Inteligentní softwarové prostředky,

softwaroví agenti

13. 12. 2002

Inteligentní softwarové prostředky aneb inteligentní softwaroví agenti

Definition: Agent

Agent:
One that is authorized to act for another. Agents possess the characteristics of *delegacy*, *competency*, and *amenability*.

Delegacy:
Discretionary authority to autonomously act on behalf of the client. Actions include making decisions, committing resources, and performing tasks.

Competency:
The capability to effectively manipulate the problem domain environment to accomplish the prerequisite tasks. Competency includes specialized communication proficiency.

Amenability:
The ability to adapt behavior to optimize performance in an often non-stationary environment in responsive pursuit of the goals of the client. Amenability may be combined with accountability.

Examples of human agents include booking agents, sales agents, and politicians.

Definition: Software Agent

Software Agent:
An artificial agent which operates in a software environment.

Software environments include operating systems, computer applications, databases, networks, and virtual domains.

Delegacy for software agents centers on persistence. "Fire-and-forget" software agents stay resident, or persistent, as background processes after being launched. By making decisions and acting on their environment independently, software agents reduce human workload by generally only interacting with their end-clients when it is time to deliver results. Additionally, autonomous automation can lead to super-human performance in terms of volume and speed.

Competency within a software environment requires knowledge of the specific communication protocols of the domain. Protocols such as SQL for databases, HTTP for the WWW, and API calls for operating systems must be preprogrammed into the software agents, limiting their useful range.

Amenability for non-intelligent software agents is generally limited to providing control options and the generation of status reports that require human review. Such agents often tend to be brittle in the face of a changing environment, necessitating a modification of their programming to restore performance.

Definition: Intelligent Software Agent

Intelligent Software Agent (ISA):
A software agent that uses Artificial Intelligence (AI) in the pursuit of the goals of its clients.

Artificial Intelligence is the imitation of human intelligence by mechanical means. Clients, then, can reduce human workload by delegating to ISAs tasks that normally would require human-like intelligence.

Many researchers that formerly referred to their work as AI are now actively engaged in "agent technology". Thus the word "agent" by itself generally connotes ISAs in the terms of the present-day research community.

Delegacy for ISAs is far more absolute. ISAs have the capability to generate and implement novel rules of behavior which human beings may never have the opportunity or desire to review. As ISAs can engage in extensive logical planning and inferencing, the relationship of trust between the client and the agent is or must be far greater, especially when the consumption of client resources is committed for reasons unexplained or multiple complex operations are actuated before human observers can react.

Competency as practiced by ISAs adds higher order functionality to the mix of capabilities. In addition to communicating with their environment to collect data and actuate changes, ISAs can often analyze the information to find non-obvious or hidden patterns, extracting knowledge from raw

data. Environmental modes of interaction are richer, incorporating the media of humans such as natural language text, speech, and vision.

Amenability in ISAs can include self-monitoring of achievement toward client goals combined with continuous, online learning to improve performance. Adaptive mechanisms in ISAs mean that they are far less brittle to changes in environment and may actually improve. In addition, client responsiveness may go so far as to infer what a client wants when the client himself does not know or cannot adequately express the desired goals in definitive terms.

Agent Variants

Mobile Agents:
Also known as traveling agents, these programs will shuttle their being, code and state, among resources. This often improves performance by moving the agents to where the data reside instead of moving the data to where the agents reside. The alternative typical operation involves a client-server model. In this case, the agent, in the role of the client, requests that the server transmit volumes of data back to the agent to be analyzed. Oftentimes the data must be returned by the agent to the server in a processed form. Significant bandwidth performance improvements can be achieved by running the agents within the same chassis as the data. Mobile agent frameworks are currently rare, however, due to the high level of trust required to accept a foreign agent onto one's data server. With advances in technologies for accountability and immunity, mobile agent systems are expected to become more popular.

Distributed Agents:
Load-balancing can be achieved by distributing agents over a finite number of computational resources. Some mobile agents are self-distributing, seeking and moving to agent platforms that can offer the higher computational resources at lower costs.

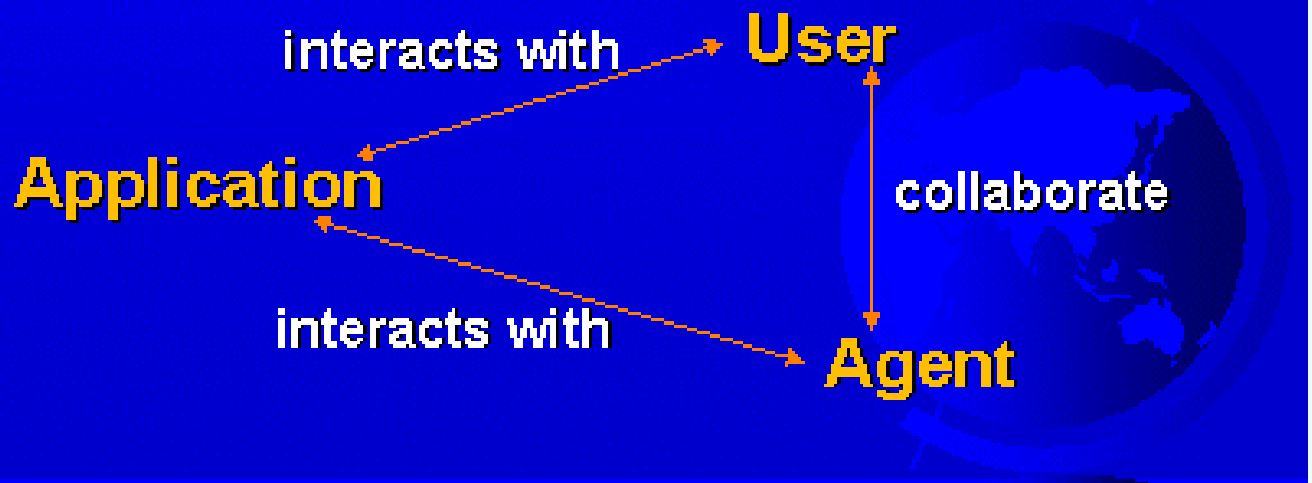
Multiple Agents:
Some tasks can be broken into sub-tasks to be performed independently by specialized agents. Such agents are unaware of the existence of the others but nonetheless rely upon the successful operations of all.

Collaborative Agents:
Collaborative agents interact with each other to share information or barter for specialized services to effect a deliberate synergism. While each agent may uniquely speak the protocol of a particular operating environment, they generally share a common interface language which enables them to request specialized services from their brethren as required.

Social Agents:
Anthropomorphism is seen by some researchers as a key requirement to successful collaboration between humans and agents. To this end, agents are being developed which can both present themselves as human-like creations as well as interpret human-generated communications such as continuous speech, gestures, and facial expressions.

What is a Software Agent?

"Software Agent" — particular type of agent, inhabiting computers & networks, assisting users with computer-based tasks.



Pattie Maes --- MIT Media Laboratory 8 Software Agents Tutorial --- 97

How is an Agent different from other Software?

- personalized, customized
- pro-active, takes initiative
- long-lived, autonomous
- adaptive

Why do we need Software Agents?

- more everyday tasks are computer-based
- vast amounts of dynamic, unstructured information
- more users, untrained

Why do we need Software Agents?

Change of Metaphor for HCI:

old: direct manipulation

new: indirect management

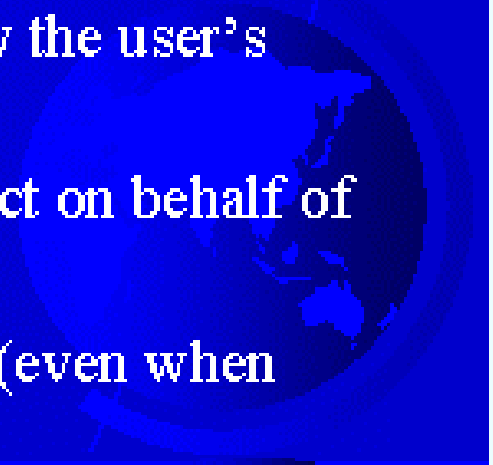
Pattie Maes --- MIT Media Laboratory 11 Software Agents Tutorial --- 97

Direct Manipulation

- task for which it was designed:
  - closed, static, relatively small and structured information world
- method used:
  - visualize the objects
  - actions on objects in interface correspond to actions on real objects
  - nothing happens unless the user makes it happen

Indirect Management/Agents

- task for which it is designed
  - open, dynamic, vast and unstructured information world
- method used
  - user delegates to agents that know the user's interests, habits, preferences
  - agents make suggestions and/or act on behalf of user
  - lots of things happen all the time (even when user is not active)



Pattie Maes --- MIT Media Laboratory 13 Software Agents Tutorial --- 97

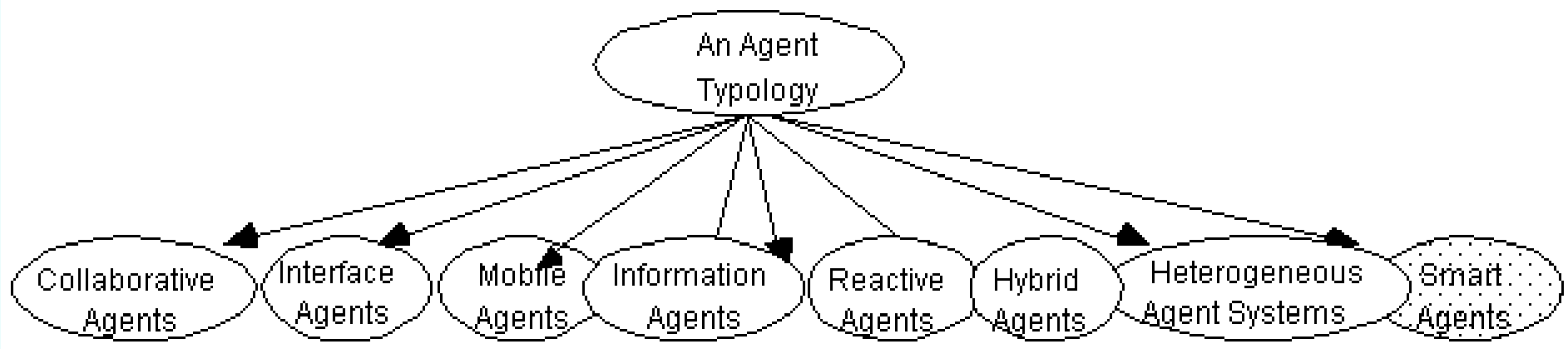
Criticisms of Software Agents (Lanier, Schneiderman)

1. Well-designed interfaces always better.
   I may not want to do the task myself
2. Agents make the user dumb.
   I may not need to learn about this task
3. Agents will never be smart.
   don't need to be; get their smarts from users
4. Agents do not exist (yet)
   agents are here to stay

Pattie Maes --- MIT Media Laboratory 14 Software Agents Tutorial --- 97

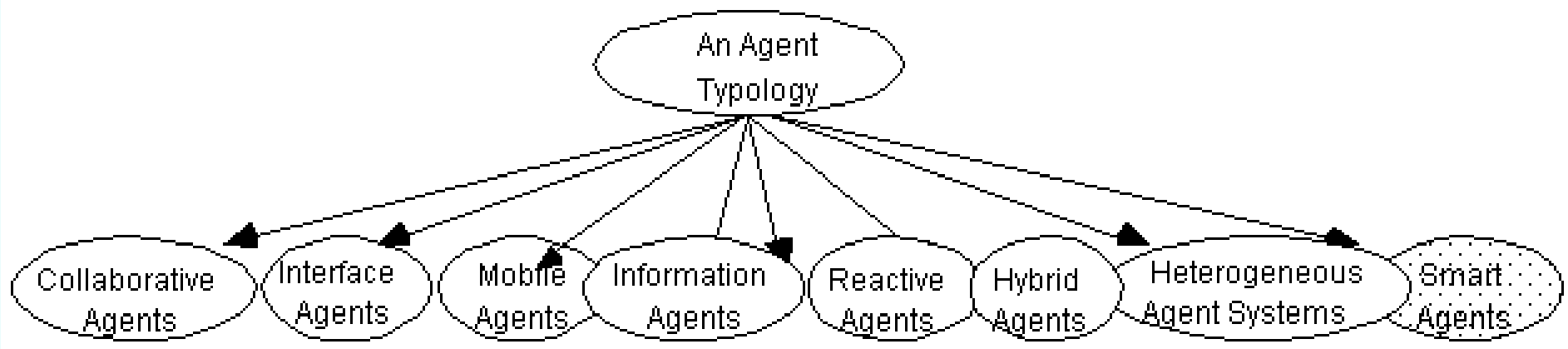
A Panoramic Overview of the Different Agent Types

A Classification of Software Agents:



A Panoramic Overview of the Different Agent Types

A Classification of Software Agents:



Collaborative Agents

emphasise autonomy and cooperation (with other agents) in order to perform tasks for their owners. They may learn, but this aspect is not typically a major emphasis of their operation. In order to have a coordinated set up of collaborative agents, they may have to *negotiate* in order to reach mutually acceptable agreements on some matters.

The *motivation* for having collaborative agent systems may include one or several of the following:

- To solve problems that are too large for a centralised single agent to do due to resource limitations or the sheer risk of having one centralised system;
- To allow for the interconnecting and interoperation of multiple existing legacy systems, e.g. expert systems, decision support systems, etc.;
- To provide solutions to inherently distributed problems, e.g. distributed sensor networks (cf. DVMT, Durfee *et al.,* 1987) or air-traffic control;
- To provide solutions which draw from distributed information sources, e.g. for distributed on-line information sources, it is natural to adopt a distributed and collaborative agent approach;
- To provide solutions where the expertise is distributed, e.g. in health care provisioning;
- To enhance modularity (which reduces complexity), speed (due to parallelism), reliability (due to redundancy), flexibility (i.e. new tasks are composed more easily from the more modular organisation) and reusability at the knowledge level (hence shareability of resources);
- To research into other issues, e.g. understanding interactions among human societies.

Interface Agents

Interface agents emphasise autonomy and learning in order to perform tasks for their owners. Pattie Maes, a key proponent of this class of agents, points out that the key metaphor underlying interface agents is that of a *personal assistant* who is *collaborating with the user* in the same work environment.

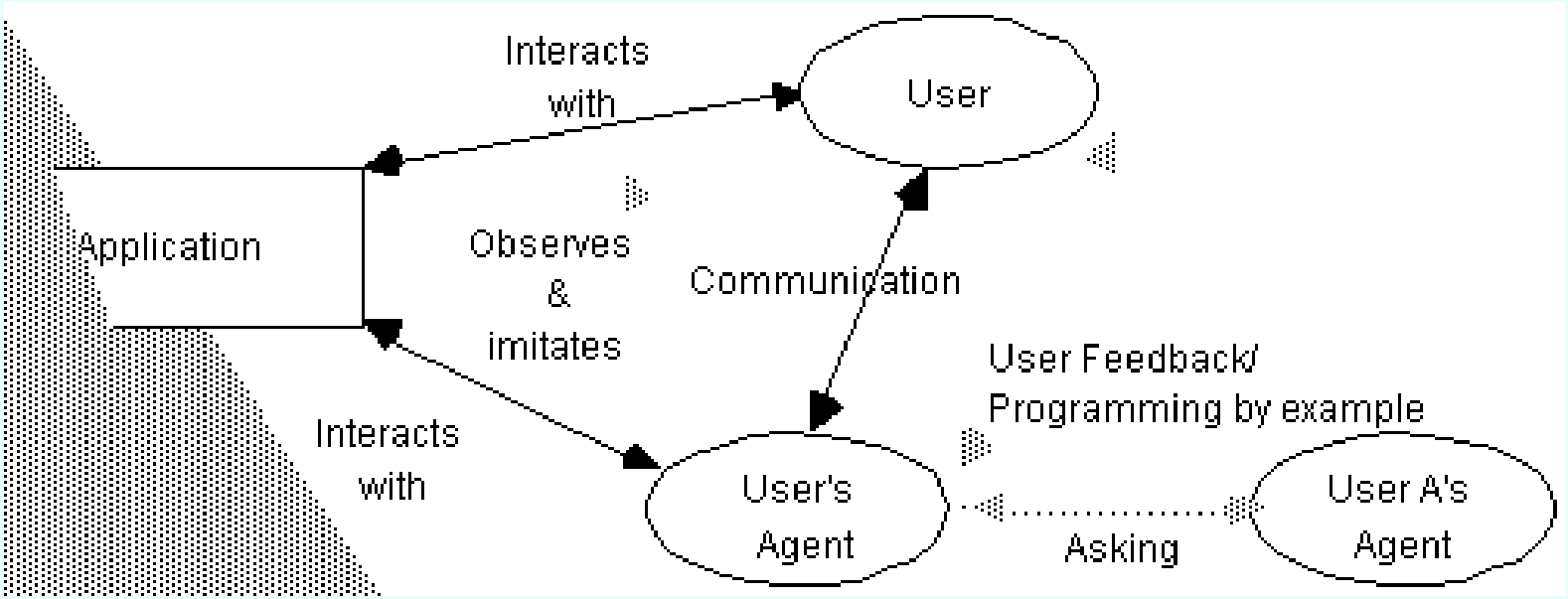
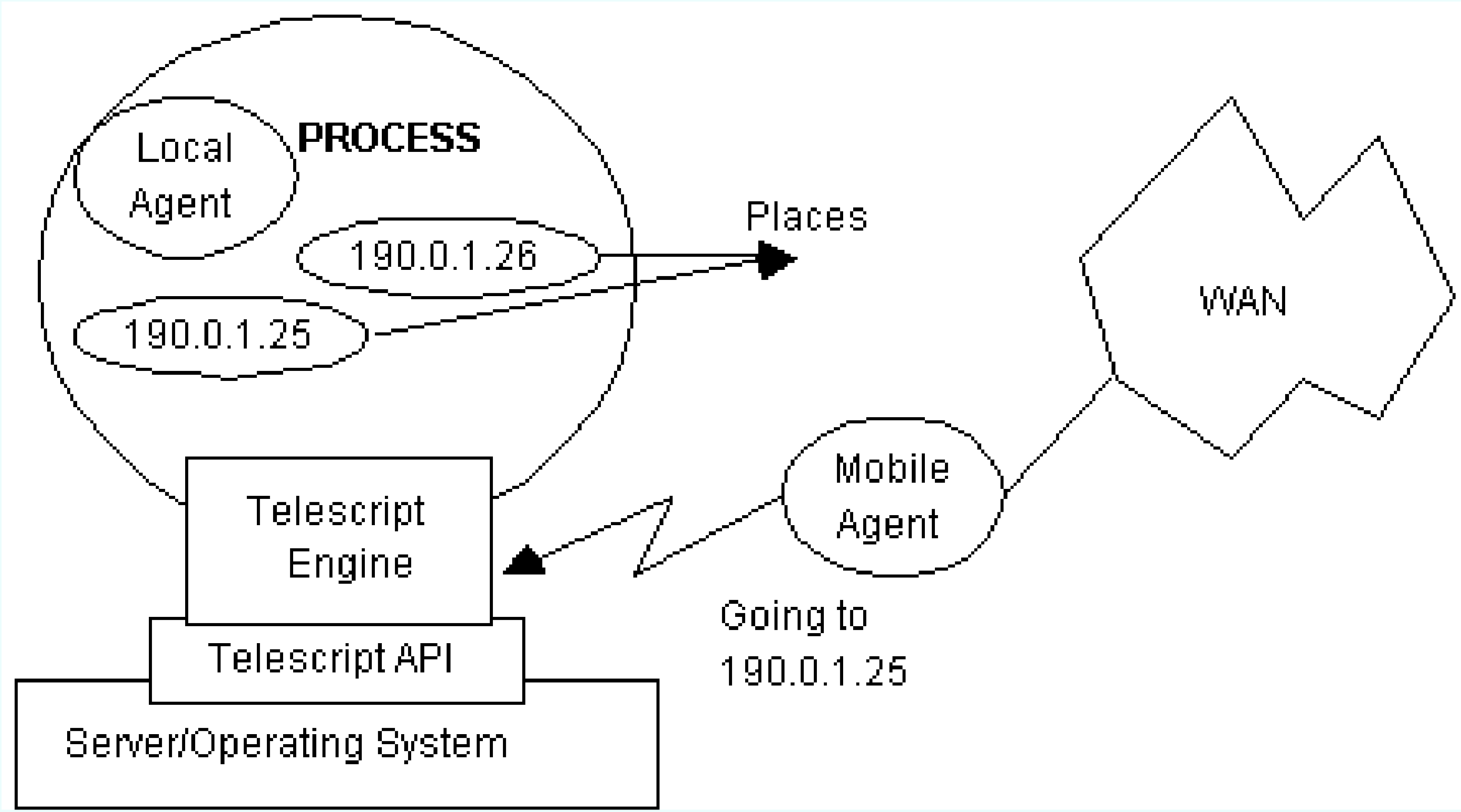


Figure depicts the functioning of interface agents. Essentially, interface agents support and provide assistance, typically to a user learning to use a particular application such as a spreadsheet or an operating system.

Mobile Agents

Mobile agents are computational software processes capable of roaming wide area networks (WANs) such as the WWW, interacting with foreign hosts, gathering information on behalf of its owner and coming **back home** having performed the duties set by its user. These duties may range from a flight reservation to managing a telecommunications network. However, *mobility is neither a necessary nor sufficient condition for agenthood*. Mobile agents are agents because they are autonomous and they cooperate, albeit differently to collaborative agents. For example, they may cooperate or communicate by one agent making the location of some of its internal objects and methods known to other agents. By doing this, an agent exchanges data or information with other agents without necessarily giving all its information away.

The key *hypothesis* underlying mobile agents is that agents need *not* be stationary; indeed, the idea is that there are significant benefits to be accrued, in certain applications, by eschewing static agents in favour of their mobile counterparts. These benefits are largely *non-functional*, i.e. we could do without mobile agents, and only have static ones but the costs of such a move are high.



Example - A Part View of Telescript Architecture

Information/Internet Agents

Information agents have come about because of the sheer demand for tools to help us manage the explosive growth of information we are experiencing currently, and which we will continue to experience henceforth. Information agents perform the role of managing, manipulating or collating information from many distributed sources.

The underlying *hypothesis* of information agents is that, somehow, they can ameliorate, but certainly not eliminate, this specific problem of information overload and the general issue of information management in this information era.

The *motivation* for developing information/internet agents:

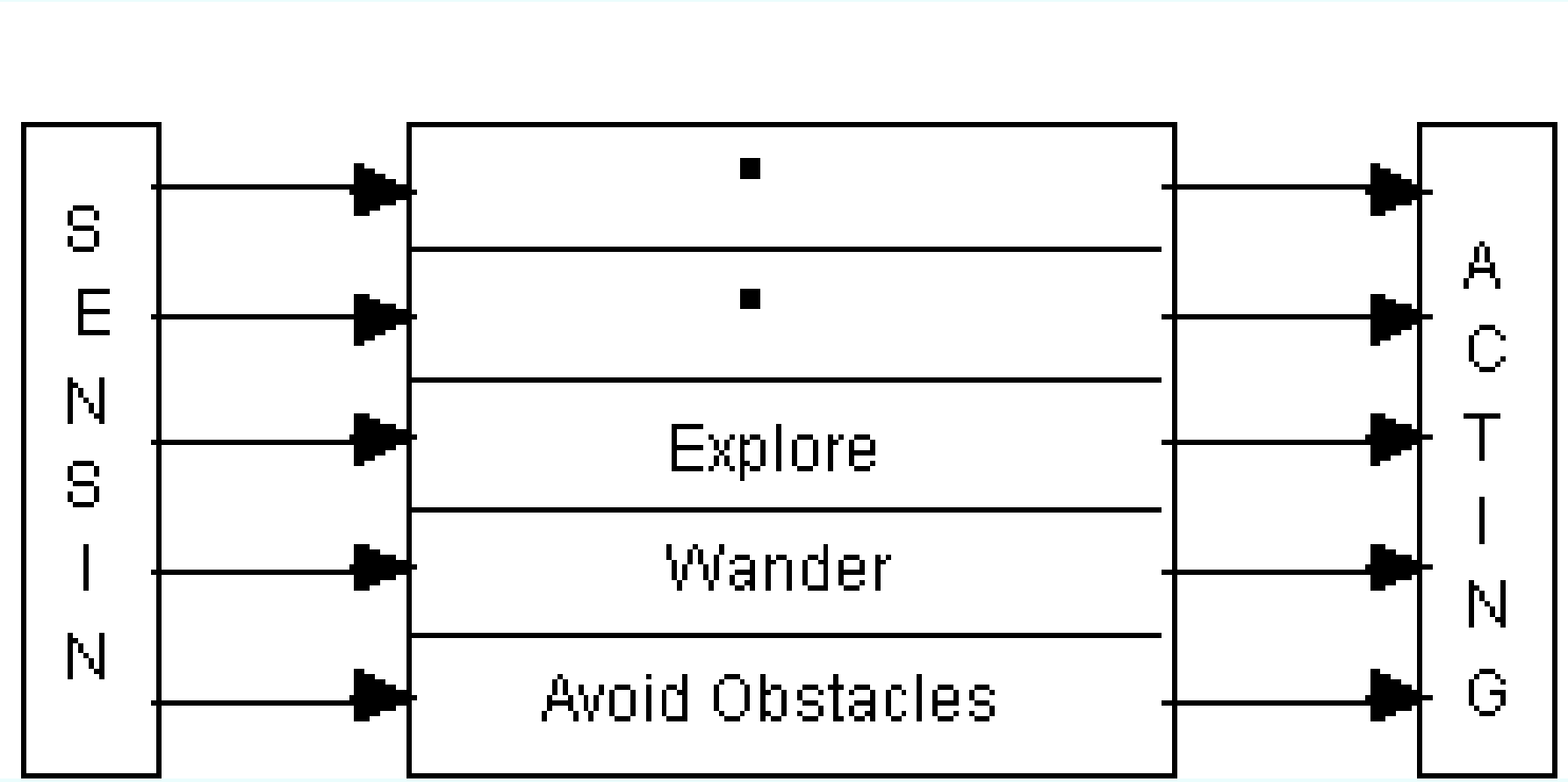
Firstly, there is simply a yearning need/demand for tools to manage such information explosion. Everyone on the WWW would benefit from them in just the same way as they are benefiting from search facilitators such as Spiders, Lycos or Webcrawlers. As Bob Johnson, an analyst at Dataquest Inc., notes: *"in the future, it [agents] is going to be the only way to search the Internet, because no matter how much better the Internet may be organised, it can't keep pace with the growth in information ...".*

Secondly, there are vast financial benefits to be gained. Recall that Netscape Corporation grew from relative obscurity to a billion dollar company almost overnight – and a Netscape or Mosaic client offers generally browsing capabilities, albeit with a few add-ons. Whoever builds the next killer application – the first usable Netscape equivalent of a proactive, dynamic, adaptive and cooperative agent-based WWW information manager – is certain to reap enormous financial rewards.

Reactive Software Agents

Reactive agents represent a special category of agents which do *not* possess internal, symbolic models of their environments; instead they act/respond in a stimulus-response manner to the present state of the environment in which they are embedded. Reactive agents work dates back to research such as Brooks (1986) and Agre & Chapman (1987), but many theories, architectures and languages for these sorts of agents have been developed since. However, a most important point of note with reactive agents are not these (i.e. languages, theories or architectures), but the fact that the agents are relatively simple and they interact with other agents in basic ways. Nevertheless, complex patterns of behaviour *emerge* from these interactions when the ensemble of agents is viewed globally.

The essential *hypothesis* of reactive agent-based systems is a specification of the *physical grounding hypothesis*, not to be confused with the *physical symbol system hypothesis*. Traditional AI has staked most of its bets on the latter which holds that the necessary and sufficient condition for a physical system to demonstrate intelligent action is that it be a physical symbol system. On the contrary, the physical grounding hypothesis challenges this long-held view arguing it is flawed fundamentally, and that it imposes severe limitations on symbolic AI-based systems.



Brook's Subsumption Architecture

Hybrid Agents

The key *hypothesis* for having hybrid agents or architectures is the belief that, for some application, the benefits accrued from having the combination of philosophies within a singular agent is greater than the gains obtained from the same agent based entirely on a singular philosophy. Otherwise having a hybrid agent or architecture is meaningless. Clearly, the *motivation* is the expectation that this hypothesis would be proved right; the ideal *benefits* would be the set union of the benefits of the individual philosophies in the hybrid.

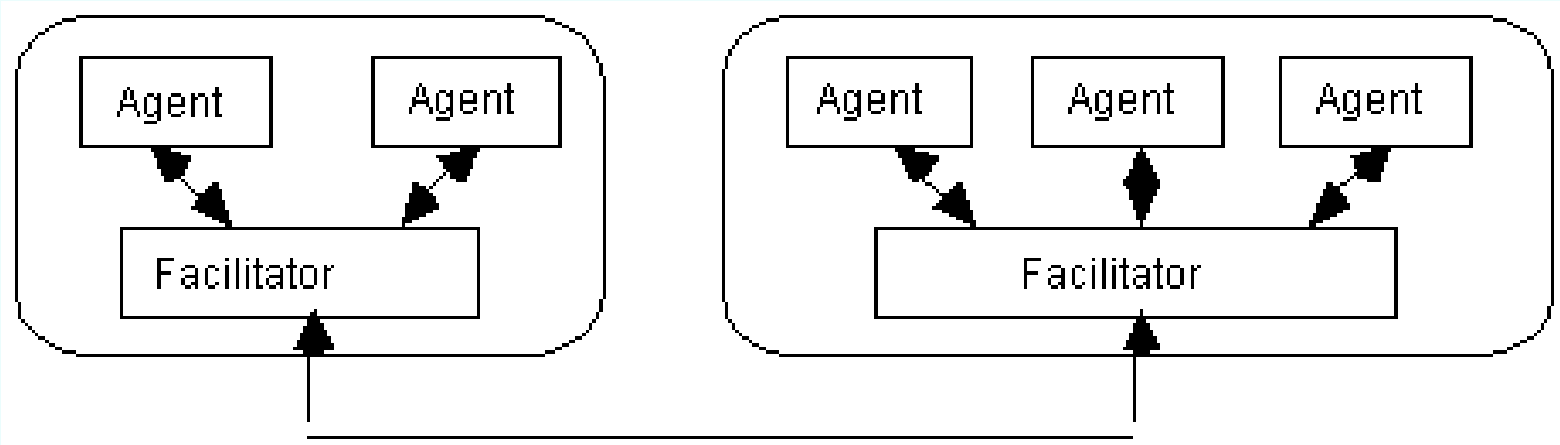
Consider the obvious case of constructing an agent based on both the collaborative (i.e. deliberative) and reactive philosophies. In such a case the reactive component, which would take precedence over the deliberative one, brings about the following benefits: robustness, faster response times and adaptability. The frame problem is also better ameliorated by the reactive component.

Heterogeneous Agent Systems

Heterogeneous agent systems, unlike hybrid systems described in the preceding section, refers to an integrated set-up of at least two or more agents which belong to two or more different agent classes. A heterogeneous agent system may also contain one or more hybrid agents.

The potential *benefits* for having heterogeneous agent technology are several:

- Standalone applications can be made to provide **value-added** services by enhancing them in order to participate and interoperate in cooperative heterogeneous set-ups;
- The legacy software problem may be ameliorated because it could obviate the need for costly software rewrites as they be given new leases of life by getting them to interoperate with other systems. Heterogeneous agent technology may cushion or lessen the blow or effect of routine software maintenance, upgrade or rewrites;
- Agent-based software engineering provides a radical new approach to software design, implementation and maintenance in general, and software interoperability in particular. Its ramifications (e.g. moving from passive modules in traditional software engineering to proactive agent-controlled ones) would only be clear as this methodology and its tools become clearer.



A Federated System

Genesereth & Ketchpel (1994) note that agent-based software engineering is often compared to object-oriented programming in that an agent, like an object, provides a message-based interface to its internal data structures and algorithms. However, they note that there is a key distinction: in object-oriented programming, the meaning of a message may differ from object to object (this is the principle of polymorphism); in agent-based software engineering, agents use a common language with an agent-independent semantics. They highlight three important questions raised by the new agent-oriented software engineering paradigm. They include:

- **What is an appropriate agent communication language?**
- **How are agents capable of communicating in this language constructed?**
- **What communication architectures are conducive to cooperation?**

Application areas:

- Logical Inferencing and Deduction
- Contextual Domain Knowledge
- Pattern Recognition
- Learning and Adaptivity
- Data Collection and Filtering
- Event Notification
- Data Presentation
- Planning and Optimization
- Rapid Response Implementation

Vlastnosti softwarových agentů

An agent is a software entity with (some of) the following chatacteristics:

- ongoing execution
- environmental awareness – reactivity
  – proactiveness
- agent awareness
- autonomy
- adaptiveness
- mobility
- anthropomorphism
- reproduceness

Inteligentní softwarové prostředky

Agents typically possess several (or all) of the following characteristics; they are:

- autonomous
- adaptive/learning
- mobile
- persistent
- goal oriented
- communicative/collaborative
- flexible
- active/proactive

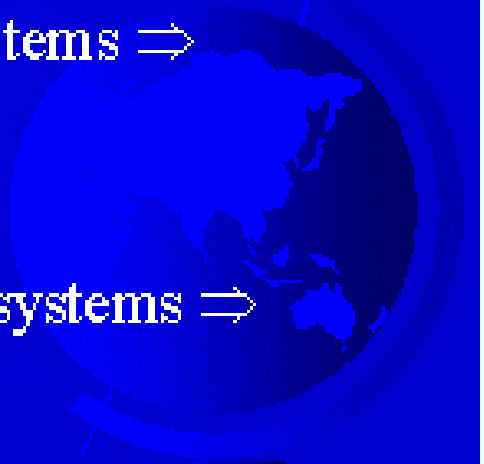
Agents also tend to be small in size. They do not, by themselves, constitute a complete application. Instead, they form one by working in conjunction with an agent host and other agents. In many ways, agents are of the same scope as applets. Small and of limited functionality on their own.

Software Agent =? Expert System

- "naive" user
  - agents ⇒ "average" users; expert systems ⇒ expert users.
- common task
  - agents ⇒ common tasks; expert systems ⇒ high-level tasks

Agents vs Expert Systems (cont.)

- personalized
  - agents ⇒ different actions; expert system ⇒ same actions.
- active, autonomous
  - agents ⇒ on their own; expert systems ⇒ passively answer.
- adaptive
  - agents ⇒ learn & change; expert systems ⇒ remain fixed.



Types of Software Agents

Dimensions along which they differ:

- Nature of Task Performed
  - user facing vs background task?
- Nature & Source of "Intelligence"
  - how is it built? who programs it?
- Mobility/Location
  - where does it reside? Can it move?
- Role Fulfilled
  - what task does it help the user with?

Pattie Maes --- MIT Media Laboratory 17 Software Agents Tutorial --- 97

Types of Software Agents

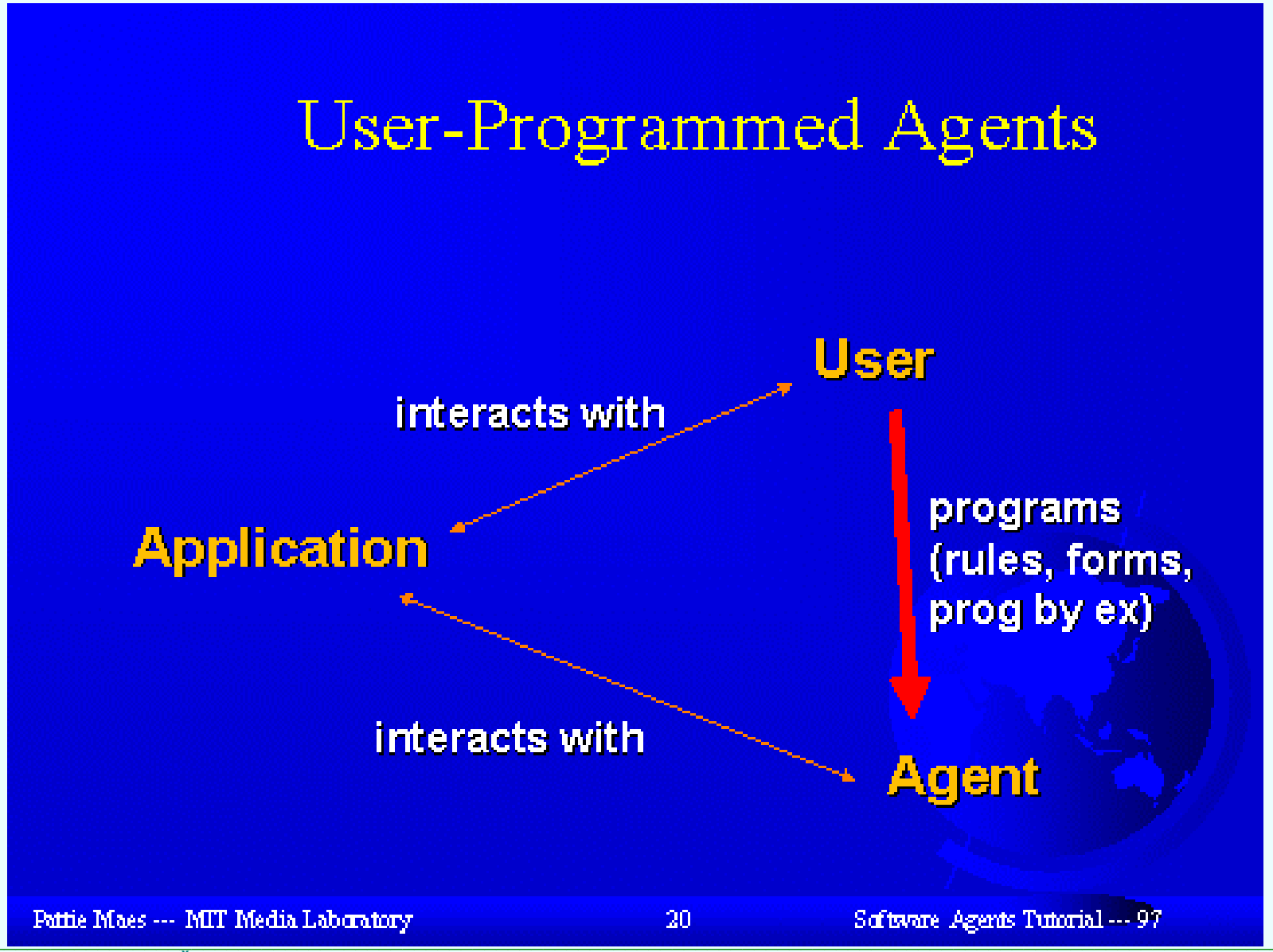
Nature of Task Performed:

- User agents: assist user, know interests/preferences/habits, may act on user's behalf.
  - e.g. personal news editor, personal e-shopper, personal web guide
- Service agents: perform more general tasks in the background.
  - e.g. web indexing, info retrieval, phone network load balancing

Types of Software Agents (cont.)

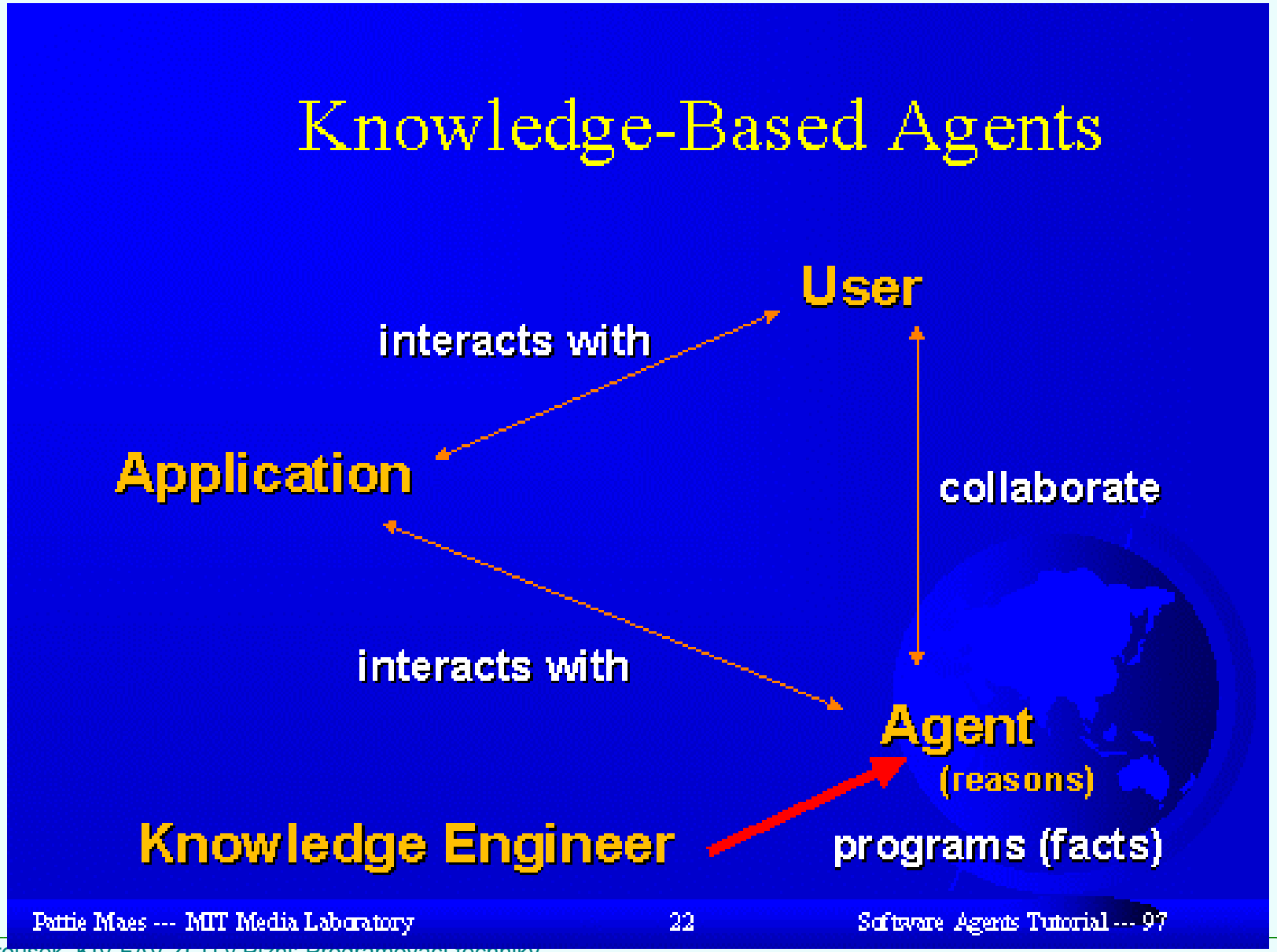
Nature of "Intelligence:

- User programmed - person provides rules and criteria directly
  - simplest
  - not very smart
  - relies on user's programming skill
  - commercially available


User-Programmed Agents
User
interacts with
Application
programs
(rules, forms,
prog by ex)
interacts with
Agent
Pattie Maes --- MIT Media Laboratory
20
Software Agents Tutorial --- 97

Nature of "Intelligence" (cont.)

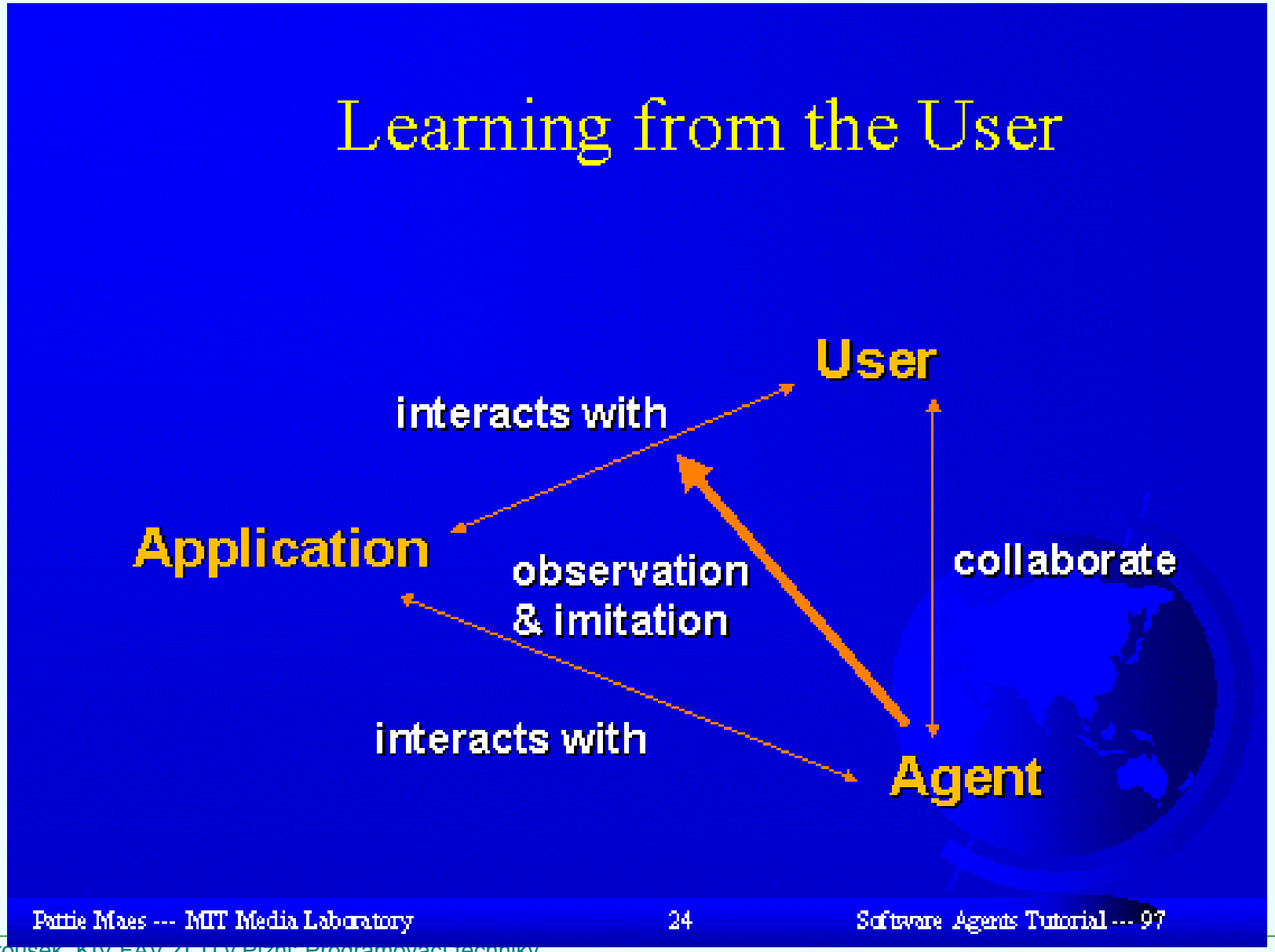
- AI engineered - created by traditional, knowledge-based AI techniques.
  - very complex
  - "smart"
  - programmed by a knowledge engineer
  - not commercially available yet

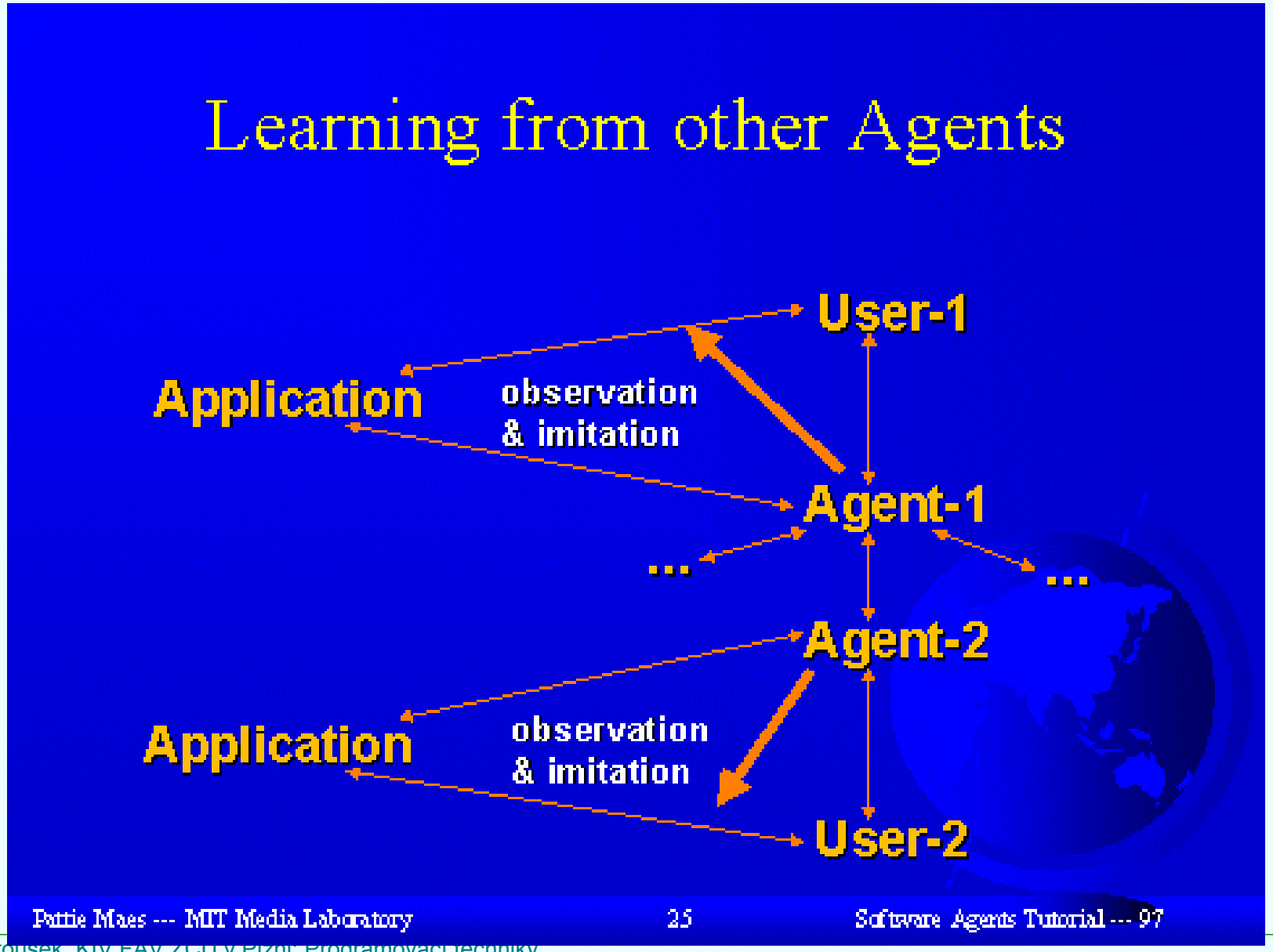

Knowledge-Based Agents
User
interacts with
Application
collaborate
interacts with
Agent
(reasons)
Knowledge Engineer
programs (facts)
Pattie Maes --- MIT Media Laboratory
22
Software Agents Tutorial --- 97

Nature of "Intelligence" (cont.)

- Learning agents - "program themselves"
  Patterns in user's actions and among users are detected & exploited
  - medium complexity
  - "smart" in key areas (where user concentrates)
  - beginning to be commercially available

Pattie Maes --- MIT Media Laboratory 23 Software Agents Tutorial --- 97


Learning from the User
User
interacts with
Application
observation
& imitation
collaborate
interacts with
Agent
Pattie Maes --- MIT Media Laboratory
24
Software Agents Tutorial --- 97


Learning from other Agents
User-1
Application
observation
& imitation
Agent-1
...
...
Agent-2
Application
observation
& imitation
User-2
Pattie Maes --- MIT Media Laboratory
25
Software Agents Tutorial --- 97

Location of Agents

- stationary in client
- stationary in server
- mobile: client->server->server->...

Location of Agents (cont.)

in server versus in client:

- easy/hard to locate other agents
- close to/far from the data operating on
- central failure point/more fault-tolerant
- less privacy/safer
- higher load on servers/more scalable
- potential trust problems running others' code on your machine (witness recent JAVA problems)

Pattie Maes --- MIT Media Laboratory 45 Software Agents Tutorial --- 97

Mobility of Agents

- NOT a necessary characteristic for some program to be an agent!
- Why move agents around?
  - reduce network traffic
  - share load among machines
  - go to the data if the data can't come to you
  - user may have only infrequent connection to network

Mobility of Agents (cont.)

Examples of languages supporting mobility:

- Telescript: agents go to centralized servers, conduct business, return to users' distributed locations with results.
- Java: agents (applets) move to distributed web browsers, bringing data and program to execute locally.
- TCL/TK: executing scripts remotely
- ...

Pattie Maes --- MIT Media Laboratory 47 Software Agents Tutorial --- 97

Roles for Software Agents

- Eager Assistant
- Guide
- Memory Aid
- Filter/Critic
- Matchmaker/Referral-giver
- Buyer/Seller (on your behalf)
- …

Multi-agent Collaboration

- if agent's confidence < tell-me threshold
- agent contacts other agents via bboard
- agent sends details of situation
- some agents respond with P_i and C_i
- agents computes score for every action $\Sigma C_i T_{i,s}$
- agent picks action with highest score
- agent updates trust levels $T_{i,s}$

Pattie Maes --- MIT Media Laboratory 82 Software Agents Tutorial --- 97

Agent Interface

- caricature faces convey state of agent: alert, thinking, working, suggestion, no clue, etc.
- agent produces a report of tasks automated
- user can look at agent's memory and instruct to forget (whole classes of) examples

Pattie Maes --- MIT Media Laboratory 83 Software Agents Tutorial --- 97

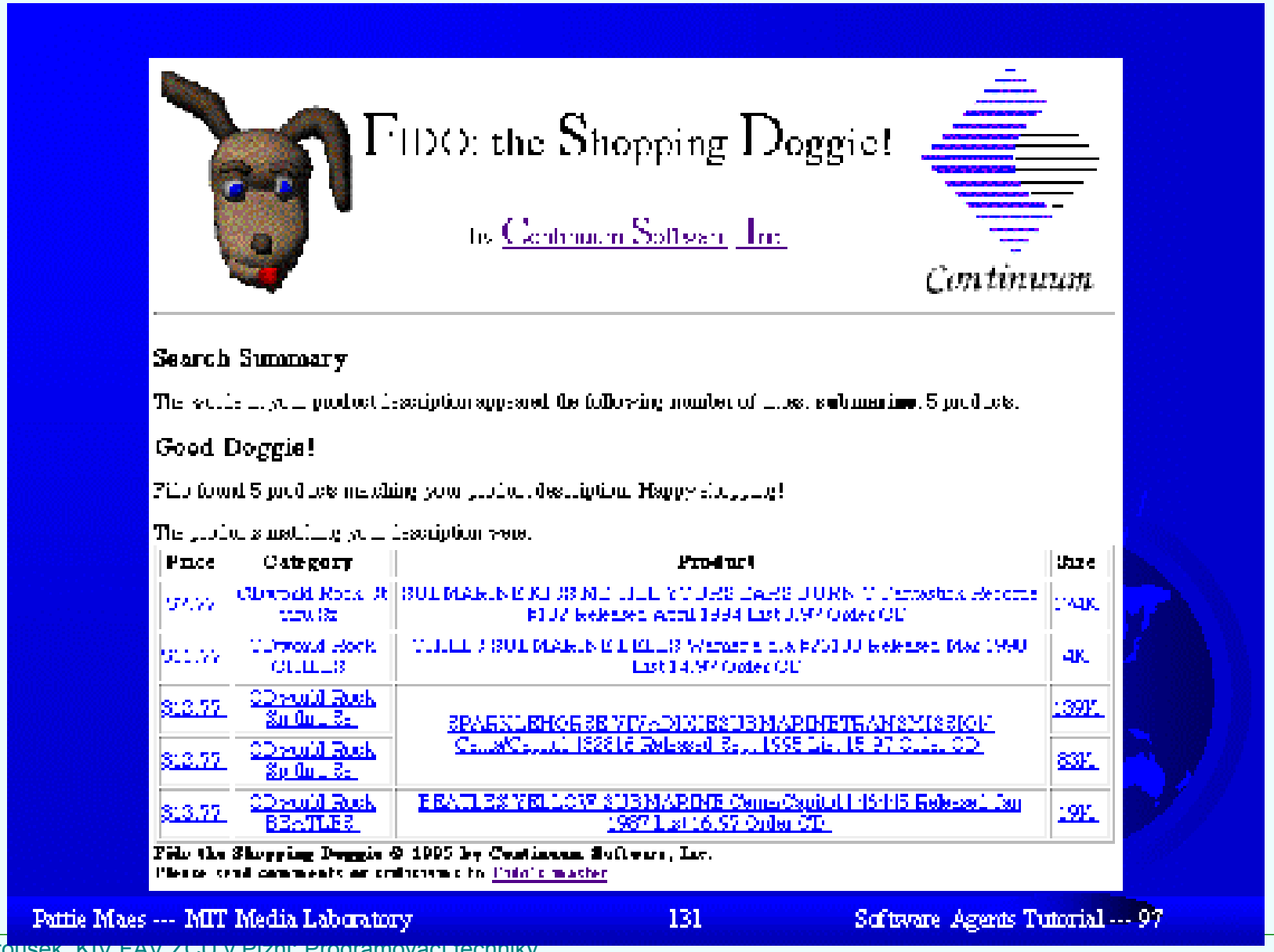
Agents for Buying/Selling

- BargainFinder (Krulwich, Anderson Consulting)
- Fido (Goodman, Continuum Software)
- ShopBot (Etzioni & Weld, UW)
- Kasbah (Chavez, MIT Media Lab)
- Bazaar (Guttman, MIT Media Lab)

Pattie Maes --- MIT Media Laboratory 126 Software Agents Tutorial --- 97

Inteligentní softwarové prostředky


Pattie Maes --- MIT Media Laboratory
130
Software Agents Tutorial --- 97


Fido: the Shopping Doggie!
by Continuum Software, Inc.
Continuum
Search Summary
Good Doggie!
Price
Category
Product
Size
Pattie Maes --- MIT Media Laboratory
131
Software Agents Tutorial --- 97

The Future of Agent Technology

- markets for/of agents
- ecologies of agents
- ...

GenericAgent.java

```
import java.io.*;
import java.util.*;
import java.lang.*;
import webl.lang.*;
import webl.lang.expr.*;
import webl.util.*;

public class GenericAgent
{
   static String script = "";
   static WeblEngine webl_engine
      = new WeblEngine();
   static int number_of_patterns = 0;
   static int selected_pattern = 0;
   int search_str_set = 0;
   int current_pattern = 0;

   public GenericAgent()
   {
      executeScript("AgtFiles/Init.agt");
   }
```

```
public void startAgain()
{
   webl_engine = new WeblEngine();
   executeScript("AgtFiles/Init.agt");
   search_str_set = 0;
}
public String executeScript(String filename)
{
   String line = null;
   String return_val = "";
   try
   {
      BufferedReader in = new BufferedReader(
         new FileReader(filename));
      while ((line = in.readLine()) != null)
      {
         script += line;
      }
      in.close();
      System.out.println("executeScript():" + filename + "\n"+ script + "\n");
      return_val = webl_engine.executeScript(script).print();
      script = "";
```

```
      }
      catch (FileNotFoundException fnf)
      {
         System.err.println("GenericAgent(): " +
            filename + " not found");
      }
      catch (IOException ex) { System.err.println(ex); }
      return return_val;
   }

   public String executeLine(String line)
   {
      Expr result;
      System.out.println("executeLine(): " + line + "\n");
      result = webl_engine.executeScript(line);
      return result.print();
   }

   public String searchWebSite(String website, String type_of_site,
      String search_string, String matching_power)
   {
      String search_str_line =
         "import Str; import Browser; var searchInput = \""
```

```
            + search_string + "\";" ;
      String type_of_site_line =
         "var searchType = \"" + type_of_site + "\";" ;
      String website_line =
         "var website = \"http://" + website + "\";" ;

      script = search_str_line +
         type_of_site_line + website_line;

      if (matching_power.length() != 0)
         executeLine("repeatThreshold = "
            + matching_power + ";");
      executeScript("AgtFiles/Search.agt");
      return "hello";
   }

   public String setSearchString(String search_str)
   {
      if (search_str_set != 0)
         return ";";
      search_str_set = 1;
      executeLine("searchString =\"" + search_str + "\";");
      return executeScript("AgtFiles/InitQuery.agt");
```

```
}

public void setStarttoParent()
{
   executeLine("childTag = Name(startQueryPiece);");
   executeLine("startQueryPiece = Parent(startQueryPiece);");
}

public void  executeQuery(int dir)
{
   String query = null;
   if (dir != 0) {
      String child_tag = executeLine("childTag;");
      String query_stmt = executeLine("queryStmt;");
      query = "Elem(P,\"" + child_tag + "\") directlyinside "
         + query_stmt;
   } else {
      query = executeScript("AgtFiles/Query.agt");
   }
   executeLine("records = " + query + ";");
   String ShowRec = "DummyRec = " + query
      + " contain (Pat(P,searchString)[0]);";
   executeLine(ShowRec);
```

```
      executeScript("AgtFiles/ShowRec.agt");
   }

   public int getNumberOfPatterns()
   {
       String result = executeLine(" Size(ToList(patterns));" );
       number_of_patterns = Integer.parseInt(result);
       return number_of_patterns;
   }

   public String getFirstPattern()
   {
       String index = "";

       if (number_of_patterns == 0)
          return null;

       current_pattern = 0;
       index = String.valueOf(current_pattern);
       executeLine("import Str; tags = Str_Search(patterns[" +
          index + "][1], \"[<](.*?)[>]\");");
       executeScript("AgtFiles/Pattern.agt");
       showPattern();
```

```
        return "hello";
    }

    public String getLastPattern()
    {
        String index = "";

        if (number_of_patterns == 0 )
           return null;

        current_pattern = number_of_patterns - 1;
        index = String.valueOf(current_pattern);
        executeLine("import Str; tags = Str_Search(patterns[" +
           index + "][1], \"[<](.*?)[>]\");");
        executeScript("AgtFiles/Pattern.agt");
        showPattern();

        return "hello";
    }

    public String getNextPattern()
    {
```

```
        String index = "";

        if (number_of_patterns == 0 )
           return null;
        if (current_pattern >= number_of_patterns)
           return null;
        current_pattern ++;
        index = String.valueOf(current_pattern);
        executeLine("import Str; tags = Str_Search(patterns[" +
           index + "][1], \"[<](.*?)[>]\");");
        executeScript("AgtFiles/Pattern.agt");
        showPattern();

        return "hello";
    }

    public String getPreviousPattern()
    {
        String index = "";

        if (number_of_patterns == 0 )
           return null;
        if (current_pattern == 0)
```

```
            return null;
        current_pattern --;
        index = String.valueOf(current_pattern);
        executeLine("import Str; tags = Str_Search(patterns[" +
            index + "][1], \"[<](.*?)[>]\");");
        executeScript("AgtFiles/Pattern.agt");
        showPattern();
        return "hello";
    }

    public String showPattern()
    {
        String result = executeLine("Size(ToList(weblstmt));");
        int numberStmt = Integer.parseInt(result);
        String[] weblstmt = new String[numberStmt];

        for (int i = 0; i < numberStmt; i++)
        {
            result = executeLine("weblstmt[" + String.valueOf(i) + "];");
            weblstmt[i] = result;

            String webltext1 = "weblobj[" + String.valueOf(i) + "] := "
                + weblstmt[i] + "; weblmarkuptxt = weblmarkuptxt + Markup(weblobj[" +
```

```
                String.valueOf(i) + "][3]);";

            String webltext2 = "if (Size(weblobj[" + String.valueOf(i) +
               "]) > 0 and minweblobj == -1) then minweblobj = " +
                   String.valueOf(i) + "; end;";

            String webltext3 = "if (Size(weblobj[" + String.valueOf(i) +
               "]) < Size(weblobj[minweblobj]) ) then minweblobj = " +
                   String.valueOf(i) + "; end;minweblobj;";

            String webltext = webltext1 + webltext2 + webltext3;
            executeLine(webltext);
          }
          String browser_text1 = "import Browser;";
          String browser_text2 = "Browser_ShowPage(\"<html><body><hr>\"
+weblmarkuptxt+\"<hr></body></html>\");";
          String browser_text3 = "weblmarkuptxt = \"\";";
          executeLine(browser_text1 + browser_text2 +
             browser_text3);

          return "hello";
      }
```

```
public int selectPattern()
{
   selected_pattern = current_pattern;
   executeScript("AgtFiles/Select.agt");
   return selected_pattern;
}

public int getNumberOfResults()
{
   String result = executeLine("Size(txt);");
   int number_of_results = Integer.parseInt(result);
   return number_of_results;
}

public String findPattern(int index, String pattern)
{
   String line1 = "import Str; currList = nil;";
   String line2 = "currList = Str_Search(txt[" +
      String.valueOf(index) + "],\"" + pattern + "\");" ;
   String line3 =
      "if ( Size(currList) != 0) then Str_Trim(currList[0][1]); else \"empty\";
         end;";
   String result = executeLine(line1 + line2 + line3);
```

```
        return result;
    }

    public String executeStmt(int index, String stmt)
    {
        String line1 = "dummy =\"\";CurrentRec = NewPiece(Markup(records["
              + String.valueOf(index) +
                  "]), \"text/html\");";

        String line2 = "import Str;";
        String line3 = stmt;
        executeLine(line1);
        //String CheckNilStmt = executeLine("CheckNilStmt;");
        //if (executeLine(CheckNilStmt).equals("empty"))
        ///   return "empty";
        String pre_result = executeLine(line2 + line3);
        StringTokenizer st = new StringTokenizer(pre_result);
        String result = "";
        while (st.hasMoreTokens())
           result += st.nextToken() + " ";
        return result;
    }
}
```

The word *agent* has found its way into a number of technologies. It has been applied to aspects of artificial intelligence research and to constructs developed for improving the experience provided by collaborative online social environments (MUDS, MOOs, and the like). It is a branch on the tree of distributed computing. There are agent development toolkits and agent programming languages.

Hucksters claim that agents can sort your mail, buy you a car, and solve your distributed computing woes -- in one fell swoop. Agents have tremendous potential to be sure, but this claim is a little far fetched -- at least today.

What is an agent?

It's difficult to find a succinct definition for agent that *includes* all of the things that most researchers and developers consider agents to be, and *excludes* all of the things they aren't. I recommend you read "Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents" by Stan Franklin and Art Graesser for a thorough, well-thought-out classification scheme. (See Resources.)

In this article, I'll limit myself to illustrating rather than defining.

Inteligentní softwarové prostředky

Agents typically possess several (or all) of the following characteristics; they are:

- Autonomous
- Adaptive/learning
- Mobile
- Persistent
- Goal oriented
- Communicative/collaborative
- Flexible
- Active/proactive

Agents also tend to be small in size. They do not, by themselves, constitute a complete application. Instead, they form one by working in conjunction with an agent host and other agents. In many ways, agents are of the same scope as applets. Small and of limited functionality on their own.

Why study agents?

Agents make an interesting topic of study because they draw on and integrate so many diverse disciplines of computer science, including objects and distributed object architectures, adaptive learning systems, artificial intelligence, expert systems, genetic algorithms, distributed processing, distributed algorithms, collaborative online social environments, and security -- just to name a few.

Agent technology is significant because of the sustained commercial interest surrounding it. You've most likely heard of General Magic and Telescript, and maybe even IBM's Aglets Workbench (now called IBM Aglets SDK) and Mitsubishi's Concordia. Agent technology may not have hit prime time quite yet, but it does seem to be gathering its share of investment money. Take a gander at the Resources section for a host of other companies engaged in agent technology development.

Agent technology is also interesting for its potential to solve some nagging productivity problems that pester almost all modern computer users. Many agents are meant to be used as intelligent electronic gophers -- automated errand boys. Tell them what you want them to do -- search the Internet for

information on a topic, or assemble and order a computer according to your desired specifications -- and they'll do it and let you know when they've finished.

What problems do agents solve?

Agent technology solves, or promises to solve, several problems on different fronts.

Mobile agents solve the nagging client/server network bandwidth problem. Network bandwidth in a distributed application is a valuable (and sometimes scarce) resource. A transaction or query between a client and the server may require many round trips over the wire to complete. Each trip creates network traffic and consumes bandwidth. In a system with many clients and/or many transactions, the total bandwidth requirements may exceed available bandwidth, resulting in poor performance for the application as a whole. By creating an agent to handle the query or transaction, and sending the agent from the client to the server, network bandwidth consumption is reduced. So instead of intermediate results and information passing over the wire, only the agent need be sent.

Here's a related situation. In the design of a traditional client/server architecture, the architect spells out the roles of the client and server pieces very precisely -- up front, at design time. The architect makes decisions about where a particular piece of functionality will reside based on network bandwidth constraints (remember the previous problem), network traffic, transaction volume, number of clients and servers, and many other factors. If these estimates are wrong, or the architect makes bad decisions, the performance of the application will suffer. Unfortunately, once the system has been built and the performance measured, it's often difficult or impossible to change the design and fix the problems. Architectures based on mobile agents are potentially much less susceptible to this problem. Fewer decisions must be made at design time, and the system is much more easily modified after it is built. Agent architectures that support adaptive network load balancing could do much of the redesign automatically.

Agent architectures also solve the problems created by intermittent or unreliable network connections. In most network applications today, the network connection must be alive and healthy the entire time a transaction or query is taking place. If the connection goes down, the client often must start the transaction or query from the beginning, if it can restart it at all. Agent

technology allows a client to dispatch an agent handling a transaction or query into the network when the network connection is alive. The client can then go offline. The agent will handle the transaction or query on its own, and present the result back to the client when it re-establishes the connection.

Agent technology also attempts to solve (via adaptation, learning, and automation) the age-old (not to mention annoying) problem of getting a computer to do real thinking for us. It's a difficult problem. The artificial intelligence community has been battling these issues for two decades or more. The potential payoff, however, is immense.

An agent architecture

In this column and in the next few down the road, I'm going to show you how to design and build an agent architecture. I'll concentrate on designing and implementing support for several of the agent characteristics mentioned earlier. Specifically, I'll consider the *tactile* characteristics of mobility and persistence, the *social* characteristics of communication and collaboration, and the *cognitive* characteristics of adaptation, learning, and goal orientation.

Requirements

Before we explore these three areas in detail, we need to build the foundation. Let's take a look at the key requirements our agent architecture must satisfy:

- An agent must have its own unique identity
- An agent host must allow multiple agents to co-exist and execute simultaneously
- Agents must be able to determine what other agents are executing in the agent host
- Agents must be able to determine what messages other agents accept and send
- An agent host must allow agents to communicate with each other and the agent host
- An agent host must be able to negotiate the exchange of agents
- An agent host must be able to freeze an executing agent and transfer it to another host

- An agent host must be able to thaw an agent transferred from another and allow it to resume execution
- The agent host must prevent agents from directly interfering with each other

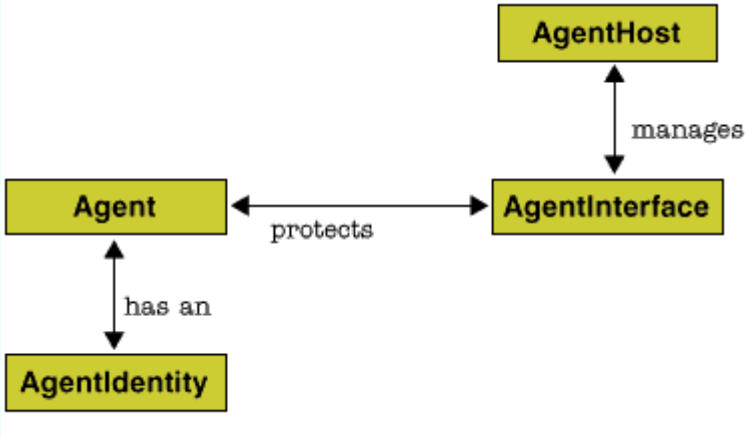
These architectural requirements provide support for the tactile and social characteristics of supported agents. Explicit support is not provided for the cognitive characteristics. We'll handle those requirements in a future column.

The objects

From the requirements listed above, we can determine what classes will be present in the system. Obviously, the system will include an Agent class and an AgentHost class. Less obviously, our system will also include an AgentInterface class. The AgentInterface class provides agents with a view of each other. This is necessitated by the last requirement -- agents must not be able to directly interfere with other agents. In practice this means that agents must not be able to directly invoke the public methods of other agents. Finally, the first requirement dictates that there be an AgentIdentity class. This class identifies agents both to themselves and to others. It allows an

agent to decide whether a message from another agent should be accepted or merely discarded.

The figure below illustrates the relationship between the classes described above.



Our agent architecture

Let's take a look at each class in more detail.

- The AgentHost class defines the agent host. An instance of this class keeps track of every agent executing in the system. It works with other hosts in order to transfer agents.

- The Agent class defines the agent. An instance of this class exists for each agent executing on a given agent host.
- The AgentInterface class defines the agent interface. An instance of this class envelopes an agent and provides access to it via a well-defined interface. It is also the primary conduit for communication between agents.

  An AgentInterface instance is the only handle an agent gets to the other agents executing on a given host.
- The AgentIdentity class defines agent identity. An instance of this class uniquely identifies an agent. Agents use this information to identify the agents with whom they are interested in collaborating.

Because the associated body of code is large, and the classes are difficult to use without additional explanation, I'm not going to provide any source code this month. But don't fret. Tune in next month, and you'll get plenty of source code along with detailed instructions on how to put it to good use.

Conclusion

With the foundation in place, we're ready to erect the walls. In coming months, I'll explore each of the three groups of characteristics mentioned above -- the tactile, the social, and the cognitive. I'll begin with the tactile characteristics, so expect a demonstration of how to weave agent mobility into the framework as we develop it next month.

Before I finish, I thought I'd leave you with some guidelines that should help you determine where agent technology might find a home in your projects.

The most robust and well-developed areas of agent technology are those revolving around autonomy and mobility. Applications built around unreliable or intermittent network connections will almost certainly find benefit, as will applications that must perform offline processing.

Oddly enough, the weakest areas of agent technology (though not for lack of trying) are those that seem to receive the most hype -- the aspects related to intelligence. If your application requires intelligent agents, you'll probably need to wait a while longer to get them. The artificial intelligence community has been working diligently for over two decades on this single problem.

Inteligentní softwarové prostředky

Remember, a computer has to do a better, more accurate job at a given task than we're capable of, or we won't use it.

I hope to hear from readers who are currently using, deciding whether or not to use, or developing agent-based technology or solutions -- especially with regard to the guidelines provided above.

The Open Agent Architecture™

A framework for integrating a community of heterogeneous software agents in a distributed environment.

What is an Agent?

The term "agent" has been used by many people to mean many different things. Even within the Agent Research Community, there are at least the following variants on the term *agent*: Mobile Agents (e.g. Voyager, Grasshopper), Learning Agents, Autonomous Agents (e.g. robots), Planning Agents, Simulation agents, Distributed Agents.

In the context of the Open Agent Architecture™ (OAA®), we are focused on building distributed *communities* of agents, where *agent* is defined as any software process that meets the conventions of the OAA society. An agent satifies this requirement by registering the services it can provide in an acceptable form, by being able to speak the Interagent Communication Language (ICL), and by sharing functionality common to all OAA agents, such as the ability to install triggers, manage data in certain ways, etc. In our community of agents, we are able to include, and make use of, each of the different types of agents mentioned above.

Agent Architectures as a programming methodology

Distributed Agent technology can be thought of as the next step in the evolution of programming methodologies. *In the beginning,* there were machine and assembly languages. These evolved into higher level programming languages able to break apart programming steps into *subroutines*. A next generalization allowed programmers to group collections of subroutines into *libraries* or *modules*. A subsequent innovation added the notion of object orientation: data and routines could be grouped into a single *object*, which further encapsulated the internals of the routines and increased modularity and reuse. Distributed Object technologies, such as CORBA or DCOM, then broke the rule that every object must reside on the local machine; now object libraries could post services through a broker, and the objects themselves could even be written in different programming languages, as long as they used the same Interface Definition Language.

So, what can *Distributed Agents* possibly add to the Distributed Object paradigm? With distributed objects, even though objects may run on different platforms, applications generally form a single monolithic entity of tightly-bound objects, with hand-coded calls to known methods of pre-existing objects.

In a distributed *agent* framework, we conceptualize a dynamic community of agents, where multiple agents contribute services to the community. When external services or information are required by a given agent, instead of calling a known subroutine or asking a specific agent to perform a task, the agent submits a high-level expression describing the needs and attributes of the request to a specialized *Facilitator agent*. The Facilitator agent will make decisions about which agents are available and capable of handling sub-parts of the request, and will manage all agent interactions required to handle the complex query. **The advantage?** Such a distributed agent architecture allows the construction of systems that are more flexible and adaptable than distributed object frameworks. Individual agents can be dynamically added to the community, extending the functionality that the agent community can provide as a whole. The agent system is also able to adapt to available resources in a way that hardcoded distributed objects systems can't.

Human calling Agent, come in Agent...

When designing the Open Agent Architecture, we realized that it is imperative that the human user must be able to interact with the collection of distributed agents as an equal member of the community, not just as an outsider to whom is presented a result once real agents have done all the work. Multiple agents can provide services for retrieval, combination, and management of the growing amount of online information, but this is only useful if controlling and interacting with the network of agents remains less complicated than interacting with the online services themselves!

With this in mind, we designed the InterAgent Communication Language (ICL) to be a logic-based declarative language capable of representing natural language expressions. In addition, we incorporated techniques into the architecture for communicating with agents using simulataneous multiple (natural) input modalities; humans can point, speak, draw, handwrite, or use standard graphical user interface when trying to get a point across to a collection of agents. The agents themselves will compete and cooperate in parallel to translate the user's request into an ICL expression to be handled. These techniques, in combination with the use of special class of agents called Facilitator agents (Facilitator agents reason about the agent interactions necessary for handling a given complex ICL expression), allow human users to closely interact with the ever-changing community of distributed agents.

Technical Features

Characteristics

- *Open*: agents can be created in multiple programming languages and interface with existing legacy systems.
- *Extensible*: agents can be added or replaced individually at runtime.
- *Distributed*: agents can be spread across any network-enabled computers.
- *Parallel*: agents can cooperate or compete on tasks in parallel.
- *Mobile*: lightweight user interfaces can run on handheld PDA's or in a web browser using Java or HTML and most applications can be run through a telephone-only interface.
- *Multimodal*: When communicating with agents, handwriting, speech, pen gestures and direct manipulation (GUIs) can be combined in a natural way.

Platforms & Languages

The OAA 2.x Facilitator is distributed in binary form for Windows, Solaris, and Linux platforms.

OAA 2.x agent libraries exist for the following languages, and have been used on (at least) the following platforms:

Quintus Prolog	SunOs/Solaris, Windows 9x/NT/2000, other Quintus-supported platforms
Sicstus Prolog	SunOs/Solaris, Linux, Windows 9x/NT/2000, other Sicstus-supported platforms
ANSI C/C++ (Unix, Microsoft, Borland)	SunOs/Solaris, Linux, Windows 9x/NT/2000
Java	Any Java platform
Compaq's Web Language	Any Java platform

OAA 1.0 agent libraries exist for the following languages, and have been used on (at least) the following platforms:

Quintus Prolog	SunOs/Solaris, Windows 9x/NT, other Quintus-supported platforms
ANSI C/C++ (Unix, Microsoft, Borland)	SunOs/ Solaris, SGI IRIX, Windows 9x/NT
Common Lisp (Allegro & Lucid)	SunOs/Solaris, Linux
Java	Any Java platform
Borland Delphi	Windows 3.1, Windows 9x/NT
Visual Basic	Windows 3.1, Windows 9x/NT
Compaq's Web Language	Any Java platform
Perl	Unix

Data mining a deduktivní databáze

Prostředky, jak analýzou rozáhlých automaticky získaných dat formulovat či odvozovat nové informace či znalosti.

Nová vědní disciplina – ”objevování” znalostí v databázích (knowledge discovery in databases – KDD)

Celý proces je interaktivní, řízený uživatelem, využívající jeho schopnosti, zkušenosti a znalosti.

Postup “dobývání”, resp. “objevování” znalostí v databázích:

- získání apriorních znalostí o datech
- přesná formulace cílů uživatele
- výběr (pod)množiny cílových dat, v níž se budeme snažit znalosti “objevit”
- předzpracování dat (např. doplnění chybějících hodnot)
- transformace dat (transformace proměnných, redukce dimenze, ...)
- výběr techniky “dobývání” – klasifikace, regrese, shlukování, generalizace, ...
- výběr konkrétního algoritmu pro řešení úlohy “dobývání”
- vlastní výběr (“dobývání”) dat, vyhledávání souvislostí, funkčních závislostí, logických pravidel, ...
- interpretace a prezentace získaných (odvozených) znalostí
- dokumentování a integrace nových znalostí do systému

Metoda: Induktivní logické programování

Inteligentní softwarové prostředky