Agent Communication Network -

A Mobile Agent Computation Model for Internet Applications

Timothy K. Shih
Multimedia Information NEtwork (MINE) Lab
Department of Computer Science and Information Engineering
Tamkang University
Tamsui, Taipei Hsien
Taiwan 251, R.O.C.
email: TSHIH@CS.TKU.EDU.TW
fax: Intl. (02) 2620 9749

Abstract

We propose a graph-based model, with a simulation, for the mobile agents to evolve over the Internet. Based on the concepts of Food Web (or Food Chain), one of the natural laws that we may use besides neural networks and genetic algorithms, we define agent niche overlap graph and agent evolution states for the distributed computation of mobile agent evolution. The proposed computation model can be used in distributed Internet applications such as e-commerce programs, intelligent Web searching engine, and others.

Key words: Search Engine, Information Retrieval, Internet, Evolution Computing, Mobile Agent, Intelligent Agent

1 Introduction

Mobile agents are computer programs that can be distributed across networks to run on a remote computer station. The technique can be used in distributed information retrieval which allows the computation load to be added to servers, but significantly reduces the traffic of network communication. Many articles indicate that this approach is a new direction to software engineering. However, it is hard to find a theoretical base of mobile agent computing and interaction over the Internet. On the other hand, communication over Internet is growing increasingly and will have profound implications for our economy, culture and society. From mainframe-based numerical computing to decentralized downsizing, PCs and workstation computers connected by Internet have become the trend of the next generation computers. With the growing popularity of World Wide Web, digital libraries over Internet plays an important role in the academic, the business, and the industrial worlds. In order to allow effective and efficient information retrieval, many search engines were developed. However, due to the limitation of now-a-day network communication bandwith, researchers [15] suggest that distributed Internet search mechanisms should overcome the traditional information retrieval technologies, which perform the controls of searching and data transmission on a single machine.

A mobile agent, in general, can be more than just a search program. For instance, a mobile agent can serve as an emergency message broadcaster, an advertising agent, or a survey questionnaire collector. A mobile agent should have the following properties:

- It can achieve a goal automatically.
- It should be able to clone itself and propagate.
- It should be able to communicate with other agents.
- It has evolution states, including a termination state.

The environment where mobile agents live is Internet. Agents are distributed automatically or semi-automatically via some communication paths. Therefore, agents meet each other on the Internet. Agents have the same goal can share information and cooperate. However, if the system resource (e.g., network bandwidth or disk storage of a station) is insufficient, agents compete with each other. These phenomena are similar to those in the ecosystem of the real world. A creature is born with a goal to live and reproduce. To defense their natural enemies, creatures of the same species cooperate. However, in a perturbation in ecosystems, creatures compete with or even kill each other. The natural world has built a law of balance. Food web (or food chain) embeds the law of creature evolution. With the growing popularity of Internet where mobile agents live, it is our goal to learn

0-7695-0250-4/99 $10.00 © 1999 IEEE

from the natural to propose an agent evolution computing model over the Internet. The model, even it is applied only in the mobile agent evolution discussed in this paper, can be generalized to solve other computer science problems. For instance, the search problems in distributed Artificial Intelligence, network traffic control, or any computation that involves a large amount of concurrent/distributed computation.

We propose a logical network for agent connections/communications called *Agent Communication Network* (or ACN). ACN is dynamic. It evolves as agent communication proceeds. It also serves as a graph theoretical model of agent evolution computing. Our research purposes include:

- Provide a model for agent evolution and define the associated rules.
- Construct simulation facilities to estimate agent evolution.
- Suggest guidelines to write intelligent mobile agent programs.
- Suggest strategies to construct efficient ACNs. And,
- Ensure network security in the simulation environment.

Given an ACN, the model finds which agent evolution policy produces the maximum throughput (i.e., the goal of agents achieved). Or, changing the structure of an ACN, the model is able to find out how to adjust the agent evolution policy in order to recover from the change (or how is the throughput affected).

We have surveyed articles in the area of mobile agents, personal agents, and intelligent agents. The related works are discussed in section 2. Some terminologies and definitions are given in section 3, where we also introduce the detail concepts of agent communication network. In our model, an agent evolves based on a state transition diagram, which is illustrated in section 4. A graph theoretical model describes agent dependencies and competitions is given in section 5. Agent evolution computing algorithms, which we used to construct our simulation, are addressed in section 6. And finally, we discuss our conclusions and possible extensions in section 7.

2 Related Works

The concept of agent-based software engineering is discussed in a survey paper [5]. The author presents two important issues: agent communication language and agent architecture. Agent communication languages allow agents to share information and send messages to each other. Agent architecture, on the other hand, inculdes network infrastructure and software architecture that encure agent computing. An open agent architecture for kiosk-based multimedia information service is proposed in [3].

The concept of mobile agent is discussed in several articles [13, 17, 11, 12]. Agent Tcl, a mobile-agent system providing navigation and communication services, security mechanisms, and debugging and tracking tools, is proposed in [9, 6, 7]. The system allows agent programs move transparently between computers. A software technology called Telescript, with safety and security features, is discussed in [19]. The mobile agent architecture, MAGNA, and its platform are presented in [11]. Another agent infrastructure is implemented to support mobile agents [12]. A mobile agent technique to achieve load balancing in telecommunications networks is proposed in [18]. The mobile agent programs discussed can travel among network nodes to suggest routes for better communications. Mobile service agent techniques and the corrseponding architectural principles as well as requirements of a distributed agent environment are discussed in [10]. The evaluation of several commercial Java mobile agents is given in [8].

3 Agent Communication Network

Agents communicate with each other since they can help each other. For instance, agents share the same search query should be able to pass query results to each other so that redundent computation can be avoided. An *Agent Communication Network* (ACN) serves this purpose. Each node in an ACN represents an agent on a computer network node, and each link represents a logical computer network connection (or an agent communication link). Since agents of the same goal want to pass results to each other, they are modeled as a complete graph. Therefore, an ACN of agents hold different goals is a graph of complete graphs.

We define some terminologies used throuth this paper. A *host station* (or *station*) is a networked workstation on which agents live. A *query station* is a station where a user releases a query for achieving a set of goals. A station can hold multiple agents. Similarly, an agent can pursue multiple goals. An *agent society* (or *society*) is a set of agents fully connected by a complete graph, with a common goal associated with each agent in the society. A goal belongs to different agents may have different priorities. An agent society with a common goal of the same priority is called a *species*. Since an agent may have multiple goals, it is possible that two or more societies (or species) have intersections. A *communication cut set* is a set of agents belong to two distinct agent societies, which share common agents. The removing of all elements of a communication cut set results in the separation of the two distinct societies. An agent in a communication cut set is called an *articulation agent*. Since agent societies (or species) are represented by complete graphs and these graphs have communication cut sets as intersections, articulation agents can be used to suggest a shortest network path between a query station and the station where an agent finds its goal. Another point is that an articulation agent can hold a *repository*, which contains the network communication statuses of links of an agent society. Therefore, network resource can be evaluated when an agent checks its surviving environment to decide its evolution policy.

It is necessary for us to give formal definitions of these terminologies to be used in our algorithms. In the following definitions, "==" read as "is defined by" and "$\mathbb{P}\,X$" represents a set of object X:

$$\begin{aligned}
&Host_Station == URL \times Resource \times \mathbb{P}\ Agent \\
&Resource == Network \times CPU \times Memory \times Information \\
&Agent == \mathbb{P}\ Goal \times Policy \\
&Goal == Query_Return_URL \times Query \times Priority \\
&Agent_Society == \mathbb{P}\ Agent \\
&Species \subset Agent_Society
\end{aligned}$$

A *Host_Station* has a uniform resource locator (i.e., *URL*)[1] which represents the station's unique network address. A host station has system resources (i.e., *Resource*) and can hold some agents (i.e., $\mathbb{P}$ *Agent*). *Network* represents the network facility available to a station. *CPU* represents the computation power of a station. *Memory* represents the storage of a station. It could be the main memory or the secondary memory. *Information* is available on a station. Each *Agent* has some *Goal*s and a *Policy*, which is a set of application dependent factors the agent depends on to perform its evolution computation. *Query_Return_URL* is the URL where an agent should return its query results. *Query* is an application dependent specification which represents a user request to the agent. *Priority* is an integer represents the priority of a goal. The larger the integer, the higher the goal priority. *Agent_Society* is a set of agents share a common goal. *Species* is a *Agent_Society* of the same goal priority.

We use a simple notation to obtain a component of an object. For example, in our algorithm, if agent A is used, then $A.Goal$ represents the goals of that agent, where A is unique in its belonging agent society (or species). We will discuss the usage of these terms in algorithms which are given in section 6. But, firstly, we should address the concepts of agent evolution states and species food web in section 4 and 5, respectively.

[1] We could use an IP address. But, since our implementation of agents is based on the Web, a unique URL is used instead.

4 Agent Evolution States

An agent evolves. It can react to an environment, respond to another agent, and communicate with other agents. The evolution process of an agent involves some internal states of an agent. An agent is in one of the following states after it is born and before it is killed or dies of natural:

- **Searching**: the agent is searching for a goal
- **Suspending**: the agent is waiting for enough resource in its environment in order to search for its goal
- **Dangling**: the agent losts its goal of surviving, it is waiting for a new goal
- **Mutating**: the agent is changed to a new species with a new goal and the agent survives in a new host station

An agent is born to a *searching state* to search for its goal (i.e., information of some kind). All creatures must have goals (e.g., search for food). However, if its surviving environment (i.e., a host station) contains no enough resource, the agent may transfer to a *suspending state* (i.e., hibernation of a creature). The searching process will be resumed when the environment has better resources. But, if the environment is lack of resources badly (i.e., natural disasters occur), the agent might be killed. When an agent finds its goal, the agent will pass the search results to other agents of the same kind (or same society). Other agents will abort their search (since the goal is achieved) and transfer to a *dangling state*. An agent in a dangling state can not survive for a long time. It will die after some days (i.e., a duration of time). Or, it will be re-assigned to a new goal with a possible new host station, which is a new destination where the agent should travel. In this case, the agent is in a *mutating state* and is reborn to search for the new goal. In order to maintain the activity of agents, in a distributed computing environment, we use message passing as a mechanism to control agent state transition.

5 Species Food Web and Niche Overlap Graph

Agents can suspend/resume or even kill each other. We need a general policy to decide which agent is killed. By our definition, a species is a set of agents of the same goal with a same priority. It is the priority of a goal we base on to discriminate two or more species. We need to construct a direct graph which represents the dependency between species. We call this digraph an *species food web* (or *food web*). Each node in the graph represents a species. All species of a connected food web (i.e., a graph component of the food web) are of the same goal. We assume that, different users at different host stations may issue the same query. Each directed edge has an origin represents a species of a higher goal priority and has a terminus with a lower priority. Since an agent (and thus a species) can have multiple goals, each goal of an articulation agent should have an associated food web.

Each food web describes goal priority dependencies of species. Form a food web, we can further derive an *niche overlap graph*. In an ecosystem, two or more species have an *ecological niche overlap* (or *niche overlap*) if and only if they are competing for the same resource. A *niche overlap graph* can be used to represent the competition among species. The niche overlap graph is used in our algorithm to decide agent evolution policy and to estimate the effect when certain factors are changed in an agent communication network. Based on the niche overlap graph, the algorithm is be able to suggest strageies to re-arrange policies so that agents can achieve their highest performance efficiency. This concept is similar to the natural process that recover from perturbations in ecosystems.

6 Agent Evolution Computing

We have described how an agent evolves and how agents compete. The algorithms proposed in this section use the agent evolution state diagram and the niche overlap graphs discussed for agent evolution computing. First, we present some naive approaches, which also explain the basic concepts of agent searching and agent distribution. We then present a set of agent evolution computing algorithms over an ACN.

6.1 Agent Searching versus Agent Cloning

An agent wants to search for its goal. At the same time, since the searching process is distributed, an agent wants to find a destination station to clone itself. Searching and cloning are essentially exist as a *co-routining relation*. A co-routine can be a pair of processes. While one process serves as a producer, another serves as a consumer. When the consumer uses out of the resource, the consumer is suspended. After that, the producer is activated and produces the resource until it reaches an upper limit. The producer is suspended and the consumer is resumed. If the searching process is a consumer, then the cloning process is a producer who provides new URLs. The following algorithms describe agent searching and cloning:

Co-routining algorithm

```
Algorithm Search(G) :
given a goal G
    repeat
        if goal G is found then
            terminate Search
        else
            if URL_queue is empty then
                suspend Search until Clone returns
            else
                search on a URL for goal G
                and delete the URL from the queue
Algorithm Clone :
    repeat
        if URL_queue is full then
            suspend Clone until Search returns
        else
            find and put next URL in the URL_queue
```

The co-routining algorithms use a queue to store URLs. When the queue is empty, algorithm **Search** is suspended until **Clone** returns. Otherwise, a URL in the queue is used to propagate the agent. Algorithm **Clone** collects some new URLs via search engine until the URL queue is full. The co-routining processes communicate through the URL queue. However, it is not an efficient approach since **Search** or **Clone** wait for each other until the URL queue is full or empty. The drawback can be eliminated using a concurrent algorithm of two separated processes:

Concurrent algorithm

```
Algorithm Search_Clone(G) :
given a goal G
    cobegin
        process Search :
            repeat
                if goal G is found then
                    terminate Search_Clone
                else
                    if URL_queue is not empty then
                        search on a URL for goal G
                        and delete the URL
        process Clone :
            repeat
                if URL_queue is not full then
                    put next URL in the URL_queue
    coend
```

The concurrent algorithm searches and propagates at the same time when the queue is not empty or not full. Two processes are used concurrently (specified in-between "cobegin" and "coend)". When the agent implemented in the concurrent or co-routining algorithm travels to a station, a local URL queue is used and the computation proceeds independently.

The above two approaches describes the relation between searching and cloning of agents. But, there is no communication among agents. All agents compute for the same goal and multiple copies of the same result will be sent back to the query station. this approach not only waste CPU time, but also waste network resource. In the next section, we want to overcome this drawback by using an agent communication network, where agents evolve.

6.2 Agent Evolution Computing over an ACN

The co-routining and concurrent algorithms discussed in section 6.1 works on a signle station. However, agent evolution on the agent communication network is an asynchronized computation. Agents live on different (or the same) stations communicate and work with each other. The searching and the cloning processes of an agent may run as a co-routine on a station. However, different agents are run on the same or separated stations concurrently. Algorithm **Agent_Search** is the starting point of agent evolution simulation. If system resource meets a basic requirement, the algorithm activates an agent in the searching state. If the search process finds its goal (e.g., the requested information is found), goal abortion results in a dangling state of all agents in the same society (including the agent who finds the goal). At the same time, the search result is sent back to the original query station. Suppose that the goal can not be achieved in an individual station, the agent is cloned in another station (agent propagation). The **Agent_Clone** algorithm is then used. On the other hand, the agent may be suspended or even killed upon the availability of system resource. Some auxiliary algorithms, which are self-explanatory, describe these processes.

Agent Searching Algorithm

```
Algorithm Agent_Search(A, G, X):
    given a goal G to agent A on station X of S
    if Resource_Available(A, G, X) >
            low_requirement then
        agent A searches for G in its station X
        if G is found then
            A sends an abort message to agents in S
            A sends search result to query station
            Agent_Search is complete
        else
            call Agent_Clone(A, G, S)
            terminate Agent_Search
    else if Resource_Available(A, G, X) >
            min_requirement then
        call Agent_Suspend(A, G, X)
    else
        call Agent_Kill(A, G, X)
```

Note that, *low_requirement* must be greater than *min_requirement* so that different levels of treatment are used when the resource is not sufficient. But the resource available factor depends on agent policy, as defined in **Resource_Available**.

Agent Cloning Algorithm

```
Algorithm Agent_Clone(A, G, S):
    given a source agent A searches for goal G of society S
    use search engine to find a new URL
        on an arbitrary station X that may contain goal G
    if station X has an agent A' then
        if goal of A' contains G then
            let S' be the society associated with G
                where A' belongs
            union S' and S
        else
            assign G to A'
            make A' join S
        call Agent_Search(A', G, X)
    else
        copy a new agent A'' of A on station X
        make A'' join society S
        call Agent_Search(A'', G, X)
```

Agent cloning is achieved by the **Agent_Clone** algorithm. When the cloning process finds new URLs to broadcast an agent, two strategies can be used. The first is to broadcast the agent to all URLs found by one search engine. But, considering the network resource available, the second strategy may check for the common URLs found by two or more search engines. The cloning algorithm must check whether there is another agent in the destination URL (or station). If so, the algorithm checks whether the agent at that URL shares the same goal with the agent to be cloned. If two agents share the same goal, there is no need of cloning another copy of agent. Basically, the goal can be computed by the agent at the destination URL. In this case, the union of the two societies is necessary. On the other hand, if the two agents do not have a common goal, to save computation resource, we may ask the agent at the destination URL to help searching for an additional goal. This case makes a re-organization of the society where the source agent belongs. The result also ensure that the number of agents on the ACN is kept in a minimum. Whether the two agents share the same goal, the **Agent_Search** algorithm is used to search for the goal again. When there is no agent running on the destination station, we need to increase the number of agents on the ACN by duplicating an agent on the destination URL. The society is reorganized. And the **Agent_Search** algorithm is called again.

Auxiliary Algorithms

Algorithm **Agent_Suspend**(A, G, X) :
given a goal G to agent A on station X
wait until **Resource_Available**(A, G, X) > *low_requirement*
call **Agent_Search**(A, G, X)

Algorithm **Agent_Kill**(A, G, X) :
given a goal G to agent A on station X
terminate agent A on station X

Algorithm **Resource_Available**(A, G, X) :
given a goal G to agent A on station X
switch *A.Policy*
case *descrete_sim* $\wedge$ *network_bound* then
Available = *X.Resource.Network*
case *descrete_sim* $\wedge$ *cpu_bound* then
Available = *X.Resource.CPU*
case *descrete_sim* $\wedge$ *memory_bound* then
Available = *X.Resource.Memory*
case *descrete_sim* $\wedge$ *cpu_bound* $\wedge$ *memory_bound* then
Available = *X.Resource.CPU* $*$ $w1$ + *X.Resource.Memory* $*$ $w2$
case *descrete_sim* $\wedge$ ...
Available = ...
case *internet_sim*
Available = resource available on X
if *G.Priority* is low then
Available = *Available* $*$ r

Note that, $w1$ and $w2$ are weights ($w1 + w2 = 1.0$). In the **Resource_Available** algorithm, we only describes some cases of using agent policies (i.e., *A.Policy*). Other cases are possible. If the goal priority (i.e., *G.Priority*) is low, we let r be a constant less than 1.0. Therefore, resources are reserved for other agents.

The above algorithms describe how an agent evolves from a state to another. The factor that agents affect each other depends on the system resource available. However, in an ACN, it is possible that agents suspend or even kill each other, as we described in previous sections. The niche overlap graphs of each goal play an important role. We revise the **Agent_Suspend** and **Agent_Kill** algorithms to take the niche overlap graphs into consideration. In the revised **Agent_Suspend** algorithm, if there exists a goal that has a lower priority comparing to the goal of the searching agent, a suspend message is sent to the goal to delay its search. The searching agent may be resumed after that since system resources may be released from those goal suspension. In the revised **Agent_Kill** algorithm, however, a kill message is sent instead. The system resource is checked against the minimum requirement. If resuming is feasible, the **Agent_Search** algorithm in invoked. Otherwise, the system should terminate the searching agent.

Algorithm **Agent_Suspend**(A, G, X) :
given a goal G to agent A on station X
check the niche overlap graph of G
for each goal G' in the graph that has a priority lower than G
send a suspend message to G' to delay search
wait until **Resource_Available**(A, G, X) > *low_requirement*
call **Agent_Search**(A, G, X)

Algorithm **Agent_Kill**(A, G, X) :
given a goal G to agent A on station X
check the niche overlap graph of G
for each goal G' that has a priority lower than G
send a kill message to G' to terminate search
if **Resource_Available**(A, G, X) > *min_requirement*
call **Agent_Search**(A, G, X)
else
terminate agent A on station X

7 Conclusions

Mobile agent based software engineering is interesting. However, in the literature, we did not find any other similar theoretical approach to model what mobile agents should act on the Internet, especially how mobile agents can cooperate and compete. A theoretical computation model for agent evolution was proposed. Algorithms for the realization of our model were given. Consequently, our contributions in this paper are:

- We proposed a model for agent evolution computing based on food web, the law of natural balancing.
- We developed a set of algorithms for the distributed computing of agent programs.
- We implemented a simulation environment based on JATLite to support our theory.

However, there are other extensions to the evolution model. For instance, species in the natural world learn from their enemies. In our future model, agents can learn from each other. We can add a new state, the "learning" state, to the agent evolution state diagram. When an agent is in the dangling state, it can communicate to other agents via some agent communication languages. Computing methods can be replicated from other agents. And the agent transits to the mutating state to wait for another new goal. In addition, when a station lacks of system resource, an agent in the suspending state can change its policy to admit to the environment before it transits to the searching state. These are the facts that agents can learn. On the other hand, in the cloning process, two agents on a station sharing a common goal can be composed to a new agent (i.e., marriage of agents). This agent may have more goals compares to its parents. An agent composition state could be added to the agent evolution state diagram. But, the destination station where this new agent lives should be compromised.

The evolution of computers has changed from mainframe-based numerical computation to networked stations. In line with the success of Internet technologies, in the future, computation and information storage are not limited to a single machine. It is possible that, an individual buy a primitive computer that only has a terminal connected to Internet. Personal data and the computation power are embedded within the Internet. Mobile agent and agent evolution computing will be very interesting and important. Our agent evolution model addresses only a small portion of the icefield, which should be further studied in the societies of network communications, automatic information retrieval, and intelligent systems.

References

[1] Jose G. Annunziato, "A Review of Agent Technology," in Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, California, U.S.A., June 12 - 14, 1995.

[2] Alper Caglayan and Colin Harrison, *Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents*, Wiley Computer Publishing, 1997.

[3] P. Charlton, Y. Chen, E. Mamdani, O. Olsson, J. Pitt, F. Somers, and A. Wearn, "An Open Agent Architecture for Integrating Multimedia Services," in Proceedings of the Autonomous Agents 97 conference, Marina Del Rey, California, U.S.A., 1997, pp. 522 - 523.

[4] Chanda Dharap and Martin Freeman, "Information Agents for Automated Browsing," in Proceedings of the 1996 ACM CIKM conference (CIKM'96), Rockville, MD, U.S.A., 1996, pp. 296 - 305.

[5] Michael R. Genesereth, "Sotftware Agents," communication of the ACM, Vol. 37, No. 7, Jul. 1994, pp.48 - 54.

[6] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko, "Mobile agents: the next generation in distributed computing," in Proceedings of the 1997 2nd Aizu International Symposium on Parallel Algorithms/Architecture Synthesis, Fukushima, Japan, 1997, pp. 8 - 24.

[7] Robert S. Gray, "Agent Tcl," Dr. Dobb's Journal of Software Tools for Professional Programmer, Vol. 22, No. 3, Mar 1997.

[8] Joseph Kiniry and Daniel Zimmerman, "Hands-on look at Java mobile agents," IEEE Internet Computing, Vol. 1, No. 4, July 1997, pp. 21 - 30.

[9] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko, "Agent Tcl: targeting the needs of mobile computers," IEEE Internet Computing, Vol. 1, No. 4, July 1997, pp. 58 - 67.

[10] S. Krause and T. Magedanz, "Mobile service agents enabling intelligence on demand in telecommunications," in Proceedings of the 1996 IEEE Global Telecommunications Conference, London, UK, 1996, pp 78 - 84.

[11] Sven Krause, Flavio Morais de Assis Silva, and Thomas Magedanz, "MAGNA - a DPE-based platform for mobile agents in electronic service markets," in Proceedings of the 1997 3rd International Symposium on Autonomous Decentralized Systems (ISADS'97), Berlin, Germany, 1997, pp. 93 - 102.

[12] Anselm Lingnau and Oswald Drobnik, "Making mobile agents communicate: a flexible approach," in Proceedings of the 1996 1st Annual Conference on Emerging Technologies and Applications in Communications, Portland, OR, USA, 1996, pp. 180 - 183.

[13] T. Magedanz and T. Eckardt, "Mobile software agents: a new paradigm for telecommunications management," in Proceedings of the 1996 IEEE Network Operations and Management Symposium (NOMS'96), Kyoto, Japan, 1996, pp 360 - 369.

[14] Jide B. Odubiyi, David J. Kocur, Stuart M. Weinstein, Nagi Wakim, Sadanand Srivastava, Chris Gokey, and JoAnna Graham, "SAIRE - A Scalable Agent-Based Information Retrieval Engine," in Proceedings of the Autonomous Agents 97 conference, Marina Del Rey, California, U.S.A., 1997, pp. 292 - 299.

[15] Michael Pazzani and Daniel Billsus, "Learning and Revising User Profiles: The Identification of Interesting Web Sites", Machine Learning, Vol. 27, 1997, pp. 313 - 331.

[16] Charles J. Petrie, "Agent-Based Engineering, the Web, and Intelligence," IEEE Expert, Vol. 11, No. 6, 1996, pp. 24 - 29.

[17] Peter S. Sapaty and Peter M. Borst, "WAVE: mobile intelligence in open networks," in Proceedings of the 1996 1st Annual Conference on Emerging Technologies and Applications in Communications, Portland, OR, USA, 1996, pp. 192 - 195.

[18] Ruud Schoonderwoerd, Owen Holland, and Janet Bruten, "Ant-like agents for load balancing in telecommunications networks," in Proceedings of the 1997 1st International Conference on Autonomous Agents, Marina del Rey, California, U.S.A., 1997, pp. 209 - 216.

[19] Joseph Tardo and Luis Valente, "Mobile agent security and telescript," in Proceedings of the 1996 41st IEEE Computer Society International Conference (COMPCON'96), Santa Clara, CA, USA, 1996, pp. 58 - 63.

[20] Christoph G. Thomas, and Gerhard Fischer, "Using Agents to Personalize the Web," in Proceedings of the 1997 ACM IUI conference (IUI'97), Orlando Florida, U.S.A., 1997, pp. 53 - 60.