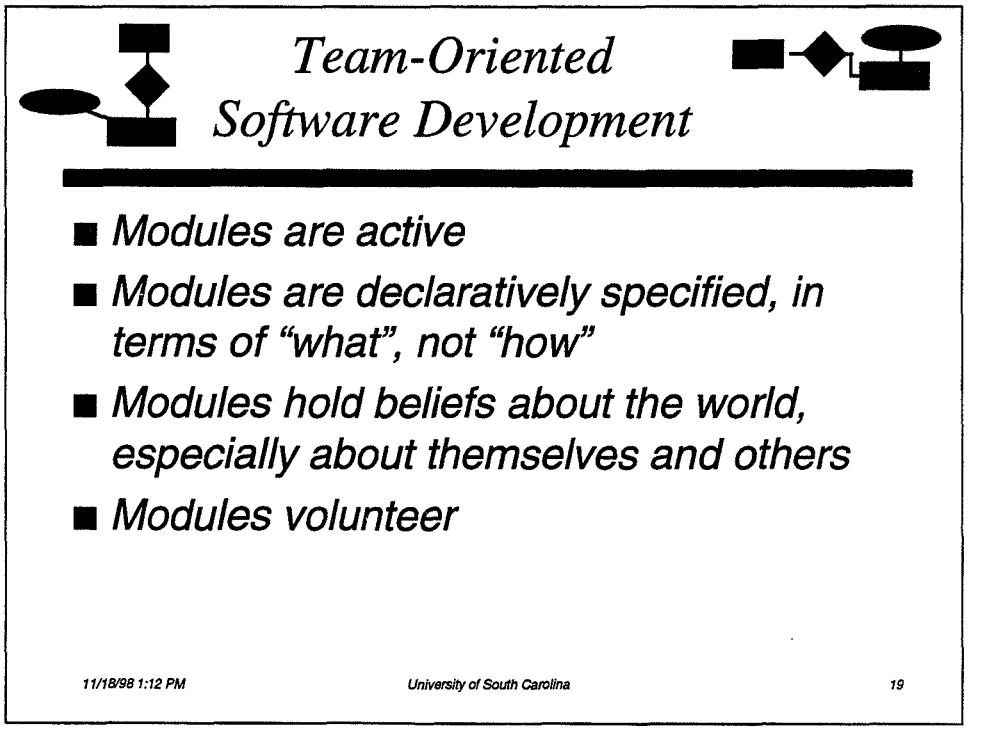
Features of Languages and Paradigms

A procedural language, an object language, and a multiagent language can be compared according to a number of criteria



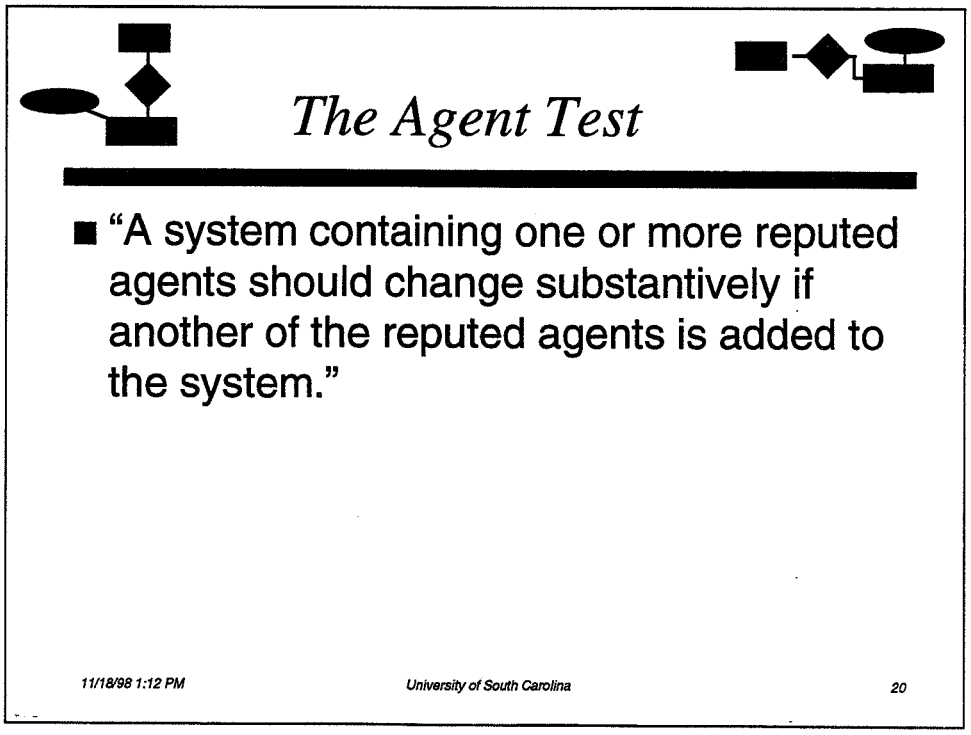
Team-Oriented Software Development

The most important characteristics of a team-oriented paradigm for software development are that the modules (1) are active, (2) are declaratively specified in terms of what behavior they should exhibit, not how they should achieve that desired behavior, (3) hold beliefs about the world, themselves, and others (whether humans or computational modules), and (4) the modules *volunteer* to be part of a software system. This last characteristic is a key to the reuse of software.



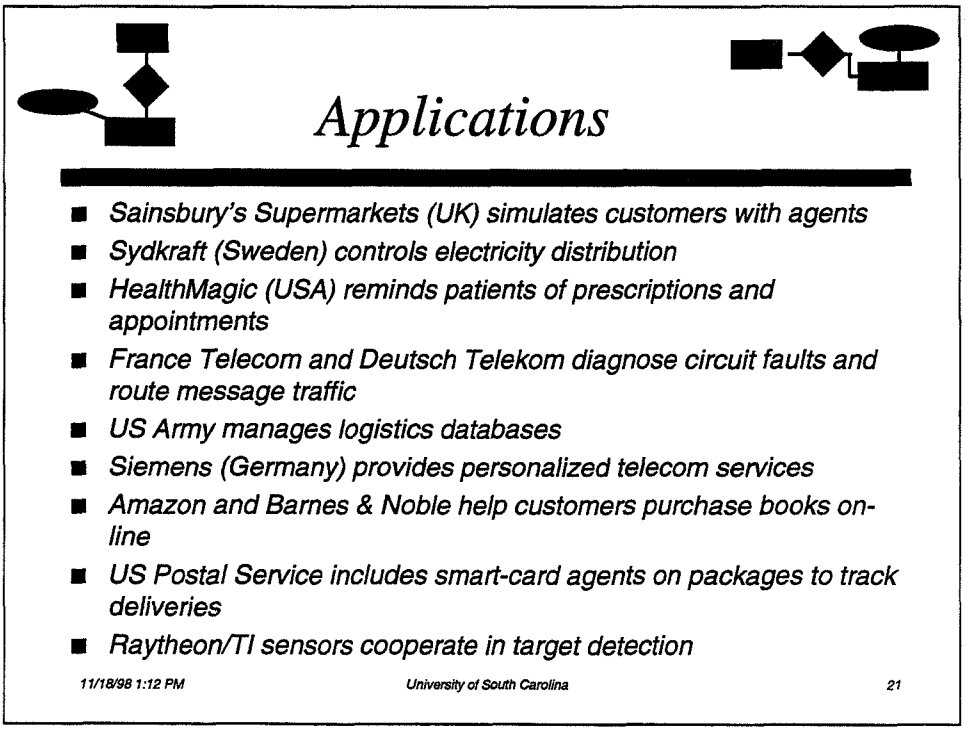
The Agent Test

Most researchers in agent technology have put forth their own definition of an agent. These definitions are usually a list of characteristics that an agent should possess, such as autonomy, persistence, reasoning ability, intelligence, communication ability, etc. Munindar Singh at NCSU and I instead propose a test for agenthood, which implies some of the above characteristics but does not require any of them. The test, to be useful, should be both necessary and sufficient, i.e., any software component that passes it should be considered generally to be an agent, and any component that fails it should be considered generally not an agent.



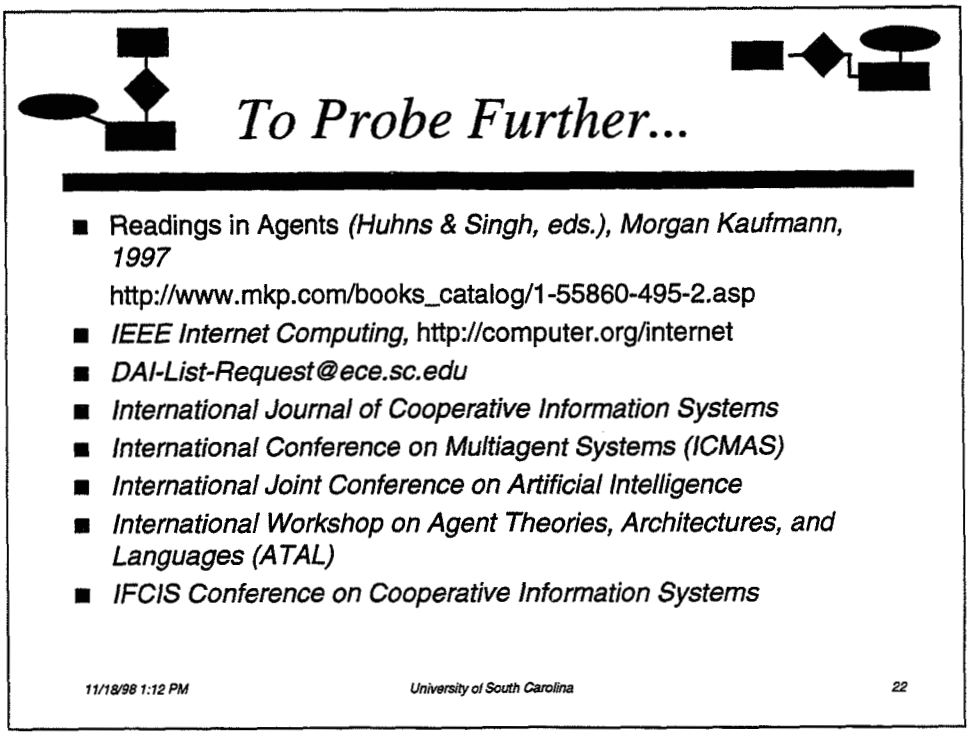
Applications

There are many important applications of agents in a wide variety of domains that are under development at the University of South Carolina in its Center for Information Technology.



To Probe Further...

There are many sources of information available on agent technology.



3/2-60
032526
370176
24P

The Open Agent Architecture™

Adam Cheyer, David Martin
and Douglas Moran
Artificial Intelligence Center
SRI International
Menlo Park, CA

Page intentionally left blank

The Open Agent Architecture™

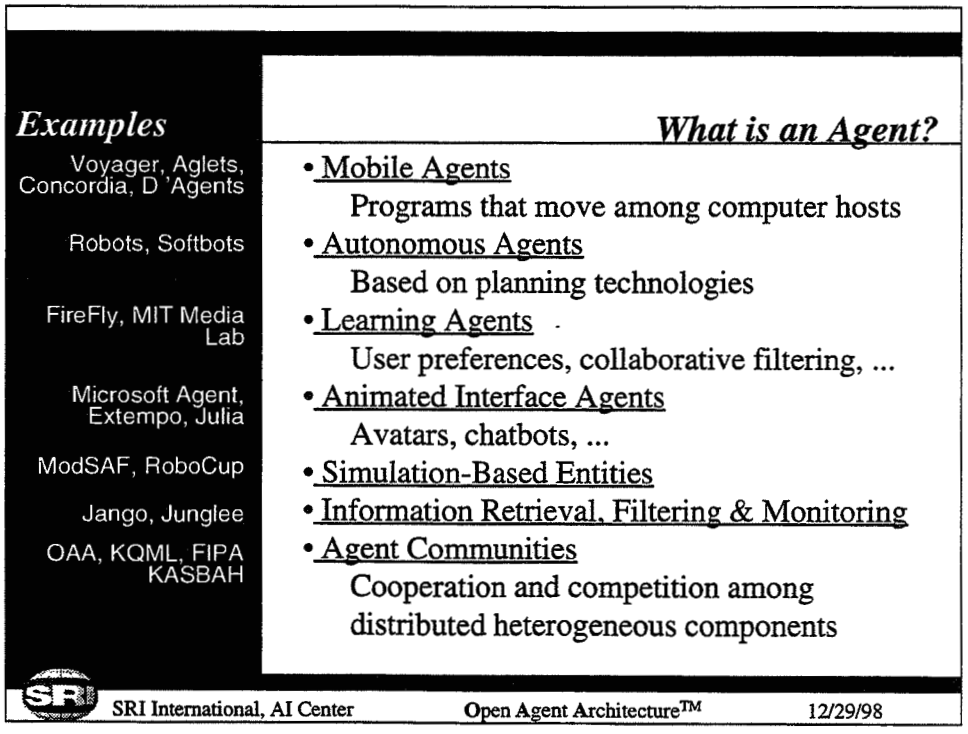
Adam Cheyer, David Martin
and Douglas Moran

Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025

This presentation provides an overview of the motivations, implementation, and application of SRI's Open Agent Architecture™ (OAA™), a new framework for constructing dynamic, distributed systems.

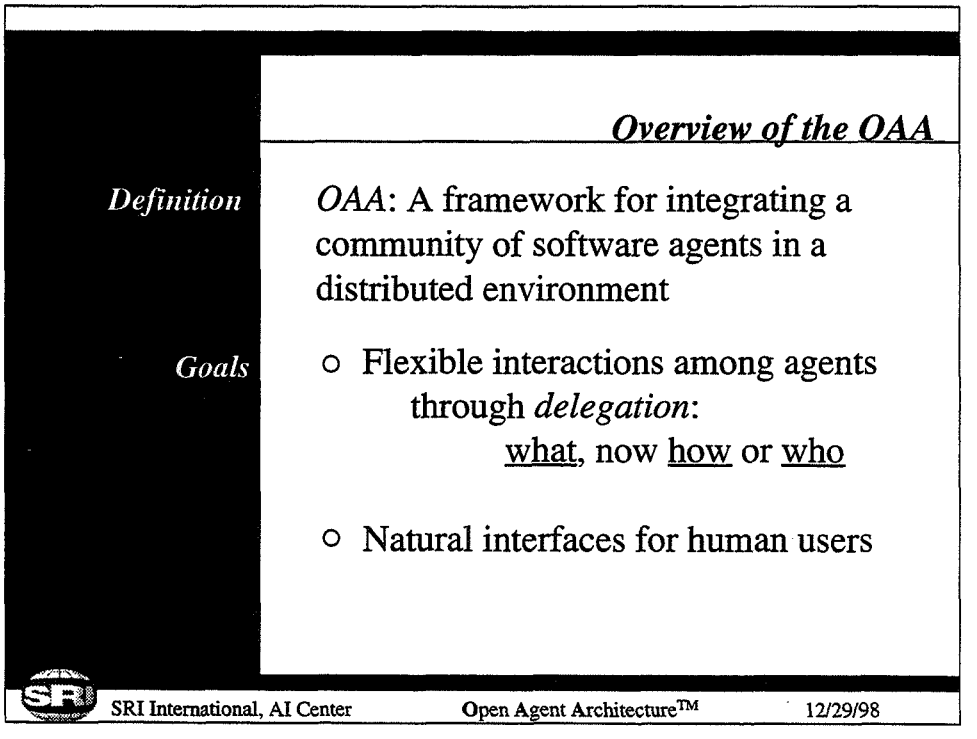
What is an Agent?

Many very different types of technologies have made use of the term "agent" to describe themselves. The OAA belongs to the class known as "cooperative" or "distributed agents," and can be thought of as a more powerful extension of "distributed object" frameworks such CORBA or DCOM. As you will see, OAA agents possess a number of features beyond distributed objects: a higher-level interface specification, an inter-agent communication language which can be translated to and from human natural language, and the ability to proactively monitor the state of the environment and autonomously take action based on various types of events.



Overview of the OAA

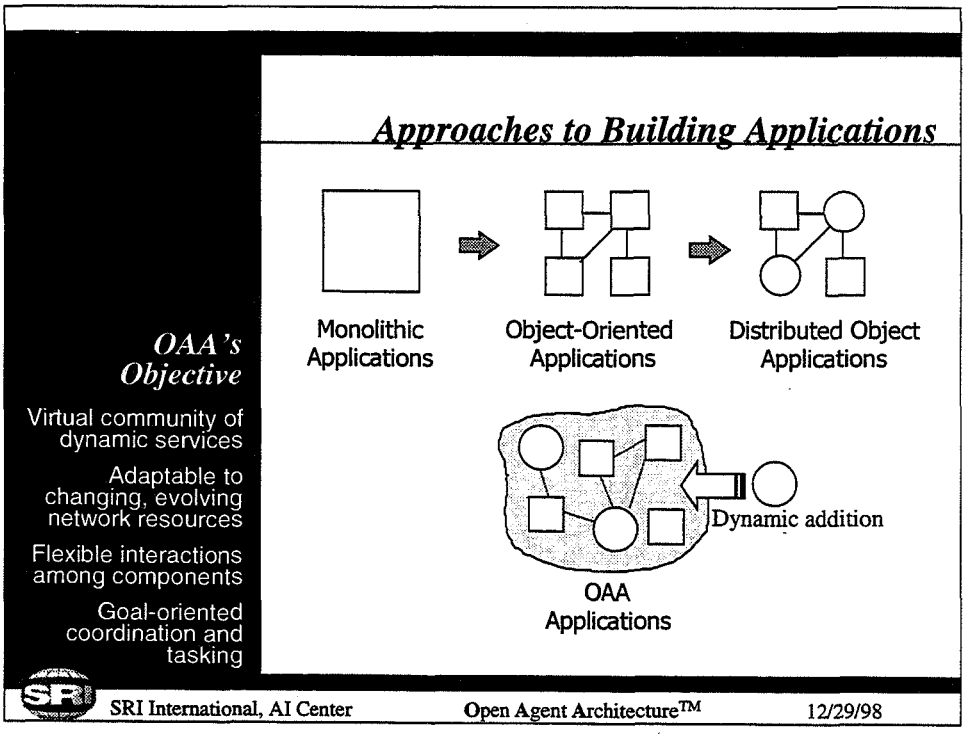
OAA research may be unique in that it simultaneously pursues two areas rarely grouped together in the same framework: 1) how to build more flexible, adaptable distributed systems, and 2) how can a human user interact more naturally with this virtual community of agents. As we hope to show, there are a number of surprising synergies between these two, seemingly disparate, objectives.



Approaches to Building Applications

In "the old days" (of which there still remain many remnants...), programmers constructed large monolithic applications which ran standalone on a desktop computer. Object-oriented technology encouraged improved code reusability somewhat, allowing a large program to be constructed from many individual components. However, the interactions among components were hard-coded by programmers. Distributed object frameworks enable the component pieces to be spread across multiple computers, but inflexible interactions among components remains a problem.

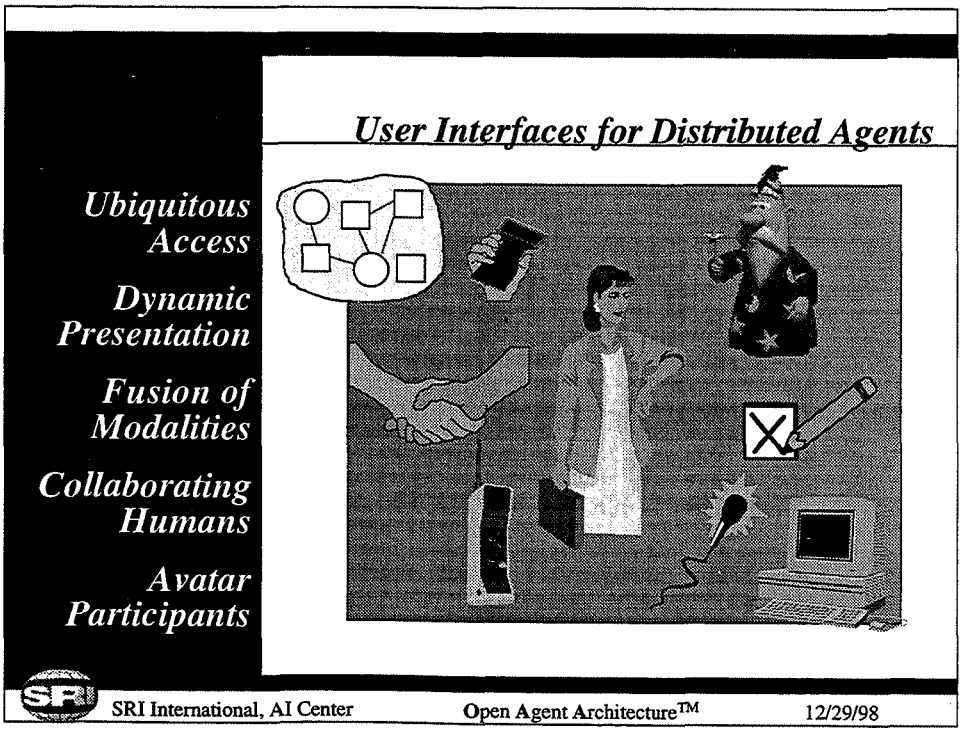
Current technology is not suitable to the dynamic nature of the Internet, where new, unimagined resources become available every day, and other network services disappear. What is required is the ability to create programs from a dynamic, virtual community of services which cooperate and interact in a flexible manner. This is one of the major goals of the Open Agent Architecture.



User Interfaces for Distributed Agents

If applications are going to be made up of many cooperative network services, it is essential that human users have efficient and natural methods for interacting with them. We believe in a multimodal approach, where any data or services can be accessed using flexible combinations of many input modalities - if graphical user interfaces are available, fine. If, in addition, a telephone or microphone is present, speech recognition might be used in conjunction with the more standard interfaces. Likewise for electronic pens. A distributed architecture must be able to adapt to the changing set of input and output resources available to the user. We also envision an architecture that supports multiple humans, and even computer avatars, to share the same workspace on collaborative tasks.

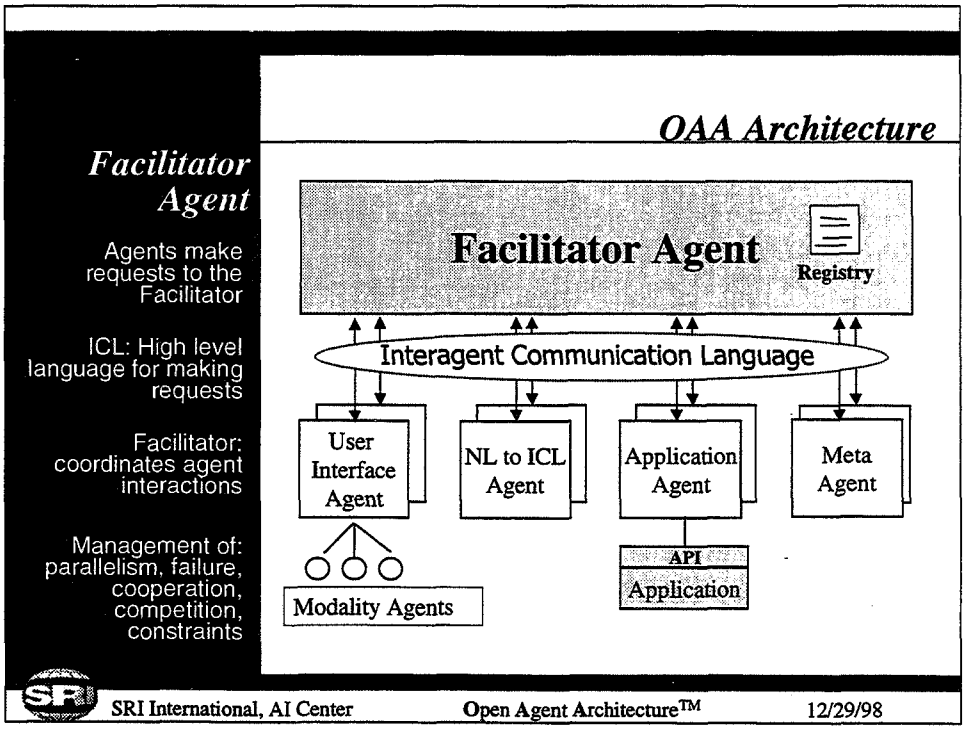
In this talk, we will highlight these user interface characteristics through demonstrations of several OAA applications.



OAA Architecture

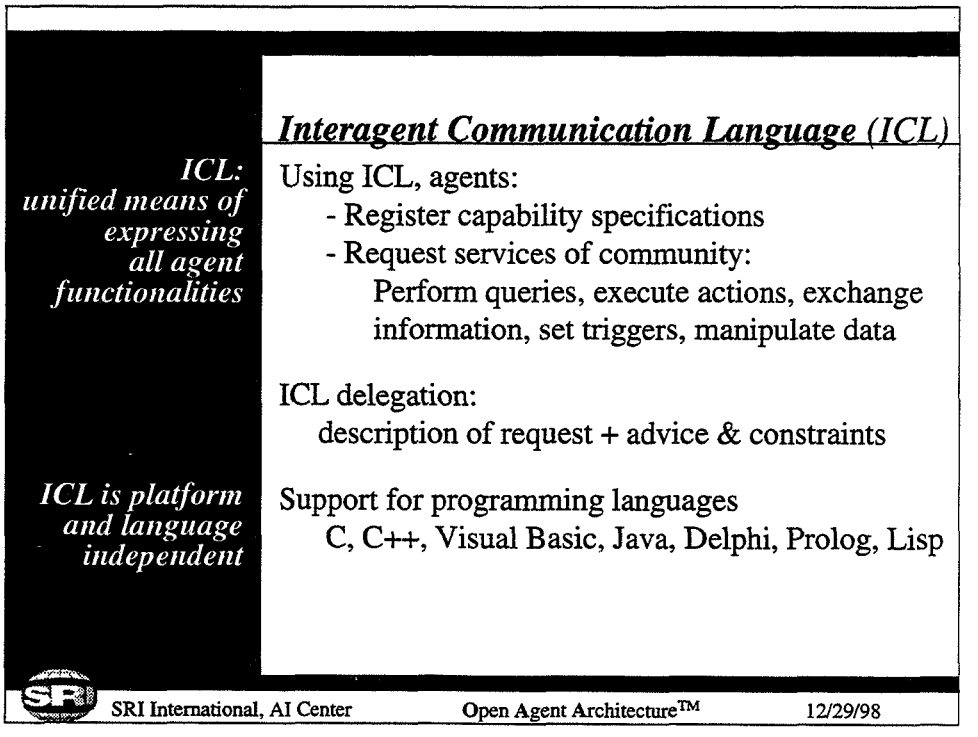
In the OAA, a Facilitator agent provides the agent community with a number of services for routing and delegating tasks and information among agents. Upon connection, each agent registers its functional capabilities and specifications of its public data. Then, when a human user or automated agent makes a request of the agent community, specifying at a high level the description of the task along with optional constraints and advice on how the task should be resolved, the Facilitator agent distributes subparts of the task among agents and coordinates their global activity.

All OAA agents share exactly the same characteristics, from Facilitator agents to User Interface agents: they publish their capabilities and communicate among themselves using the Inter-agent Communication Language. However, it is often useful to conceptualize several classes or types of agents as illustrated in this slide: UI agents, NL agents, Facilitator agents, Application agents, and Meta agents.



Inter-agent Communication Language (ICL)

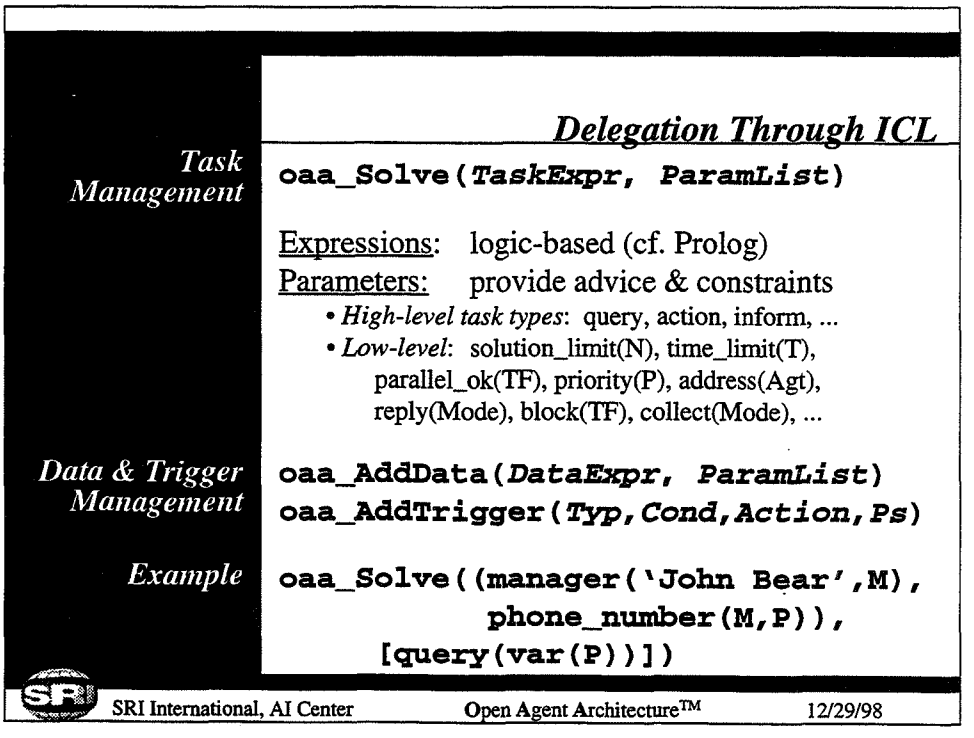
Perhaps the key innovation of the OAA is the Inter-agent Communication Language, the means with which agents exchange information, requests and notifications.



Delegation Through ICL

The ICL is a logic-based language that can represent complex, multi-step tasks. The language was designed to be compatible with the output of many natural language systems. The result is that a human user can make a request in English using vocabulary provided by the dynamic set of registered agents, and this request will be translated into a task description directly executable by the community.

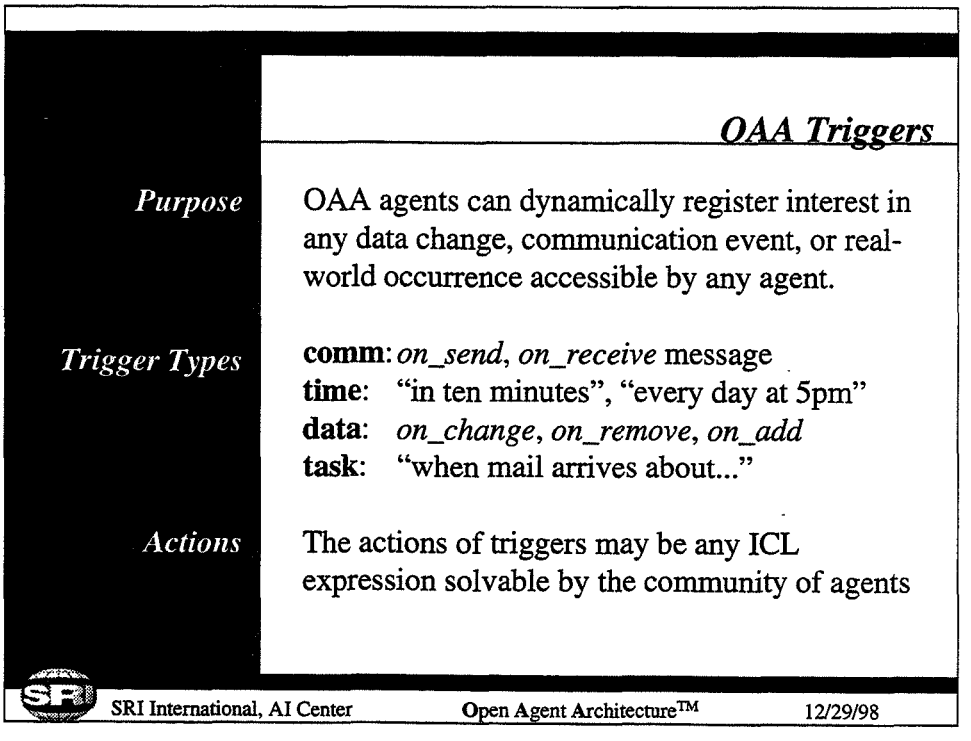
The ICL allows an agent (or human user) to delegate complex tasks to an agent community with a configurable level of detail. Generally, an agent will make a request supplying only basic suggestions about how the task should be executed: perhaps specifying the type of task, or special time constraints under which the task is to be performed.



OAA Triggers

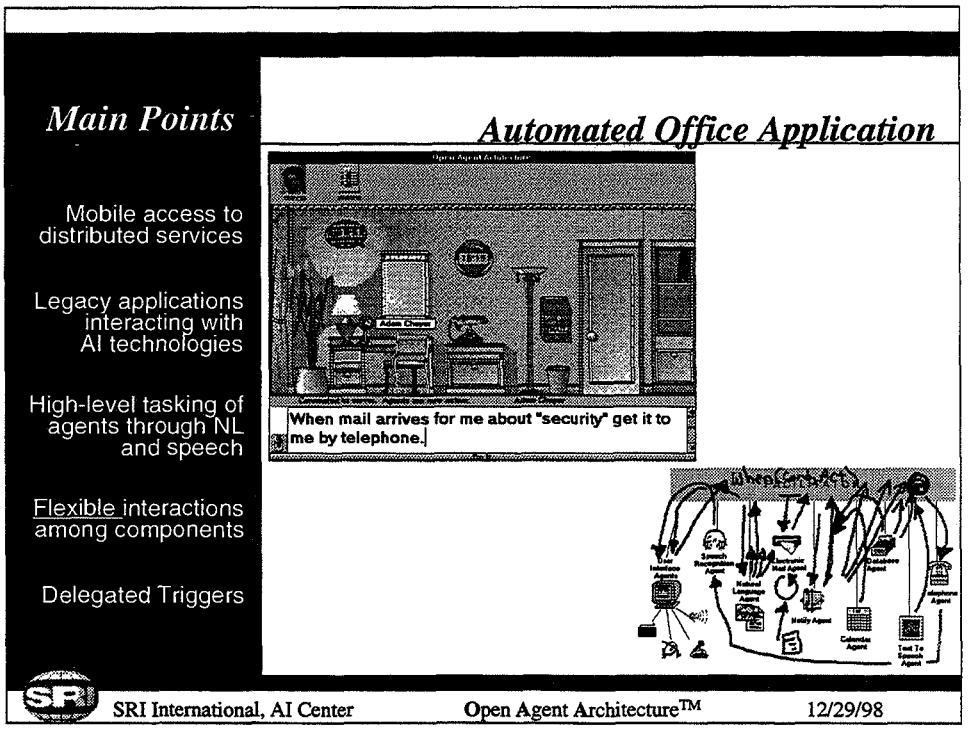
Triggers are managed by the agent library and can be delegated under the Facilitator's control across multiple agents in the system. As noted in the slide below, four types of triggers are built into the OAA infrastructure: triggers for communication messages, data changes, time conditions, or domain-specific (task) events such as the arrival of an email message.

- Triggers are stored using the OAA data management predicates, so agents are free to examine, search, add, or modify triggers on any other agent or agents.
- Creating a trigger requires that the user or agent specify at least its type, a conditional statement to test, and an action or other ICL expression to perform when the trigger fires. Optional parameters include recurrence values (how many times should the trigger fire before being removed), additional test conditions to try before the trigger fires, when the trigger should expire, etc.



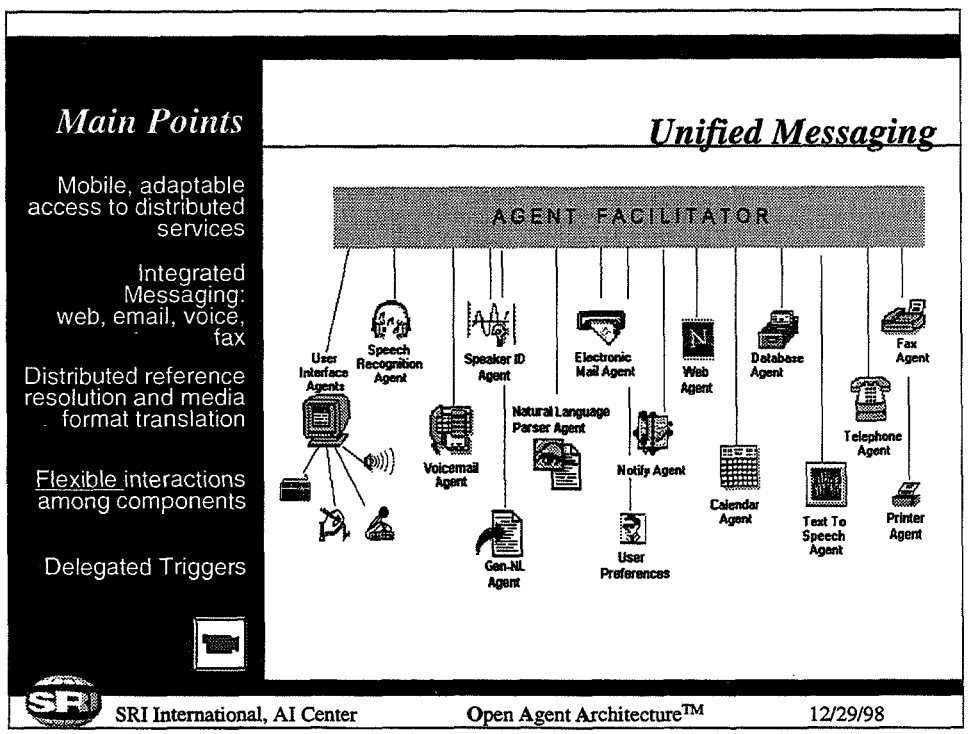
Automated Office Application

OAA characteristics such as flexibility agent cooperation can be best shown through an actual example. In the Automated Office system, we see that from one simple English request spoken into a telephone, many OAA agents, written in several programming languages and spread across multiple computers, can cooperate and compete (when appropriate in parallel) to resolve a task for the user. The system is extensible beyond many other distributed systems -- as new agents are added at runtime to the system, what the user can say and do literally changes. In addition, the execution process is highly distributed: there is no single agent (not even the Facilitator agent) who has knowledge pre-coded into it specifying how agents will work together for all given user input.



Unified Messaging

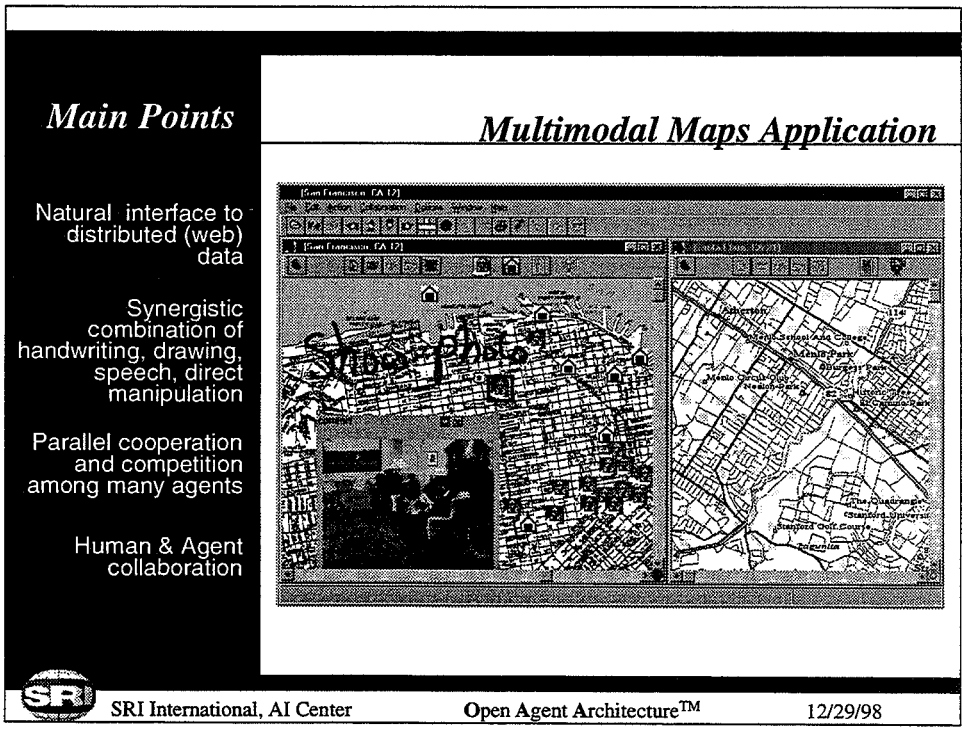
The Unified Messaging application is a direct extension of the Automated Office application, but provides greater support for media translation, distributed reference resolution, and adaptable presentation. It also adds a number of media and presentation agents such as fax, printer, voicemail, etc. The focus of this application is on how to build a dynamic community of agents that can adapt to the input and output media used to access them (e.g., graphical user interface, telephone).



Multimodal Maps Application

The Multimodal Maps application illustrates a natural user interface to distributed agent services, using services provided by a distributed, parallel infrastructure. Ambiguities at many levels during the interpretation process (modality fusion) are resolved by competing and cooperating agents operating in parallel.

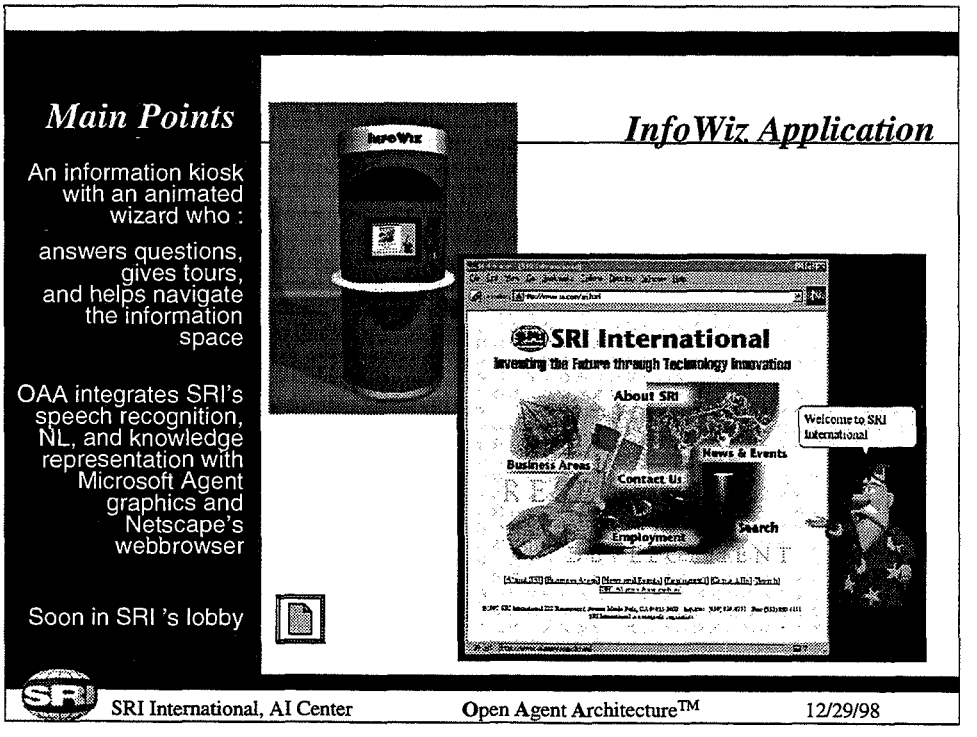
The Multimodal Maps application also uses the collaborative services of the OAA infrastructure, enabling multiple human participants to share a common workspace from remote locations, exchanging information and requests with each other and with automated agents.



InfoWiz Application

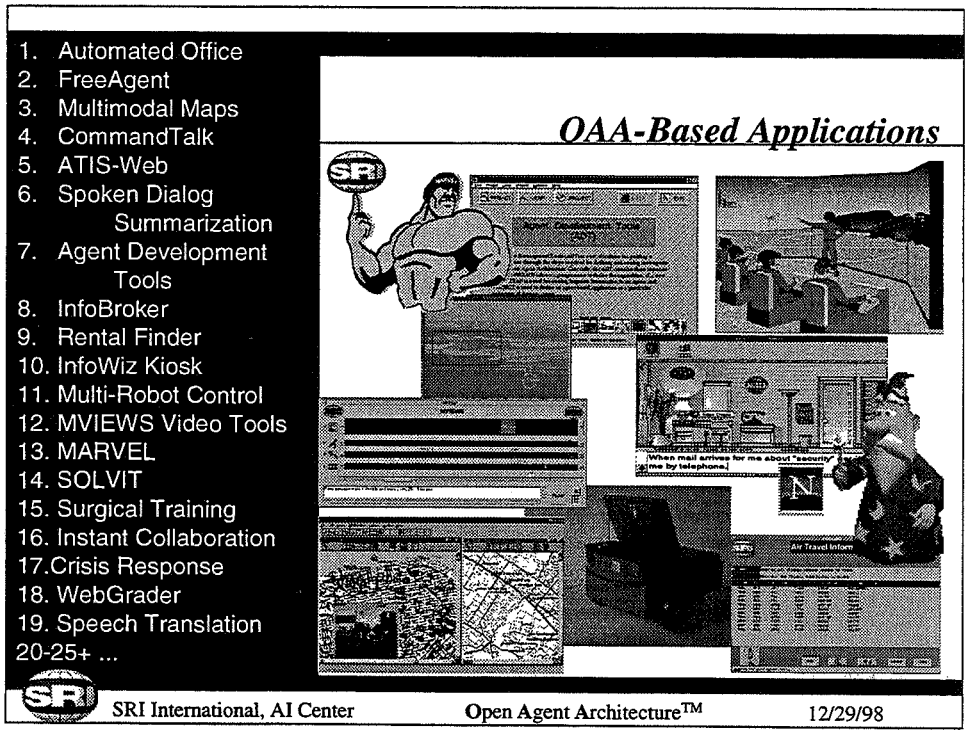
The InfoWiz project is centered around the idea of putting an interactive kiosk into the lobby of SRI. People who have a few minutes to spend should be able to learn something about SRI, enjoy themselves, and walk away with a good feeling of having seen something interesting and unusual.

One of the design decisions of the project has been to use speech recognition as the main form of user input to the system. In order to encourage spoken interaction with the system, we have created an animated character, a cartoon wizard, who attempts to engage the user in conversations about SRI. The InfoWiz can answer questions, provide supplementary information, make suggestions, and take the user on guided tours of the information space. This is our first attempt at allowing an animated avatar agent to interact directly with humans.



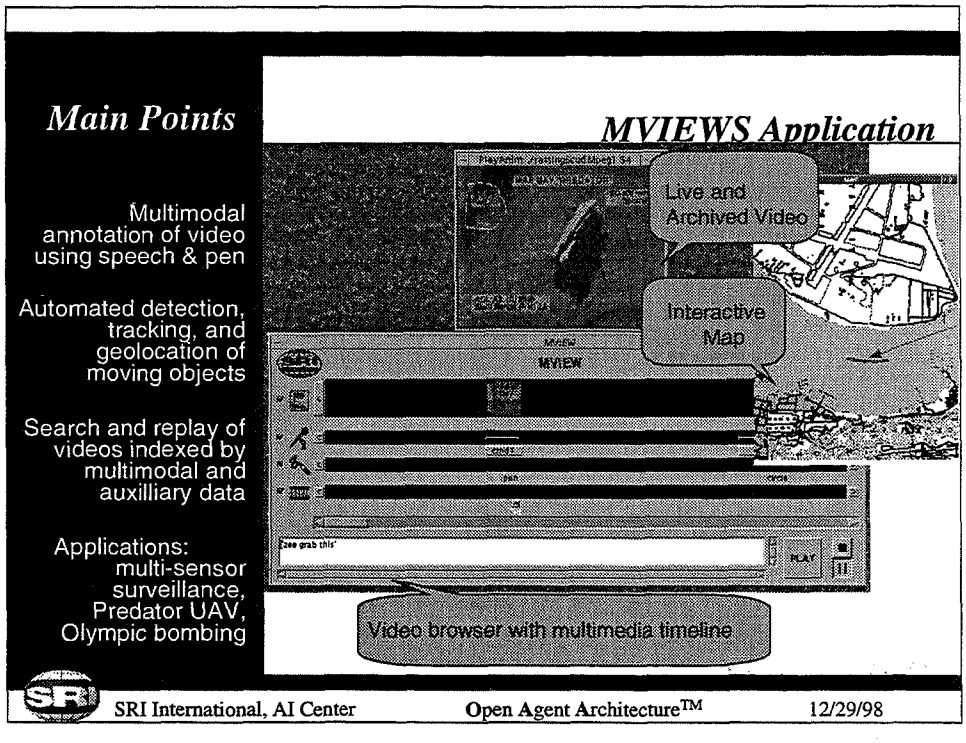
OAA-Based Applications

The OAA is a general-purpose framework which has been applied to a wide number of distributed applications in diverse domains. So far, we have presented several applications which illustrate OAA's unique capabilities for including the human user as a special member of the agent community. In the following slides, additional applications of OAA technology will be briefly discussed.



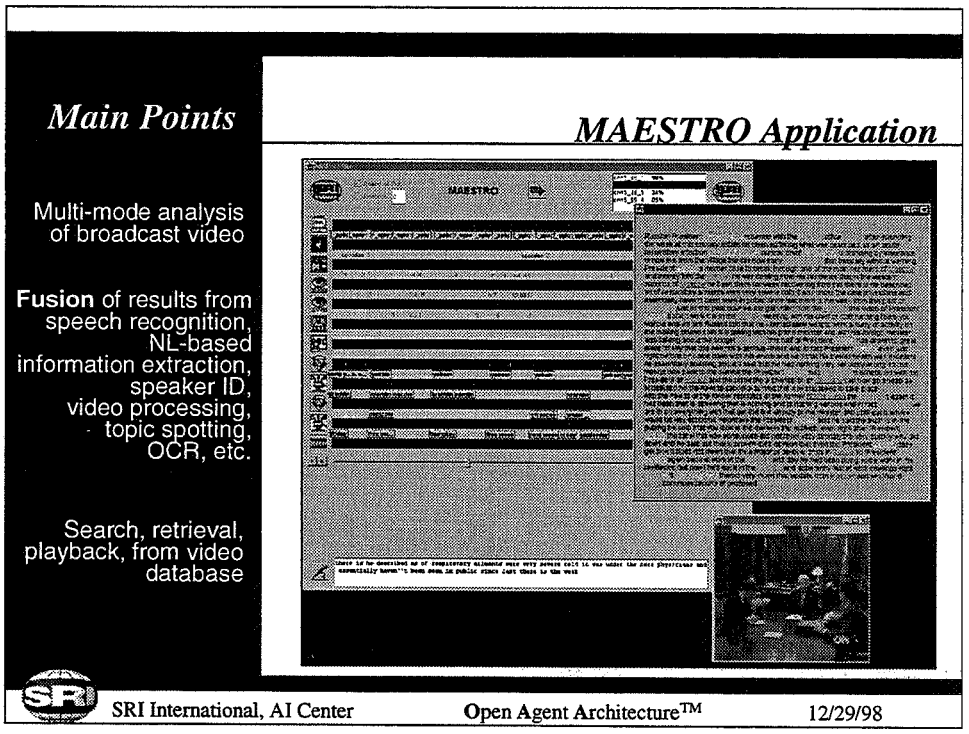
MVIEWS Application

Full-motion video has inherent advantages over still imagery for characterizing events and movement. Military and intelligence analysts currently view live video imagery from airborne and ground-based video platforms, but few tools exist for efficient exploitation of the video and its accompanying meta-data. In pursuit of this goal, SRI has developed MVIEWS, a system for annotating, indexing, extracting, and disseminating information from video streams for surveillance and intelligence applications. MVIEWS integrates technologies such as pen and voice recognition and interpretation, image processing and object tracking, geo-referenced interactive maps, multimedia databases, and human collaborative tools.



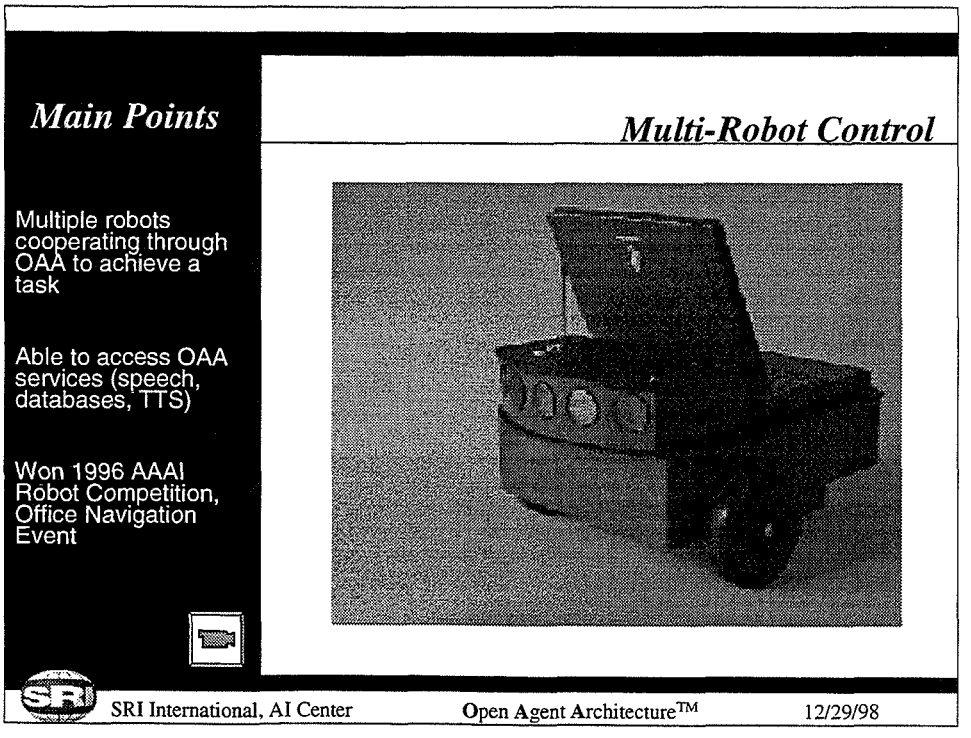
MAESTRO Application

The MAESTRO system, integrated with the MVIEWS video tools, uses fusion of numerous recognition technologies to improve automated recall and analysis of broadcast news videos.



Multi-Robot Control

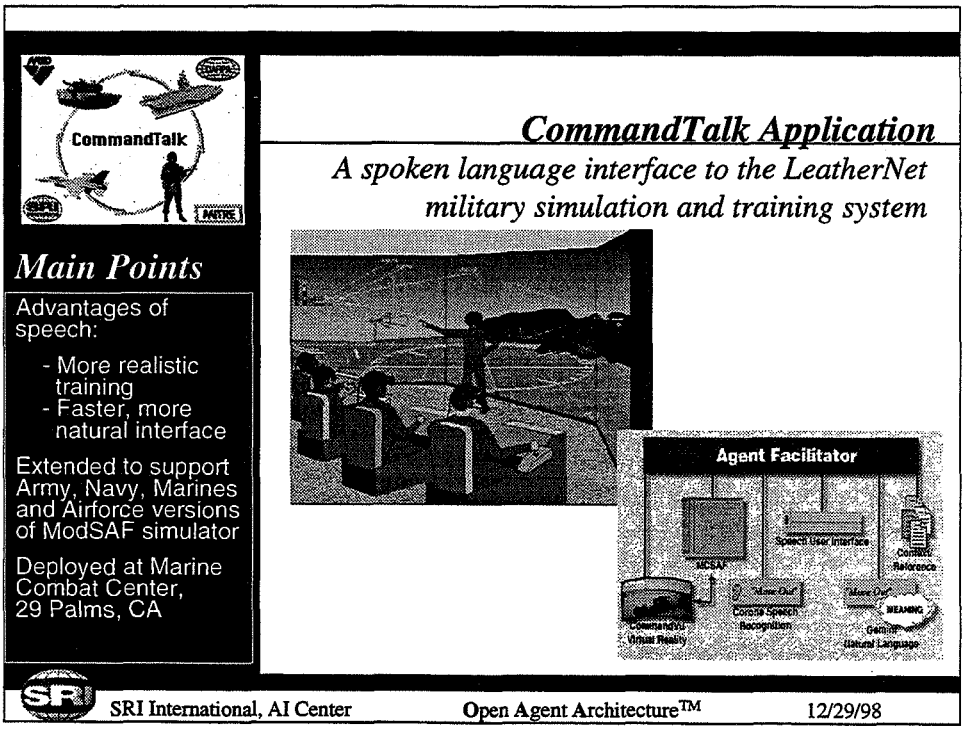
Robots, integrated as members of an OAA community, can access distributed services such as speech recognition, text-to-speech, map software, and so forth, and can communicate with each other to accomplish coordinated tasks. Using this approach, a multi-robot team captured first place at the 1996 AAAI robot competition, office navigation event.



CommandTalk Application

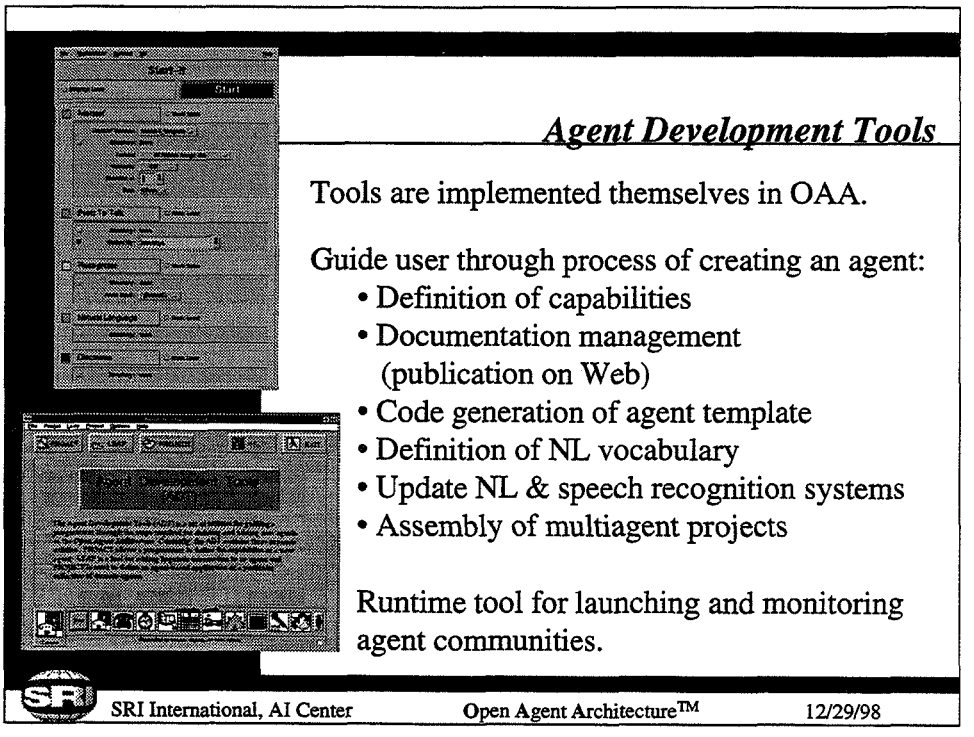
CommandTalk is a spoken-language interface to synthetic forces in entity-based battlefield simulations, developed by SRI International under our DARPA-sponsored project on Improved Spoken-Language Understanding. The principal goal of CommandTalk is to let simulation operators interact with synthetic forces by voice in a manner as similar as possible to the way that commanders control live forces.

CommandTalk was initially developed for LeatherNet, a simulation and training system for the Marine Corps developed under direction of the Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD). Recently, CommandTalk has been extended to Navy, Air Force, and Army versions of ModSAF, to provide control of all synthetic forces in DARPA's STOW97 Advanced Concept Technology Demonstration.



Agent Development Tools (ADT)

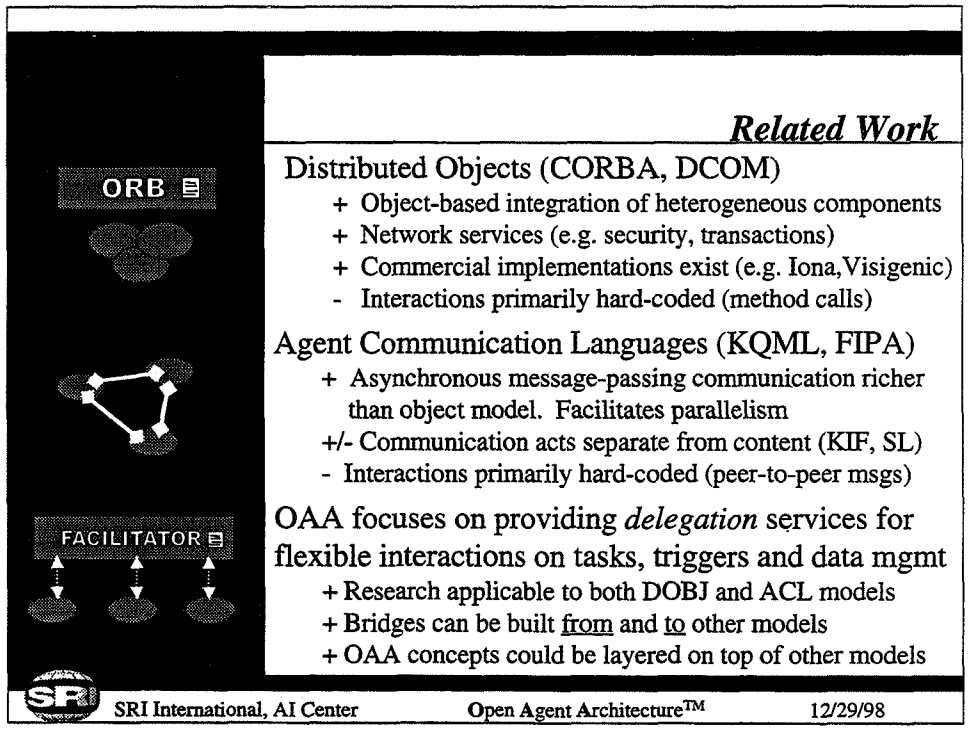
A set of runtime and development tools have been created to guide an agent developer through the steps of creating a new agent and including the agent within an application community. The tools are implemented as agents themselves, allowing them to collaborate with each other and with other agents.



Related Work

OAA, like distributed object frameworks such as OMG's CORBA or Microsoft's DCOM, supports applications formed of distributed, heterogeneous components. But interactions among objects in the latter systems are hand-coded by programmers, who must know which objects are available and what services they provide. Even distributed agent frameworks like KQML and FIPA, which rely primarily on higher-level message passing, produce applications with tightly-coupled component interdependencies.

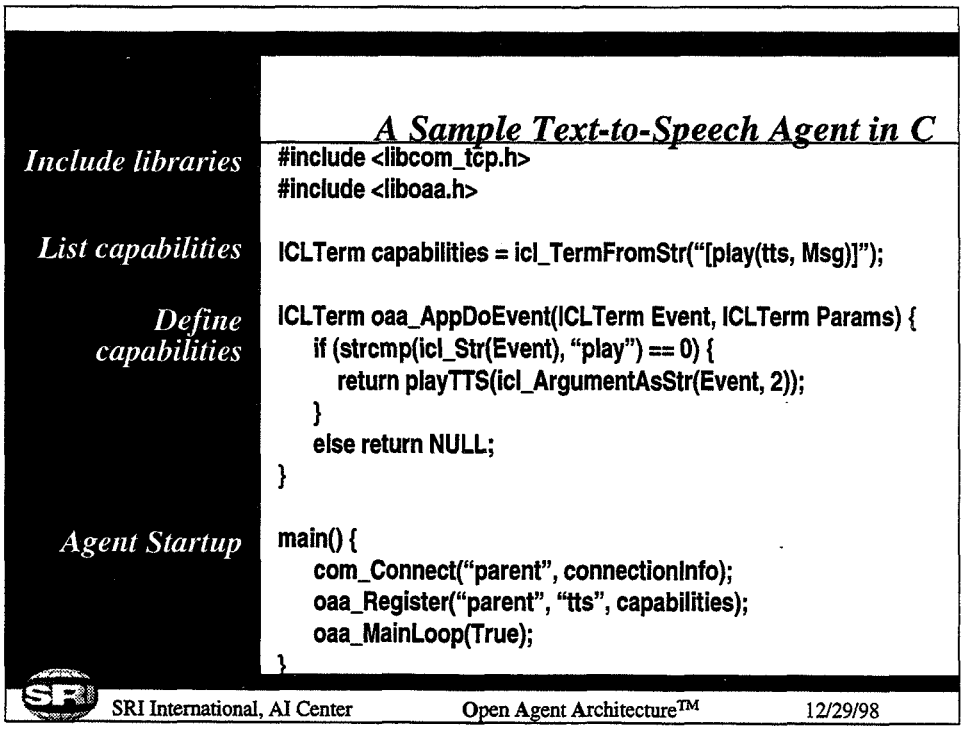
The OAA tries to relax some of these constraints by making the process of defining agent interactions a cooperative task between the programmer and an automated Facilitator agent. The programmer specifies the capabilities of an agent using a rich description language, and then defines needs in abstract terms. The Facilitator agent then instantiates these requirements in assignments to agents, managing parallelism, failure conditions, conflicts, etc., for the task.



A Sample Text-to-Speech Agent in C

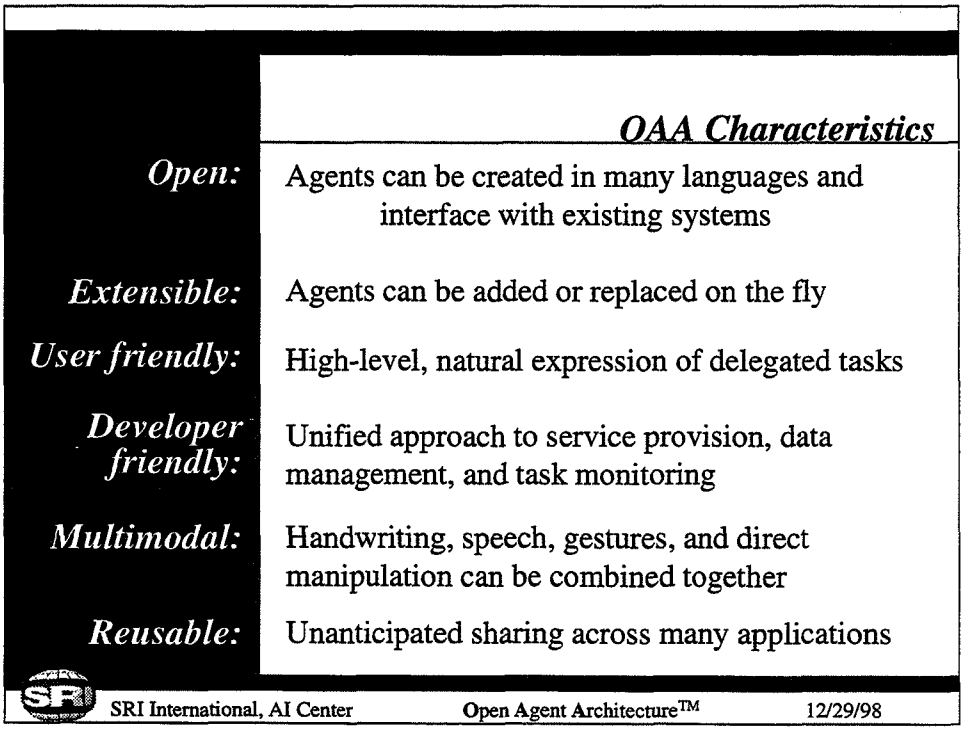
Creating a new OAA agent involves the following steps:

- Include libraries for OAA and choose a communications subsystem.
- Define the list of capabilities your agent can solve using ICL expressions. These may be either simple patterns to unify against an incoming request (e.g., play(tts, Msg)), or more complex specifications that include translations and synonyms, executable test conditions or constraints, etc.
- For each capability declared, the agent should parse an incoming request using the built-in icl_ routines, map the request to the API of the underlying application, and then return solutions to the request by constructing ICL return values.
- The body of the agent should initialize a connection to a Facilitator agent, register the agent's capabilities and name with it, and then enter a loop waiting for incoming requests from other agents.



OAA Characteristics

In summary, here are some of the main characteristics of the OAA framework for distributed computing.



513-62
032 527
37078
26p

KR for the World Wide Web

James Hendler
University of Maryland
College Park, MD

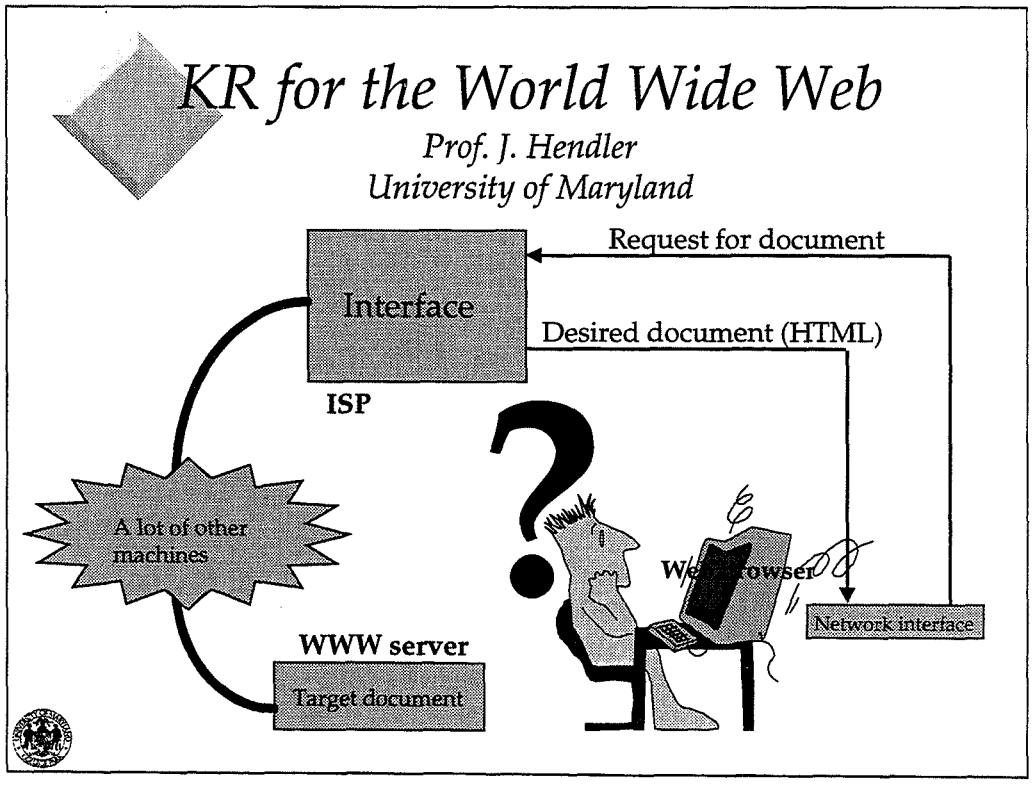
Page intentionally left blank

KR for the World Wide Web

James Hendler
University of Maryland
College Park, MD

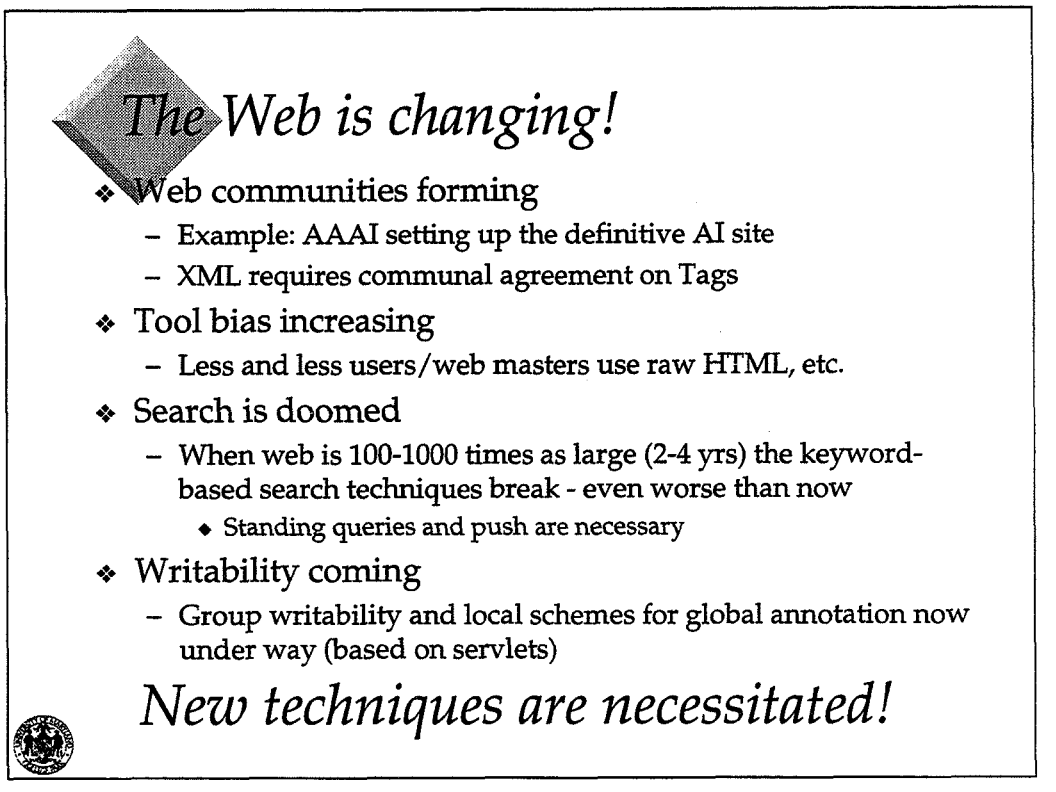
One of the most exciting changes in computing in the past decade has been the introduction of the world wide web and the internationally expanding internet, making enormous amounts of information available to users regardless of location. This access, however, has created information management problems beyond the capabilities of most systems. AI systems, combining intelligent agent technologies with large knowledge bases, have long been proposed as a leading contender for dealing with searching, managing, and filtering this wealth of knowledge.

In this talk, we look at some work aimed at bridging the gap between AI and databases, and the use of this research in support of world wide web applications. In particular, we describe the SHOE language, an ontology mark-up language for web documents. We show how SHOE is being used to support an emerging community-wide web management and search tool for researchers in microbiological epidemiology (particularly concerning documents on "transmissible spongiform encephalopathies," such as the well-known "mad cow disease"). Also described is the Parka-DB™ system, a combination of AI and database technologies which allows for collection of SHOE data into knowledge bases that can be browsed or queried by users.



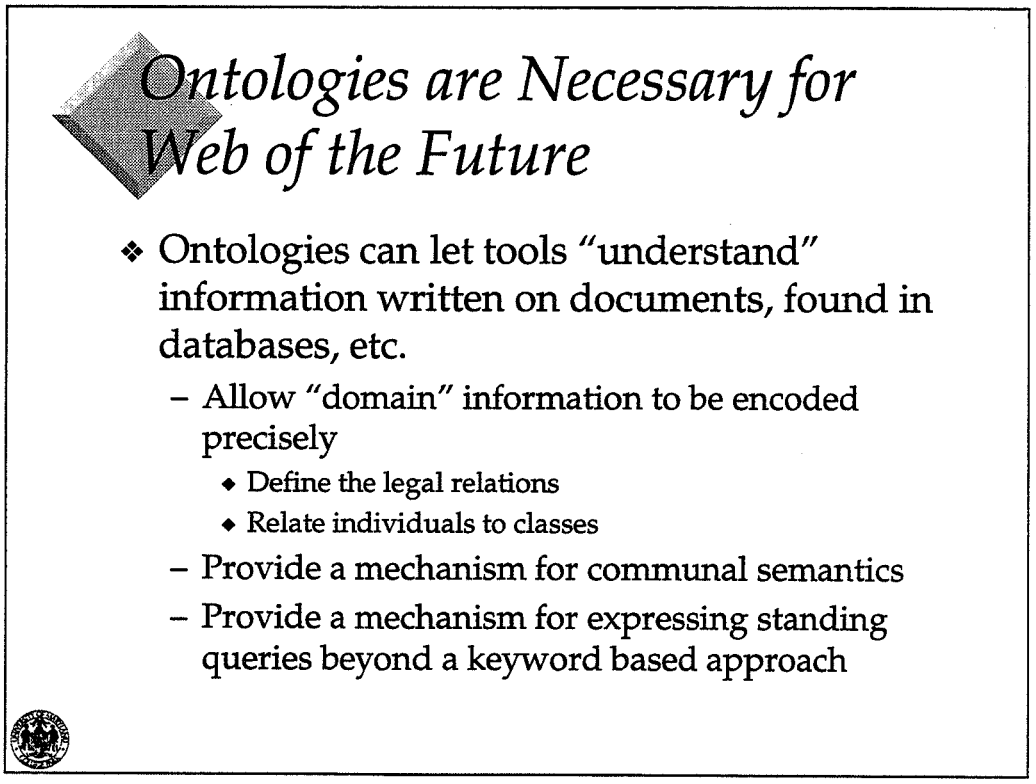
The Web is Changing

Trends in computing and information technology indicate a number of new directions in which we see the world wide web changing from its current form. These trends are the motivation for our work, which focuses on building tool-based mechanisms to support groups of web users who form on-line communities.



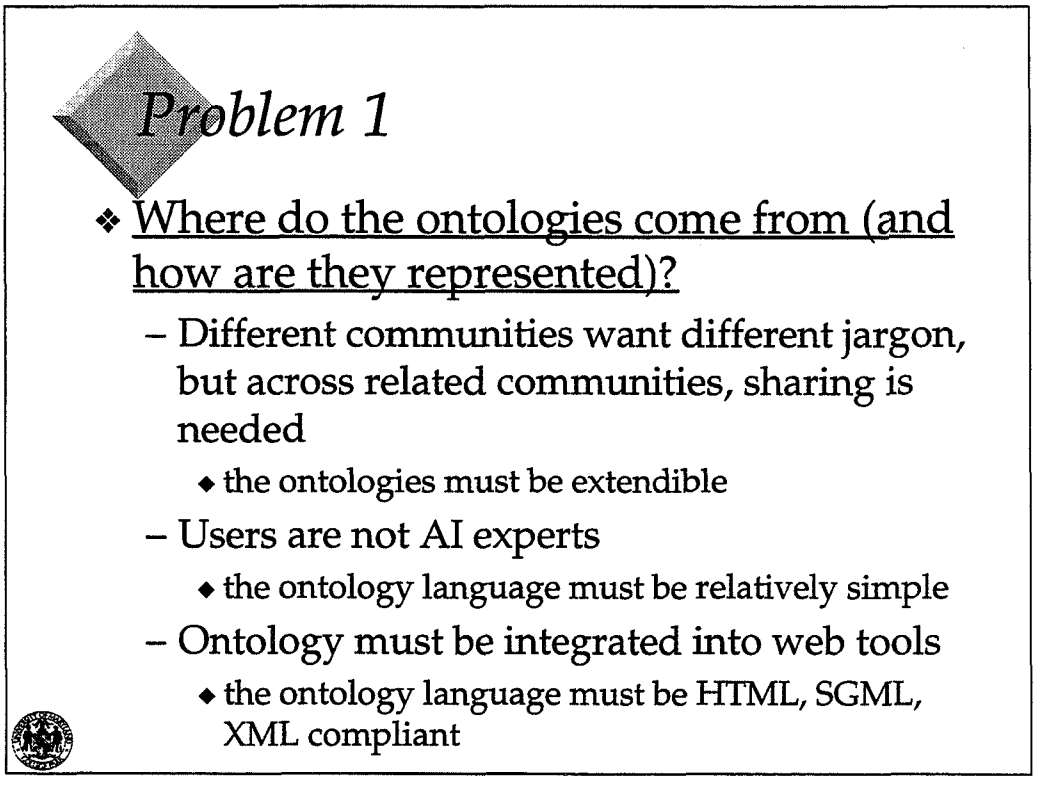
Ontologies for Web Use

The changing nature of the web mandates that many applications must have access to information beyond the simple English words appearing on the page. Ontologies are formal languages that let us specify the particular terms for individuals and classes and the relationships between these.



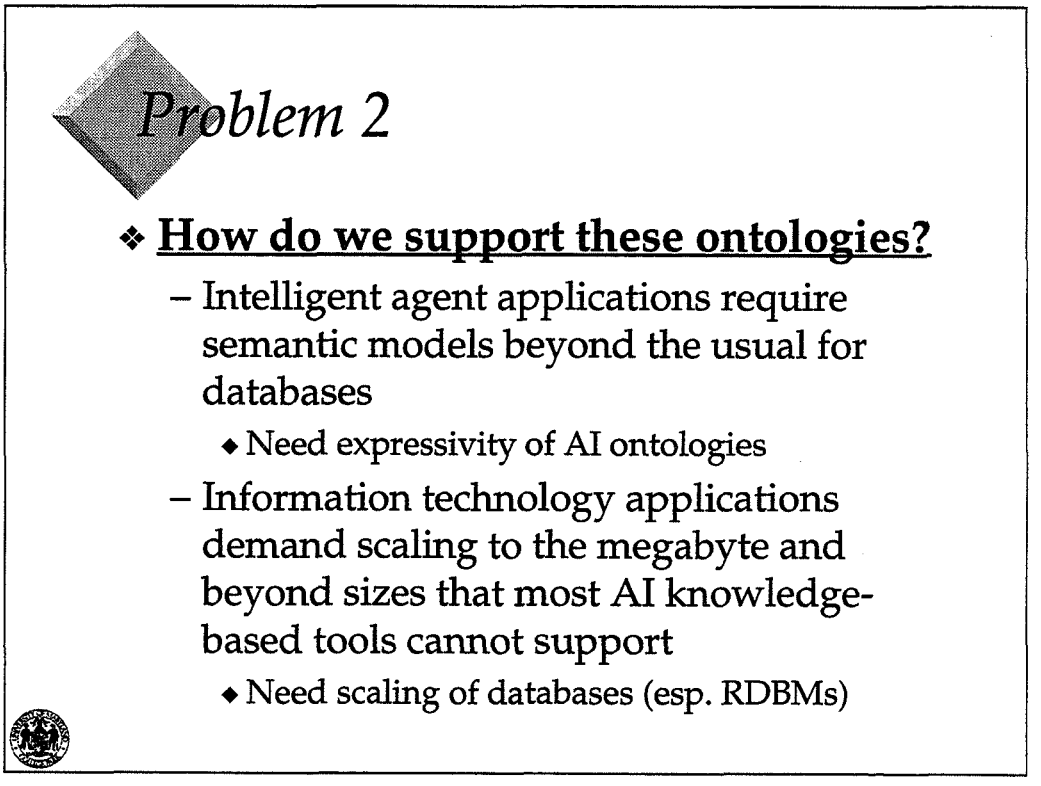
Problem 1: Where Do Ontologies Come From?

To provide a machine readable ontology, it must be specified in a formal language. Most previous work has focused on languages based on formal logics, which are notoriously difficult for untrained users to author.



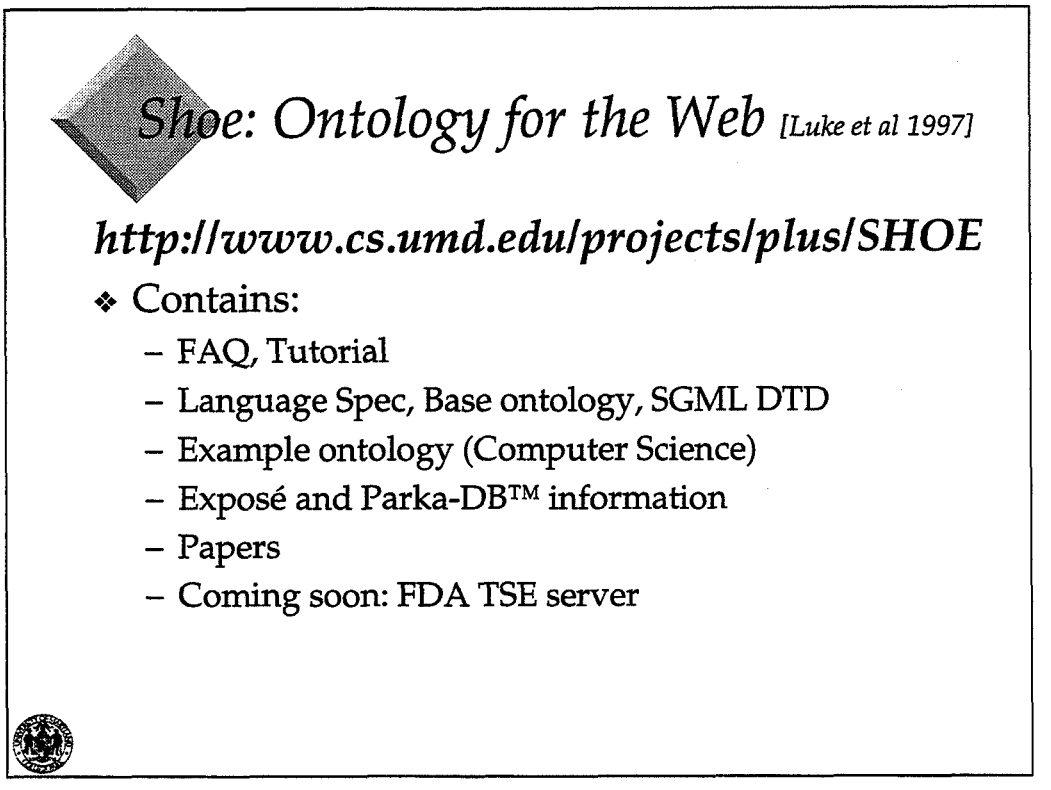
Problem 2: Ontology Support Tools

The second problem arises from the difficulty in finding tools that allow the expressivity of AI languages without sacrificing efficiency. Database languages, whether relational or object-oriented, do not permit the scope of expressive forms and inferencing permitted by AI languages.



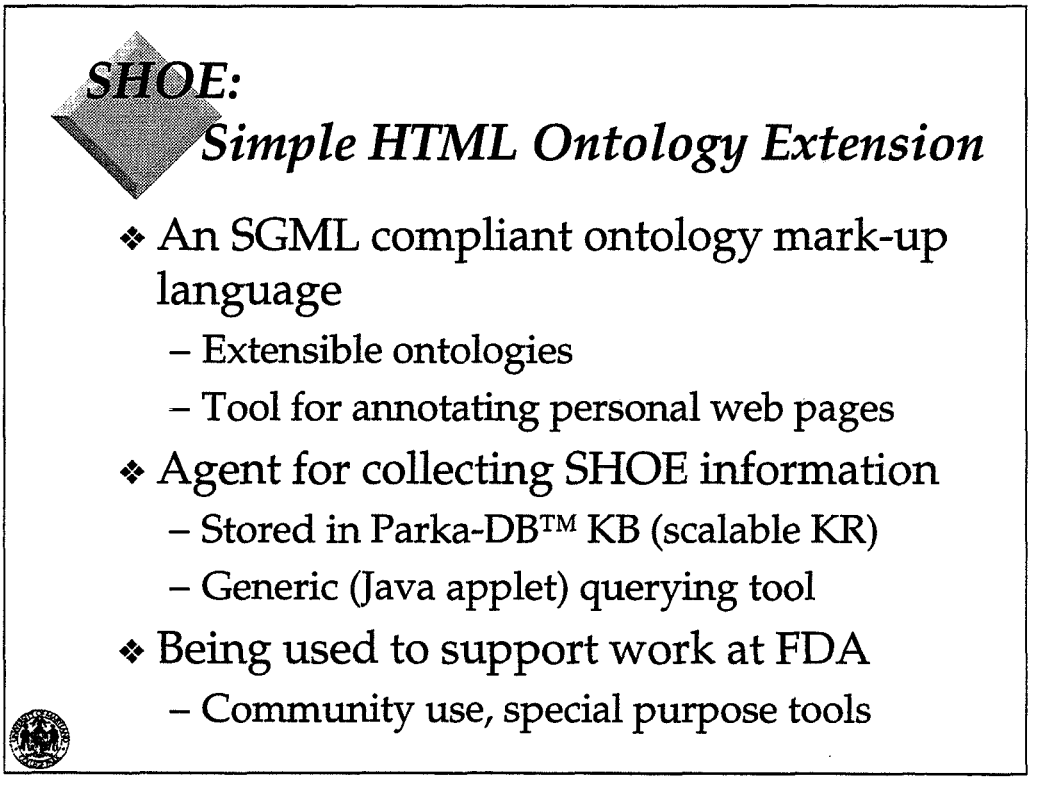
SHOE

SHOE is a language for creating machine-readable ontologies that can be recorded on web pages. The home page for more information about the SHOE language is http://www.cs.umd.edu/projects/plus/SHOE.



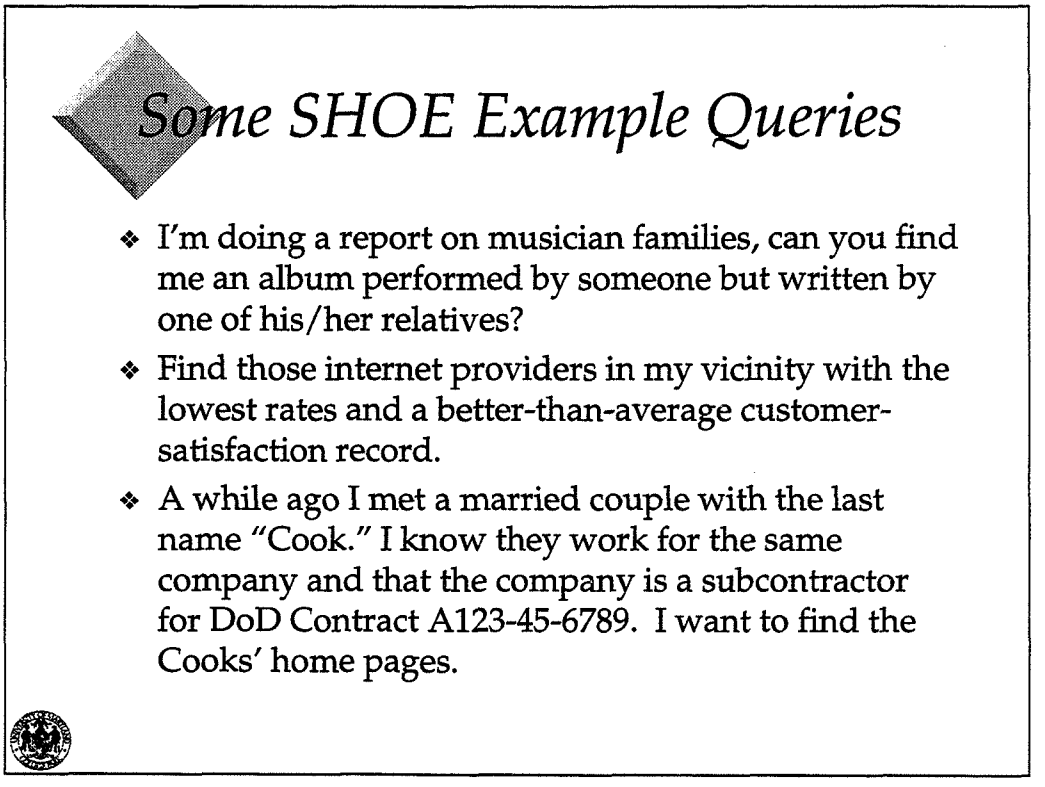
SHOE Description

SHOE has been designed to be interoperable with many current web tools and techniques. It has been designed with SGML and XML considered, and this makes SHOE much more usable than other AI techniques. SHOE is also designed to interact with the PARKA-DB ontology management system. Parka-DB is a patent-pending system that allows ontological information to be stored in database formats. It is described in more detail later in this presentation.



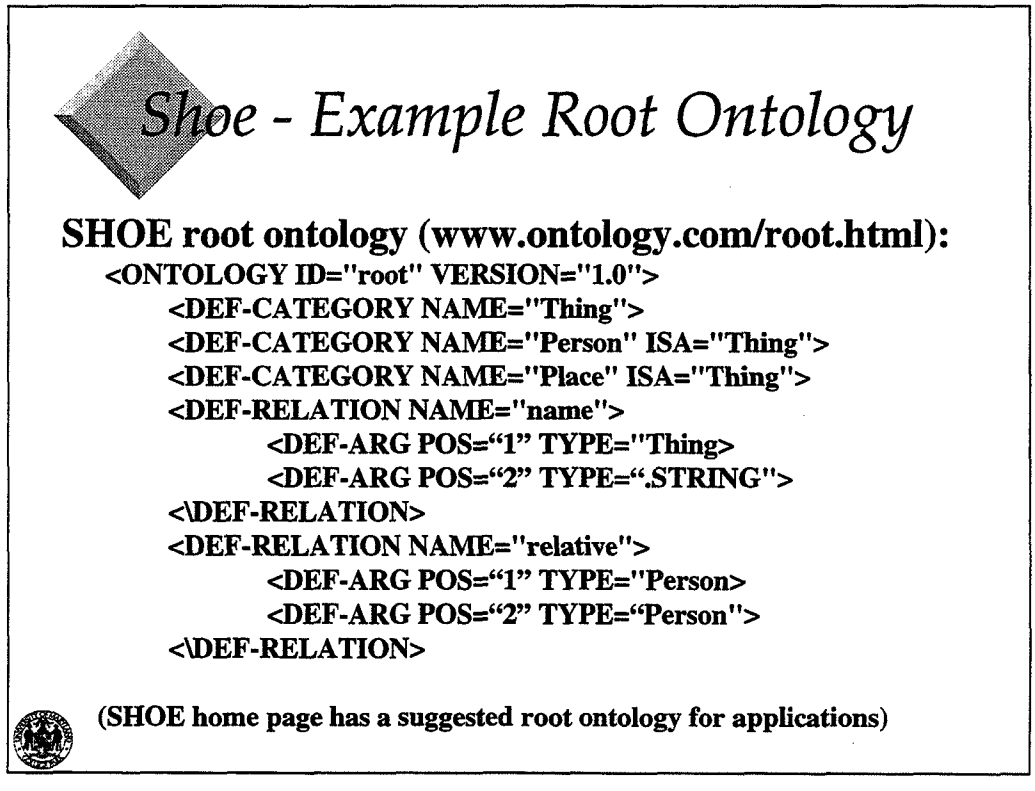
Example Queries

The examples below illustrate the need for SHOE. They point out that while there are many queries that could be answered from existing web sources, but which are very difficult to answer with current web tools. We will work through one of these examples in detail on successive pages.



Root Ontology

SHOE allows one ontology to modify and extend another. These must be rooted in some very basic terms, and can be found in a root ontology that is located on the SHOE web page or which can be created by others at some other web site. Here we show some of the basic facts from the SHOE root ontology defining person, places, things and names. Also shown is the fact that one person can be a "relative" of another.

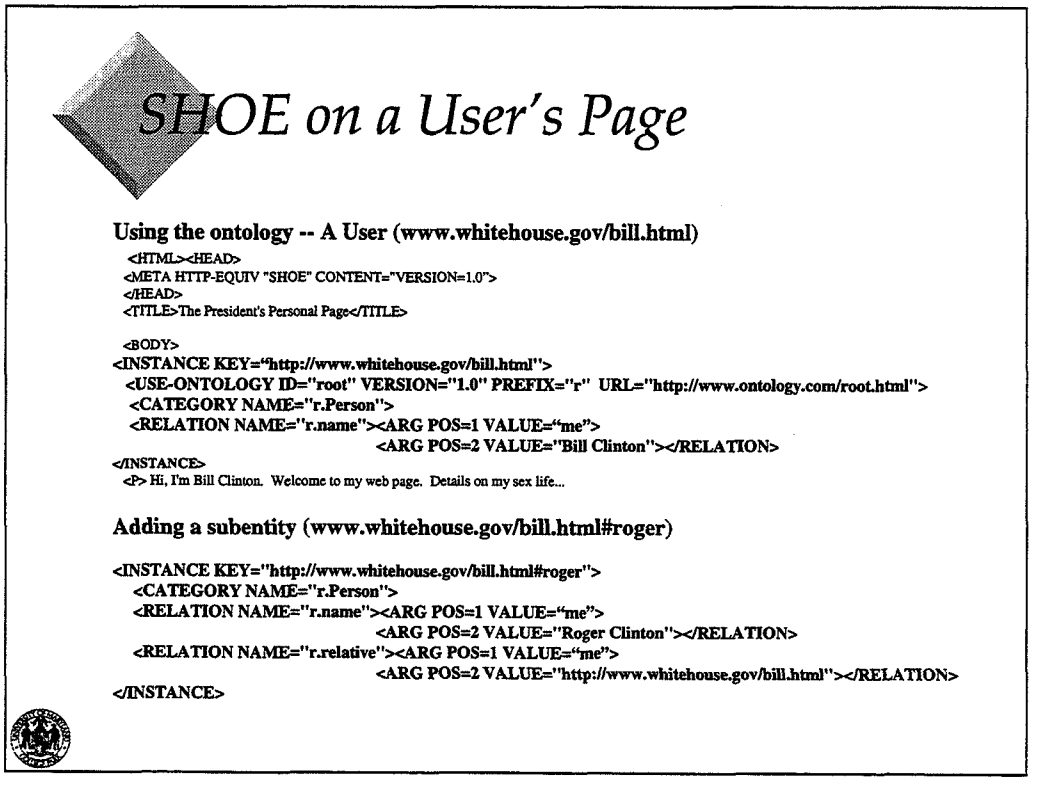


Extending a SHOE Ontology

An ontology defined in SHOE can be extended. The example below extends the root ontology shown previously by defining the terms *album, image, cover* and *performer* as would be expected in normal use of these terms.

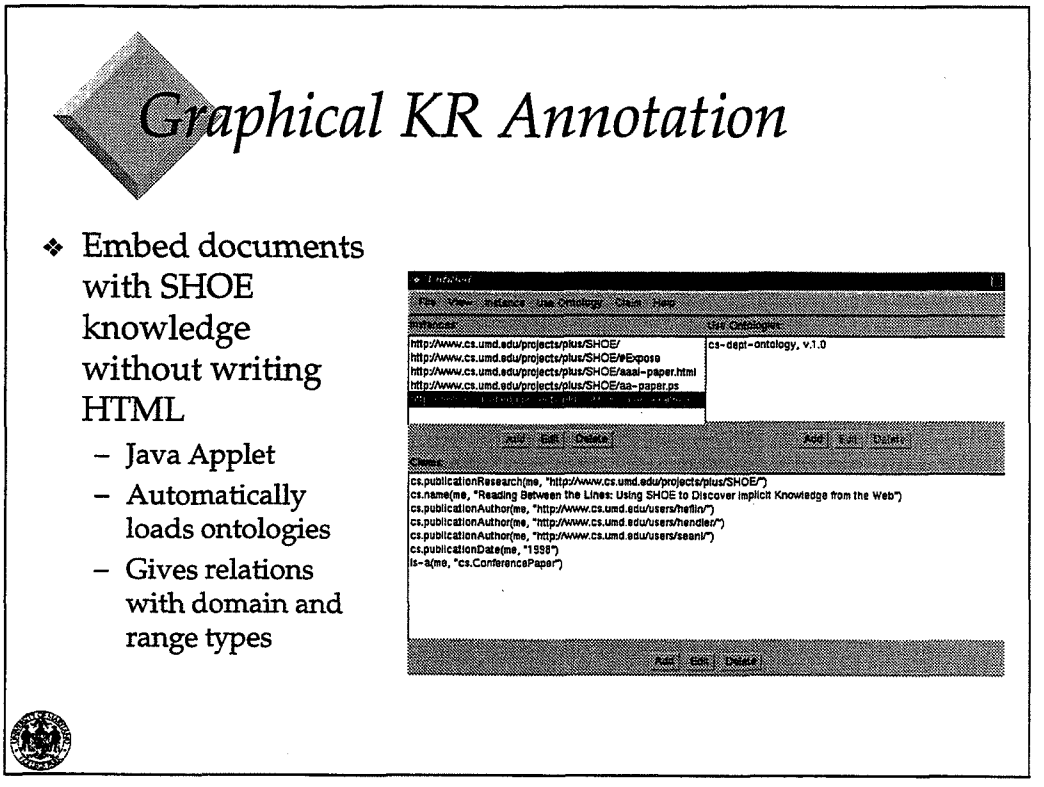
Using SHOE

Users' pages can be marked with SHOE annotations as defined in an ontology. In the example below, clearly a fictitious use, we examine a user named "Bill Clinton" who has added information about himself to the web in SHOE readable form. Later on the same page, information about "Roger Clinton" is added, including the fact that he is a relative of Bill Clinton.



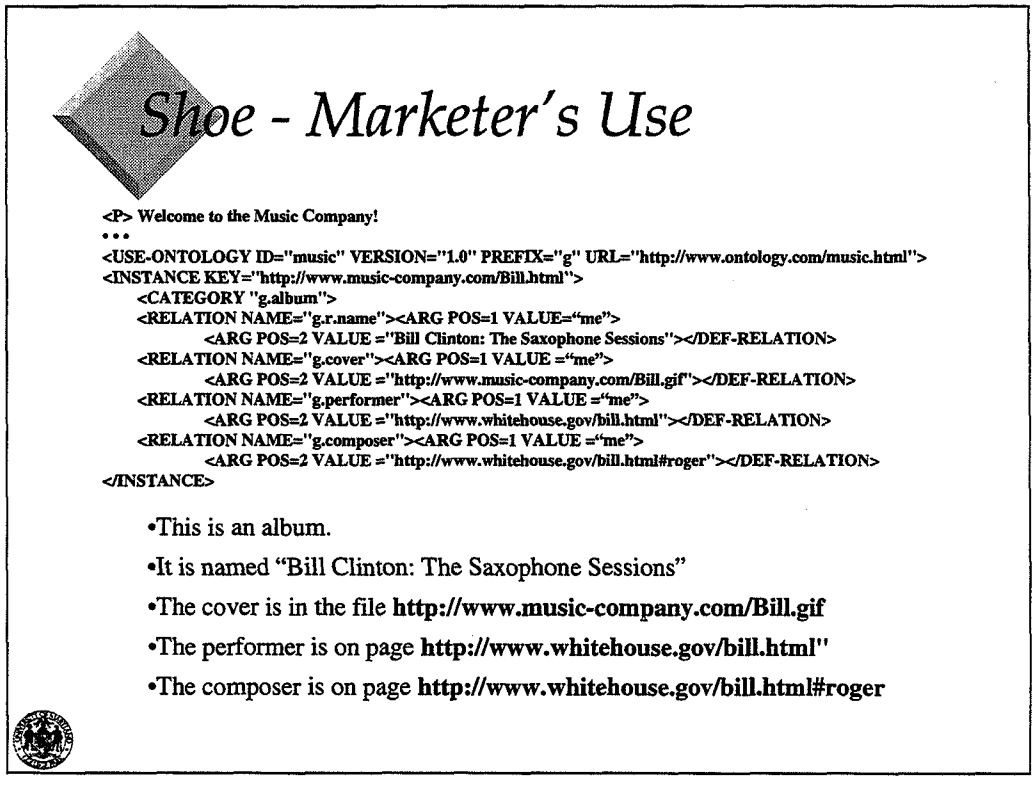
The Knowledge Annotator

As users would not generally be able to handle the complexity of the notations in SHOE, a special Java™ applet has been created to make it easier to do this markup. This applet, known as the knowledge annotator, is available from the SHOE web page.



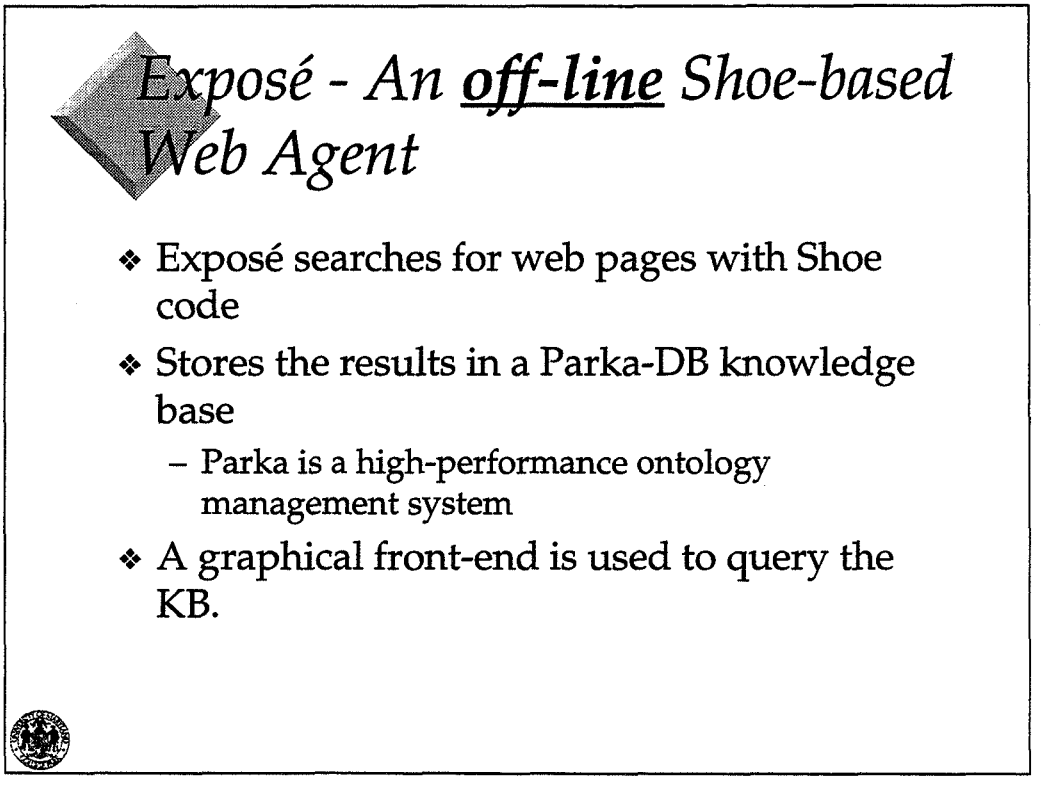
Further Use

Since SHOE is compliant with SGML, it is easily generated from a user's database in the same way that they can generate the HTML that users see. Thus, a marketing page can contain a fair amount of product information written in the SHOE markup form.



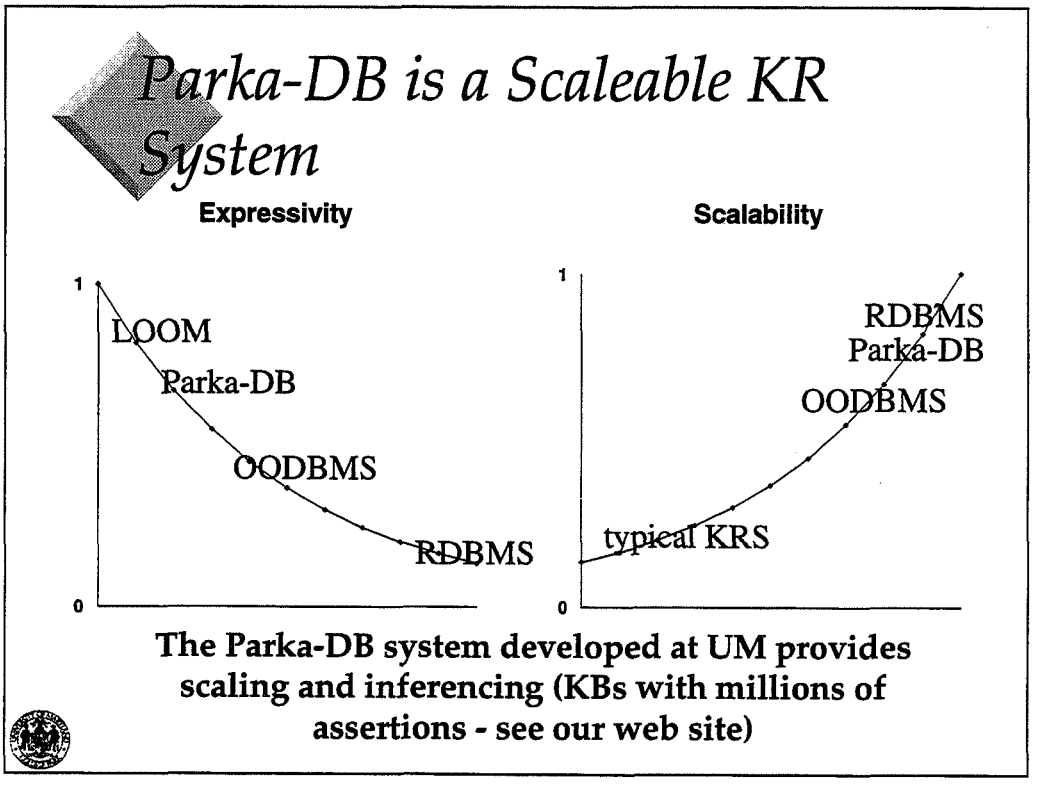
Exposé

Exposé is a web agent that searches the web for pages written with SHOE annotations. When one is found, Exposé imports this knowledge into a server-based version of Parka-DB for use by other tools.



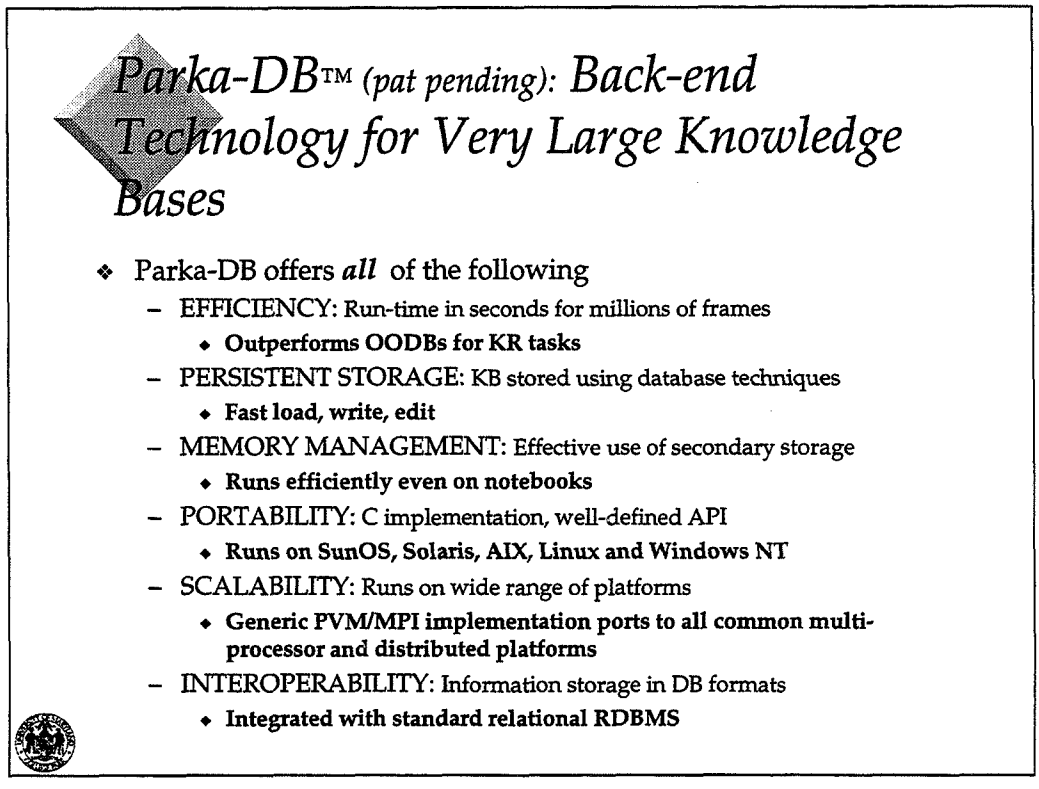
PARKA-DB™ (pat pending)

As mentioned earlier, Parka-DB is a scaleable system for keeping information in a form so that it is efficiently accessed (like a database) but with expressive power allowing ontological constructs and some forms of inferencing. Details of the Parka-DB system can be found on the web page http://www.cs.umd.edu/projects/plus/Parka.



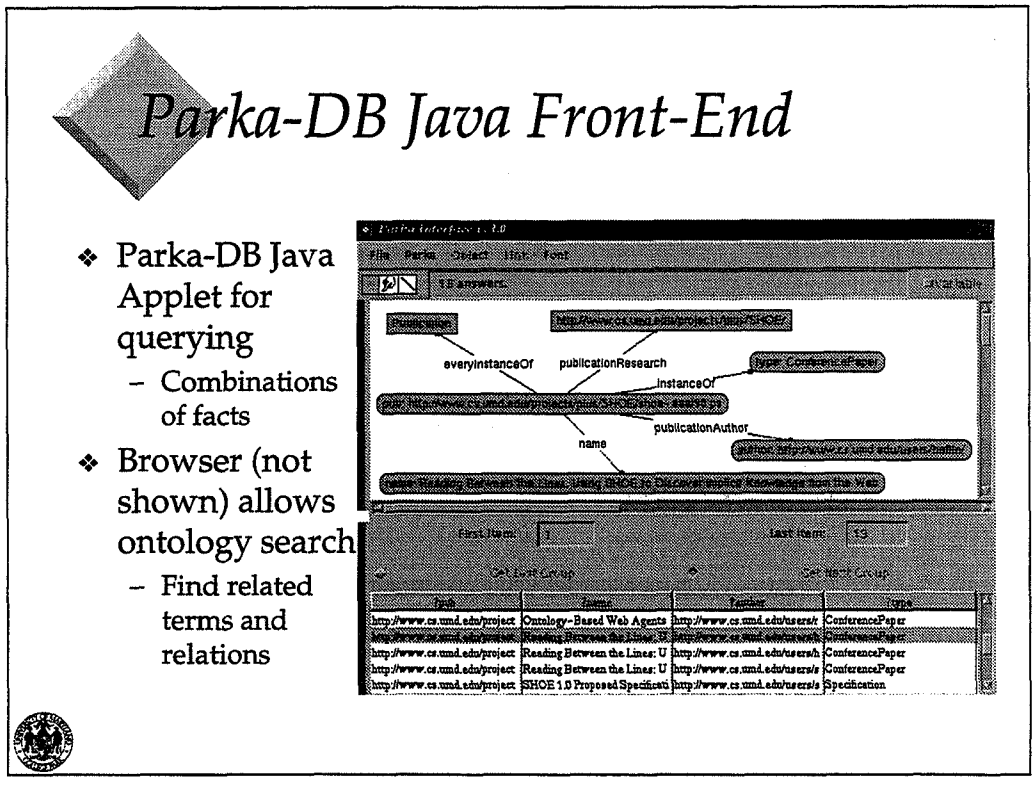
PARKA-DB Features

Parka-DB was developed at the University of Maryland over a period of about ten years. It provides unmatched capabilities for information storage combined with knowledge management. Using its unique combination of knowledge- and data-based capabilities, Parka-DB provides a wide range of capabilities not found in other knowledge representation systems.



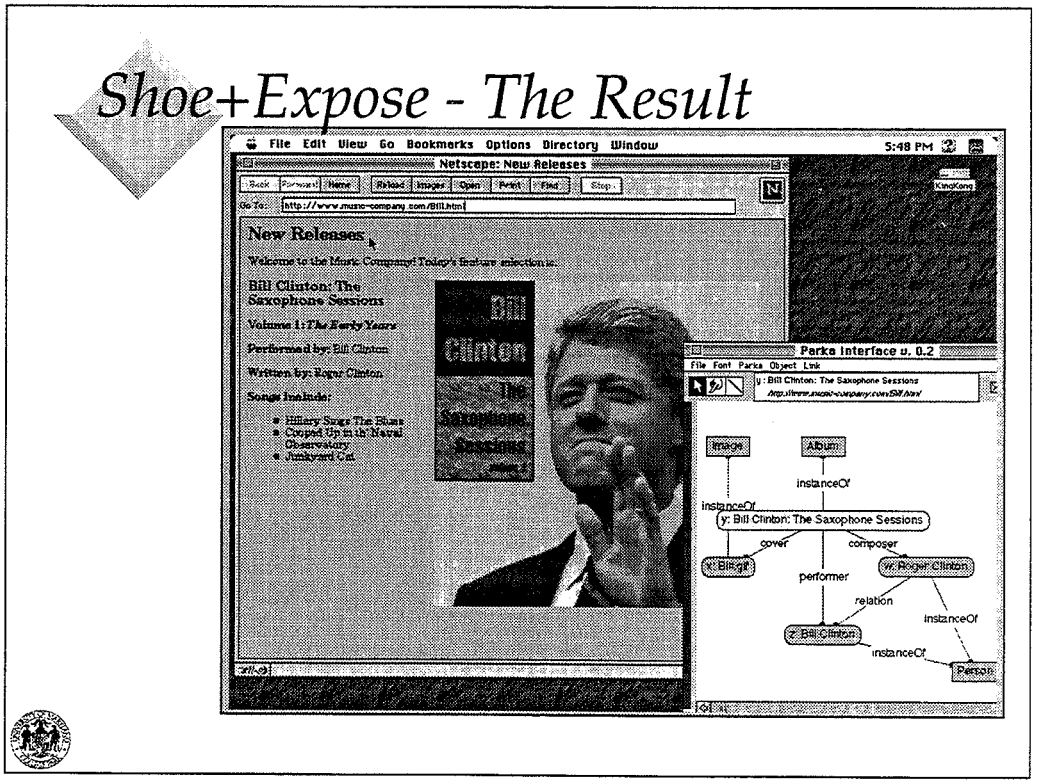
PARKA-DB Server

SHOE interacts with a server-based version of Parka-DB. Using a Java™ Applet, a user can query or browse information stored in the knowledge base. The figure below shows a typical query using the front-end. Parka-DB is also implemented in a modular fashion that allows remote queries using a simple API, allowing other front-ends to be built easily.



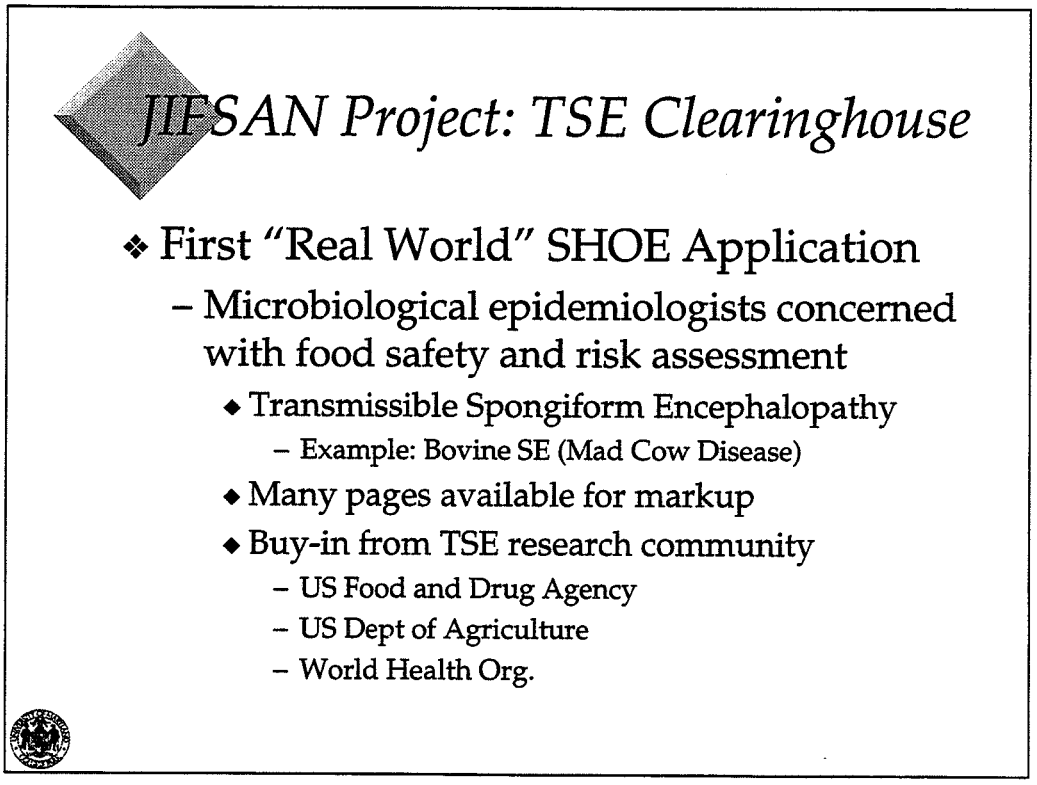
Example Concluded

Using all the tools described so far, the user is able to express a query that asks to see the cover of an album that is performed by one person, composed by a second, and these two people are relatives. Using the information from the music ontology, the (fictitious) Clinton home page, and the album vendors page, the query is answered with a search in the Parka-DB server and the page is displayed in the user's browser.



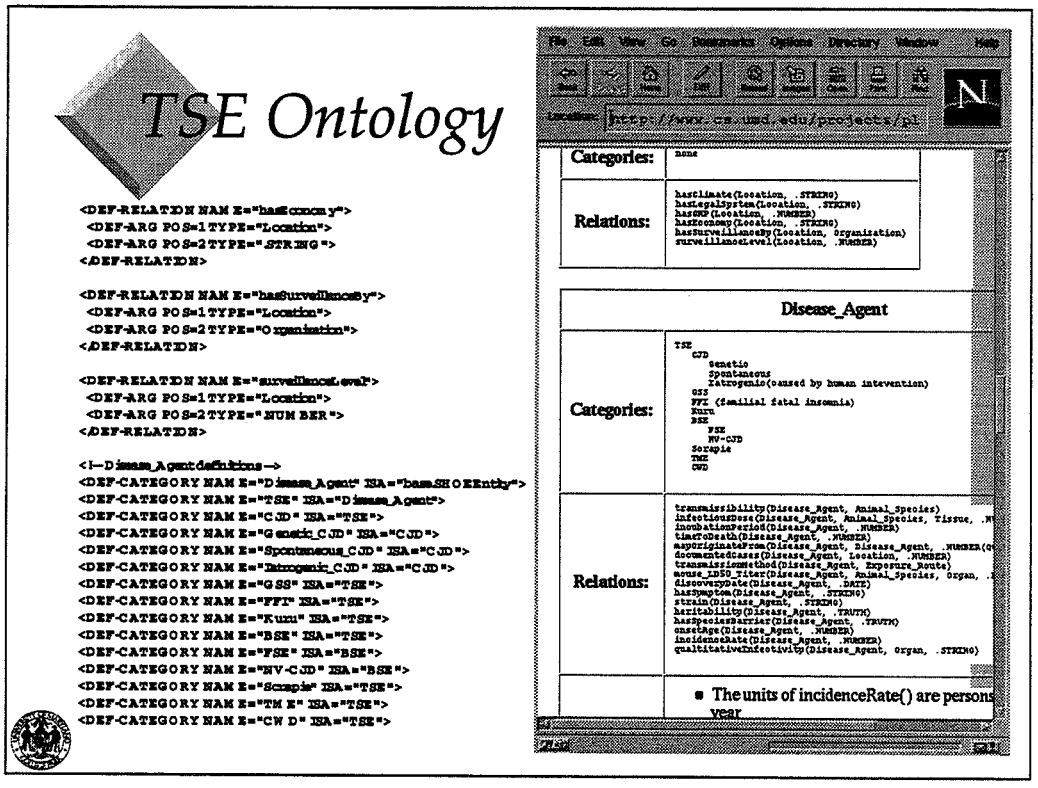
SHOE in Real Use

The first real use of SHOE is in a project we are performing jointly with the "Joint Institute for Food Safety and Nutrition" (JIFSAN). This project uses SHOE as part of a web project for providing information to regulators, scientists and the general public about the safety of foods. The currently used sample pages focus on "Transmissible Spongiform Encephalopathies," and particularly, the well-known "Mad Cow Disease" (Bovine Spongiform Encephalopathy).



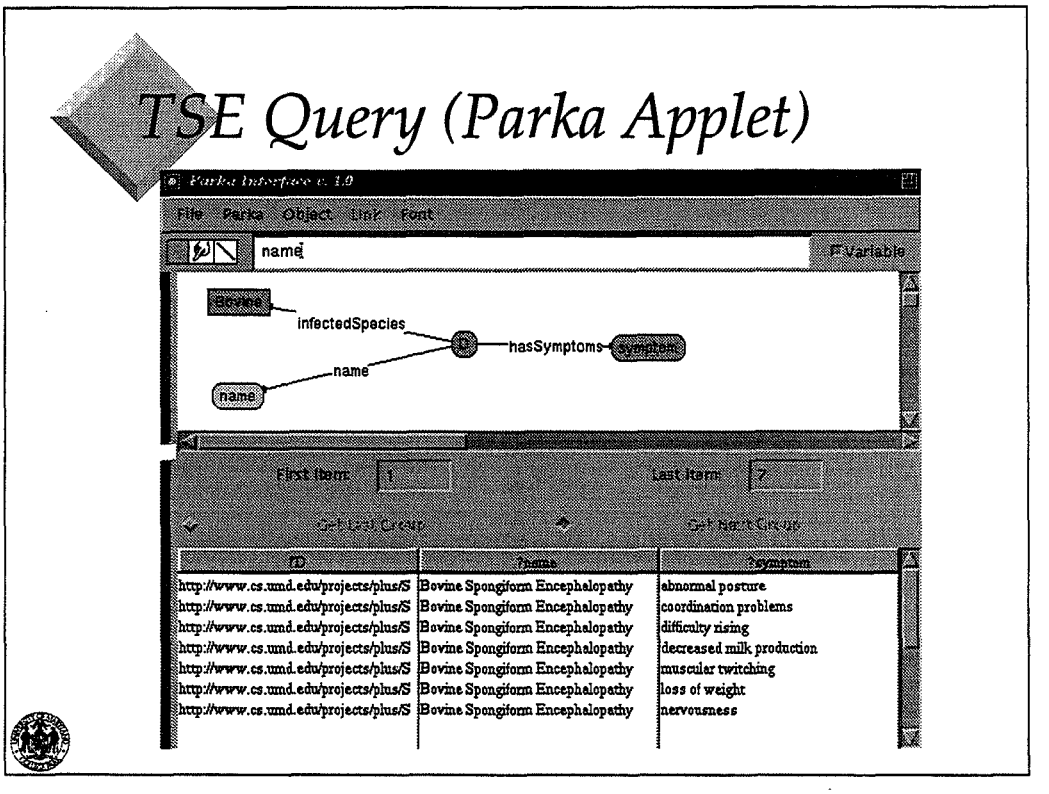
TSE Ontology

The figure below shows part of the TSE ontology as recorded in SHOE and as it appears in a human-readable form on the ontology page. This ontology was created by extending the root ontology described earlier.



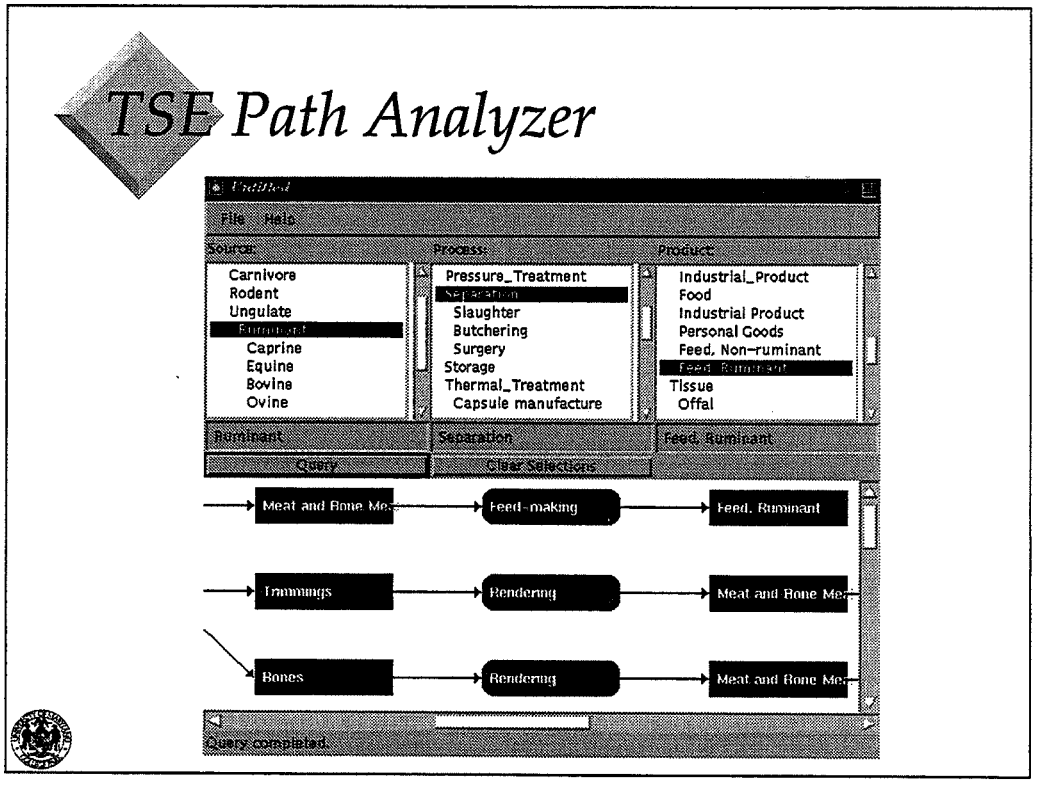
TSE Example

This is an example of a query that asks for the symptoms of the diseases that might effect bovine species. The query returns a list of the symptoms and the web pages to go to for more information on each. This list was built by use of the Exposé agent visiting annotated TSE pages and building these into the separate knowledge base. This is what the query looks like using the default Parka-DB front-end as described earlier.



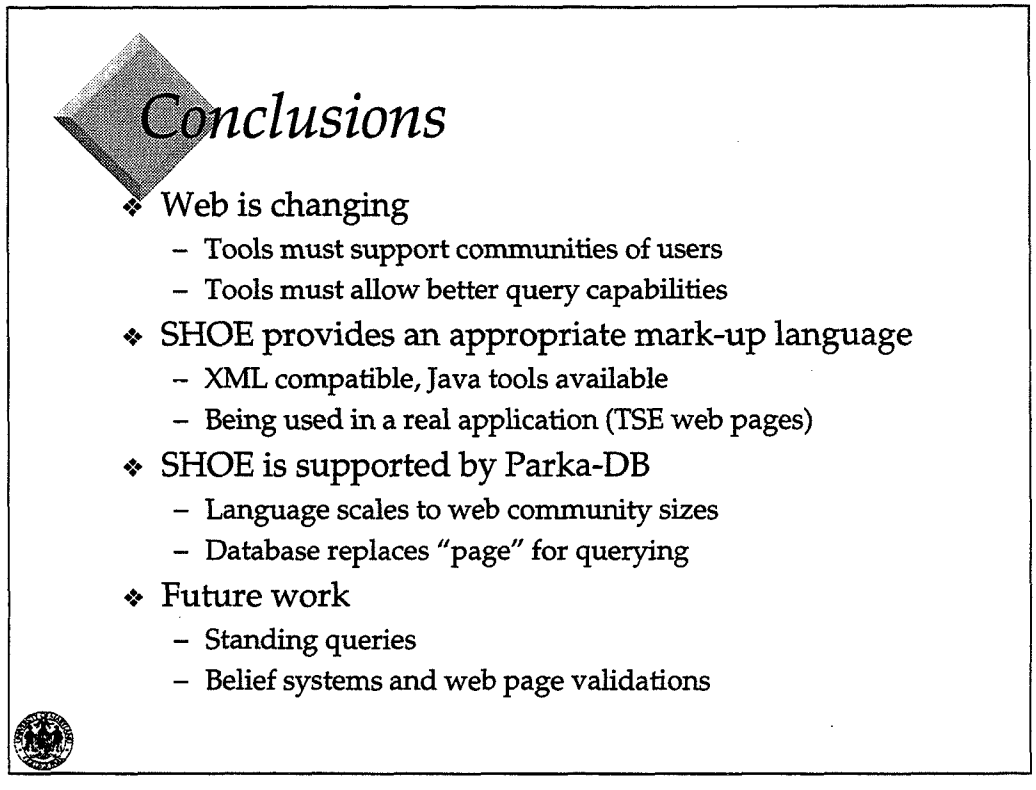
TSE Path Analyzer

As mentioned previously, other front-end applets can be easily built to query the Parka-DB server. The example below is from a "path analyzer" provided so that a regulator can examine how components from a particular species might end up in particular products. The analyzer makes a set of queries from Parka-DB and displays the results graphically. Clicking on any of the returned items will open a menu of web pages on which the particular item or process is described.



Conclusions

This presentation has shown how new trends in web use require tools with better querying capabilities. To support this, we have developed SHOE, an extension to HTML that allows users to define and use ontologies on web pages. SHOE uses a unique server-based knowledge representation language (Parka-DB™) to support querying. Parka-DB provides both scaling and inferencing capabilities. Future work (not described in this presentation) is focusing on combining Parka-DB and SHOE to provide a basis for better push technology using a standing query approach.



Page intentionally left blank

S/4-62
032 528
C2054
37079
14p

Why Surf Alone? Exploring the Web with Reconnaissance Agents

Henry Lieberman
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA

Page intentionally left blank

Why Surf Alone? Exploring the Web with Reconnaissance Agents

Henry Lieberman

MIT Media Lab

Cambridge, Massachusetts

Http://www.media.mit.edu/~lieber

Traditional Web Surfing

Either...

- *Browsing* [= Unconstrained link following]
  - Good news: People love to explore
  - Bad news: Tend to get "lost in hyperspace"
- Or *Searching* [= Precisely targeted retrieval]
  - Good news: Computers can search faster
  - Bad news: What if I don't know exactly what I want?

Browsing Should be a *Co-operative Activity* Between an Agent and the User

Let an agent produce browsing suggestions while the user is browsing.

Let each partner do what they do best

- The user is better at deciding which pages are interesting.
- The computer is faster at searching pages.

Agent performs search, but learns how to evaluate pages from the user.

Letizia: An Autonomous Interface Agent for Web Browsing

Letizia acts as an *advance scout* for Web browsing:

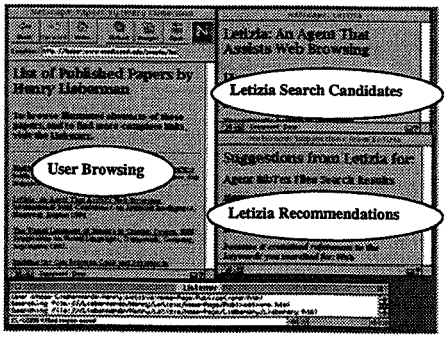
- It *watches your Web browsing* to try to learn what topics you are interested in.
- While you are reading a Web page, Letizia searches the *neighborhood* of the page to discover other pages you might be interested in.

Why the Name Letizia?

"Letizia Álvarez de Toledo has observed that this vast library is useless: rigorously speaking, a single volume would be sufficient, a volume of ordinary format, printed in nine or ten point type, containing an infinite number of infinitely thin leaves."

- Jorge Luis Borges, *The Library of Babel*

Letizia is a "Channel Surfing" Interface for the Web



Letizia is a *Reconnaissance* Agent

Browsing a link requires a "leap of faith"

- You don't know what's behind the link.
- Letizia's lookahead can try to anticipate whether you'll be interested in the link.

Breadth-First vs. Depth-First Search

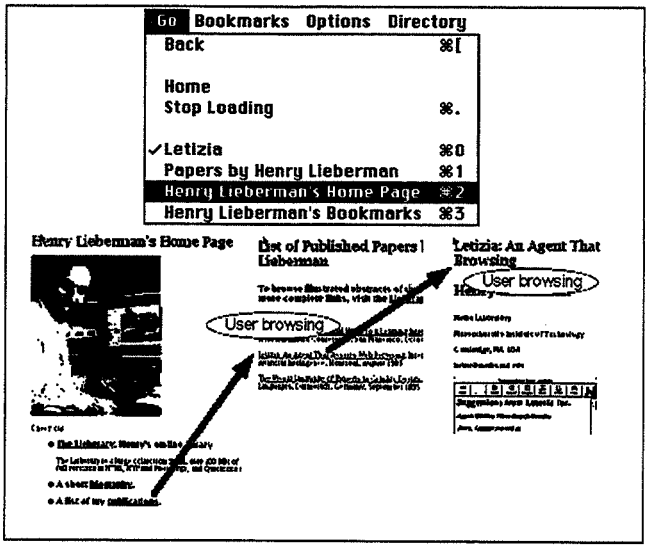
Web browsers encourage *depth-first* browsing strategy.

But most information of interest is not deep in Web.

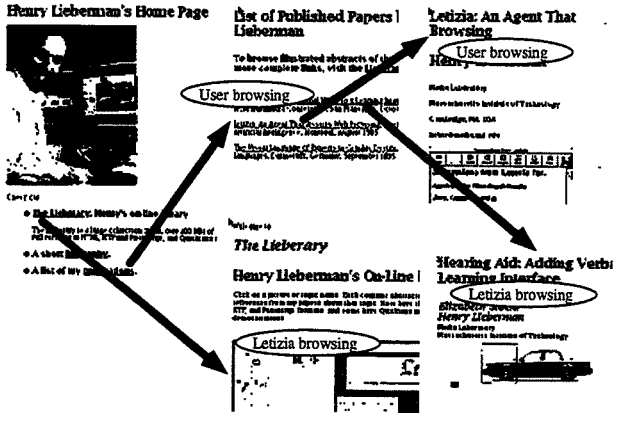
Letizia conducts a *breadth-first* search in parallel with the user's browsing activity.

User's browsing actions immediately refocus the search.

User's Search [Depth-First]



User's Search & Letizia's Search



Inferring User Intent From Browsing Behavior

Infers interests from content of document.

Applies weak heuristics based on keywords

- e.g. TFIDF

Would be better to use partial parsing techniques, knowledge-based content inference.

Inferring User Intent From Browsing Behavior

Inferences of interest from sequence of browsing events

- Selection of link indicates interest in containing document.
- "Passing over" links indicates disinterest.

User actions other than browsing

- Search, entering on hotlist, mailing, downloading, dwell time...

Persistence of Interest

Users tend to remain interested in a topic long after they have performed a search or browsed a document containing that topic.

It is too much trouble to restate interest at each opportunity.

Agent can play the role of maintaining persistence of interest.

Discovering *Serendipitous Connections*

Browsing tends to be "chunky"; you browse a set of related topics for a while, then switch to an unrelated topic.

If I browse for "Paris", then "Jazz", ...

then "Calendar of Events in Boston"

... agent can show "French Jazz Festival".

I could type all the topics into a search engine, but I won't.

Let's Browse

An agent to assist multiple users browsing together

- What does it mean for a group of people to browse together?
- How could an intelligent agent assist the browsing process?

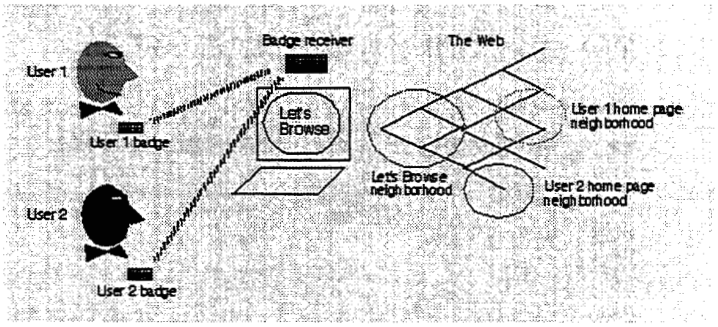
Assistance Provided by Let's Browse

Automatic detection of the presence of users.

Automated "channel surfing" browsing.

Dynamic display of profiles, recommendations.

Let's Browse Setup



User Profiles

In advance, Web crawler scanned a breadth-first neighborhood surrounding the user's home page.

Organization's page used if home page not found.

TFIDF keyword frequency analysis.

Weighted list of keywords.

Can also compute profile dynamically.

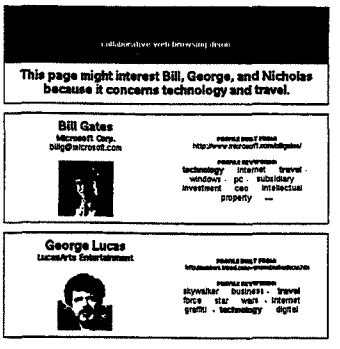
Discovering Common Interests

Intersections of user profiles.

Changes dynamically as users come and go.

Result: Needed to keep lower-frequency terms to ensure non-empty intersections of interests.

Visualization of the Recommendation Process



Design Principles for Autonomous Interface Agents

If you're only trying to make suggestions, each decision isn't so critical.

There's a tradeoff between deliberation and action.

Take advantage of information that the user gives the agent "for free".

Take advantage of the user's "think time".

The user's attention may be time-shared.

Autonomous interface agents may fit the cognitive styles of some users, but not others.

Letizia Shows How Intelligent Agents Can Improve Interfaces

Letizia *works with* an interactive interface

- It doesn't try to *replace* interaction.

Letizia *learns* from interaction with the user

- It gets better over time.

Letizia *anticipates* the user's needs and interests

- It works proactively, while the user may be doing something else.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 07704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1999	Conference Publication

4. TITLE AND SUBTITLE
Intelligent Agents and Their Potential for Future Design and Synthesis Environment

5. FUNDING NUMBERS
WU 282-10-01-42

6. AUTHOR(S)
Ahmed K. Noor and John B. Malone, Compilers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17802

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/CP-1999-208986

11. SUPPLEMENTARY NOTES
Ahmed K. Noor: University of Virginia Center for Advanced Computational Technology, Hampton, VA;
John B. Malone: Langley Research Center, Hampton, VA

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 61 Distribution: Standard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

This document contains the proceedings of the Workshop on Intelligent Agents and Their Potential for Future Design and Synthesis Environment, held at NASA Langley Research Center, Hampton, VA, September 16-17, 1998. The workshop was jointly sponsored by the University of Virginia's Center for Advanced Computational Technology and NASA. Workshop attendees came from NASA, industry and universities. The objectives of the workshop were to assess the status of intelligent agents technology and to identify the potential of software agents for use in future design and synthesis environment. The presentations covered the current status of agent technology and several applications of intelligent software agents.

Certain materials and products are identified in this publication in order to specify adequately the materials and products that were investigated in the research effort. In no case does such identification imply recommendation or endorsement of products by NASA, nor does it imply that the materials and products are the only ones or the best ones available for this purpose. In many cases equivalent materials and products are available and would probably produce equivalent results.

14. SUBJECT TERMS
Engineering education; Instructional technology; Advanced training; Curriculum reform

15. NUMBER OF PAGES
295

16. PRICE CODE
A13

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL