8-2003

ABSA: An Agent-Based Tool for System Administration

Santosh Ramakrishna
Southern Illinois University Carbondale

Shahram Rahimi
Southern Illinois University Carbondale, rahimi@cs.siu.edu

Published in Ramakrishna, S., & Rahimi, S. (2003). ABSA: an agent-based tool for system administration. Proceedings, IEEE International Conference on Industrial Informatics, 2003. INDIN 2003, 312-319. doi: 10.1109/INDIN.2003.1300288

ABSA: An Agent-Based Tool for System Administration

Santosh Ramakrishna, Shahram Rahimi
Department of Computer Science, Southern Illinois University
Mail Code 4511, Carbondale, Illinois – 62901, USA.
{srama, rahimi} @cs.siu.edu

ABSTRACT

Studies indicate that because of the difficulty and complexity, the cost of administering systems is ten times the cost of the actual hardware. ABSA is an agent-based solution to automated system administration. ABSA architecture is introduced to minimize the cost of administering computers in multi platform networks and to provide a simple, consistent, expandable, and integrated system administration tool.

Keywords: Agents, System administration and Distributed systems.

1. INTRODUCTION

Networks maintained by many sites today contain tens to hundreds of computers. Managing such a sizeable collection of computers and their software is a challenging task, generally referred to as system administration. Majority of the tasks performed by a system administrator on a day to day basis include ensuring all hardware and software is in working order, managing user accounts, dealing with the security threats, backups, software upgrades, maintenance, recovery from system failure and ensuring an adequate supply of resources such as swap and disk space. Performing all these tasks manually can prove to be very difficult, especially when dealing with a sizable collection of computers. Majority of the day to day activities performed by system administrators are procedural and recurring and hence a burden to the system administrator [1]. This complexity and difficulty of system administration has been long recognized. Studies indicate that because of complexity, cost per year of administering systems is much higher than the cost of the actual hardware itself [10]. While system administration is challenging and burdensome, most of the tasks performed by an administrator can be automated to great extent. Moreover there is a limit on the number of systems that can be maintained by an administrator, which highlights the need for a scalable approach.

In this paper, we first discuss the current approaches to automation of system administration tasks and then present ABSA, an agent-based architecture for system administration. But before preceding any further we discus software agents, some of their important characteristics and their types.

A definition of "software agent" that many agent researchers might find acceptable is: a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes [2]. The requirement for continuity and autonomy derives from our desire that an agent be able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment without requiring constant human guidance or intervention. In general, software agents are differentiated from other applications by their added dimensions of mobility, autonomy, and the ability to interact independent of their user's presence.

Figure 1 illustrates the typical difference between mobile agents and client-server architecture. In client-server architecture, the client constantly interacts with the server over the network to get its task done. In contrast, in mobile agent architecture, the agents migrate from the client to the server, perform the operations locally on the server, and then come back to the client. This architecture has advantages such as load distribution, ease of network traffic, support for off-line operation and temporal network failures etc.

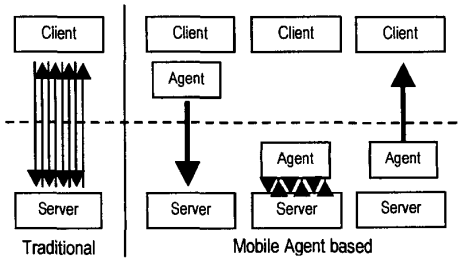

Figure 1 Client - Server Vs. Mobile Agents

There are two types of agents, namely stationary agents and mobile agents. Stationary agents are permanently attached to a place (node), while mobile agent can move from one place to another. An agent is said to be strongly mobile if its entire code and execution state move with it.

0-7803-8200-5/03/$17.00 ©2003 IEEE

In our architecture, we use stationary agents for management purposes and mobile agents to distribute the system administration tasks among the computers in the network. Agent technology provides a fresh scalable approach to system administration, which avoids the difficulties of the traditional client/server approach.

Distinctiveness of agents such as autonomous nature, intelligence, perseverance, adaptability, and of course mobility are most appropriate for their use in our architecture. The mobile nature of agents allows keeping minimum essential environment on the remote host that is just enough to allow execution of agents on it. This avoids the concentration of the operations in a single computer; instead, it uses the computing power of other computers by distributing the tasks. Moreover, using java agents in ABSA, provides the system with platform independency which further distinguishes it from other tools available in the market.

The remaining parts of the paper are organized as follows. First, a brief background on system administration tasks and software agents is given in sections 2. Then the general architecture of the system is presented in section 3. In section 4, we describe the implementation and the tools used. Finally, a brief summary concludes the paper.

2. BACKGROUND

In this section, we briefly review the current centralized system administration approach and discuss some of the existing tools that aid system administrators.

Centralized System Administration
Recently, there has been considerable amount of research to replace the traditional ad hoc system administration by client/server based applications, which aim to centralize the process. These centralized applications use mainly two protocols, the Simple Network Management Protocol (SNMP) and the Common Management Information Protocol (CMIP). Both protocols follow a client/server approach with managers invoking operations on management programs. They also provide mechanisms for reporting of events by management programs. However, there are fundamental differences between these two protocols. CMIP offers a much richer set of protocol operations both on manager and on management. However SNMP is a simpler tool for and is more popular in the market.

SNMP, CMIP, and related approaches to network and system administration are centralized paradigms based on the client/server architecture. These solutions require gathering all management functionality in a central manager which causes complexity and lower performance. Moreover, they do not address heterogeneity of the platforms. Scalability is another disadvantage of centralized approach, which loses performance to the size of the network.

Analysis of existing Tools aiding System Administrator
Automated administration of systems is becoming increasingly important due to the associated costs. Some work has been done in this regard to either partially automate the tasks or produce tools to aid administrators.

"Software Update via Mobile Agent Based Programming" [6] is one such approach for automated updating of software on the systems. This model has some limitations such as platform dependency. Moreover, the software has to be maintained on the server, which causes centralization of considerable amount of the tasks and hence a bottleneck. As a second example, we can name "The Igor System Administration Tool" [7] which is a tool for performing administration tasks simultaneously on numerous hosts. Although it eases the task of system administration, it does not deal with automation of system administration and it focuses on UNIX systems only.

"Central System Administration in a Heterogeneous UNIX Environment: GeNUAdmin" [8] is another example. In this tool, configuration profiles for clients are maintained on the central server and clients are configured based on their profile on the server. Administrator has to modify the configuration files to manage the clients. The modifications are automatically transferred to the client systems. Its disadvantages are that it may cause inconsistency among configuration files and also it is for UNIX systems only. Our last example is "WEBMIN: A Web-Based System Administration Tool for UNIX" [9], which is a web based tool for configuring UNIX systems. This one does not support platform independency either.

All the above tools are based on client-server architecture. This makes them less scalable since all the administrative tasks are done on a single computer.

3. AGENT-BASED SYSTEM ADMINISTRATION

General System Architecture
In this section, we present the architecture and the behavior of ABSA. We divide the computers present in a network into two categories, the central manager node, from which we manage other nodes in the network, and the client nodes that are managed by the central manager node. The central manager node is responsible for receiving the administration requests, analyzing the requests and dispatching necessary agents to appropriate client machines to carry out the request(s) and report the status.

Within this overall architecture, there exist multiple agent classes, both stationary and mobile, and including both intelligent, and less intelligent software agents. Central manager node has different stationary agents within itself to perform the necessary tasks. The only mobile agents in this architecture are the Action Agents which migrate to the client nodes to perform the requested tasks. We refer the reader to figure 2 for the following discussion of the architecture in a network of heterogeneous systems.

Figure 2 illustrates the general system architecture. Before progressing to describe how the system operates, we list the types of the agents that at this moment are used in the system together with a brief description of each.

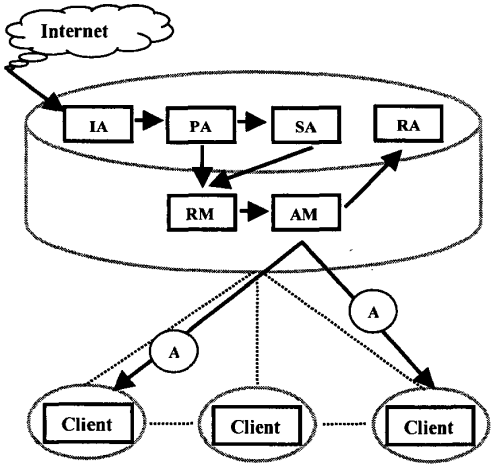

Figure 2 System Architecture

Internet Agent (IA): It receives the administration requests and also requests for the status of submitted tasks via internet and is actually the server side for web-based GUI. The IA is a stationary agent on the central manager node. For each submitted task IA generates unique ID that could be used at a later time to find the status of the task. IA sends the submitted administration requests to the processing agent and status related requests to report agent.

Processing Agent (PA): Receives requests from the IA. It processes the request and puts them into a data structure. The PA also deciphers if the task is one time task or a scheduled task. If the task is scheduled one, it is sent to the scheduler agent, else it is sent to the request manager. It is also a stationary agent on central manager node.

Scheduler Agent (SA): It is a stationary agent on the central manager node. Responsible for generating requests to the request manager for scheduled tasks and managing the tree data structure used to keep information about the scheduled tasks.

Request Manager (RM): Maintains the request queue on a priority basis. It could receive requests from PA or SA depending on the type of the task. It is stationary agent on a central manager node.

Agent Manager (AM): Responsible for generating mobile agents in the system to carryout the requested tasks. It receives a task from the RM and generates an appropriate action agent to perform the task. It then moves the action agent to the client on which the task has to be carried out. AM is again a stationary agent on the central manager node as well.

Report Agent (RA): It is a stationary agent on the central manager node. RA is responsible for maintaining status of the tasks being managed by the AM. It also maintains the log file for all the submitted tasks for each client machine. RA also responds to requests from IA by performing certain query operations on the log file and providing the query results to IA.

Action Agent (AA): These are mobile agents generated by the AM to perform the requested task. AA is a broad term given to a set of task-oriented agents. There are different action agents for different tasks. AA migrates to the client machine, performs the requested task and informs the AM about the status.

System Behavior

At this stage of the implementation of the system, we have focused in automation of just few routine tasks such as managing user accounts, backup, upgrading application software, applying patches, antivirus updates and checking printer status. This section describes a typical scenario that utilizes the above named agents for system administration.

As it was mentioned earlier, the Processing Agent receives requests from the Internet Agent. Since these requests may be simultaneous, the PA maintains a FIFO queue for the inputs. It decodes the task requests, puts the information into appropriate data structure and sends them to the Scheduler Agent or the Request Manager based on the type of the task. If the task needs to be scheduled, it would be send to SA; otherwise, one time tasks are sent to RA.

The IA provides a web-based GUI and is used for submitting tasks as well as viewing their status. Upon submission of a task request, the user is given a unique task ID. The task ID is generated based on the current time (including month and year, in order to generate a unique ID), and the user can later use it to obtain the status of the submitted task. The IA gets the status of the task from the Report Agent (not shown in the figure).

The Scheduler Agent preserves a two level tree structure in which the first level contains the hostnames of the computers in the network and the second level includes the scheduled tasks for each computer. Each node in the first level of the tree, in addition to a hostname, holds the next immediate scheduled task. The next level of the tree maintains the list of the scheduled tasks to be performed on each host. This is illustrated in Figure 3. Whenever a scheduled task is picked for operation or a new task is added to the tree, the SA searches the second level to find the next immediate task for each node and place it at the first level by the hostname. This is done in order to reduce the search time.

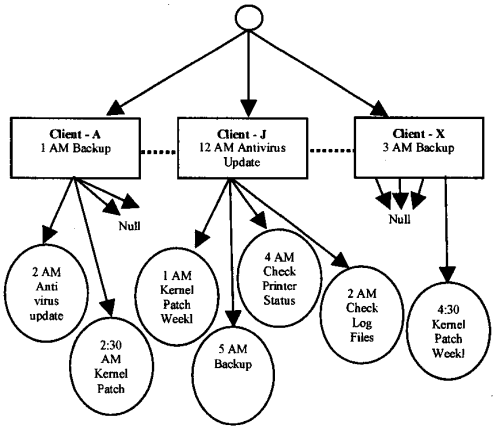

Figure 3 Scheduler Agent Data Structure

The Request Manager receives the task requests from the PA (one time tasks) or the SA (scheduled tasks). It maintains a priority queue of the requests. The priorities are assigned based on the origin and the significance of the requests. If the origin of a request is a regular user, its priority is less that a request from the administrator. In addition, the priority of an "antivirus data updating" task is higher than the priority of a "create user account" request. The system has a default priority setting; however, the administrator can change these priorities.

The Agent Manager has a threshold on the number of Action Agents it can maintain at a time. When the number of AAs in the system is less than the threshold, the AM accepts new tasks from the RM and creates appropriate AAs to be dispatched to the corresponding client computers. After creating an AA, the AM sends its task request to the RA which assigns the "in process" status to the task. Upon completion (or failure), the AA reports the status to the RA (either "completed" or "error" with a code number). The AA will be suspended after completion of its task.

As it was mentioned, there is a different AA for each of the tasks. For instance, for creating a user account we have Create User Account AA, for updating antivirus definitions we have Antivirus AA and so on. AAs are the only mobile agents in the system and most of them have some level of intelligence.

An example of one time task such as create user account will go through the following sequence of agents in the order specified: IA followed by PA, RM, AM, AA and RA. An example of scheduled task such as backup will also go through the same sequence of agents except that SA is in between PA and RM since it considered a scheduled task. In order to facilitate the operation of the system across multiple operating systems, the choice of implementation tools is vital.

4. IMPLEMENTATION TOOLS

The system is being implemented in Java and over Grasshopper agent environment, while the knowledge bases of the intelligent agents are being written in Java Expert System Shell (JESS). The choice of Grasshopper platform and JESS were based on a comparative study of existing tools and environments [3]. Version 0.2 of the system is actually functioning and is being tested at this time.

Grasshopper is implemented completely in Java and is designed in conformance with the Object Management Group's Mobile Agent System Interoperability Facility (MASIF). The platform can be enhanced with an add-on, which is compliant with the specification of the Foundation for Intelligent Physical Agents (FIPA) [4].

JESS is a rule engine and scripting environment written entirely in Java. Jess is Java implementation of CLIPS expert system shell and is a scripting environment, from which objects can be created and methods can be called without compiling any Java code [5]. Java provides APIs for network communications, implements threads, remote procedure calls, web request processing, and also gives the system the advantage of platform independence. Therefore ABSA is capable to manage networks of different operating system platforms.

Now, we further extend this discussion to important data structures followed by some implementation details for each of the agents in the system.

One of the important data structure used in the architecture is the synchronized circular shared buffer. This buffer is used by all the stationary agents in the system to communicate with one another. Since the buffers are shared between concurrently running agents, only one agent should be allowed to access the buffer in order to maintain the buffer consistency. Java provides APIs for synchronize access of objects, which allows

only one thread to access an object at a time. We use this synchronization and create synchronized circular shared buffer objects for communication.

Another important data structure is the tree structure used by Scheduler Agent, which was discussed earlier. We now extend the discussion to some implementation details of the agents in the system.

Internet Agent: It is a Java Servlet which responds to web requests. IA communicates with PA and RA using Datagram Sockets. Using sockets for communication provides us with the advantage of having IA either on the central manager node or on a different web server and still be able to communicate with PA and RA. For administration related requests, IA first verifies if the necessary parameters to carry out the task are submitted, then it concatenates the received parameters in a particular sequence and passes it on to PA. For status related requests, IA passes the received query parameters to RA and displays the output generated by RA to the user.

Moreover multiple task requests can be batched together in a file. IA can accept batch files and pass on a request for each one of the tasks in the batch file to PA. Batching is very convenient especially when a task has to be performed on multiple hosts, such as fixing bugs, installing patches, holiday shutdown etc.

Processing Agent: Receives requests from the IA using Datagram Socket. It decodes the task to be performed and forms an appropriate data structure, and then it passes it to SA or RM depending on the task type using synchronized circular shared buffer object.

Scheduler Agent: It reads from the shared buffer object of PA and writes it into the tree data structure (discussed earlier). SA processes the tree in such a way that the tree always holds the next task to be performed at the first level. At the scheduled time, SA writes the request to the shared buffer object of RM.

Request Manager: It reads from the shared buffer objects of PA and SA, and maintains a priority queue. It shares this priority queue with AM. AM reads the topmost request from this queue.

Agent Manager: It invokes an appropriate agent class for the task and migrates the agent to the client using Grasshopper Agent Platform. It also writes the status of the tasks to shared buffer object of RA.

Report Agent: It reads from the shared buffer object of AM and updates the log file for the task status. RA can search the log file for a task ID, tasks on a particular host and tasks submitted on a particular day.

Action Agents: They are mobile agent classes. Each AA is specific to the task and to the operating platform on which the task has to be performed.

Figure 4 illustrates UML sequence diagram, depicting the flow of control between agents. This diagram describes the timing sequence of method calls between different classes. The flow of control is initiated by user request to the IA. The arrows in the sequence diagram correspond to the method calls.

All the stationary agents shown in the UML diagram (figure 4) are java threads running in parallel. These agents communicate with each other using either datagram sockets or circular shared buffers as discussed earlier. The UML sequence diagram also depicts some of the important methods used. Flow control starts with a web request from user to IA either for performing a task or to know the status of a submitted task. The *doPost* method of IA handles these user requests and the *getTaskID* method generates a unique task ID for each task request. The *send* method of IA transfers the user request either to PA or RA depending on the type of the user request.

The *send* method of IA corresponds to the *receive* method in PA which receives the task request. Upon receiving the request, the *decode* method of PA determines the type of the task and calls appropriate method of the *GenericDS* class. The *writeToMPI* method of PA writes this request to shared buffer of SA or RM depending on the task (scheduled or unscheduled).

Let us assume that the task is a scheduled one such as performing a backup. The SA reads a task request from the top of the buffer and adds it to *SATree* using *addTask* method. The *SATree* is processed by *processTree* method of SA to arrange the tasks in such a way that the next task to be performed on the host is at the first level of the tree as discussed earlier. The *taskAvailableToPerform* method of SA periodically checks the *SATree* to find if any tasks are available to perform; if available the requests are written to shared buffer of RM.

RM reads the requests from the buffer shared with SA and PA and uses its *processQueue* method to rearrange the tasks in the queue on a priority basis. The priority queue maintained by RM is the shared buffer to AM. The *isWriteable* method of RM writes the task request to this priority queue.

The AM reads the priority queue maintained by RM using *isReadable* method. This method always reads the topmost task in the priority queue. For each task read, AM creates the appropriate AA and migrates it to the client to perform the task. AA upon completion of the task, reports the status to AM. AM writes the status of the performed task to the shared buffer of RA using its *report*

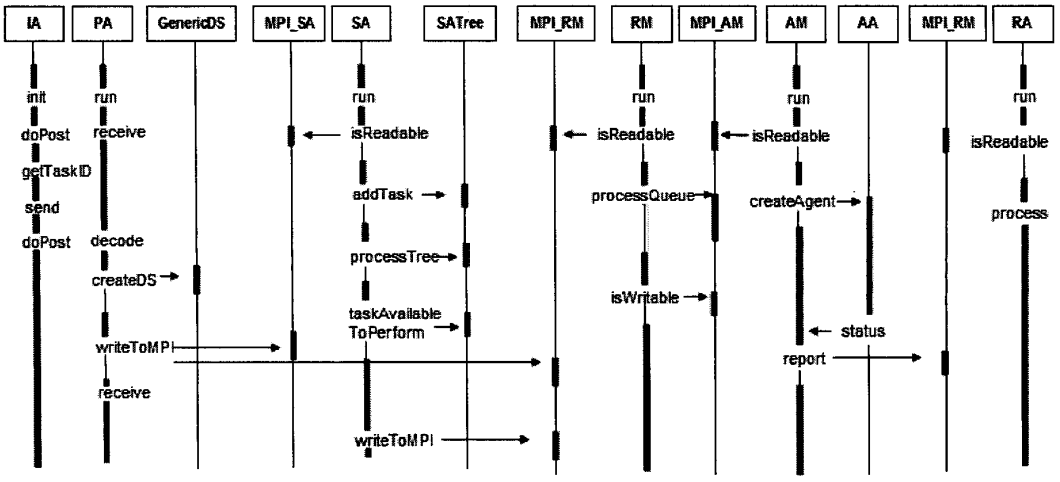

Figure 4 UML Sequence Diagram

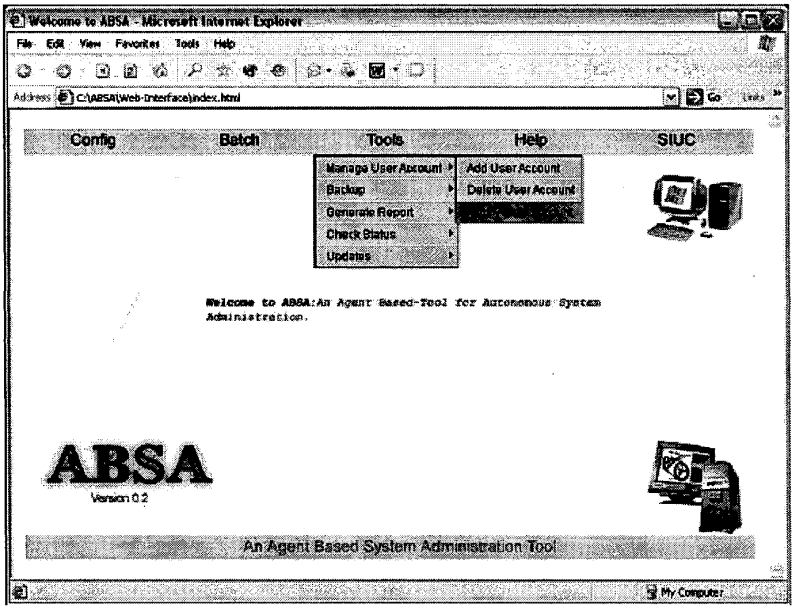

Figure 5 ABSA: Main Interface

method. RA reads the shared buffer using *isReadable* method and writes the status of the task to the log file, maintained by itself using the *process* method. The *process* method also sorts the log file based on the task ID.

Figure 5 shows the web-based graphical user interface of ABSA. System administrators can log in from anywhere in the world and use the system. Upon choosing a task, the user gets an interface with parameters specific to that task. Figure 6 shows the interface for create user account.

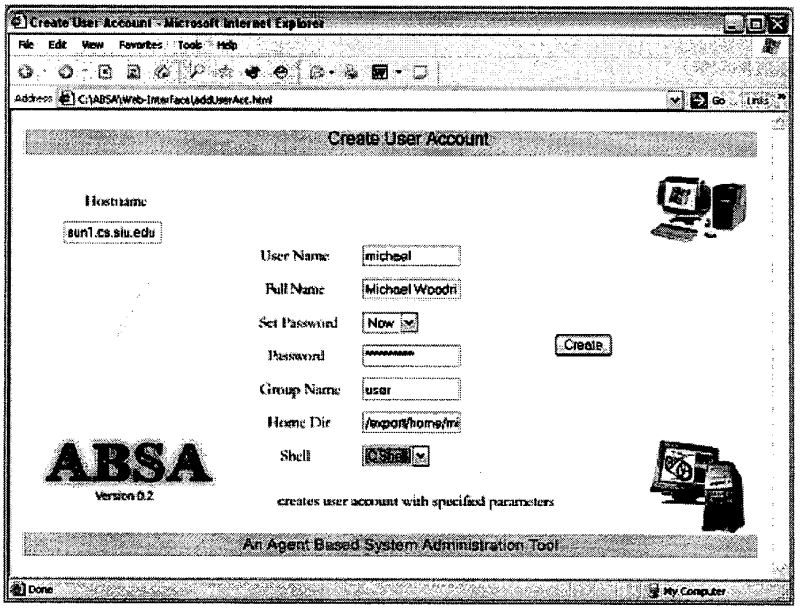

Figure 6 ABSA: Create User Account Interface

5. SUMMARY

This paper presents ABSA, a new tool for automation of system administration based on a novel agent-based architecture. System administration by itself is a challenging area; besides, the added complexity of working with different platforms in a heterogeneous environment is immense. The agent technology in our architecture, augmented by expert system capabilities, demonstrates a remarkable capacity for managing these complexities and producing satisfactory results. By employing agent technology, ABSA is capable of distributing the administrative tasks among the computers in the network and prevents the concentration of the tasks on a central computer. This gives the system scalability and more reliability. The first version of the software has been implemented and the initial results are promising.

REFERENCES

[1] Miller and Donnini, "Relieving the burden of system administration through support automation" **Proceedings-of-the-Fourteenth-Systems-Administration-Conference-LISA-XIV**. 2000: 167-80

[2] Shoham. Y, "An Overview of Agent-oriented Programming", **Software Agents**, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press. 1997.

[3] S. Rahimi, R. Angryk, J. Bjursell, M. Paprzycki, D. Ali, M. Cobb and K. Kolodziei, "Comparison of Mobile Agent Frameworks for Distributed Geospatial Data Integration," **Proceeding of the 4th Agile Conference on Geographic Information Science**, Brno, Czech, pp. 643-655, 2001.

[4] Grasshopper Programmer's Guide, URL: http://www.grasshopper.de.

[5] Jess- the Java Expert System Shell, URL: http://herzberg.ca.sandia.gov/jess.

[6] "Software Update via Mobile Agent Based Programming" Publication: 2002 ACM 1-58113-445-2/02/03

[7] "The Igor System Administration Tool", **Tenth USENIX System Administration Conference** Chicago, IL, USA, Sept. 29-Oct 4, 1996.

[8] "Central System Administration in Heterogeneous Unix Environment: GeNU Admin" LISA, pp. 1-8, September 19-23, 1994

[9] "WEBMIN:A Web-Based System Administration Tool for UNIX" **USENIX Annual Technical Conference**, San Diego, California USA, June 18-23, 2000.

[10] Gartner Group. A white paper on Gartner group's next generation total cost of ownership methodology, 1997.