Software Agents Finally Go To Market

Ross A. Gagliano, *Member, IEEE Computer Society*

Abstract - Within the zeitgeist of so-called autonomous software (agents, aglets, applets, bots, servlets, etc.), there is increasing emphasis on brokering (competing, cooperating, negotiating) vis-a-vis traditional distributed electronic problem-solving (browsing, communicating, coordinating, data collection). Towards these purposes, market models have been defined, created, and tested that are poised for possible use in e-markets of several forms (auction, barter, exchange), and key aspects for agent representation in software have been identified.

Keywords: DSC, ISY

I. Introduction

We begin by defining what is generally meant by semi-autonomous software, and then briefly describing a few of the various types; e.g., aglets, applets, bots, knowbots, softbots, software assistants, and servlets, etc. Such software is indeed semi-autonomous inasmuch as no mobile code currently acts in a totally independent fashion [14]. Nonetheless, the expanded use of software agents promises to radically transform modern commerce, perhaps developing into a technology as important as graphical user interfaces (GUIs), client/server platforms, and even the object-oriented approach itself [6].

Suffice it to say that this technology appears to be moving beyond distributed objects with the expectation that software agents operating in e-markets will transform not only the way this enterprise operates [20], but also the manner in which its associated information systems are modeled [18]. The dual aim of this paper is to provide a perspective on the current state of affairs in software agent technology, and to explain how earlier research into various market models can assist in their implementation as they assume a more prominent role as e-brokers and e-negotiators [7,9,10,11,12,13].

II. Applets

First on the scene of semi-autonomous software was the applet, or application code, that started the trend of software that was capable of "migrating" from server to client where it would then run executable threads, typically Java, within the host [24]. Normally, several other things would occur when a Web browser causes a Java applet to be sent to a host. The first is that the applet installs a security manager that both restricts its actions and protects its activities. The second thing that the applet does is to create a class loader that downloads any requested class files from the server. One major shortcoming is that an applet does not carry along its state when migrating as do agents and aglets that will be further explained later [4].

III. Bots

A second form of semi-autonomous software is known by several names; e.g., bot, softbot, knowbot, or even a software assistant [6]. The most common task for such software is searching the Web for specific data and creating databases that are accessible by search engines. However, bots are now being installed at customer service call centers to interact with human clients through either voice recognition or computer-generated images. At an estimated cost of $33 per call [5], customer service centers have embraced the use of such specialized bots equipped with artificial intelligence (AI) heretofore found only in research. When a customer enters inquiries in plain text into chat boxes, these bots respond in "natural language," anticipating questions, even those inappropriate or irrelevant, to provide the best possible answers. Thus far, bot search results have been found to be more accurate than those of human counterparts, plus bots operate at a lot less cost.

Many large commercial sites, however, oppose the use of bots for several reasons [19]. Besides their obvious inability to respond to advertising, bots react faster than most servers and can disrupt the site operation. In a recent court case, a judge sided with the online auction site eBay in enforcing rules that dictate where bots may go and what they may do [5]. As a result, bots are no longer permitted at many auction aggregator sites.

IV. Servlets

This type of code is sent from client to server, just the reverse of the applet. The name servlet is derived from "<u>serv</u>er app<u>let</u>," and these are also being implemented in Java. As applets have became more popular, servlets have emerged as the preferred method for server applications. Earlier, Common Gateway Interface (CGI) programs and scripts were the only interface (or gateway) for user-server interaction, such as databases applications. CGI also

The author is a retired computer science professor and now a consultant. See web site at http://www.RosarioPub.Com.

allowed real time generation of HTML documents where the information could be changed dynamically to fit the needs of a given task. Prior to Java, CGI was written principally in Perl, C/C++, Visual BASIC, Tcl or other shell scripts. But as the number of Java-enabled browsers has increased, the number of servlets written in CGI scripts has decreased [15].

V. SOFTWARE AGENTS

To the layman, a software agent is simply a piece of code that "thinks;" i.e., it is a "smart" applet. Technically, a software agent is semi-autonomous software that not only can delegate tasks, but is also adaptive, long-lived, and proactive. Suffice it to say, it can transfer itself from one machine to another via a network, and start and stop its own execution on these distant machines. In addition to this mobility, a software agent contains both its executable code and its state description [16]. Henceforth, in this paper, we will use the term "agent" to mean "software agent."

However, an agent is more than just mobile code [3]. In fact, agents have also been labeled "intelligent," "sensible," and "representative." Intelligent agents are endowed with AI, as explained above, but do not necessarily have to be mobile [22]. Sensible agents, on the other hand, possess the capability to interact with other agents [2]. Representative agents, like personal software assistants, are pieces of code that represent or stand in for humans; and, depending on their instructions, make decisions and even consummate deals. Representative agents can be mobile or intelligent or both, but generally are not mobile.

An agent has three essential properties: autonomy, reactivity, and communication. Autonomy means that an agent exercises exclusive control over its behavior and state. Reactivity is its ability to sense changes in its environment and respond to them. Lastly, even the most basic software agent has the ability to communicate with other users, agents, or objects. An agent is, therefore, a distinct category of software that incorporates local knowledge of tasks and resources in operating across a network alone or in a group.

A major difference between applets and agents is itinerary. Applets usually migrate from just one site to another, whereas agents typically employ an itinerary to visit several sites sequentially. It is customary, therefore, for an agent to collect information from many sites. A good example is a network maintenance agent that periodically checks the auxiliary memory on various machines in a network, collects information about the status of their disks, and returns to the point of origin to create a report.

VI. AGLETS

One of the more recent developments in mobile agents is the aglet. An agent applet (hence, ag-let) is Java-based software that, once launched, "decides" where to go and what to do, independent of itinerary or external control [25]. Aglets can receive requests from external sources, but are not compelled to comply. They are mobile Java programs that can halt execution at one site, "determine" where to travel next in the network (again like agents, with both code and state intact), and resume execution at the new site [15].

An important aspect of aglets is that they represent the next step in the evolution towards a higher level of software abstraction, particularly for network computing, even possibly answering the question of 'what comes after object-oriented (OO) technology?' Aglets appear to embody this paradigm shift better than applets, servlets, or remote procedure calls (RPCs) because they support the notion of the 'network as a computer' in which all accessible computing resources are "visualized" as being in one single virtual computational space.

VII. AGENT ACTIVITIES

Taken collectively, all of these forms of semi-autonomous software possess some unique capabilities: they can represent knowledge, make inferences, and even "learn" [4]. Moreover, they do improve efficiencies while dramatically cutting costs, plus they can be integrated easily into existing software systems, including networks for electronic commerce. However, there is a pressing need for a more comprehensive framework through which agents can be developed and managed that are capable of brokering and negotiating.

Traditional tasks involve agents searching for information on behalf of a user, and performing some type of transaction when the appropriate situation is encountered. Currently, the more common agent tasks include: searching, monitoring, distributing, information brokering, parallel processing, and gaming [4].

1. Searching. This involves necessary, but costly and time-consuming data filtering and analysis due to the ever increasing Internet information. Rather than an individual user searching numerous Web pages and subsequently filtering irrelevant information, agents are able to visit many more sites, search each more thoroughly, then build link sets to match search criteria.

2. Monitoring. When needed information appears across time but not space (memory of various machines on the same network), as new information is published on the network, agents can be sent to access it as it becomes available. A popular example is an agent that acts as a stock market monitor, waiting for a particular stock to attain a certain price, then buying or selling shares on behalf of its user. The user does not have to stay connected to the network as the agent is able to wait for a subsequent reconnection to make its report.

3. Information Distribution. Targeted information dissemination is another current use of agents to distribute interactive news or advertise to specific (interested?) parties. Unfortunately, indiscriminate electronic information distribution is viewed like "spam" e-mail, insuring that incoming agents charged with such a task are more thoroughly screened.

4. Information Brokering. This is the case where agents gain information through interaction with other agents. For example, to schedule a meeting, one agent is sent to interact with the agents of other potential participants. The agents then "negotiate" to establish meeting parameters by exchanging information about users' schedules, preferred meeting times and places.

5. Parallel processing. Given that agents can spawn subagents, as will be discussed below, this technology offers a serious alternative to the usual parallel processing. For instance, a computational task requiring a large amount of CPU time is broken into smaller pieces, each represented by a subagent. A dedicated infrastructure of subagents then negotiates for the necessary processing capability assigned to each sub-computation.

6. Simulation and Gaming. This last activity is either viewed as frivolous entertainment or serious operations research. Here, agents represent game players, competing with one another using strategies set by the players. For more serious work, agents are charged with exploring for optimum solutions, while as entertainment, agents are entrusted with real money.

More information on commercial agent development can be found on the websites of various corporations including Crytaliz, FTP Software, IBM, Microsoft, Mitsubishi, Oracle, Toshiba [27]. Agent-based research has also been underway at the following U. S. universities: Caltech, Carnegie-Mellon, Cornell, Dartmouth, Maryland, Michigan, MIT, Purdue, Stanford, and Washington; and in European universities at Berlin, Grenoble, Kaiserlautern, and Vrije; plus Canadian universities at Carleton and Ottawa.

VIII. Agents in Markets

Beyond the standard applications, several issues must be resolved prior to the implementation of agent brokering and negotiating tasks. First, agents must be able to recognize the type of market encountered; and, second, they must be prepared to react to the different types of markets and various market settings; i.e., local conditions for a particular market. For agents to perform this recognition, a scenario similar to that described earlier with bots would be employed permitting an agent entering a site to request a text file containing a description of the market type and other special local conditions.

To illustrate these points, we now present three simple examples that show how agents could operate in three different market settings: a simple exchange (one-to-one bidder and offeror); an auction (many-to-one bidders to a single offeror); and a challenge process (similar to bartering in that no funds are involved). In the first two examples, the same sequence of events is followed: sellers (offerors) and buyers (bidders) self identify; sellers make offers and buyers make bids on those offers; and, transactions are consummated, if possible.

A. Simple Exchange Model

This first exchange process is as follows: agents identify themselves as willing offerors; offerors pair with willing bidders; and agent offeror-bidder pairs negotiate. In each negotiation, the offeror sets a selling price (the current minimal value it will accept); and a bidder makes a bid (the maximal value that currently it is willing to give). A primary test for a bidder is to determine what portion of its available funds it will bid, although there is no minimum bid. Bidders calculate bids as a portion of their own particular ceiling price and how seriously they "desire" an item. The difference between selling price and bid is then determined by both parties, and no exchange is made unless the difference is reduced to zero through iterative bargaining steps.

At each step, a new selling price, a new bid, and the new difference are calculated until there is either agreement or a new round is entered. The selling price is reduced by an amount based on how "eager" the offerer is to sell while the bid is increased based on how "eager" the bidder is to buy. Such calculations can also be performed using fuzzy arithmetic [1]. An exchange is made if and only if the bid equals or exceeds the current selling price. When an exchange is made, the amount of the final bid transfers from the bidder to the offeror, and the item under negotiation is removed from the control of the offeror to that of the bidder.

From an earlier exchange example between two agents [11], we present the results in the negotiations of each step of an exchange process as shown in the following Table I. Note that in a series of six recalculations, the offering agent lowered the selling price by 8 units of funds (from 20.1 to 12.1) while the bidding agent raised its bid by just over 2 units of funds (9.9 to 12.13) until the difference went to (actually below) zero. The final agreed upon price between the two agents was 12.13 units of funds. As might be expected, these values converged because both agents remained sufficiently "eager" for the exchange, the offeror apparently more eager than the bidder as its changes were larger. The less eager or uncertain the agents, then the less likely the difference between selling price and bid will decrease. This latter case would probably result in "no sale."

Table I. An Exchange Example.

Selling Price (S)	Difference	Amount Bid (B)	Exchange?
20.1	10.2	9.9	No
19.1	8.9	10.2	No
17.9	7.1	10.8	No
16.5	5.4	11.1	No
14.9	2.7	12.2	No
13.5	0.9	12.6	No
12.1	-0.03	12.13	Yes

Table II. An Auction Example.

Agent	Bid	Minimum Bid	Exchange?
A	15	35	No
B	25	35	No
C	30	35	No
A	25	30	No
B	30	30	No
C	40	30	Yes

B. Agents in Auctions

In an auction, several agents bid on a single item offered by one other agent. The bidding agents must "decide" whether to bid and how much to bid, and these decisions are based on what items are available and the amount of available funds. Again, an important variable is the “eagerness” to participate which can be a function of the agents’ task requirements and possible time deadlines. We exclusively employed first-price, sealed-bids auctions; i.e., the values of all bids are not made “known” to all agents. In reality, there are a great variety of auction types, including open versus sealed-bid, English (monotonically increasing bids) versus Dutch (monotonically decreasing bids), first-price versus Vickery (second-price), and others.

In our simulations, if an agent “anticipates” being unsuccessful in completing a task by its scheduled completion time, an additional incentive in bidding is provided. In such cases, bidding agents "escalate" attempts to obtain items by increasing their bids, and this eagerness is a function of the difference between currently computed and originally scheduled completion times. At each stage of an auction when agreement is not reached, participating agents re-compute their requirements and make new bids based on a revised eagerness. Agents are also able to increase their bids by offering unnecessary resources through subsequent auctions. In the next table, a series of actions is shown that is similar to the activity of the well known auction site eBay.

In Table II, three agents (A, B, and C) bid in two rounds on an item held by a fourth offering agent (D). The bids are sealed in that no agent “knows” the others’ bids. In the first round, the offeror D set a minimal acceptable value of 35 which no bidding agent met. In the second round, the bidding agents increased their bids while the offeror reduced the minimum bid or its current acceptable amount. Although two agents (B and C) then bid more than the new minimum, the item was “won” by agent C because it had the highest bid. The amounts of increase by the bidders (A, B, C) and the decrease by the offeror (D) were calculated based on the eagerness of each plus factors associated with their requirements and deadlines. Again, the less eager the agents, the less likely agreement is reached.

C. Challenge Process

The third market example is similar to bartering, and e-commerce appears to be an ideal forum for this form of agent technology. For example, a person who wants to exchange opera tickets, computer peripherals, or musical instruments, etc., for other specific items of value would launch an agent to find sites of other interested parties to negotiate a possible exchange.

Rather than using such direct bartering, the challenge process that we developed allowed the shared use of specific resources [9]. We called it a challenge ring (CR) model because it involves challenges for resources arranged in a network ring topology. Specifically, we performed experiments comparing results of the CR to two more familiar configurations: a queue-like server ring (SR) and a "no-wait" mover ring (MR). In all three cases, if an agent's current needs matched the resource at a site in the ring, and that resource was available (not currently being used), then the agent took control of it. If the resource was being used, different things happened based on the particular configuration as follows.

In the CR, a current user was “challenged” for use of the resource according to game theoretic strategies [26] and the “cooperativeness” of the strategy of the agents involved. In general, the more cooperative an agent’s strategy, the more likely that it succeeded; i.e., it gained access to resources and completed on time. In the SR, agents waited in a queue, observing either a first-in-first-out (FIFO) or a last-in-first-out (LIFO) discipline for the resources. In the MR, agents did not wait if desired resources were unavailable (not there or in use), but "moved " or migrated to the next site in the ring (the ring structure became the itinerary).

Further explanation of this model is too involved to be provided in this short paper. However, several detailed examples do appear in [9]. Suffice it to say that the challenge experiments did yield some interesting insights, particularly into the dichotomies of wholes (systems) and parts (agents).

For example, individual agent outcomes were often at odds with collective system performance across a wide array of inputs: ring configurations, agent mixes, arrival rates, rules of service (SR and MR) and combinations of strategies (CR). Yet the advantage was that individual agent progress and activities could be monitored. Additionally, novel system measures were also created that proved to be effective indicators; e.g., the ratio of agent completion times in a congested ring compared to the minimum time required in a less busy (relatively empty) ring; and the proportion of agents completing (both on time and late) versus those unable to complete.

IX. Electronic Market Models

Now that we have described how agents would operate in three different market settings, let us look at several issues involved in the basis for their protocols and the analysis of their subsequent design and coding. First, we found that a market provides a natural setting for developing decentralized control and "hostless" resource allocation. Second, this development of markets employing individual entities anticipates a discrete element methodology (DEM) that is easily extended to software object and agent technologies.

A. PARC Market Models

Market protocols for system control and resource allocation are not new as they have been investigated for some time. Two research projects occurred at the Xerox Palo Alto Research Center (PARC) in the late 1980's involving distributed computation. The first was a joint PARC-MIT activity called Project Enterprise in which a "market-like" scheduler used a bidding scheme to allocate processors for complex, distributed scientific calculations [21]. The second, called Project Spawn, also involved computational experiments that were conducted on a network of graphical workstations [23].

Although Spawn capitalized on earlier Enterprise concepts, there were at least three major differences between the two projects. First, instead of using the Enterprise bidding scheduler, Spawn employed an auction. Second, although both involved computational tasks, Spawn was restricted mainly to large-scale concurrent Monte Carlo simulations. Third, and perhaps most important, Spawn drew its name from the UNIX (Web operating system backbone) feature that allows tasks to decompose into sub-tasks (or "spawn") for the purpose of fitting fragments of calculations onto, thus utilizing to the maximal extent, networked high-performance workstations.

The motivation behind these PARC projects was that as networks of more powerful workstations were installed, their computing resources became idle for fairly long periods of time; i.e., nights and weekends. The researchers conjectured that these resources could be utilized more effectively if their allocation was dynamic and flexible, and their control supported concurrent applications with remote access through rapidly evolving, more extendable networks.

B. GSU Market Models

Independent of, yet similar to and during the same period of time as the work at PARC, we at Georgia State University (GSU) in Atlanta used simulation to investigate decentralized control in generalized computing environments [7,9,10,11,12,13]. We designed and built several market protocols including sealed-bid auctions, single (with funds) and barter (no funds) exchanges, and the challenge protocol described above.

Our primary guiding principle was that computing resources, such as processors, memory, remote access, and peripheral devices, need not be allocated through priority schemes, controllers, or schedulers, but utilized through simple yet self-organizing systems (hence, markets). Some of these ideas originated in our earlier work involving systems synchronization, discrete cybernetics, and organizational theory [8].

Simulated computing tasks (in a form like agents) were then programmed to: "bid" through auctions; "exchange" by bartering or buying; or "share" through challenges in order to gain needed resources. Simulated system configurations, while greatly simplified, are applicable to a wide variety of environments including single and multiple machine architectures, various operating systems, and different network configurations.

C. Discrete Representation

Because these market protocols were based on individual surrogates (agents) operating in the so-called hostless, or decentralized, environment, our discrete design approach (the DEM) was apparently well suited for two reasons. First, the DEM better accommodated the representation of competition and cooperation of individuals, phenomena essential to decentralization and self-organization. Second, market functions in this series of models of agents buying, selling, exchanging, and contending were more readily and realistically captured through the DEM [7].

Needless to say, discrete representation of markets dates back at least to a seminal treatise on the economic theory of games [26]. This trend continued in some oligopolistic models [36], and is still evident in recent cases [13,21,23]. A collateral contribution of this discrete evolution was an identification of the dichotomy between micro specifications (individual traits or discrete parameters) and macro behavior (system global variables) [12].

With the DEM, market models are prescribed in which each participating agent (buyer or seller) is defined by the following traits: connections (identification of agents pairs that "connect" through transactions); strategies (formulation of individual plans to buy or sell); memory (retention of information about previous transactions); reputation (how previous transactions are perceived by other agents); disposition (eagerness to participate); and intensity of participation (aggressiveness). Not all of these traits are used in all models, however.

X. Objects Versus Agents

Indeed, the implementation of DEM concepts follows an obvious progression from procedures to objects to agents. We discovered that implementing discrete agents provides a way to think about solving problems in networks that fits more closely with reality; i.e., users “send” representatives electronically from site to site to accomplish needed tasks. Thus, we found that models of discrete entities automatically led to distributed systems that are more robust and adaptive.

The object-oriented (OO) approach enhanced this implementation by prescribing objects that produced systems that are likewise adaptive and distributed. OO control functions lend themselves well to agent activation, deactivation, coordination, synchronization, and selection, as well as system adaptation, learning, and evolution. In this fashion, agent paradigms continue the OO focus on evolving system complexity through individual component change.

We recognize that agents and objects have many similarities and differences. They are similar in that they both can be organized into hierarchies that exploit inheritance; e.g., a portfolio managing agent can be an instance of a personal assistant class. Also, while agents can advertise their services using different means, like objects their implementation of services is transparent; i.e.; the encapsulation of particular behavior. Finally, two agents could respond to the same service request differently without the requester knowing about such differences, resembling object information hiding.

On the other hand, agents are different from objects in several crucial ways. First, agents manipulate objects to perform tasks; and, in the process, agent behavior is modified either through “learning” or the influence of other agents. Second, unlike objects, agents "cooperate" to solve problems using both task and domain-level protocols, often obtaining solutions without complete or consistent knowledge or data. Lastly, OO notions of class, association, and message are strongly biased towards software implementation, thus are not sufficiently expressive to capture market rules, or agent constraints and goals.

XI. Agent Oriented Network Software

In our opinion, agents can assume the next step in the evolution of the OO paradigm, particularly for networks. An agent-oriented approach is a more comprehensible paradigm for constructing e-markets, as well as supplying network techniques that are enabled, but not driven, by object technology. The latter approach is based on active, goal-seeking agents rather than static, passive objects.

As networks become more complex, and the size of OO systems grow, the number of messages between objects can increase until "controller objects" become performance bottlenecks. Rather than strict dependence on lower-level communication, agents participate in higher-level task-oriented dialogues using domain knowledge in conjunction with interaction protocols. In this fashion, lower-level communication can be reduced drastically with agents using higher-level dialogues to exchange procedural and declarative knowledge along with detailed state information.

On the other hand, agent network applications raise other concerns, not the least of which is security. As with any established technology that gives non-native software access to native resources, viruses and hackers pose a very serious threat. Although difficult, this problem is not insoluble; e.g., existing security mechanisms, such as those provided by Java, offer protection from such attacks [17]. However, there may be another security aspect not as straightforward nor as easy to solve; that is, protecting agents from malicious hosts. If agents carry financial (credit card or e-cash) information embedded in their state, 'e-pickpockets' pose a new threat. As hosts upload, say, aglets with their class files and states, such private information could be stolen at a particular site or in transit, or even altered.

But assuming these more complex security issues can be addressed, market agent solutions are preferable over traditional client/server models or other mobile code such as applets or servlets. And, for most distributed processing models, none of the agent disadvantages appears truly overwhelming. Finally, software agents induce a more powerful framework for common network services; and while alternatives have been suggested for each disadvantage, no single alternative currently exists to the cumulative functionality of agents.

XII. Conclusions

A variety of semi-autonomous software has been developed and is becoming more widely accepted in e-commerce. Network developers have shown interest in moving beyond the status quo of objects, servlets, and client/server platforms to more novel electronic enterprise solutions. Despite these developments, we suspect that there is yet unmet demand for software agents that are capable of brokering and negotiating. Major handicaps apparently are that agents are less well known and perceived as more difficult to code.

Our experience has shown that generally once an acceptable model is developed, the programming becomes relatively straightforward. Moreover, in conjunction with an expanded use of aglets, the programming language of choice is becoming Java. Irrespective of language issues, agents involved in markets must be able to move from site to site, determine the type of market involved, and participate in a rational, albeit constructively “selfish” way.

In this paper, three market models were presented to illustrate agent brokering and negotiating in a simple fashion. These models were developed earlier in response to a need for more flexible control and dynamic allocation of computing resources. Two aspects of our approach to agent design are unique. First, the software components are provided the means to "cooperate" in carrying out complex functions. Second, using discrete elements based on the DEM, simpler yet more robust markets are developed, leading to improved resource availability and system performance.

Lastly, modern networks, including those associated with multi-, parallel, and distributed processing, are gradually moving away from centralized control towards more decentralized and even market-based approaches. In our ever interconnected world of computing and communicating, software agents are better positioned for future electronic markets. Over the past few years, highly distributed yet increasingly demanded information assets have become more accessible via the Internet, starting with the early successes of systems like Gopher and Veronica, and moving towards ever more diverse forms of e-commerce as demonstrated by online sites such as eBay, E-trade, Priceline, and others. It is now feasible that software agents will assume a greater role in such markets.

XIII. References

1. Alem, L., R. Kowalczyk, and M. Lee, "Recent Advances in e-Negotiation Agents," ***Proc. of the SSGRR-2000 International Conference on the Advances in Infrastructure for Electronic Business, Science, and Education on the Internet***.

2. Barber, K, A. Goel, D. Han, T. Liu, C. Martin, and R. McKay, "Sensible Agents Capable of Dynamic Adaptive Autonomy: An Architecture and Infrastructure to Support Reconfiguration of Problem Solving Frameworks for Electronic Commerce Applications," ***Proc. of the SSGRR-2000 International Conference on the Advances in Infrastructure for Electronic Business, Science, and Education on the Internet***.

3. Bellavista, R., A. Corradi, and C. Stefanelli, "Mobile Agent Middleware for Mobile Computing," ***IEEE Computer***, Vol. 34, No. 3, 73-81, March 2001.

4. Caglayan, A. and C. Harrison. **Agent Sourcebook: A Complete Guide to Desktop, Internet and Intranet Agents**. Wiley Computer Books, 1999.

5. Denison, D., "Bots for Business," *Boston Globe*, C1, May 13, 2001.

6. Farhoodi, F. and P. Fingar, "Competing for the Future with Intelligent Agents," ***Distributed Object Computing***, Oct-Nov 1997.

7. Gagliano, R. A., M. D. Fraser, and M. E. Schaefer, "Simulation of a Market Model for Distributed Control," *21st Annual Simulation Symposium Proceedings*, IEEE Computer Society Press (ISBN 0-8186-0845-5), 171-187, March 1988.

8. Gagliano, R. A. and K. S. Lebkowski, "Cybernetics and a General Theory of Organization," in ***Progress in Cybernetics and Systems Research***, Vol. VII, Hemisphere Publishing Co., 143-148, 1979.

9. Gagliano, R. A. and M. E. Schaefer, "Decentralized Control of Access to Resources in a Network: Interdependence of Strategies in a Challenge Ring Model," in M. Fraser (ed.) **Advances in Control Networks and Large Scale Parallel Distributed Processing Models**, Ablex Publishing Co. (ISBN 0-89391-647-1), Chapter 5, 145-170, 1991.

10. Gagliano, R. A., M. D. Fraser, and M. E. Schaefer, "Auction Allocation of Computing Resources," ***Communications of the ACM***, Vol. 38, No. 6, 88-99, June 1995.

11. Gagliano, R. A. and P. A. Mitchem, "Valuation of Network Computing Resources," in S. Clearwater (ed.) **Market-Based Control: A Paradigm for Distributed Resource Allocation**, World Scientific (ISBN 981-02-2254-8), Chapter 2, 28-52, 1996.

12. Gagliano, R. A., "Markets: Paradigms for Distributed, Parallel and Multiprocess Control," *Adaptive Distributive Parallel Computing Symposium Proceedings*, Dayton, OH: Wright Laboratory, 91-101, August 1996.

13. Gagliano, R. A. and M. D. Fraser, "Software Agents and the Role of Market Protocols," *35th Annual ACM Southeast Conference Proceedings* (ISBN 0-8979-1925-4), 88-90, April 1997.

14. http://www.artima.com/underthehood/point.html

15. http://www.trl.imb.co.jp/aglets

16. http://www.javaworld.com/javaworld/jw-04-1997/jw-04-agents.html

17. Karjoth, G., D. Lange, and M. Oshima, "A Security Model for Aglets," ***IEEE Internet Computing***, Vol. 1, No. 4, 68-77, Jul-Aug 1997.

18. Kumova, B. "Software Design Patterns for Intelligent Agents," ***Proc. of the SSGRR-2000 International Conference on the Advances in Infrastructure for Electronic Business, Science, and Education on the Internet***.

19. Luh, J., "No Bots Allowed," ***Interactive Week***, Vol. 8, No. 15, 62, Apr. 16, 2001.

20. Ma, M. "Agents in E-commerce," ***Communications of the ACM***, Vol. 42, No. 3, 79-80, Mar. 1999.

21. multiagent @ http://www.parc.xerox.com/istl/groups/iea/topics/multiagent.shtml.

22. Papazoglou, M. "Agent-Oriented Technology in Support of E-business: Enabling the Development of 'Intelligent' Business Agents for Adaptive, Reusable Software," ***Communications of the ACM***, Vol. 44, No. 4, 71-77, April 2001.

23. "PARC Brings Adam Smith to Computing," ***Science***, Vol. 244, 145-146, April 14, 1989.

24. Sommers, B. "Agents: Not Just For Bond Anymore," ***JavaWorld***, April 1997.

25. Venners, W. "The Architecture of Aglets," ***JavaWorld***, April 1997.

26. von Neumann, J. and O. Morgenstern. **Theory of Games and Economic Behavior**. Princeton University Press, 1953.

27.http://msdn.microsoft.com/workshop/imedia/agent/default.asp.

Ad Patres. This paper is dedicated to the memories of my grandfather who emigrated to the U.S. from Italy 96 years ago and to my father who returned to Italy for study 74 years ago.