A Multi-Agent Approach to Distributed Computing

C.R. Muthukrishnan, Suresh T.B.

Department of Computer Science and Engineering

Indian Institute of Technology, Madras

India

Email: crm@iitm.ernet.in, sunny@indy.iitm.ernet.in

Abstract

A network of heterogeneous workstations is an inherently dynamic distributed environment. The number of lightly loaded workstations available for use continually changes; the parallel machine presented by the network, in effect, continually undergoes reconfiguration. Application programs must effectively adapt to the changing number of processing nodes while maintaining computational efficiency. The present work develops methods using software agents which adapt to a dynamic environment. Distributed applications can be written using a programming interface to the agents which carry out the computations thus providing maximum flexibility. We use multi-agent technology for distributed computing. Our multi-agent approach speeds up data parallel applications by allocating workstations fairly and efficiently. We present the design of a multi-agent system that adaptively executes data parallel programs on a network of workstations. Two examples, the Mandelbrot set computation and the Traveling Salesman Problem are implemented and studied using our approach and performance results are presented.

Keywords: software agents, multi-agent systems, parallel/distributed computing, adaptive parallelism

1 Introduction

Parallel distributed environments [9, 11, 14, 16] consisting of a network of heterogeneous workstations introduce an inherently dynamic environment. The number of workstations available for use is continually changing; the parallel machine presented by the network is in effect continually reconfiguring itself. Application programs must effectively adapt to the changing number of processing nodes and achieve computational efficiency.

Recent years have seen a marked interest in agent-oriented technology [4, 12], spanning applications as diverse as information retrieval, user interface design and network management. An agent is a computational entity which acts on behalf of other entities in an autonomous fashion, performs its actions with some level of pro-activity and/or reactiveness, exhibits some level of the key attributes of learning, cooperation and mobility. Software agents (often simply termed agents) are software systems that loosely conform to this description [6].

Our objective is to use a multi-agent system for distributed computing on a network of workstations. We develop methods using agents which adapt to a dynamic environment where machines can come into and leave the network over time. Distributed applications can be written with a programming interface to the agents so that agents can take care of the computations with the maximum flexibility. To facilitate the use of heterogeneous machines in a network, we use the Java programming language in our implementation. We use an agent communication language for information exchange at a higher level among agents for distributed computing. Two examples, viz, Mandelbrot set computation and the Traveling Salesman Problem are implemented to study this approach and their performance results are presented.

2 Design

A schematic design of our multi agent system is given in Figure 1. Agents reside on the nodes of the network of workstations. Agents get information from the user and other agents. The daemon part of an agent always listens for communication from other agents in the network.

The application program accesses the agent system using a programming interface to the agents. This interface defines the communication between the application program and the agent system. This facilitates developing the application program and the agent system separately. It is always better to make the agent system separate from the application program so that many application programs can be developed using the same interface.

Initially the problem is given to a single agent which decomposes it into subproblems

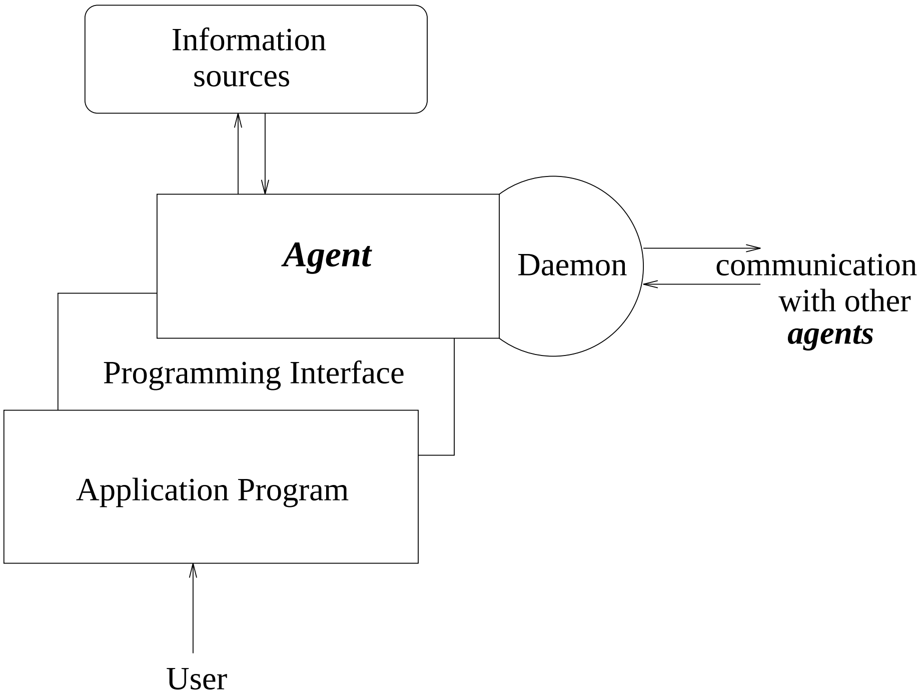

Figure 1: Design of our Multi-Agent System

and distributes them among other agents. For this, the agents in the system need to communicate and coordinate among themselves.

3 Behaviour

Our multi-agent system is decentralized. It consists of a distributed set of agents, each running on a node in the network. There is no single point of control. Each agent is capable of both assigning tasks originating on the local machine and offer its processing resources for tasks from other nodes/agents. All agents are identical and have the same behaviour.

Agent behaviour is based on a market/bidding metaphor. The underlying protocol on which this metaphor is implemented is the Contract Net protocol [15]. The basic mechanism in this protocol is to send requests for accepting work (call for bids) and then assign the work to the agents(s) according to some criteria (for example, load of the machine on which the agent is running) which the participating agents provide when they respond to the request for bids.

Agents can join the computation by responding to the request for bids. Agents inform their availability by submitting a bid. Assignment of subproblems depends on the bid the agent submits.

3.1 Bidding and Distribution of Subproblems

An agent divides the problem given to it into subproblems. It then sends requests to all other agents for bids for a subproblem. The responses received from the agents are evaluated and the subproblem is assigned to one of the agents with best bid. All subproblems are handled in this manner by the agents.

Each agent which takes part in solving a problem can distribute work to other agents. Subproblems originate at the nodes and agents which solve these subproblems need to cooperate among themselves to take into account dependencies in the computation.

3.2 Dynamic Addition and Leaving of Agents

Agents can join the computation by responding to the request for bids. The request for bids is a broadcast and hence any agent who is listening can respond with a bid if it wants to join the computation.

Agents in our multi-agent system can leave the system when the subproblem assigned to it has been solved. Before leaving, the agent communicates the solution of the subproblem to the agent from which it had received the subproblem.

When a solution to the problem is found, this information is broadcast to all other agents. On receiving this information all agents terminate the local computation.

4 Communication

Large-scale computation consisting of a collection of independent, autonomous entities requires communication between the entities to support coordination and synchronization. The communication between agents is by asynchronous message passing. The interaction among agents need not be, and is often not, predetermined [7].

For software agents to interact and inter-operate effectively three fundamental and distinct components are required: 1) a common language 2) a common understanding of the knowledge exchanged and 3) the ability to exchange whatever is included in (1) and (2). Agents are commonly taken to be "high-level". Hence the communication language should be powerful enough to support communication at this high level; otherwise the agents will have to perform the task of translating the information exchanged between levels.

```
(tell   :sender agentA
        :receiver agentB
        :language KIF
        :content (load 50))
```

Figure 2: Example KQML Performative

We use the Knowledge Query and Manipulation Language (KQML) [17, 18] in our implementation. Knowledge Interchange Format (KIF) [5, 20] is used as the content language. KIF is a prefix version of first order predicate calculus with extensions to enhance its expressiveness. An example of a KQML performative that uses KIF as the content language is shown in Figure 2. This performative is used by agentA to convey to agentB that its load is 50%.

5 Implementation

Two different approaches have been explored in the literature [5] for communication between the agents in a multi-agent system: direct communication (in which agents handle their own coordination) and assisted coordination (in which agents rely on special system programs called *facilitators* to achieve coordination).

We employ direct communication among agents. The advantage of direct communication is that it is simple, efficient and does not rely on the existence or capability of any other programs. We use the popular contract-net approach for direct communication.

Agent systems are built in Java so that agents can run on heterogeneous platforms. JATLite (Java Agent Template, Lite) [19] is an agent writing tool in Java. It provides a Java agent platform that makes it easy to build systems in a common way without imposing any particular theory of autonomous agents. We have used JATLite to build our multi-agent system. Application programs are written in Java and JATLite library calls are invoked from the Java program. JATLite facilitates construction of agents that send and receive messages using the standard communication language KQML.

6 Case Study 1: Mandelbrot Set Computation

The Mandelbrot set, named after Benoit Mandelbrot, is a fractal. Fractals are objects that display self-similarity at various scales. The process of generating the Mandelbrot set is based on an extremely simple equation involving complex numbers [1].

6.1 Implementation

Agents are organized according to the master-slave model for this example. There is a master agent that coordinates the computations of the worker agents. The computation involves applying a recursive function to a collection of points in the complex plane until the function values either reach a specific value or begin to diverge. Depending upon this condition, a graphical representation of each point in the plane is constructed. Since the function outcome depends only on the starting value of the point (and is

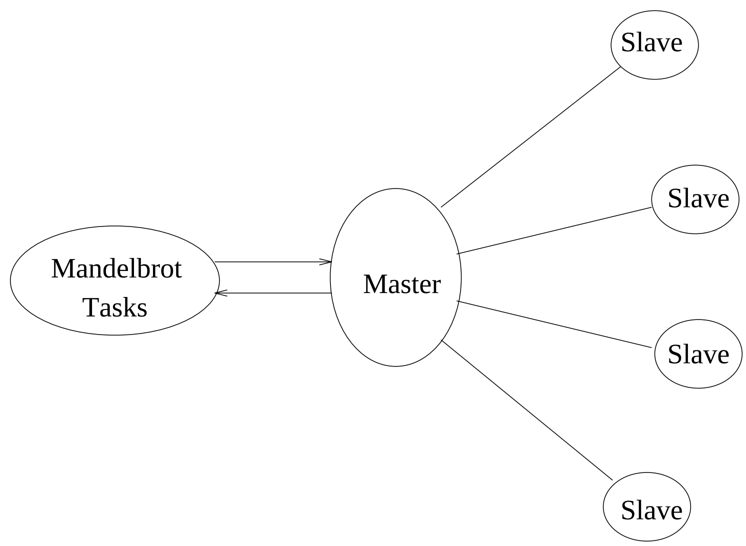

Figure 3: Master-slave paradigm

independent of other points), the problem is partitioned into completely independent portions. The algorithm is applied to each portion and the partial results obtained are combined using simple combination schemes. The control structure of the master-slave class of applications is shown in Figure 3.

6.2 Performance Results

Machines	Type of Machines	Time (sec)	Speedup	Efficiency of processor usage
1 (sequential)	SG Ver. 5.3	19	n/a	100%
1	SG Ver. 5.3	21.8	0.87	87.2%
2	2 SG (Ver. 5.3 & 6.2)	12.8	1.48	74.22%
3	2 SG & 1 SP	9.6	1.98	66%
4	2 SG & 2 SP	9	2.11	52.8%
5	2 SG & 3 SP	7.8	2.44	48.7%

SG - Silicon Graphics , SP - SUNSPARC

Table 1: Performance Results (Mandelbrot Set Computation)

The implementation has been done on two Silicon Graphics workstations (one running version 6.2 and the other running version 5.3 of the IRIX operating system) and three SUNSPARC workstations (each running Solaris 2.3). The workstations are connected using a 10Mbps Ethernet network.

The performance results are as shown in Table 1. All times are actual *wall* clock times. The speedup and efficiency are calculated using the formula:

$speedup = (sequential\, time / time\, to\, run\, under\, our\, system)$

$efficiency = sequential\, time / (time * number\, of\, machines)$

Speedup and efficiency are plotted against the number of machines in Figure 4. We note that the performance of the system improves as more machines come into and take part in the computation.

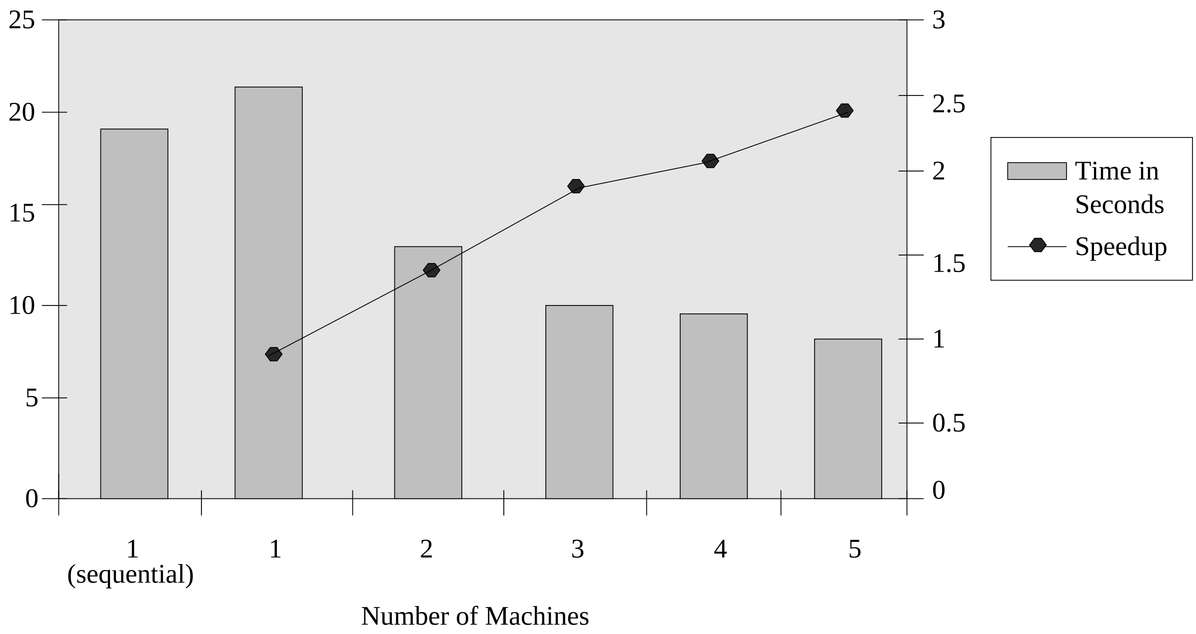

Figure 4: Performance Evaluation (Mandelbrot Set Computation)

7 Case Study 2: The Traveling Salesman Problem

The traveling salesman problem [10] is to find a shortest possible circuit through n cities, where the distances between the cities are given.

The algorithm developed by Little, et al. [3] uses a branch-and-bound method to solve the traveling salesman problem. Little takes advantage of one of the trade-offs possible in branch-and-bound methods [2, 8]. A poor but easily calculated bound results in more branching operations but less computation at each node. This makes the parallelization of the algorithm easier. The calculation of bounds is referred to as matrix reduction. The reduced matrices are used for branching by partitioning the tours in a given subset into two subsets. This is done by committing an arc in one of the subsets and prohibiting that arc in the other subset.

7.1 Implementation

We are using a parallelized version of Little's algorithm to solve the TSP on a network of workstations. The same program (agent) runs on all the workstations that take part in the computation. Agents running on different workstations explore different tours and communicate their lower bounds when they need to cut off some of the branches of the tour tree.

7.2 Performance Results

The implementation has been done on three Silicon Graphics workstations running version 6.2 of the IRIX operating system. The workstations are connected using a 10Mbps Ethernet network.

No. of Machines	Time Taken (minutes)	Nodes Explored	Problem Solved?
1	72	3069	No
2	76	5146	Yes
3	31	5217	Yes

Table 2: Performance Results (Traveling Salesman Problem)

We use the 25 city problem of Held and Karp [13] as input. With one agent the system runs out of memory after one hour of computation. Two agents solved the problem in 76 minutes. The total number of nodes of the tree explored was around 5000. With three agents the time required to solve the problem reduced to 31 minutes. Here also the total number of nodes explored was around 5000. When a solution is found, some of the branches which lead to higher cost are cut off, thus decreasing the storage requirements. It is due to this that the problem can be solved with more than one agent but not with a single agent. Agents communicate with each other after 500 iterations. The performance results are tabulated in Table 2. We note that performance of the system improves as more machines join the computation.

Agents can dynamically enter and leave the system during a computation. Any agent who wants to leave the system assigns its work to any other agent who is willing to accept the work. The performance results when agents are dynamically added to the system are plotted in Figure 5. Agent 1 starts at time 0. When the total number of nodes explored becomes 400, agent 2 joins the system (i.e. after 15 seconds of computation). Agent 3 comes into the system when the total number of nodes explored becomes 700 (i.e. after 37 seconds of computation). All three agents then try to solve the problem together. The problem was solved in 24 minutes and the total number of nodes explored

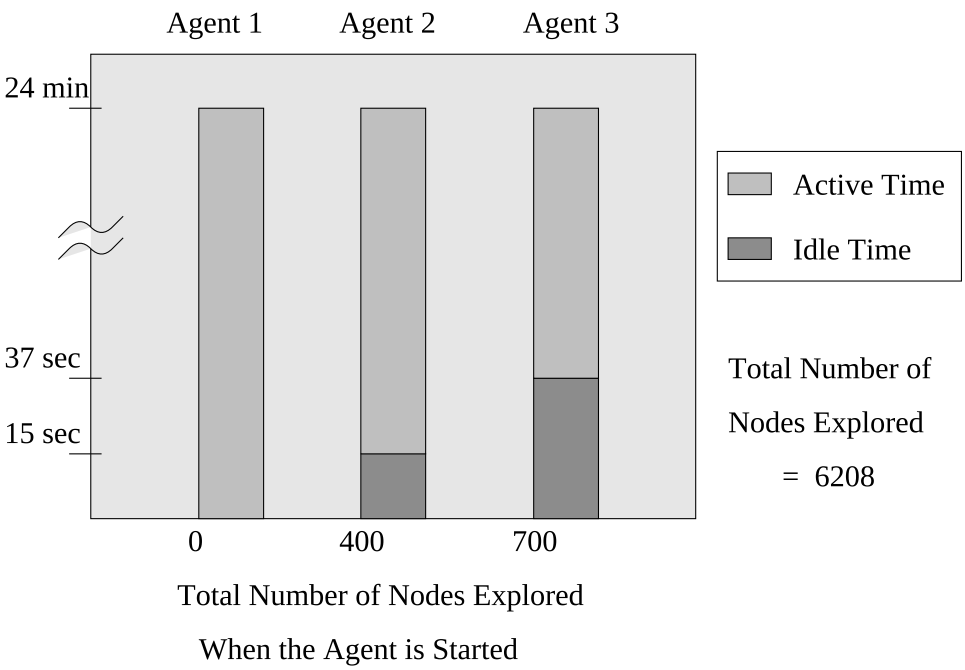

Figure 5: Dynamic Addition of Agents (Performance Evaluation)

was 6208. Here the agents communicate with each other after 200 iterations.

Performance results for dynamic addition and leaving of agents are plotted in Figure 6.

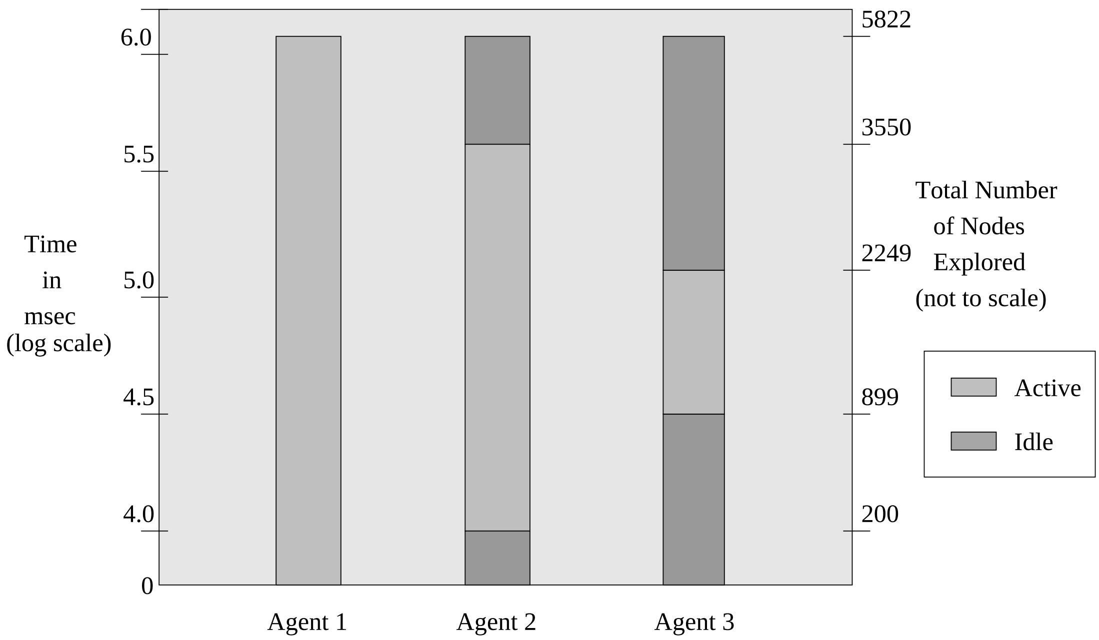

Figure 6: Dynamic Addition and Leaving of Agents (Performance Evaluation)

It shows agents 1, 2 and 3 entering the system in that order. Then agents 3 and 2 leave the system. The graph plots the amount of time spent by each agent in the logarithmic scale. The total number of nodes explored is also given (not to scale). Here also the communication takes place after 200 iterations.

8 Conclusions

In this paper, we have presented the design of a multi-agent system for distributed computing on a network of workstations. Two examples with their performance results are presented. The contributions of this work are

1) Use of multi-agent approach for distributed computing on a network of workstations.

2) Adapting a computation to a dynamic environment where machines can come into and leave the network over time.

3) Using an agent communication language for information exchange at a higher level among agents for distributed computing.

References

[1] **Michael Barnsley.** *Fractals Everywhere.* Academic Press Inc., 1988.

[2] **Teodor Gabriel Crainic** and **Bernard Gendron.** Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066, November-December 1994.

[3] **J.D.C. Little, D.W. Sweeney, C. Karel** and **K.G. Murty.** An algorithm for the traveling salesman problem. *Operations Research*, 11:972–989, 1963.

[4] **Stan Franklin** and **Art Graesser.** Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Third International Workshop on Agent Theories, Architectures, and Languages.* Springer-Verlag, 1996.

[5] **M. Genesereth** and **S. Ketchpel.** Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[6] **N.R. Jennings** and **M.J. Wooldridge.** Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[7] **Dennis Kafura** and **Jean-Pierre Briot.** Actors and agents. *IEEE Concurrency*, pages 24–29, April-June 1998.

[8] **E.L. Lawler** and **D.E. Wood.** Branch-and-bound methods: A survey. *Operations Research*, 14:669–719, 1966.

[9] **Mike Litzkow** and **Miron Livny.** Experience with the Condor distributed batch system. In *IEEE Workshop on Experimental Distributed Systems*, Huntsville, October 1990.

[10] **M. Bellmore** and **G.L. Nemhauser.** The traveling salesman problem: A survey. *Operations Research*, 16:538–558, 1968.

[11] **David Kaminsky, Nicholas Carriero, David Gelernter** and **Jeffery Westbrook.** Adaptive parallelism with Piranha. Technical Report 954, Yale University, February 1993.

[12] **H. Nwana.** Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, September 1996.

[13] **R.M. Karp** and **M. Held.** A dynamic programming approach to sequencing problems. *J. Soc. Indust. and Appl. Math.*, 10:196–210, 1962.

[14] **Steve Otto, Robert Prouty** and **Jonathan Walpole.** Adaptive execution of data parallel computations on networks of heterogeneous workstations. Technical Report CSE-94-012, Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Baltimore County, March 1994.

[15] **R.G. Smith.** The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[16] **David A. Patterson, Thomas E. Anderson, David E. Culler** and **the NOW Team.** A case for networks of workstations : NOW. *IEEE Micro*, February 1995.

[17] **Don Mckay, Tim Finin, Rich Fritzson** and **Robin McEntire.** KQML - a language and protocol for knowledge and information exchange. Technical Report CS-94-02, Department of Computer Science, University of Maryland, UMBC, Baltimore County, 1994.

[18] Current KQML standard. http://www.cs.umbc.edu/kqml/.

[19] JATLite (Java Agent Template, Lite). http://java.stanford.edu/java_agent/html/.

[20] UMBC AgentWeb. http://www.cs.umbc.edu/agents/.