Communications of the Association for Information Systems

Volume 14 Paper

September 2004

Intelligent Agents

Ira Rudowsky
Brooklyn College, rudowsky@brooklyn.cuny.edu

Recommended Citation
Rudowsky, I. (2004). Intelligent Agents. Communications of the Association for Information Systems, 14, pp-pp. https://doi.org/10.17705/1CAIS.01414

Communications of the Association for Information Systems

INTELLIGENT AGENTS

Ira S. Rudowsky
Brooklyn College
rudowsky@brooklyn.cuny.edu

ABSTRACT

A search on Google for the keywords “intelligent agents” will return more than 330,000 hits; “multi-agent” returns almost double that amount. Over 5,000 citations appear on www.citeseer.com. What is agent technology and what has led to its enormous popularity in both the academic and commercial worlds? Agent-based system technology offers a new paradigm for designing and implementing software systems. The objective of this tutorial is to provide an overview of agents, intelligent agents and multi-agent systems, covering such areas as:

1. what an agent is, its origins and what it does,
2. how intelligence is defined for and differentiates an intelligent agent from an agent,
3. how multi-agent systems coordinate agents with competing goals to achieve a meaningful result, and
4. how an agent differs from an object of a class or an expert system.

Examples are presented of academic and commercial applications that employ agent technology. The potential pitfalls of agent development and agent usage are discussed.

Keywords: agents, intelligent agents, multi-agent systems, artificial intelligence

I. WHAT IS AN AGENT?

HISTORICAL CONTEXT

Over the last thirty years, Artificial Intelligence (AI) and agent systems were closely related. AI is interested in studying the components of intelligence (e.g., the ability to learn, plan) while the study of agents deals with integrating the same components. This distinction may seem to imply that all the problems within AI must be solved to build an agent. But, as Etzioni points out, this is not the case: ‘Intelligent agents are ninety-nine percent computer science and one percent AI’ [Etzioni, 1996]. While AI techniques may be drawn upon to build agents, not all AI capabilities are required by an agent. Thus not all AI problems need be solved before building an agent. For example, the ability to learn may not be a desirable trait for an agent in some situations while it is certainly a component of AI.

Between 1960 and 1990, AI witnessed a great deal of progress in many sub-areas such as knowledge representation and inference, machine learning, vision, robotics. In addition, various advancements in computer science and computing (e.g., multitasking, distributed computing, communicating processes, real-time systems and communication networks) made the design, implementation, and deployment of agent based systems possible, at least in principle. The potential applications in distributed databases, mobile computing, information gathering, and collaborative computing that take advantage of these advances in AI and computer systems pose a strong argument for the development of intelligent agents and multi-agent systems.

But is all this in touch with the reality? One need look no further than NASA's Deep Space 1 (DS1) project where an artificial intelligence system was placed on board to plan and execute spacecraft activities. In contrast to remote control, this sophisticated set of computer programs acts as an agent of the operations team on board the spacecraft. Rather than requiring humans to do the detailed planning necessary to carry out desired tasks, Remote Agent formulates its own plans/ It combines the high level goals provided by the operations team with its detailed knowledge of both the condition of the spacecraft and how to control it. It then executes that plan, constantly monitoring its progress. If problems develop, Remote Agent in many cases will be able to fix them or work around them. If it is unable to find a fix or a work around, it can request help from its human counterparts.

Remote Agent operated DS1 spacecraft during two experiments that began on May 17, 1999, when it ran the on-board computer more than 60,000,000 miles from Earth. The tests were a step toward robotic explorers of the 21st century that are less costly, more capable, and more independent from ground control. These intelligent agents can have the potential of making space exploration of the future more productive while staying within NASA's limited budget. By transferring functions normally performed by people to a remote agent, a spacecraft may be more agile in responding to unexpected situations it encounters. In addition, by assuming responsibility for on-board tasks that currently require human intervention from Earth, Remote Agent permits spacecraft to fulfill their mission while greatly reducing the time consuming and labor intensive communications to and from mission control. This ability will enable NASA to achieve its goal of launching many more spacecraft into the solar system while staying within budget.

So we have what looks like a winning idea. What is it all about?

DEFINING AN AGENT

No definition of the term agent is accepted universally. Here are a few:

- Russel and Norvig [1995) define an agent as an entity that can be viewed as perceiving its environment through sensors and acting upon its environment through effectors.
- Coen [1995] views software agents as programs that engage in dialogs and negotiate and coordinate the transfer of information.
- Wooldridge and Jennings [1995] state that an agent is a hardware and/or software-based computer system displaying the properties of autonomy, social adeptness, reactivity, and proactivity. Others [Brustolini, 1991; Franklin and Graeser, 1996; Maes, 1995; Hayes-Roth et al, 1995; Gilbert et al, 1995] offer variations on this theme.

A consensus among researchers indicates that autonomy, the ability to act without human intervention or other systems, is a key feature of an agent. Beyond that, different attributes take on different importance based on the domain of the agent.

Figure 1 is a high-level view of an agent within its environment. An agent receives input from its environment. Through a repertoire of actions available to it, the agent reacts to the environment

in order to modify it. Generally, in domains of reasonable complexity, an agent will not control its environment completely. Thus, the same action performed twice in seemingly identical situations might appear to result in completely different outcomes. Failure is also a possibility i.e., the action taken by the agent may not produce the desired effect at all.

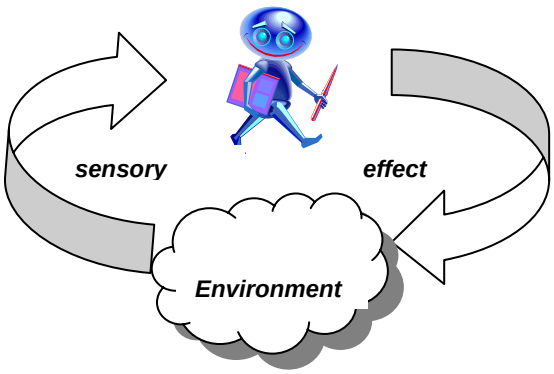

Figure 1. Agent Interacting with its Environment

AGENT ENVIRONMENTS

The critical decision an agent faces is determining which action to perform to best satisfy its design objectives. Agent environments are classified based on different properties that can affect the complexity of the agent's decision-making process [Russell and Norvig, 1995]. They include:

- *Accessible vs. inaccessible*
  An accessible environment is one in which the agent can obtain complete, timely and accurate information about the state of the environment. The more accessible an environment, the less complicated it is to build agents to operate within it. Most moderately complex environments are inaccessible.

- *Deterministic vs. non-deterministic*
  Most reasonably, complex systems are non-deterministic. The state that will result from an action is not guaranteed even when the system is in a similar state before the action is applied. This uncertainty presents a greater challenge to the agent designer than deterministic systems.

- *Episodic vs. non-episodic*
  In an episodic environment, the agent's actions depend on a number of discrete episodes with no link between the agent's performance in different scenarios. This environment is simpler to design due to the lack of need to reason about interactions between previous and future episodes; only the current environment needs to be considered.

- *Static vs. dynamic*
  Static environments remain unchanged except for the results produced by the actions of the agent. Other processes operate in a dynamic environment, thereby changing the environment outside the control of the agent. A dynamic environment obviously requires a more complex agent design.

- *Discrete vs. continuous*
  If the number of actions and percepts are fixed and finite, then the environment is discrete. A chess game is a discrete environment while driving a taxi is an example of a continuous one.

AGENT EXAMPLES

A simple example of an agent in a physical environment is a thermostat for a heater. The thermostat receives input from a sensor, which is embedded in the environment, to detect the temperature. Two states: (1) temperature too cold and (2) temperature OK are possible. An action is associated with each state: (1) too cold ➡ turn the heating on and (2) temperature OK ➡ turn the heating off. The first action raises the room temperature. However its action does not guarantee a higher room temperature. If cold air continuously comes into the room, say from an open window, the added heat may not create the desired effect of raising the room temperature.

Background software processes which monitor a software environment and perform actions to modify it can be viewed as agents. A software daemon that continually monitors a user's incoming e-mail and indicates via a GUI icon that some messages are unread can also be viewed as a simple agent.

AGENTS AND OBJECTS

Doesn't object-oriented programming provide these agent features? What does an agent offer that Java can't? After all, objects encapsulate data that can represent the state of the object, have methods that enable the objects to perform actions, and can communicate by message passing. Despite these similarities, agents differ significantly from objects. An object may be said to exhibit autonomy over its state (by defining its instance variables as private) but it does not exhibit control over its behavior. The designers of an object-oriented system work towards a common goal: if an object O_i invokes method m of object O_j then that method will be executed because the designers ensured that it is in the best interest of the system. In many types of multi-agent systems, where agents may be built by and/or for different and competing organizations, no such common goal can be assumed. Thus, the agent decides whether to execute the requested method based on its own design goals. "Objects invoke, agents request" or as Wooldridge [1995] indicates he heard it said "Objects do it for free; agents do it for money".

A second important distinction is that objects do not inherently say anything about how to build a system that integrates flexible, autonomous behavior. Of course, such systems could be built with objects but the standard object-oriented programming model is not concerned with these types of behavior.

Finally, a multi-agent system is intrinsically multi-threaded (each agent is assumed to contain at least one thread of control). Object-oriented languages may enable multi-threading but autonomy is not a sine qua non.

AGENTS AND EXPERT SYSTEMS

What about expert systems? Couldn't they be considered agents? Expert systems typically do not exist in an environment, they are disembodied. They obtain their information not through sensors but through a user acting as a middle man. MYCIN, [Shortlie and Rhame, 1975] the expert system whose purpose was to assist physicians in the treatment of blood infections in humans, acted as a consultant. It did not operate directly on humans or any other environment. Similarly, expert systems do not act on any environment. Instead they give feedback or advice to a third party. In addition, expert systems are generally not required to be capable of cooperating with other expert systems. The foregoing does not mean that an expert system cannot be an agent. In fact, some real-time (typically process control) expert systems, such as the ARCHON system discussed in Section III [Jennings, 1996a], are agents.

II. INTELLIGENT AGENTS

The idea of intelligent software agents captures the popular imagination. Tell the agent what you want done, set it free, and wait for it to return results sounds too good to be true. We'll come back to that later. In the meantime, we address the question of what makes an agent intelligent. Wooldridge and Jennings [1995] define an intelligent agent as one that is capable of flexible autonomous action to meet its design objectives. Flexible means:

- *reactivity*:
  intelligent agents perceive and respond in a timely fashion to changes that occur in their environment in order to satisfy their design objectives. The agent's goals and/or assumptions that form the basis for a procedure that is currently executing may be affected by a changed environment and a different set of actions may be need to be performed.

- *pro-activeness*:
  reacting to an environment by mapping a stimulus into a set of responses is not enough. As we want intelligent agents to do things for us, goal-directed behavior is needed. In a changed environment, intelligent agents must recognize opportunities and take the initiative if they are to produce meaningful results. The challenge to the agent designer is to integrate effectively goal-directed and reactive behavior.

- *social ability*:
  intelligent agents are capable of interacting with other agents (and possibly humans), through negotiation and/or cooperation, to satisfy their design objectives.

Other properties sometimes mentioned in the context of intelligent agents include:

- *mobility:* the ability to move around an electronic environment
- *veracity:* an agent will not knowingly communicate false information
- *benevolence:* agents' goals do not conflict and every agent will therefore always try to do what is asked of it
- *rationality:* an agent will act to achieve its goals insofar as its beliefs permit
- *learning/adaptation:* agents improve performance over time

What drives the interest and need for intelligent agents? Users of the Web are faced with information overload. The amount of data available doubles annually. Individuals can analyze only about 5% of the data and most efforts do not provide real meaning. Thus, intelligent agents are needed to assist in searching, filtering, and deciding what is relevant to the user. Forrester Research [Coolidge, 2001], in the latest forecast found, estimated that by 2005, 20 million households will be using the Web for investment and financial planning advice; quite an important task for a critical life decision without some means of assistance.

To put these concepts into a reality based framework, here is a scenario of what an intelligent agent might be able to do in the future [Wooldridge and Jennings, 1995].

> *You are editing a file when your PDA requests your attention: an e-mail message arrived that contains notification about a paper you sent to an important conference. The PDA correctly predicted that you would want to see it as soon as possible. The paper was accepted. Without prompting, the PDA begins to look into travel arrangements by consulting a number of databases and other networked information sources. A short time later, a summary of the cheapest and most convenient travel options is presented to you for selection and approval.*

KNOW HOW, WITH NO HOW

One way to convey to an agent the task it should perform is to simply write a program that the agent should execute. The agent will do exactly as told and no more - if an unforeseen circumstance arises, the agent will have no clue as to how it should react. Thus, what we really want is to tell our agent *what* to do without really telling it *how* to do it. Associating a performance measure or utility with each state is one technique for doing so.. A utility is a number representing the 'goodness' of the state – the higher the utility the better the state. The task of the agent is to maximize utility without being told how to do so.

An example of the use of such a utility function is in Tileworld [Pollack, 1990]. Tileworld is a simulated, two-dimensional grid environment, both dynamic and unpredictable, which contains agents, tiles, obstacles, and holes. An agent can move in four directions – up, down, left, or right. If it is located next to a tile, it can push it. An obstacle is a group of immovable grid cells; agents are not allowed to travel freely through obstacles. An agent scores points by filling holes with tiles, the aim being to fill as many holes as possible. A number of parameters can be set, including the rate of appearance and disappearance of holes, obstacles, and tiles. The performance of an agent on a run *r* is defined as the number of holes filled in run *r* divided by the number of holes that appeared in run *r*.

Despite its seeming simplicity, Tileworld enables the study of a number of important capabilities of agents. Chief among them is the ability of an agent to react to changes in the environment and to exploit opportunities when they arise. If an agent is pushing a tile to fill a hole and the hole disappears before being filled, the agent should realize its original goal is no longer in effect and 'rethink' its objective by searching for a new hole to fill. In a similar vein, if an agent is pushing a tile to fill a hole that's four grid cells in the north direction and an empty hole suddenly appears one cell to the east of the agent's current position, the agent should capitalize on this change and fill the closer hole. All other things being equal, the chances of the hole to the east not disappearing in one move is four times greater than the hole to the north which is four moves away.

Tileworld represents an oversimplification of real-world scenarios but it is a useful environment for experimentation.

BUT HOW DO THEY DO IT?

How is this know-how incorporated into software? Shoham introduced a new programming paradigm based on societal views of computation that he called agent-oriented programming [Shoham, 1993]. He called the programming language AGENT0. The key idea is programming agents in terms of "mentalistic" notions such as belief, desire, and intention (BDI), which have been developed by agent theorists to represent the properties of agents. In AGENT0, an agent is specified in terms of a set of capabilities (things the agent can do), a set of initial beliefs, a set of initial commitments (an agreement to perform a particular action at a particular time) and a set of commitment rules. Capabilities are used by the agent to decide whether to adopt commitments; an agent will not adopt a commitment to perform an action if the agent can never be capable of performing that action.

The set of commitment rules determines how the agent acts. Each commitment rule contains a message condition, a mental condition and an action. To determine whether such a rule fires, the message condition is matched against the message the agent received and the mental condition is matched against the agent's beliefs. If the rule fires, the agent becomes committed to performing the action. For example, agent A sends a commitment request in a message to agent B. Agent B will accept or reject the request based on the details of the request, its behavioral rules, and its current mental model. B will then send a message to A indicating acceptance or rejection of the request. If B accepts the request, it agrees to attempt to perform the requested action at the requested time if possible. For example, agent B may commit itself to make an inquiry into a database on behalf of A. Even if B can connect and query the database, it may not

be possible at the specified time due to a disk crash during the database access. B will monitor the execution of the query and send a message back to A to report success or failure of the commitment.

III. THE NEXT STEP: MULTI-AGENT SYSTEMS

As the field of AI matured, it broadened its goals to the development and implementation of multi-agent systems (MASs) as it endeavored to attack more complex, realistic, and large-scale problems which are beyond the capabilities of an individual agent. [Sycara, 1998]. The capacity of an intelligent agent is limited by its knowledge, its computing resources, and its perspective [Simon, 1957]. By forming communities of agents or agencies (Figure 2), a solution based on a

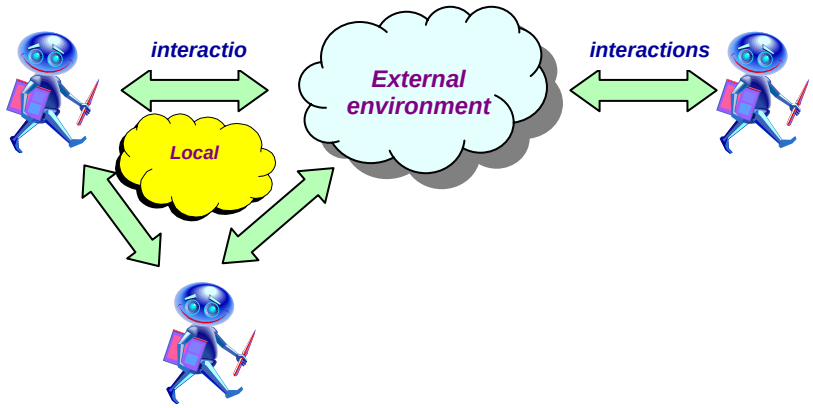

Figure 2. Multi-Agent System

modular design can be implemented where each member of the agency specializes in solving a particular aspect of the problem. Thus, the agents must be able to interoperate and coordinate with one another in peer-to-peer interactions. The characteristics of MASs are defined as [Sycara, 1998]:

- Each agent's information or capabilities for solving the problem is incomplete. Thus, the agent's viewpoint is limited.
- No global control system
- Data are decentralized
- Computation is asynchronous

What can MASs do that generate so such interest in them?

- They can be used to solve problems that are too large for a centralized agent to solve because of resource limitations and/or to avoid a one point bottleneck or failure point.
- To keep pace with changing business needs, legacy systems (which may not be able to be rewritten due to a combination of cost, time, and technical know) how, can be made to interoperate with other agents in an agent society by building an agent wrapper around them [Genesereth and Ketchpel, 1994]. In addition, those agencies which are not self-contained but interact with outside agents e.g., buying and selling, contract negotiation, meeting scheduling [Garrido and Sycara, 1996], are by nature MASs.

- MASs enhance performance in the following areas: (1) computational efficiency through concurrency, (2) reliability via redundancy, (3) extensibility of the agency by changing the number and capabilities of the agents, (4) maintainability via modularity and (5) reuse of agents in different agencies to solve different problems.

MASs sound great, but did anyone implement one that is useful? Here is but a small sample of applications:

- ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems) [Jennings et al, 1996a; Parunak, 1999] is one of the largest and probably best known European multi-agent system development project to date. Multi-agent technology was developed and deployed in a number of industrial domains. The most significant is a power distribution system currently operational in northern Spain for the electricity utility Iberdrola. The ARCHON technology was subsequently deployed in particle accelerator control for CERN [Perriolat et al., 1996].

- In workflow and business process control, the ADEPT system [Jennings et al, 1996b] models numerous departments at British Telecom involved in installing a network to deliver a particular type of telecommunications service. Beginning with the initial customer contact, followed by customer vetting, requirements definition, determination of legality of service, design plan, and final quote, departments and individuals within the departments are modeled as agents that negotiate with each other to reach a mutually agreeable contract for the customer.

- The OASIS system (Optimal Aircraft Sequencing using Intelligent Scheduling) [Ljungberg, 1992] is an air-traffic control system whose purpose is to assist an air-traffic controller in managing the flow of aircraft at an airport. OASIS contains (1) global agents which perform generic domain functions e.g., arranging the landing sequence of aircraft and (2) an aircraft agent for each aircraft in the system airspace. It was trialed at Sydney airport in Australia.

- A proliferation in online auctions led to the need to monitor and bid in multiple auctions to procure the best deal for the desired item. Both of these actions are complex and time consuming, particularly when the bidding times for different auctions may or may not overlap and when the bidding protocol may differ. Anthony and Jennings [2003] describe the development of a heuristic decision making framework that an autonomous agent can exploit in such situations. An agent-based architecture for bidding on the New York Stock Exchange also has been proposed [Griggs, 2000] as well as a trading simulation that merges automated clients with real-time, real-world stock market data [Kearns and Ortiz, 2003].

IV. JACK BE NIMBLE, JACK BE QUICK

To appreciate the process of constructing intelligent agents better, the following example, coded in the JACK™ Agent Language from Agent Oriented Software Group[1], is offered. The JACK™ Agent Language is an agent-oriented development environment built on top of and fully integrated with the Java programming language. It defines new base classes, interfaces, and methods as well as provides extensions to the Java syntax to support new agent-oriented classes, definitions and statements. By enabling an agent to pursue its given goals (desires) and adopt the appropriate plans (intentions) according to its current set of data (beliefs) it follows the BDI model of artificial intelligence.

[1] http://www.agent-software.com/shared/home (consulted August 30, 2004)

The class-level constructs JACK™ employs include Agents, Events, Plans and BeliefSets. Agent classes are used to define the behavior of an intelligent software agent by specifying:

- all internal and external events that it will handle
- events the agent can post internally to be handled by other plans
- events the agent can send externally to other agents
- plans the agent can execute
- beliefsets the agent can refer to

When an agent is instantiated, it waits until it is given a goal to achieve or experiences an event that it must respond to. The types of events an agent responds to include internal stimuli representing events an agent sends to itself or external stimuli which are messages from other agents or percepts that an agent receives from its own environment. JACK™ provides two categories of events.

1. A normal event in which the agent reacts to transient information in the system e.g., the location of the ball in a soccer game. The agent selects the first applicable plan instance for the event and executes only that plan.

2. BDI or goal directed events commit the agent to a desired outcome rather than a specific method to achieve that outcome. In this case the agent selects from a set of plans based on relevancy and applicability. If the selected plan fails to execute, the agent executes an alternative plan until it succeeds or runs out plans from which to choose.

A plan is analogous to an agent's functions i.e., the instructions the agent follows to try to achieve its goals and handle its designated events. Each plan handles a single event, but multiple plans may handle the same event. An agent can discriminate further between plans by executing a plan's *relevant()* method to determine whether it is relevant for the instance of a given event. From those selected as relevant, the agent can further decide which plans are applicable by executing the each plans *context()* method.

An agent's beliefs about the world are stored in a beliefset using a tuple-based relational model. In a Closed World relation the tuples stored are believed to be true, those not stored are assumed false. In an Open World relation both true and false tuples are stored; anything not stored is "unknown". Events can be posted when changes are made to the beliefset and thus initiate action within the agent based on a change of beliefs.

THIS IS HOW IT HAPPENS

With this overview of how agent oriented programming is supported by JACK™ , we can move on to the problem at hand. A relational database contains a field SubjectType within a table named Experiment. Whenever the value of SubjectType is either 'monkey' or 'mouse' the system will detect this state and invoke an agent to change the value to 'animal'. The system contains two agents, Monitor and Updater, two events, Update and UpdateRequest and three plans, SendUpdateCommand, UpdateMonkey and UpdateMouse. Appendix I presents the source code of the system components.

Figure 3 illustrates the flow of action steps. When the driver class, Program, finds a record that contains 'monkey' or 'mouse' in the field SubjectType (box(1)), it invokes the submitUpdateRequest() method of the Updater agent (box(2)). This method, in turn, posts a synchronous UpdateRequest event and invokes the request() method of the UpdateRequest event (box(3)). The UpdateRequest event is added to the event queue of the Updater agent and awaits processing. The Updater agent includes the statement `#uses plan SendUpdateCommand;` which informs the agent what plan it should execute to handle any events it receives. Thus, the system progresses to box(4) where the SendUpdateCommand plan handles

the UpdateRequest event. First the plan instantiates an Update event and invokes the update method of Update (box(5)). Then the SendUpdatePlan "sends" an update event to the Monitor agent and waits for a reply before continuing. The Monitor agent evaluates the relevant() and context() methods of UpdateMouse and UpdateMonkey in order to choose between two plans of action (box (6)). The selected plan executes and updates the value of SelectType to 'animal' for the given record (box(7)). Upon completion of the plan, an @reply with a Finished event is issued which invokes the finished method of the Finished event (box(8)). The @wait_for command in SendUpdateCommand receives this message and interprets the response (box(9)). SendUpdateCommand now terminates and control returns to Program (box(1)). Thus through a series of events and messages agents, the system monitors the database table and when under the proper conditions will trigger a sequence of steps to update the record.

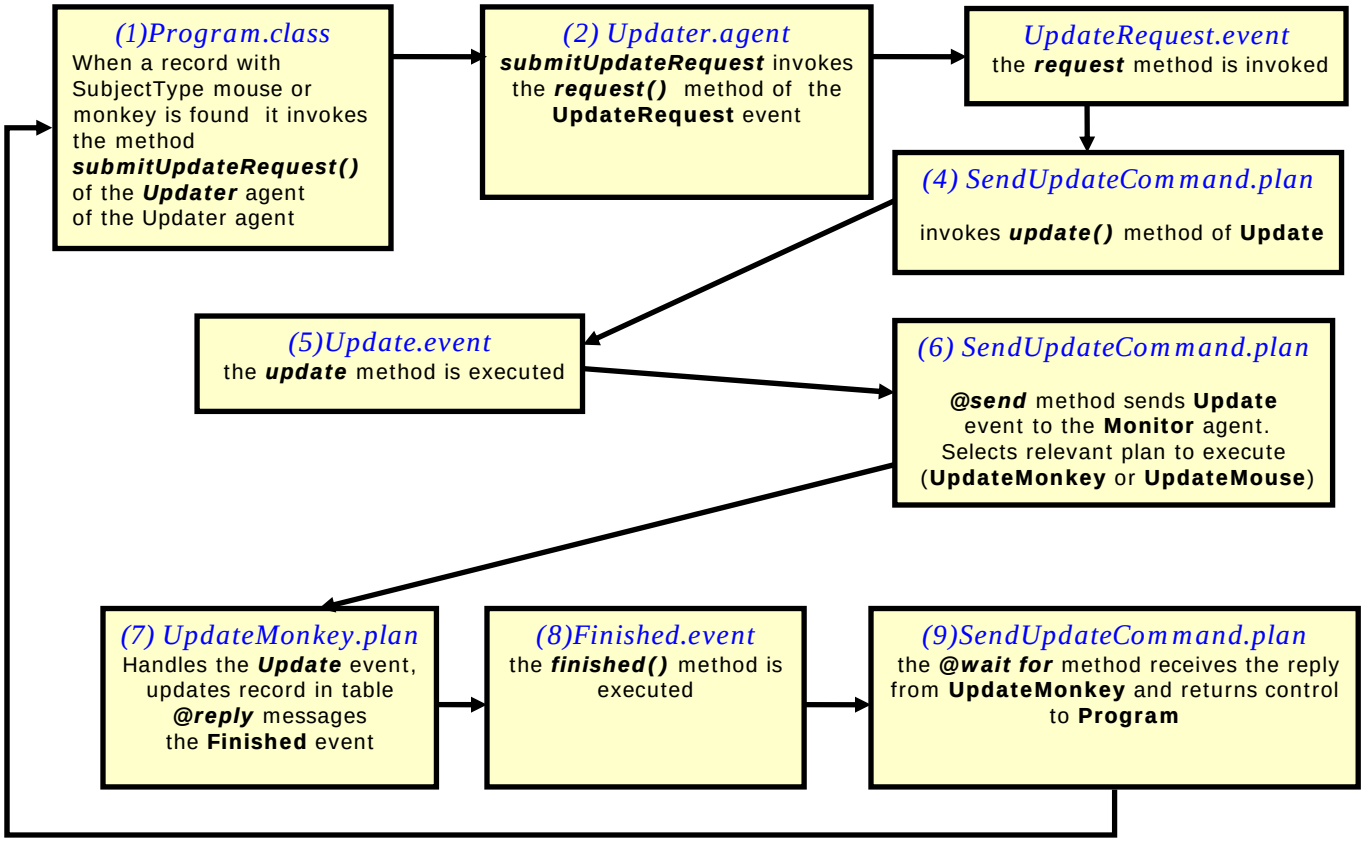

Figure 3. Sequence of Events within the System

IV. THE DOWNSIDE

Despite the significant advances made in the science of agent systems, the pragmatic engineering of such systems is not as well understood (Wooldridge and Jennings, 1998]. Some of the common pitfalls include:

- Agents do not make the impossible possible – they are not a magical problem solving paradigm

- Agents are not a universal solution; in some situations a conventional software development paradigm (e.g., object oriented) may be far more appropriate
- Projects that employ agents because of the hype about agents but with no clear picture of what benefits agents will bring to the project are likely doomed to failure.
- Building a successful prototype with agents does not guarantee it will prove scalable and reliable in solving the full blown, real-world problem.
- The design does not leverage concurrency
- The design contains too few or too many agents
- A multi agent system can not be developed successfully by throwing together a number of agents and letting the system run – anarchy may result. Instead, a great deal of a priori system-level engineering is required, especially for large-scale systems.

Even with a well-designed and implemented MAS, other issues can prevent the acceptance of the system by the user community:

- Cost justification: is it worth the price?
- Security: will data be secure, particularly in a distributed environment? Will an agent respect restrictions from other servers and go only where allowed?
- Legal/Ethical issues: will agents' goals and plans be designed so as not to do anything illegal or unethical? Are there guidelines as to what determines a well-behaved agent? With whom does liability rest for the decisions, actions and/or recommendations of these systems? [Mykytyn et al, 1990].
- Accuracy: can the correctness of the results be guaranteed?
- Acceptance by society: An impediment to the widespread adoption of agent technology is social – for individuals to be comfortable with delegating tasks they must first have trust in agents. [Bradshaw, 1997]

V.CONCLUSION

Agent-based systems technology is a vibrant and rapidly expanding field of academic research and business applications. By providing a new paradigm for designing and implementing systems for a complex, dynamic, and distributed environment where the common currency is negotiation, unplanned for events can be managed in a way that is beneficial to the overall system performance. Agent technology is greatly hyped as a panacea for the current ills of system design and development, but the developer is cautioned to be aware of the pitfalls inherent in any new and untested technology. The potential is there but the full benefit is yet to be realized. Agent technology will achieve its true potential only if users understand its business value [Radjou, 2003]. Much work is yet to be done.

Editor's Note: This article was received on August 8, 2004 and was published on September 8, 2004

REFERENCES

Anthony, P. and Jennings, N.R. (2003) Developing a Bidding Agent for Multiple Heterogeneous Auctions, *ACM Transactions on Internet Technology*, (3)3, pp. 185-217.

Bradshaw, J. (1997) *Software Agents*, Menlo Park, CA: AAAI Press.

Brustolini, J.C. (1991) *Autonomous Agents: Characterization and Requirements, Report* CMU-CS-91-204, Pittsburgh, PA: School of Computer Science, Carnegie-Mellon University,.

Coen, M.H.(1991) *SodaBot: A Software Agent Construction System*, Cambridge, MA: MIT AI Laboratory,

Coolidge, C. (2001) Financial Planning, Forbes – Best of the Web, Spring (167)5, p. 84.

Etzioni, O. (1996) Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web*, Proceedings of the 13th national Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, pp.4-8.

Franklin S.P. and Graesser A.G. (1996) Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents, In M.J. Wooldridge & N.R. Jennings (eds.), *Intelligent Agents III*, Heidelberg: Springer-Verlag.

Gilbert, D., et al. (1995) *IBM Intelligent Agent Strategy*, White Paper, IBM Corporation.

Griggs, K. (2000) An Agent Oriented Business Model for E-Commerce Based on the NYSE Specialist System, *ACM SIGCPR 2000*, Evanston, IL, pp.136-143.

Hayes-Roth, B., et al. (1995) A Domain Specific Software Architecture for Adaptive Intelligent Agents, *IEEE Transactions on Software Engineering*, (21)4, pp. 288-301, April.

Honavar, V. (1999), Intelligent Agents and Multi Agent Systems, IEEE Conference on Evolutionary Computation, Washington, DC,

Jennings, N. R., et al. (1996a) Using ARCHON to Develop Real-Word DAI Applications, *IEEE Expert*, (11)6, pp. 64-70.

Jennings, N. R.,et al. (1996b) Agent-Based Business Process Management, *International Journal of Cooperative Information Systems*, (5)2&3, pp. 105-130.

Kearns, M. and Ortiz, L (2003) The Penn-Lehman Automated Trading Project, *IEEE Intelligent Systems*, (1)6, Nov/Dec, pp. 22-31.

Kendall, E.A.et al. (2000), An Application Framework for Intelligent and Mobile Agents, *ACM Computing Surveys*, (32)1 March.

Ljungberg, M., Lucas, A. (1992) The OASIS Air Traffic Management System, *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence*, PRICAI '92, Seoul, Korea.

Maes, P. (1995) Artificial Life Meets Entertainment: Lifelike Autonomous Agents, *CACM*, (38)11, pp. 108-114, November..

Mykytyn, K., Mykytyn, P., Slinkman, C. (1990) Expert Systems: A Question of Liability?, *MIS Quarterly*,(14)1 March, pp. 27-42.

Parunak, H.V.D. (1999) Industrial and Practical Applications of DAI, in *Multi-Agent Systems,* (ed. G. Weis), pp. 377-421, Cambridge, MA:MIT Press,

Perriolat, F., et al.(1996) Using Archon, Part 3: Particle Accelerator Control, *IEEE Expert,* (11)6, pp. 80-86.

Pollack, M.E. and Ringuette, M. (1990) Introducing the Tileworld: Experimentally Evaluating Agent Architecture, *Proceedings of the 8th National Conference on Artificial Intelligence* (AAAI-90), pp. 183-189, Boston, MA.

Radjou, N. (2003) Software Agents in Business: Still An Experiment, *Forrester Research Brief*, March 27.

Russell, S. and Norvig, P. (1995) *Artificial Intelligence: A Modern Approach,* Englewood Cliffs, NJ: Prentice-Hall,.

Shoham, Y. (1993) Agent Oriented Programming, *Journal of Artificial Intelligence,* (60)1, pp. 51-92, 1993.

Shortlie, E.H. and Rhame, F.S. et al. (1975) Mycin: A Computer Program Providing Antimicrobial Therapy Recommendations*, Clinical Medicine,* (34), 1975.

Simon, H., (1957) Models of Man: Social and Rational; Mathematical Essays on Rational Human Behavior in a Social Setting, New York, Wiley.

Sycara, K.P. (1998) Multiagent Systems, *Artificial Intelligence*, (10)2, pp. 79-93.

Turban, E. and Aronson, J.E. (2001) *Decision Support Systems and Intelligent Systems 6/e*, Upper Saddle River, NJ: Prentice Hall,

Wooldridge, M. (2002) *An Introduction to MultiAgent Systems,* Chichester, UK: Wiley.

Wooldridge, M. and Jennings, N.R. (1995) Intelligent Agents: Theory and Practice, *The Knowledge Engineering Review,* (10)2, pp.115-152.

Wooldridge, M. and Jennings, N.R. (1998) Pitfalls of Agent-Oriented Development, *Proceedings of the 2nd International Conference on Autonomous Agents* (Agents 98), Minneapolis/St. Paul, MN, pp, 385-391.

APPENDIX I. SOURCE CODE FOR AGENTS, EVENTS AND PLANS

```
public class Program
{
 public   static   void   main(String   args[])throws   SQLException,
ClassNotFoundException,
   NullPointerException
   {   Monitor monitor = new Monitor("ResearchMonitor");
       Updater updt = new Updater("ResearchUpdater");

       //SQL CODE TO CONNECT TO DATABASE
       // AND SEARCH TABLE FOR RECORDS WITH SUBJECTTYPE EQUAL TO mouse
OR monkey

       ResultSet r = s.executeQuery("SELECT ExpID,SubjectID,SubjectType
FROM
       EXPERIMENT  WHERE  SUBJECTTYPE  =  'monkey'  OR  SUBJECTTYPE  =
'mouse'" );

       while (r.next())
    {    String eid = r.getString("ExpId");
         String stype = r.getString("SubjectType");
         System.out.println("ID #:"+ eid + " Subject ID:" +
               r.getString("SubjectID") + " Subject Type:" + stype);
         updt.submitUpdateRequest("ResearchMonitor", eid, stype);
    };
  }
```

```
}

public agent Updater extends Agent {
   #handles external event UpdateRequest;
   #sends event Update;
   #uses plan SendUpdateCommand;
   #posts event UpdateRequest ev;

   public Updater(String name) { super(name);}
   public void submitUpdateRequest(String monitor, String
       expID, String stype){
     postEventAndWait(ev.request(monitor, stype, expID));
   }
}

public agent Monitor extends Agent {
    #handles external event Update;
    #sends event Finished;
    #uses plan UpdateMonkey;
    #uses plan UpdateMouse;
    #posts event Update ev;

    public Monitor(String name){ super(name);}
    String stype, eid;
    public void setVars (String s, String e) {stype = s; eid = e; }
}


public event Update extends BDIMessageEvent   {
    public String stype, eid;
    #posted as
    update (String s, String e) {stype = s; eid = e; }
}

public event UpdateRequest extends BDIGoalEvent {
    public String monitor, stype, eid;
    #posted as
    request (String m, String s, String e){
      monitor = m; stype = s; eid = e;
    }
}

public event Finished extends BDIMessageEvent {
    public String stype, eid;
    #posted as
    finished(String s, String e) {stype = s; eid = e;}
}


public plan SendUpdateCommand extends Plan {
    #handles event UpdateRequest preqev;
    #sends event Update ev;

    body()
    {try
     {   Update q = ev.update(preqev.stype, preqev.eid);
         @send (preqev.monitor,q);
```

```
         @wait_for(q.replied());
         Finished response = (Finished) q.getReply();
         System.out.println(agent.name()+" has been updated in
               SendUpdateCommand "+response.eid);
     }
     catch (NullPointerException npe) {
        System.out.println("preqev.eid "+preqev.eid);
     }
   }
}


public plan UpdateMonkey extends Plan {
    #handles event Update handleUpdateEvent;
    #sends event Finished fev;
    #uses interface Monitor self;

    static boolean relevant(Update evRef)
    { return  ((evRef.stype != null) && (evRef.stype.length() > 0)); }

    context(){   handleUpdateEvent.stype.equals("monkey"); }

    body()
    { self.setVars(handleUpdateEvent.stype, handleUpdateEvent.eid);

    // SQL CODE TO CONNECT TO DATABASE AND
    // UPDATE SUBJECTTYPE FROM monkey TO animal
     s.executeUpdate("UPDATE EXPERIMENT SET SUBJECTTYPE='animal‘
       WHERE EXPID='" + handleUpdateEvent.eid + "'" );
     }
     @reply (handleUpdateEvent,fev.finished(self.stype, self.eid));
    }
}

public plan UpdateMouse extends Plan {
    #handles event Update handleUpdateEvent;
    #sends event Finished fev;
    #uses interface Monitor self;

    static boolean relevant(Update evRef)
    { return  ((evRef.stype != null) && (evRef.stype.length() > 0)); }

    context(){   handleUpdateEvent.stype.equals("mouse"); }

    body()
    { self.setVars(handleUpdateEvent.stype, handleUpdateEvent.eid);

    // SQL CODE TO CONNECT TO DATABASE AND
    // UPDATE SUBJECTTYPE FROM mouse TO animal
     s.executeUpdate("UPDATE EXPERIMENT SET SUBJECTTYPE='animal‘
       WHERE EXPID='" + handleUpdateEvent.eid + "'" );
     }
     @reply (handleUpdateEvent,fev.finished(self.stype, self.eid));
    }
}
```

ABOUT THE AUTHOR

Ira S. Rudowsky is Associate Professor of Computer and Information Science at Brooklyn College of the City University of New York at the undergraduate and graduate level. He also teaches business/technology courses in the Department of Economics. His research interests include database systems for multimedia data, intelligent agents, and management information systems. For over twenty-five years he managed and developed application software for financial services industry in companies such as Merrill Lynch, Bankers Trust, and the New York State Office of the Comptroller. He is a member of AIS, ACM, and IEEE Computer Society.

Communications of the Association for Information Systems

ISSN: 1529-3181