RC22410 (W0204-100) April 24, 2002
Computer Science

IBM Research Report

Security and Policy Management Issues in Automation Controller Design

Xin Yu, Ajay Mohindra
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Security and Policy Management Issues in Automation Controller Design

Xin Yu[?]
Dept. of Computer Science
Courant Institute
New York University
New York, NY

Ajay Mohindra
IBM T. J Watson Research Center
Yorktown Heights, NY

Abstract

Subscription computing is the provision of a computing infrastructure as a service. Subscription computing relocates much of the responsibility for selecting, configuring, and maintaining a computing infrastructure to a remote service provider. To reduce the operating costs associated with remote management, we designed and implemented an *Automation Controller* to automate some aspects of systems management. In this paper, we describe the design and implementation of the automation controller prototype for a user management system. Specifically, we address the technical challenges related to security and policy management.

1. Introduction

With the rising costs of IT, companies are moving towards a concept known as *e-sourcing* in which management and day-to-day operation of a company's IT infrastructure is performed by an outside service provider. *Subscription Computing* is a special case of *e-sourcing* in which the computing infrastructure is provisioned as a service. To improve profitability and efficiency of *e-sourcing*, service providers are investigating ways to reduce the operating costs by designing and deploying *intelligent infrastructures*. These infrastructures include technology to remotely monitor computers, diagnose problems, and remotely take over the operations of a system to help users whenever necessary.

To reduce costs associated with *e-sourcing*, we are investigating ways to automate tasks associated with IT management. There are two main challenges to achieving the goal of automation of system administration. First, the system should possess the knowledge that system administrators carry "in their heads" and should be able to reason based on the different events to determine the situation and take corrective actions to resolve the situation. Moreover, the system should also have the ability to act proactively to predict and resolve problems even before they occur. Second, the system infrastructure should be extensible so that it could be easily modified whenever needed.

[?] Xin Yu was a summer intern at IBM Research.

In our work, we address the two challenges using the intelligent agent technology. We represent the knowledge and the policy required to perform any system administration task as rules. In this paper, we discuss the architecture, design, and implementation of the automation controller using a user management system as an example.

We also examined issues related to security and policy management for the automation controller. The main technical challenge in automation controller is maintaining the security and integrity of customer's data. Since system administration tasks require very high security and all communications are on the Internet, we have to authenticate the identity of users in order to ensure that the service is provided to a trusted person. We solve client authentication problem by using digital signatures to ensure non-repudiation of the sender. To ensure user's data privacy and integrity, we used SSL for encryption. We also looked at the issue of policy management as in system administration as IT policies affect the decision-making process. We addressed the issue of policy management via a framework that flexibly combines policy with basic rules.

2. Related Work

At present, several researchers are investigating ways to automate system administration. Many of these systems rely on generating shell scripts for Unix-based systems [1][2][3][4]. Others have developed tools to create and deploy preconfigured system configurations. The most popular tool is Cfengine [5][6][7]. It can coordinate deployment of system configuration over large number of systems. In [8], authors discuss an approach of using the Prolog programming language for configuration control. The goal of the research work is to reduce the human involvement in system configuration. Our work, in contrast, focuses on automating system administration tasks for a wide range of activities such as user management, disk management, software installation, and server management.

Another approach to automating system management is based on writing custom web-based applications. In this approach, procedures and rules for performing system management tasks are embedded in the software application. Such an approach is difficult to maintain, as it requires changes to the application whenever rules and policies change. In our work, we separate system administration rules and policies from actual transaction processing, and store these rules and policies in a knowledge base. This approach enables us to easily modify the application logic by editing only the knowledge base and not modifying the application code.

3. Architecture of the Automation Controller

In this section, we present the architecture of the Automation Controller (AC) using a user management system as an example. First, as a background, we introduce the concept of a simple reflex agent model that forms the basis of the design of the Automation Controller.

3.1 Simple Reflex Agent Model

An agent is an entity that perceives its environment through sensors and acts upon that environment through effectors [9]. A simple reflex agent works by finding a rule whose conditions match the current situation and then executing the action associated with that rule. Figure 1 shows the schematic structure of a simple reflex agent.

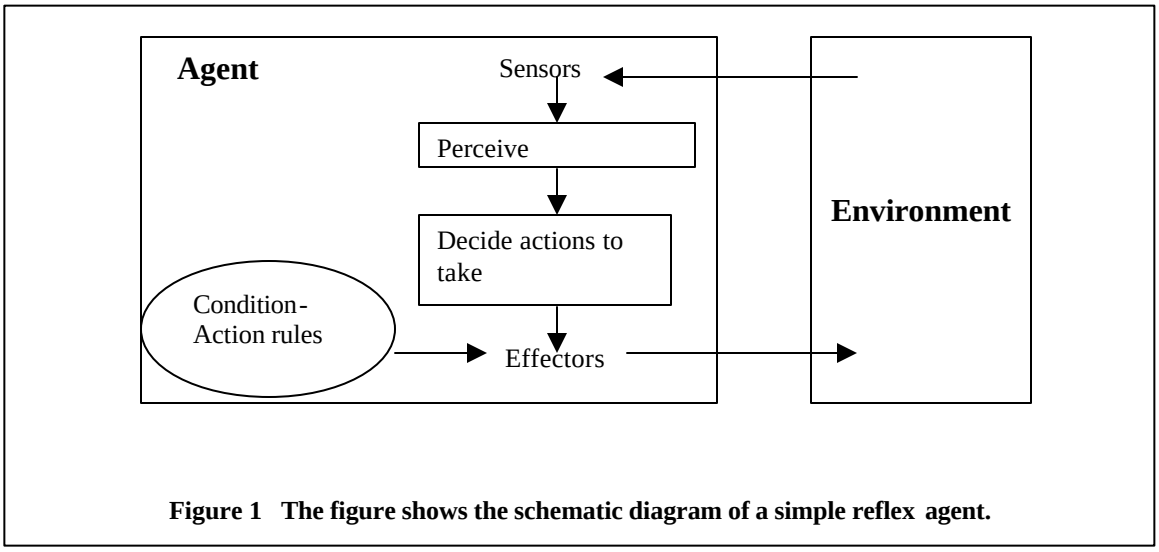

Figure 1 The figure shows the schematic diagram of a simple reflex agent.

3.2 Architecture of the Automation Controller

The design of the AC is based on the reflex agent model. It has four basic components: a rule evaluator, a situation analyzer, an action generator, and a policy manager. Figure 2 shows the architecture of the Automation Controller.

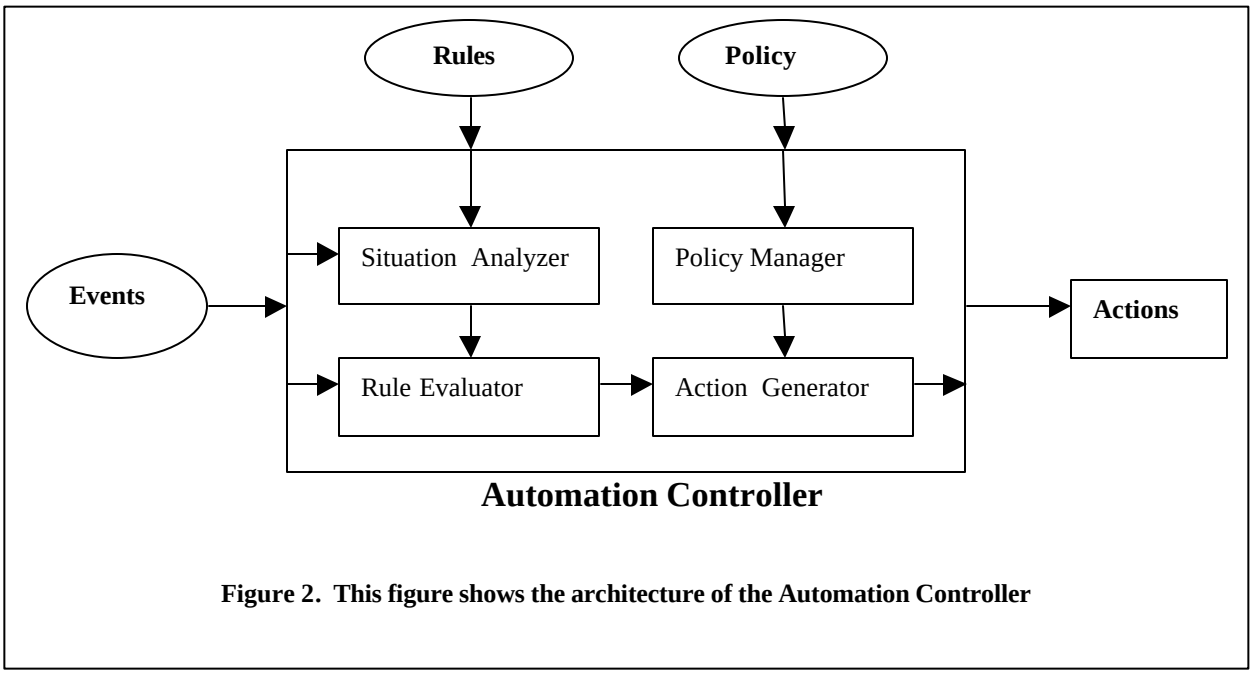

Figure 2. This figure shows the architecture of the Automation Controller

In the AC, each system management task is represented as a set of rules that specify the behavior of the system for a particular request. The requests are received as events. The situation analyzer correlates the incoming events and tries to determine the situation. The information is then passed onto the rule evaluator. As the situation analyzer is responsible for determining the problem situation, it acts like a "sensor" of an agent.

The rule evaluator is a rule-based inference engine. When an event is received by the rule evaluator, it locates the rule corresponding to that event and evaluates the rule. If the rule evaluator reaches a valid conclusion then it triggers an action associated with that rule via the action generator. The action generator component acts like the "effector" of an agent.

The policy manager is responsible for managing the operating policies associated with the AC. In system administration, policies specify the additional conditions associated with different rules before associated actions can be taken. Therefore, a policy can affect action generation. For example, a policy such as "No more than 50 users can share a server" can be associated with an action that adds new user accounts to the server.

3.3 Example: User Management System

We use the user management system as our prototype implementation of the AC. In user management, basic tasks include adding users, deleting users and resetting passwords. Next, we describe the process of how the automation controller implements these tasks. We use CommonRules [10] as our rule engine. The sensor and the effector procedures are attached to the rule engine. Figure 3 shows the architecture of the user management system.

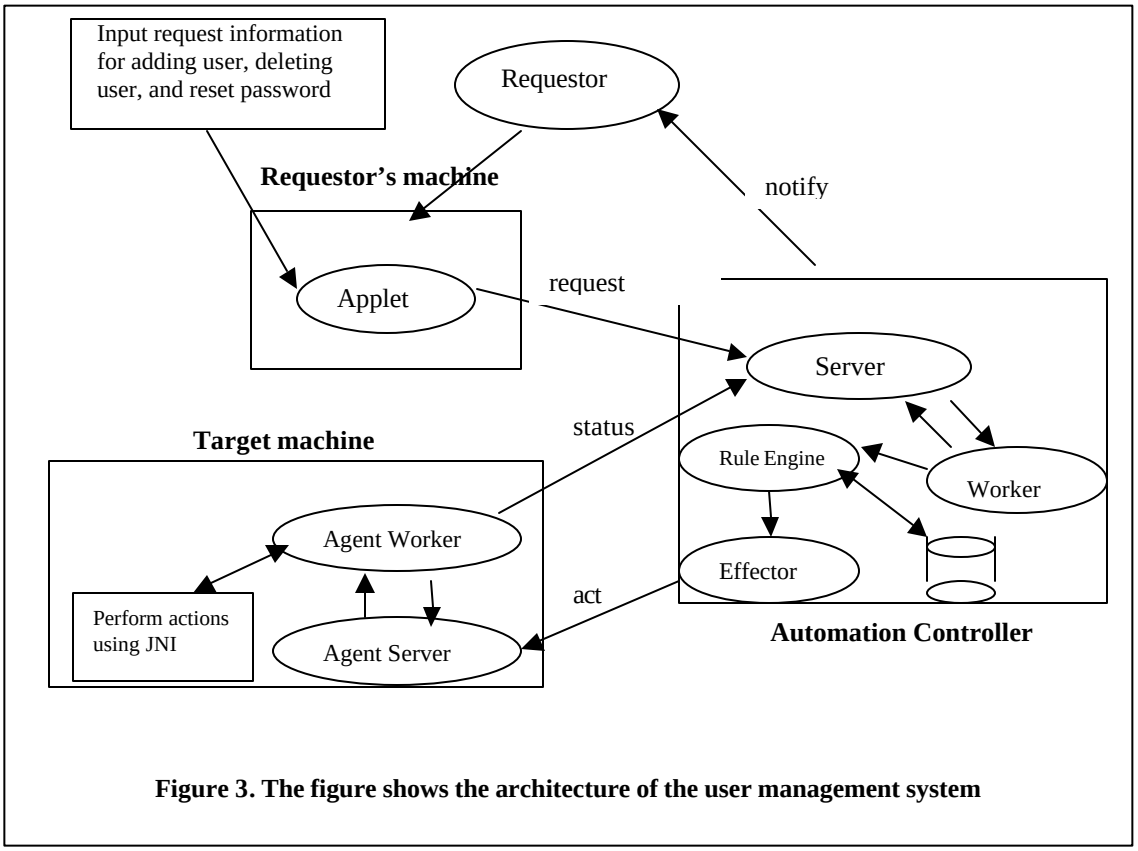

Figure 3. The figure shows the architecture of the user management system

Figure 3 shows the architecture of the user management system. The Requestor machine is the system from which a user submits a user management request using a browser. The Automation Controller is the system where the AC executes. It consists of a primary server with a set of worker threads. The server thread receives requests from the network and queues them for the worker threads to process. The rule engine also resides at this system. Finally, the Target machine is the system that is being managed by the AC. The target machine hosts a server and a set of workers for performing actions specified in the message. Associated with each action is a piece of code residing on the target machine that performs the task using JNI and the Window operating system APIs.

Upon completion of the action, the worker threads return any errors back to the Automation Controller for processing. Upon receipt of this status message, the Automation Controller uses the rule engine to take appropriate actions for the status message. For example, the status message may require notifying both the requestor and the administrator via an email message.

4. Security Issues in Automation Controller

When IT is provided as a service over the Internet, one of the biggest challenges is client authentication, security, privacy and integrity of customer data. The IT service should be provided only to authorized users. For every request that is executed, the system has to ensure the identity of each requestor.

Although, SSL provides support for both client and server authentication as part of normal SSL handshake, it does not authenticate the identity of the person initiating the request. SSL also does not provide any non-repudiation service. In this section, we describe our approach to solving the client authentication problem.

4.1 Architecture for Client Authentication

We use digital signatures for client authentication. When the receiver of the data receives the digital signature, he uses the sender's public key to verify the authenticity of signature. A receiver needs to ensure the authenticity of the public key itself before using it to check the authenticity of the signature. In our design, we use two *keystores* -- one contains the administrator's key entries, and the other contains the trusted certificate entries for the administrator from a public Certification Authority (CA). Figure 4 shows the architecture for client authentication.

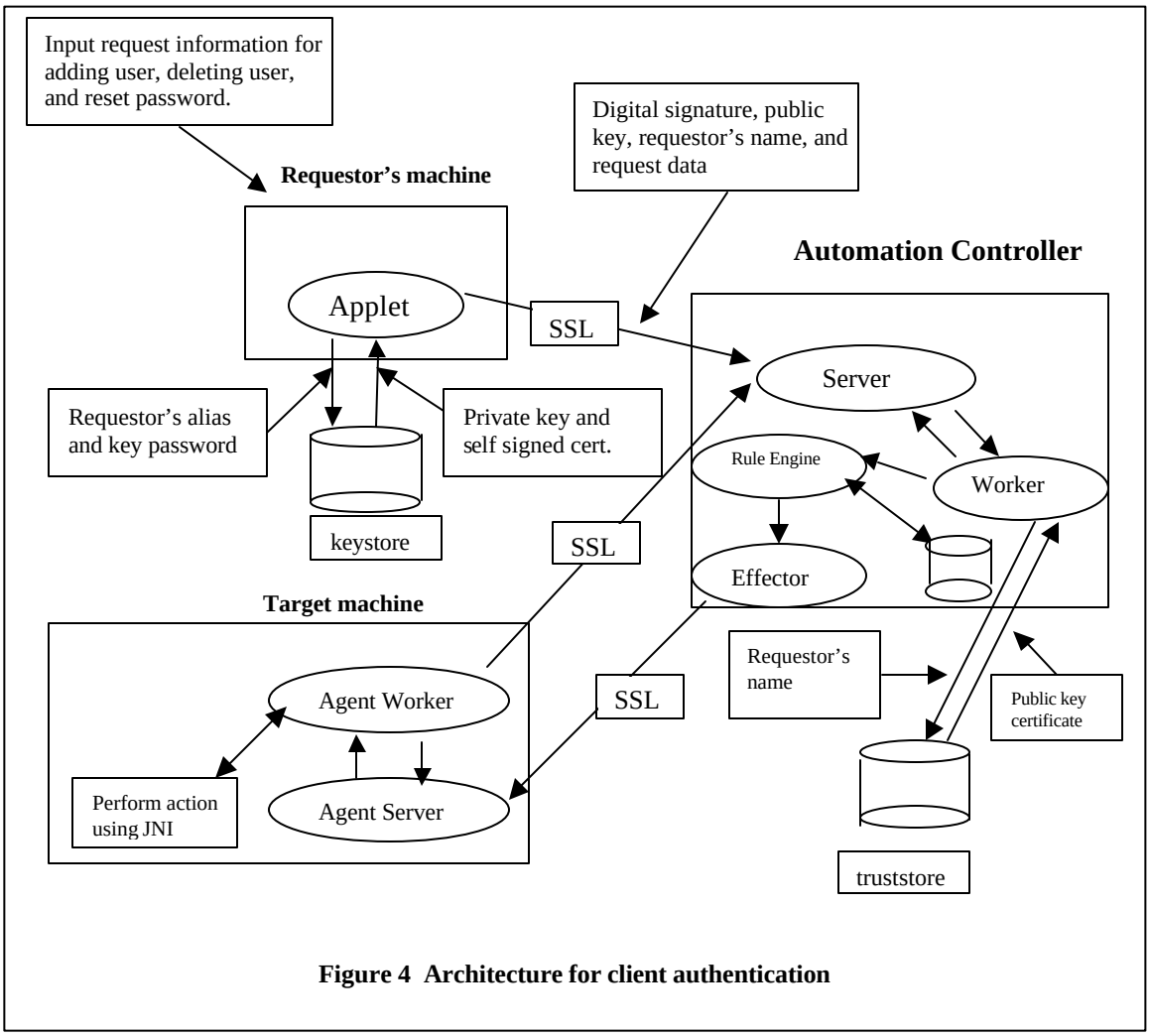

Figure 4 Architecture for client authentication

4.2 Implementation of Client Authentication

The sequence of steps in the implementation of our prototype system is as follows.

1) Create a keystore for the administrators (users of the remote administration system) using *keytool*. The keystore is protected by a keystore password and each key entry in the keystore is protected by a key password.
2) Create a key entry for each administrator. A matched public/private keypair is generated together with a self-signed certificate in which the issuer is the same as subject (the entity whose public key is being authenticated by the certificate).
3) Generate a Certificate Signing Requests (CSR) file for the specified alias in keystore. Then use the Microsoft Certificate Server to get a certificate from public Certifying Authority (CA). Alternatively, the CA can also directly issue the certificate

4) Create a *truststore* and import the certificate from the CA into the truststore. The certificate is sent to the AC.
5) When the requestor enters the information, the client program uses the alias and the key password to retrieve the private key and the certificate from the keystore. The client program then uses the private key to sign the data and generate a digital signature using the Java security API.
6) The client program then sends the data, the requestor's digital signature and the public key extracted from the self-signed certificate to the AC.
7) The worker thread picks up the request message and verifies the data integrity using the received public key to decrypt the signature. If this step fails then the request is discarded as the data could have been modified in transit. The reason for this check is to ensure that the data has not been modified.
8) If step 7 succeeds then the worker thread proceeds to authenticate the identity of the sender by using the public key associated with the requestor extracted from the truststore. For example, if Alice claims to be Bob then she will enter her name as Bob. However, she can use only her own private key to generate the signature. When we use Bob's public key to check the authenticity of signature, the impersonation is detected.
9) If the verification succeeds then worker thread invokes the rule engine to begin the evaluation of rules according to the request identifier. Upon reaching a conclusion, the worker thread sends a message to the agent server to take appropriate action. In case of failure, the AC sends an email notification to both requestor and administrator.

We use SSL to encrypt all messages transmitted over the Internet. Although SSL provides optional client and server authentication, the "client" here refers to the client machine. The *keystore* and *truststore* files are deployed both on the client and the server for use by SSL. While SSL provides authentication, privacy, and data integrity, it does not provide non-repudiation services. Non-repudiation means that an entity that sends a message cannot later deny that they sent it. We use digital signatures to provide non-repudiation.

4.3 RSA-Signed Applet

Applets typically run under the scrutiny of the Java security manager. An applet downloaded over the Internet is prevented from reading and writing files on the client. JDK1.1 supports the notion of trusted applets, which enables browsers to run trusted applets in a trusted environment. However, a trusted applet is not allowed to access local resources unless explicitly granted permission to do so by the security policy in effect.

Since an applet in our system needs to read a *keystore* file on requestor's machine, we create an RSA signed applet, and grant necessary security permission in policy file of client machine. The steps are as follows:

1) Create a policy file to grant the required permissions: read keystore and truststore file in the client machine, dynamically install SunJSSE provider, and read/write system properties. Save the file as *.java.policy* file under the user home directory on the client machine. Since the applet runs inside a Java plugin, we need to set the policy file location property in *java.security* file in the JRE home directory (typically \Program files\JavaSoft\jre\lib\security\java.security).
2) Obtain a public key certificate from the CA, import the certificate to the truststore in the client machine, and modify the java.security property file to specify the location of the truststore.
3) Archive the applet class files into a JAR file using the *jarsigner* tool to sign the JAR file with the private key associated with the certificate obtained above.
4) Install the Java Plugin 1.3.0 at the client machine, and use the HTMLConverter to convert the html page to one that can be loaded by the plugin

The trusted applet provides the user interface for the user management system.

5. Policy Management in Automation Controller

All system administration tasks and procedures are governed by company wide policies that are in line with the business. A policy refers to additional conditions or constraints that must be satisfied before an action can occur. As a policy can be often added, deleted, or modified, the policy management should make the maintenance work easy. Our goal is to construct a framework to combine policy with basic rules flexibly.

In our prototype implementation, we considered two kinds of policies. One is an assertive/proscriptive policy, while the second is a time-based policy. An example of an assertive policy is "All passwords must have at least six characters." An example of a proscriptive policy is "No more than 50 users can share a server." Time-based policies refer to policies that have a time component in the policy. For example, "the daily backup takes place at 8pm." In this section, we describe the implementation of policy management.

5.1 Design of Policy Management

There can be two approaches to applying policies in decision-making. The first approach uses the sensor procedure to load the policy database and evaluate the appropriate policy according to the request type. The second approach uses the effector procedure.

In the first approach, the rule engine is triggered in the sensor procedure, and if a conclusion is reached then a true value is returned by the sensor method. Otherwise, a false value is returned. The predicate *applyPolicy(?Requestid)* in each rule evaluates to either true or false based upon the fact whether the policy for that request is valid or not. One drawback to this approach is that it can only get a conclusion fact set, but cannot get

a particular parameter value from the policy file such as the time for a task specified for a time-based policy. Therefore, we chose not to use the sensor-based approach.

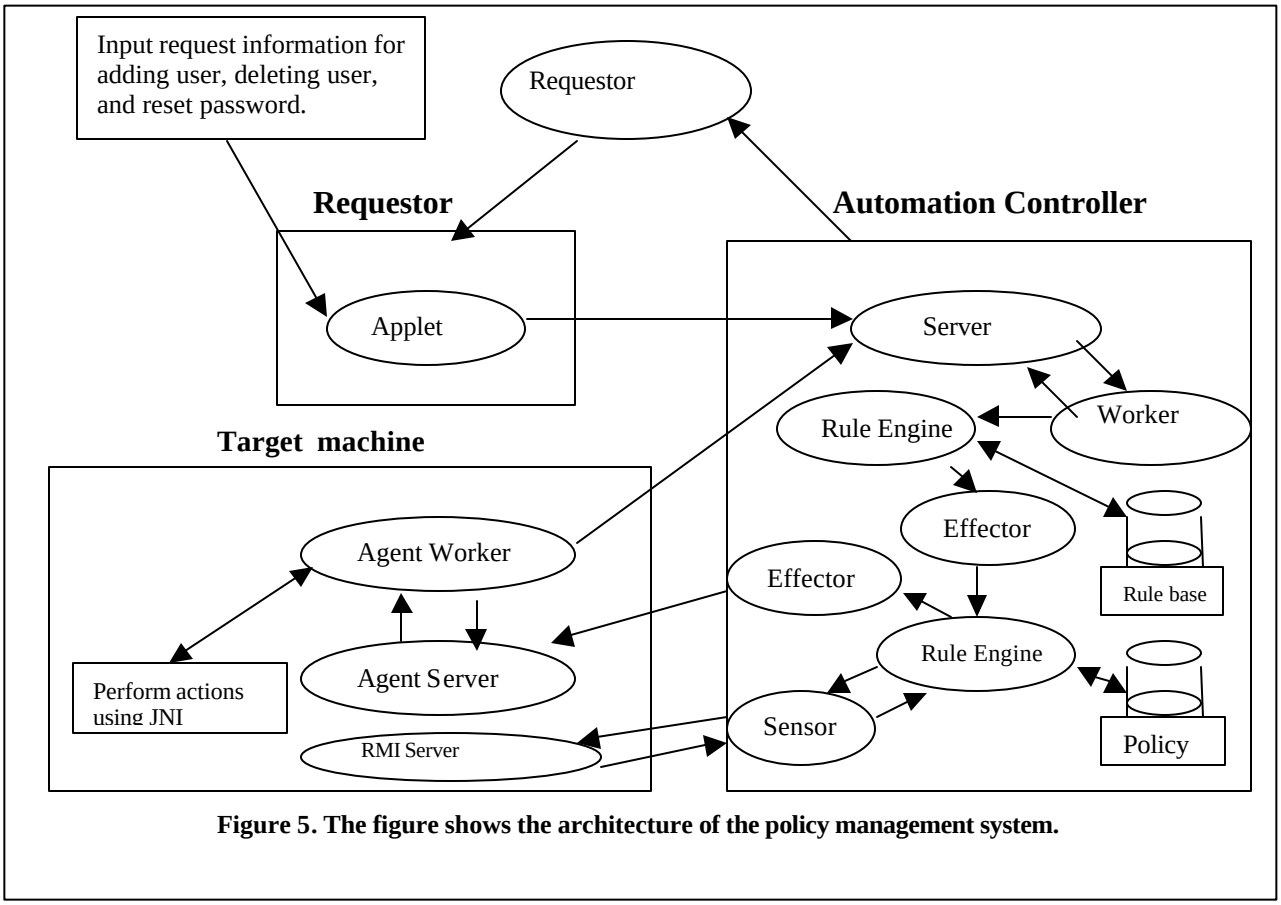

Figure 5. The figure shows the architecture of the policy management system.

We discuss the effector-based approach to policy management using the policy "No more than 50 users can share server seattle." as an example. Figure 5 shows the architecture of the policy management system.

1) In the effector procedure, another instance of the rule engine is triggered with the *Request* object and the second policy file *policy.clp*. According to the requestid, a specific policy rule will be evaluated for that action. In *policy.clp*, we represent the above policy as:

```
validAddPolicy(?Requestid,?Servername,?Number) ✍
        Request.requestid(?Requestid, ?Request) AND
        equals(?Requestid,"ADD_USER") AND
        validCompany(?Company) AND
        serverName(?Servername, ?Company) AND
        userCount(?Servername,?Number) AND
       lessThan(?Number,50);
```

2) A sensor procedure is used to get the environment state from outside of the intelligent agent. In our case, the rule engine is running at the server side, the outside environment

refers to the target machine. We use a sensor for the predicate userCount(?Servername,?Number) to get the number of valid users on the target machine.

3) The communication between the target client and the server is done using RMI and SSL.

4) In the *Sensor_GetUserCount* class, the RMI client makes a call to the method getUser() from the remote RMI server at the target machine, and gets the return value of the number of users on that machine.

5) If the number of users is less than 50 then the rule validAddUserPolicy() is true. Therefore, in the effector for this policy, a message will be sent to the target to take action of adding a user to the target machine.

The policy file can be modified without needing to change the main rule base, or the application structure and code.

5.2 Design for a time-based policy management

As described earlier, the time-based policy refers to policies that have a time component associated to them, e.g., "run system backup program at midnight", "run anti-virus program at noon." Such policies raise another problem: if a user submits a request to run backup in the morning but policy specifies that backup can be done only at midnight, then the actions taken by the AC are not clear. We use queuing events to resolve this problem.

In the design, we maintain an event queue to sequence all incoming unique requests by time, where "unique" means that the target machine, the task time, and the request type identify each event. Therefore, for a specific machine and a specific task, there can only be one action pending even if a user sends in multiple requests to the same machine with the same request type. The outgoing events are sorted by time.

Next, we describe the scheme using the example in which a user sends in request to run backup at midnight.

1) First, the rule engine evaluates the request information such as the requestor's name, and target machine by loading the main rule base *admin.clp*. Once a conclusion is reached, the effector procedure invokes another rule engine to evaluate the policy for that request specified by *requestid*.

2) When a policy is defined in the *policy.clp* file, it is tagged as being time-based for use by the effector to decide whether to construct an event message or not. Two variables are added to each policy rule, one is "TimeFlag"("True" represents this is a time-based policy) and the other is "Time." A fact is also added for each task to specify whether it is time-based or not.

Here is an example of policy “Run backup program at midnight 12”:

```
validBackupPolicy(?TimeFlag,?Requestid,?Servername,?Time) <-
        Request.requestid(?Requestid, ?Request) AND
        equals(?Requestid,"BACKUP") AND
        validCompany(?Company) AND
        serverName(?Servername, ?Company) AND
        getCurrentTime(?CurrentTime) AND
        BackupTime(?Time,?TimeFlag) AND
        lessThan(?CurrentTime,?Time);
validCompany("Doc.com");
validCompany("Legal.com");
serverName(seattle, "Doc.com");
BackupTime(12,"TRUE");
```

3) When a task is identified to be time-based and the specified time is after the current system then the effector program constructs an event message (with the target machine name, the task type, and the specific task time extracted from the *policy.clp* file), and tries to insert it in the event queue.

4) Since there can be only one unique event message (identified by the request type, the server name and the task time) in the event queue, if the effector finds the current message already in the queue then it sends an error notification to the requestor and deletes the current message. Otherwise, the effector inserts this message in the queue. The effector program then sends an acknowledgement by email to the requestor indicating that the action is pending.

6. Performance

We measured the cost of performing the security and policy management in the Automation Controller by recording system time at several points in the program flow. Figure 6 shows the points where the measurements were performed. Cold start means all classes needed for creating the secure socket are loaded to memory for the first time. The main performance bottleneck is the SSL handshake because it uses public key cryptography to distribute the secret key for encrypting data and the public key cryptography requires extensive computation. We can reduce the time associated with this computation by exploiting the SSL coprocessor chip integrated in IBM NetVisa machine. Table 1 shows the performance comparison between cold and warm start.

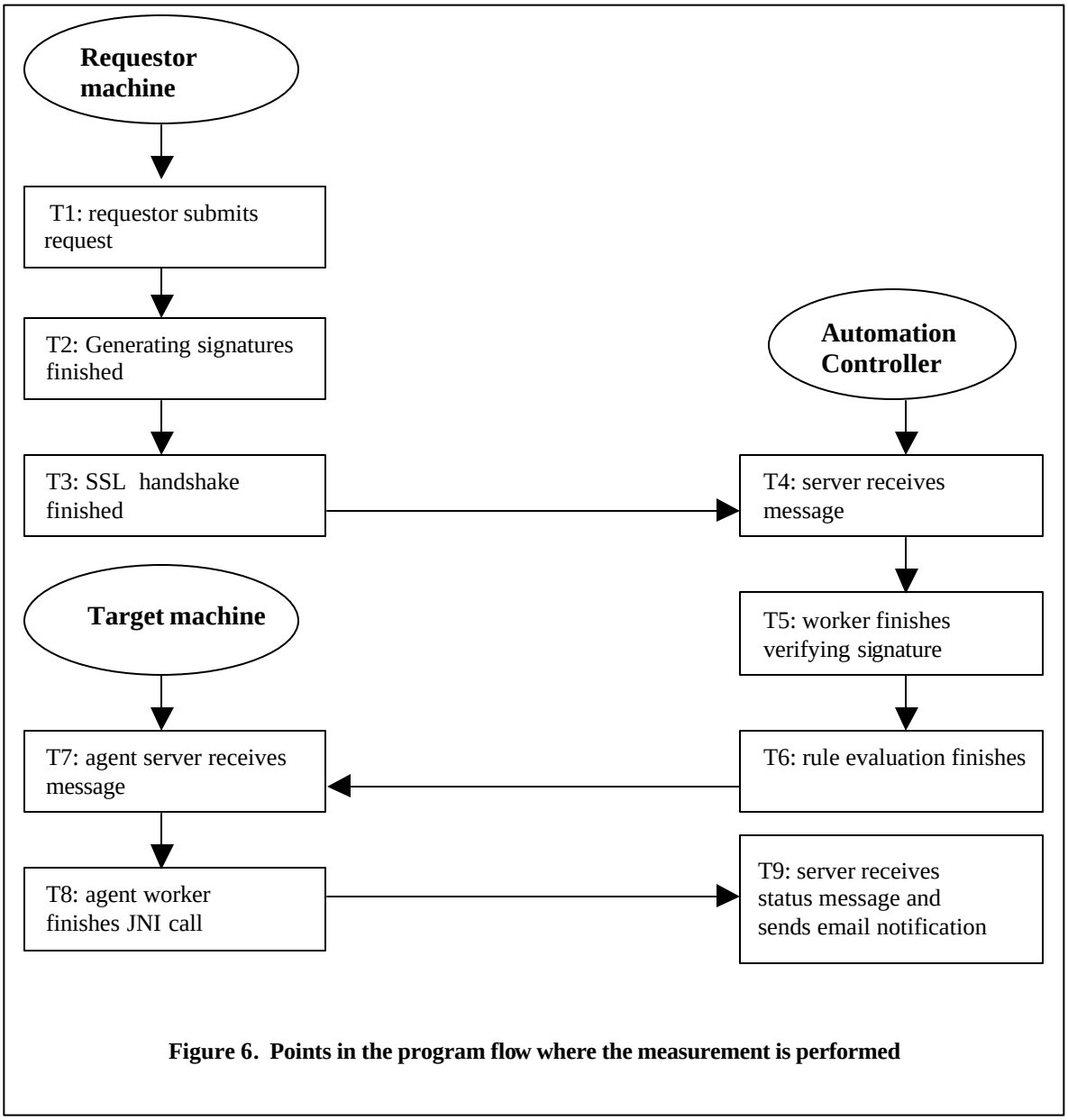

Figure 6. Points in the program flow where the measurement is performed

	Read keystore & generate signature (T2-T1)	SSL handshake (Cipher suite, authenticating server, encrypt data) (T3-T2)	Reading truststore & verify digital signature (T4 – T5)	Rule engine evaluation and trigger effector procedure (T6-T5)	JNI to finish action for the requested task (T8-T7)	Total time when the AC receives the status (T9-T1)
Cold Start	0.5	36	0.2	0.5	0.8	38 sec
Warm Start	0.5	12	0.2	0.5	0.8	14 sec

Table 1 Performance comparison between cold start and warm start (units are seconds)

7. Conclusions and future work

In this paper, we describe the security and policy management issues for the Automation Controller. Our prototype illustrates that a rule-based approach is a good way to implement automation for remote system management. We represent the knowledge and the policy as rules, and use the rule engine to provide reasoning and inferring to solve problems.

For the future, we plan to work on the situation analyzer that can process events to perform complex situation analysis. Further, we plan to extend our work in policy management by investigating ways to represent and manage general policy issues. Finally, we plan to address the issue of fault tolerance of the Automation Controller.

8. References

1) P.Anderson. *Towards a high-level machine configuration system*. Proceedings of the 8th System Administration conference (LISA), 1994.
2) M.Fisk. *Automating the administration of heterogeneous LANs*. Proceedings of the 10th System Administration conference (LISA), 1996.
3) J.P. Rouillard and R.B. Martin. *Config: a mechanism for installing and tracking system configurations*. Proceedings of the 8th System Administration conference (LISA), 1994.
4) J.Finke. *Automation of site configuration management*. Proceedings of the 11th System Administration conference (LISA), 1997.
5) M. Burgess. *A site configuration engine*. Computing system 8, 1995.
6) M. Burgess and R. Ralston. *Distributed resource administration using Cfengine*. Software: Practice and Experience 27, 1997.
7) M. Burgess, *Computer Immunology*. Proceedings of the 12th System Administration conference (LISA), 1998.
8) Alva L.Couch, Michael Gilfix. *It's elementary, Dear Watson: Applying logic programming to convergent system management processes*. Proceedings of the 13th System Administration conference (LISA), 1999.
9) Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*.
10) CommonRules: http://www.research.ibm.com/rules/
11) Joseph P. Bigus, Jennifer Bigus, *Constructing intelligent agents with Java*. John Wiley & Sons, Inc
12) Frank Dignum, Ulises Cortes. *Agent-Mediated Electronic Commerce III*.