System Administration Using Software Agents

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

Keywords: Agents, System administration, Distributed systems.

Received: August 15, 2003

The increasing complexity and difficulty of system administration has been long recognized. Studies indicate that because of the difficulty and complexity, the cost of administering systems is ten times the cost of the actual hardware. Here, we present ABSA; 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. ABSA system supports important system administration features such as domain-wise administration, automated error handling and default system configuration besides others.

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 [2]. 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 present an agent-based architecture to facilitate and automate the system administration tasks. 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 ABSA from other tools available in the market.

The remaining parts of the paper are organized as follows. First, a brief background on different system administration approaches and software agents is given in section 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.

2.1 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 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.

2.2 Analysis of existing System Administration Tools

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 develop tools to aid administrators.

"Software Update via Mobile Agent Based Programming" [3] 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" [4] 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" [5] 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. Administrators have to modify the configuration files on the server to manage the clients. The modifications are automatically transferred to the client systems. Its disadvantages are that it may cause inconsistency among configuration files on the server and the actual client configuration and also it is for UNIX systems only. Our last example is "WEBMIN: A Web-Based System Administration Tool for UNIX" [6], 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.

2.3 Software Agent Technology

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 [7]. 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.

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.

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.

3 Agent-Based System Administration

3.1 General System Architecture

In this section, we present the architecture and the behaviour of ABSA. We divide the computers present in a network into two categories, namely 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, analysing 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 I for the following discussion of the architecture in a network of heterogeneous systems.

Figure 1 illustrates the general system architecture. Before progressing to describe how the system operates, we list different agents that at this moment are used in the system together with a brief description of each.

**Internet Agent (IA):** It receives 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 IA. PA decipheres 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 using appropriate protocols. 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 the 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.

**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. RA also maintains the log file for all the submitted tasks and their current status. RA processes the log file and provides results for status related queries by 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 AM about the status.

As stated earlier, there are different action agents for different tasks. Figure 2 shows the Hierarchies of Action Agent classes. At the highest level we have Action Agents that could perform tasks independent of the operating systems. As we traverse hierarchy of action agent classes downwards, we have action agents which are very specific to operating system. The user account agent class under sun action agents is specific to sun operating system while the user account agent class under windows action agents class is specific to windows operating system. Action agents are sent out to the target systems based on the type of the operating system installed on it.

### 3.2 System Behaviour

At this stage of the system implementation, we have only focused on automation of major 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 and sends them to the Scheduler Agent or the Request Manager based on the type of the task, using appropriate protocols. 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 this ID to obtain the status of the submitted task. The IA gets the status of the task from the Report Agent.

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 the 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 Fig. 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.

The Request Manager receives the task requests from PA (one time tasks) or 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 than that of a request from the administrator. In addition, the priority of an "antivirus definitions 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 progress" status to the task. Upon completion (or failure), AA reports the status to the AM (either "completed" or "error" with a code number). AA will be suspended after completion of its task. AM then updates the status of the task with RA.

As it was mentioned, there is a different AA for each of the tasks. For instance, for creating a user account we have 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.

Domain-wise Administration

One of the key features of ABSA is its support for domain-wise system administration. In case of large computer networks, computers are logically grouped together to form domains. The domains themselves can be grouped together to form a higher level domain. This logical grouping of systems makes domain administration tasks possible. For instance the administrator could create a user account for a particular domain, which can be used on all the computers in that domain.

We now discuss how domain-wise administration is achieved in ABSA. We maintain the domain information of the network in a tree structure. As shown in Fig. 4 each node in the tree contains a domain name, a domain ID and the list of all the users allowed administering the systems in that domain.

From figure 4, users with ID 4 and 5 have same administrative rights for the domain siu.edu and all the domains under it. User with ID 3 can administer computers in cs.siu.edu domain and the domains below it; user with ID 6 can administer computers in domain sag.cs.siu.edu. Users with rights for a domain cannot administer systems in the higher levels; viz. user with ID
The intelligent agents are being written in Java.

When a user submits a task, the user enters the domain name he/she is willing to administer; the system obtains this domain name and his/her user ID. We then perform a traverse on the domain information data structure (shown in Fig. 4), find the domain's node, and check if the user is allowed to administer the system in the domain. If the user is not authorized to perform the requested task in that domain, his or her request will be denied. Furthermore, to set up user permissions, we maintain a profile for each user. Each entry in this profile contains a domain ID followed by a 32-bit number, each bit of this number determines whether a user is allowed to perform a particular task or not depending on whether the bit is set or reset. If the user is allowed to perform a particular administration task on a domain (after checking the domain information data structure), his or her profile is checked for that particular domain ID and whether he or she is authorized to carry out the particular task that he is submitting. Depending on the result of the profile check, his/her request may be accepted or denied. All the above checks are performed by the Internet Agent before accepting the user request.

**Default System Configuration**

ABSA supports configuration of computers from scratch (computers installed with an operating system and an agency to support agents). Administrator maintains a default configuration profile for every domain, which specifies the steps to be carried out and the agents to be sent out. This default configuration profile can be used to configure computers from scratch.

**Automated Error Handling**

ABSA can read the system logs generated by Windows and UNIX operating systems and take appropriate actions in response. The system logs generated by operating systems are in response to events such as information, warning, and error. At this stage of the system implementation, we are handling only the error events. As discussed earlier on every client we have an agency operating. Monitor Agent in the agency reads these system logs at regular intervals of time which is a configurable parameter.

We now discuss how the automated error handling works. The Monitor Agent checks to see if there are any new errors in the log file from the previously read time. If any new errors are found the Monitor Agent communicates with the Internet Agent about the error. The central manager then performs a check in its knowledge base to see if it has a similar error in its knowledge base and the solution for the error. If no match is found for the error in the central manager's knowledge base the error is reported to the administrator.

If a match is found appropriate agent is sent from the Agent Manager to fix the error. The whole picture of automated error handling can be viewed as the Monitor Agent on the client submitting a task request and the central manager node processing the task request.

**Autonomous Software Management**

Another important feature that is supported by ABSA, which is at its early stages of implementation, is Autonomous Software Management. Autonomous Software Management allows application software to be deployed by the users independently in a controlled form as specified by the administrator. Administrators need only to prepare the software once for the system and then leave the entire deployment and maintenance to the system itself. This greatly reduces the workload of administrators and also improves the process of software deployment significantly.

Here is how Autonomous Software Management is being implemented in ABSA. At the central manager node we have Software Manager Agent (not shown in Figure 1), which is a central control application for software management, linked to an SQL database. Administrator has to pre-configure each software by packaging it into a JAR file [11] which contains a Manifest file [12]. Using this format allows the package to be digitally signed for security and at the same time allows the package to be compressed. The SQL database is used to consolidate information pertaining to each package, this information is called an installation profile. Each of these profiles contains a variable and a fixed component, where variable component can be modified by users after application has been installed while the fixed component cannot. Users may use a web-based interface to look up a particular package available for installation on the server.

At the client end, Monitor Agent (discussed earlier), collaborates with the Software Manager Agent to automate the process of software deployment. Monitor Agent is controlled by Software Manager Agent to perform tasks on the computer in which it resides. This agent monitors applications to ensure that their installation profile is properly adhered. It also monitors usage statistics such as how often the application is used. Moreover, adjustments made by the user to the variable component of installation profile are also noted, so that the default installation profile, stored on the server, is updated.

In order to facilitate the operation of the ABSA architecture across multiple operating systems, the choice of implementation tools are 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 [8]. 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) [9].

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 [10]. 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 to 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 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 correct and consistent, then it concatenates the received parameters in a particular sequence and passes it on to PA. For status related requests, IA passes the received status related query 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 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 IA using Datagram Sockets. It decodes the task to be performed and then passes the task request 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 5 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 IA. The arrows in the sequence diagram correspond to the method calls.

All the stationary agents shown in the UML diagram (Fig. 5) 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. After sending the request to PA or RA, IA waits to receive new requests.

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 the 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 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 6 shows the web-based GUI 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 particular task, Figure 7 shows the interface for create user account.

|
| Create User Account - Microsoft Internet Explorer | Create User Account
| Hostname: sun1.cs.ui.edu |
| User Name: Michael Woodr |
| Full Name: michael |
| Set Password: Now |
| Password: |
| Group Name: staff |
| Home Dir: /home/michael |
| Shell: CShell |

![Figure 7 ABSA: Create User Account Interface](image)

**5 Conclusion**

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. ABSA v0.2 was preliminarily evaluated against few current centralized approaches for automation of system administration tasks and the results were promising. The performance tests were based on the following criteria: expandability, extent of automation, error rate of the overall architecture, overall security of the architecture, multiplexing (distributing) of tasks, and ease of use.

The performance of ABSA v0.2 architecture was better than other approaches in many of the criteria's used for the evaluation. ABSA approach was highly scalable compared to other approaches. Upon increasing the number of computers in the network, the performance of ABSA was relatively steady, while the performance of other centralized administration approaches fell due to increased load on the central server. ABSA system does not maintain client configuration files on the central
manager node as compared to other approaches, thus avoids any possibility of inconsistency between the client and the central manager. The user interface of ABSA is very friendly and is independent of the operating platform, which allows administrators familiar with only one operating platform to administer systems of other platforms as well.

ABSA has few disadvantages such as security of the agents in the system. One more disadvantage of ABSA is that it needs to keep up with new releases of operating systems. Moreover, since ABSA supports multiple operating systems, maintenance may be needed more frequent compare to other systems. Although ABSA has some limitations, the advantages outnumber these disadvantages.

References


