# Alternative Java Security Policy Model *

S. Cloutier C. Gustave R. Khoury D. Nassour A. Robison F. Samson
N. Tawbi
{ Christophe.Gustave Andrew.Robison } @alcatel.com
{ Simon.Cloutier Raphael.Khoury
Dani.Nassour Frederic.Samson } @ift.ulaval.ca
Nadia.Tawbi

March 19, 2004

## Abstract

Java language and technology [4, 6, 10] were proposed with security in mind, yet there are some limitations especially when it comes to running Java applications in a distributed context. In this work we propose a new security policy together with the relevant verification mechanisms. This model is aimed at controlling the access to the system resources in a trustable and flexible way. This model extends the existing security mechanisms offered by Java in two ways: first the authentication is based on public keys rather than global names which is more flexible. Second our model supports delegation which is of major importance in a distributed context. The new model has been integrated in a Jini based platform.

## 1 Introduction

The distributed nature of todays open environments creates new requirements and challenges for deploying services across network boundaries in a ubiquitous and secure way. In this context, establishing and subsequently maintaining security properties is a challenging process. Indeed, heterogeneous components - part of the overall information system - have to share common security policies and interoperate with various security models.

Many approaches have been proposed for securing distributed architectures. However, most of them are limited to a subset of security policies for two main reasons: inherent complexity due to the number of components, protocols and data format, and diversity of such architectures. Our approach is to provide the necessary flexibility to implement various kinds of authorization policy constraints without increasing the systems complexity. In this paper, we present the result of our research work aiming at enhancing the authentication mechanism and the authorization policy to be implemented into Jini-based platforms. We present the Distributed and Secure Java Virtual Machine (DSJVM). In this framework the authentication mechanism is more flexible than the classical one and the authorization scheme is extended in order to support delegation. The extensions to the authorization mechanism are inspired by Ponder, a formal language for expressing security policies.

### 1.1 Security Needs Into Service Brokers

The advent of distributed e-business service broker infrastructures, in essence vulnerable to various network-based threats, brings out new requirements in term of implementation of the underlying services

*This work has been funded by Research & Innovation, Alcatel, 600 March Road Kanata, ON K2K 2E6

platform. Security can not be considered anymore just as an after-thought design process. Rather, it has to be implemented at the heart of the system, ensuring both scalability and extensibility from a security deployment perspective. Furthermore, network infrastructures are undoubtedly as secure as their weakest link. Thus, it is crucial that security functions are coordinated and exchange information in a way which avoids exposing sensitive assets to unauthorized and potentially malicious third-party entities. Application services accessing underlying communication brokers need firstly to be authenticated and consequently should only be able to interact with the service entities that they are authorized for. In the context of access control policy decision, at this point, we need to distinguish two different kinds of authorization entities:

- the identified subject or participant - Human, machine, or process - initiating the request,
- the object - ultimately a set of network services resource - to be accessed.

A sound access control model should clearly identify the different kind of subjects operating in the system, along with the scheme used to map them to the various controlled resources. In a distributed environment, this can be a rather complicated process requiring a lot of digging and prone to administrative errors. Generally, the PDP (Policy Decision Points) duty is to assign the specific access control rules that applies. Ultimately, the PEP (Policy Enforcement Point) launch a query-access ressouce to the appropriated PDP. The PDP applies the rules and replies subsequently by denying or allowing access to the resource. Figure 1 shows the basic interactions involved when a subject, such as an application, request access to a network resource entity. Access requests to the targeted service must be regulated through the PEP engine.

One of the key points is to maintain a coherent access control semantic throughout the whole system. Indeed, two different policies could interfere between each other, and enforcing a policy could result in an inconsistent access rule, putting in place a back door or digging an unexpected security hole in the core

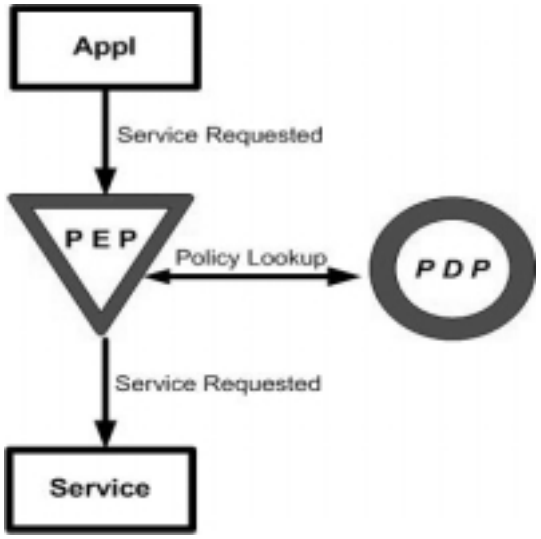


Figure 1: Access Request Enforcement

system. This brings the needs for an access control policy model capable to streamline the security administration process, decreasing the risk of administrative error while globally setting up a consistent security policy.

In the scope of our research, we carried out a new access control security policy model that meets the challenging requirements of todays complex networking environments. This model is mainly based upon the Ponder formal language, whose inherent expressiveness allowed us to define easily and consistently a wide range of security policy constraints. On the other hand, the authentication mechanism is based on public keys as identities of the participants. While focusing, in the next section, on the different security properties that are enforced in our platform, we present, in Section 3, the Jini-based platform. In Section 4, we describe a case study that illustrates how our security policy model can be used in real life applications to procure enhanced access control decisions. In Section 5, we discuss different alternatives to our solution along with potential limitations. In Section 6, we draw a brief conclusion.

# 2 Enforced Security Properties

The Java security policy model is mainly aimed at enforcing authorization policies. The extensions we bring to this model are inspired by the Ponder Specification Language [2]. This Language allows the expression of very powerful security policies. It is aimed at enhancing the expression power of security policy specification related to access control. Namely it allows the expression of delegations, exceptions and negative authorizations.

Authorization are granted or denied depending on the identity of a participant. Hence, as a first step, authentication of the participants has to be performed.

In the sequel, we list all the security properties that our DSJVM platform is able to enforce.

## 2.1 Authentication

Authentication based on the participants names could not be performed correctly unless a global name space is used. Although global name space is very convenient when the number of participants is limited, it poses severe scaling problems.

In our model, we avoided global names by performing authentication based on public keys. Each participant is associated with a public key that could be created locally. A trusted party, usually called certification authority, signs this key and delivers a certificate legitimizing the use of the key and its association to the corresponding participant. An example of such a trusted party is the famous Verisign company.

In our model, we ensure mutual authentication. Thus, a client asking for a service should authenticate itself to the server providing this service, and the server should authenticate itself to this client.

The authentication is based on the SSL (Secure Sockets Layer) protocol [12] which is a trustable and very well analyzed protocol.

## 2.2 Data integrity and confidentiality

Data integrity and confidentiality are enforced in our system in the standard way supported by Java. After the mutual authentication of the server and the client is performed, the SSL authentication protocol generates a symmetric key that is shared only by the server and the client. All the communications after the authentication step are encrypted using this symmetric key, which is a grantee for information integrity and confidentiality.

## 2.3 Authorization policies

### 2.3.1 Positive and Negative Authorization

Authorization is the security property that enables a system to restrict access to protected resources. Access is denied to some users and permitted to others based on a series of restrictions put in place by the owner or by the administrator of those resources.

There are two ways to control the access to the protected resources. The most common method is by a list of actions that a user is permitted to perform. Whenever a user attempts an action, the system checks if that action is in the list of permitted actions. If it is, the action is executed. If not, it is denied execution. This is positive authorization.

The second way is by creating a list of actions that a user is not permitted to execute. Whenever an action is attempted, the system determine if the action is listed. If so, the action is not executed. If not, it is executed. This is negative authorization.

The original Java policy provider only supports positive authorization. In our model, we support both types of authorization.

It is important to see that we follow Ponder, in our model, by giving to the negative authorization the priority over the positive authorization. If the administrator gives both a positive and a negative authorization on the same action, only the negative authorization is considered. This is done for safety reasons. Indeed, it is less dangerous to deny an action that should have been permitted than to permit an action that should have been denied.

### 2.3.2 Exceptions

Exceptions are a way to make the authorization rules more precise. For example, we may have a rule that

permits a user to access all files on a computer's hard drive. The administrator of the hard drive adds more files to the hard drive and realizes that these files are confidential and should not be read by this user so he must update the policy file.

Without using exceptions, the administrator would have to delete the general rule that gives this user access to all the files of the hard drive and add a series of new rules each giving the user access to each directory on the drive except for the one that contains the confidential files. This can be a very long policy file and it is not a very efficient way to write security policies.

Using exceptions, the administrator simply modifies the general rule that gives the user access to all the files of the hard drive by adding an exception parameter for the directory containing the confidential files. The system can now be much more powerful. If the user attempts to access the files that are not confidential, it will let this access take place. If the user attempts to access the files inside the confidential directory, it will not let this access take place. This is done without the need for long and complicated policy files.

Exceptions work on both positive and negative authorization. An exception on a positive authorization takes away the given permission and an exception given on a negative authorization takes away the negative authorization. This lets the administrator give a positive authorization elsewhere in the policy file that won't be in conflict with that negative authorization.

It is important to note that exceptions never actually give a permission. Also, they are not equivalent to negative authorization. For example, if positive permission $A$ has an exception $E$ then when the user tries to perform $E$, he or she is denied but if elsewhere in the same policy file permission $A$ appears again but this time without exception $E$ then the user is still permitted to perform this action.

Similarly, if negative permission $B$ has an exception $E$ and the user tries to perform $E$ then he is still denied. The exception $E$ does not act like a positive permission. However, if the policy file also contains a positive permission $B$ with no exceptions then the user is permitted to perform this action even though a negative permission $B$ exists.

The original Java policy provider does not support exceptions.

### 2.3.3 Constraints

The next security feature we use from Ponder is constraints. Like exceptions, constraints make the authorization security property more precise. While exceptions more efficiently perform access control on protected resources, constraints can be used by the administrator to make authorizations valid or invalid for any given period of time.

The policy file contains a list of positive and negative permissions for different users. It is normally written by the owner or the administrator of the protected resources. Over time, it often becomes necessary to add or delete some permissions. Everytime a change is needed, the administrator must open the policy file, make the necessary changes, and then make the system read the policy file again so that the new policies are taken into account.

If this process is frequent, it could lead to a waste of time and increase the possibility to introduce errors. Furthermore, sometimes the file is changed only temporarily. The administrator must make a change in the policy file knowing that he or she will have to come back to change the file to the way it was before.

To avoid this waste of time, the administrator can write a policy file and add some constraints to some of the permissions. For example, employees in a company may have positive permissions for access to all files on a hard drive. This access is permitted but only during the days of the week. If the access is attempted during the weekend, then it should be denied. The administrator may have to load a different policy file every Friday night and Monday morning for the system to perform the correct access control.

Using constraints, the administrator does not have to change the file all the time to reflect some changes in the policy authorizations. He or she can put constraints in the policy file permission entries that say that the permission is only valid if the day of the week is between Monday and Friday inclusively. It can even be refined by saying that the permissions are only valid between 8 AM and 8 PM. Constraints

can be put in place for any type of time or date including hours, minutes, seconds, day of the month, month, year, and day of the week.

Although we only used timing constraints the model would easily allow the use of many other types of constraints such as performance constraints, operating systems to be used, etc.

#### 2.3.4 Delegation

The last security property inspired by Ponder is delegation. Delegations provide users with the possibility to give to other participants some of the granted permissions they possess. Obviously, this should only be done when the administrator allows it.

Delegations only work based on permissions, not users. The user is permitted or not permitted to delegate one permission at a time. Each permission normally has a delegation parameter so the user is permitted or not permitted to delegate the entire permission. It is not possible to separate the permission to only delegate a part of it.

When writing the security policy, the administrator may want to give to some of the most trusted users the permission to give to other users some of their permissions. To do that, the administrator writes their security policy and adds a parameter to each permission regarding delegation. Using that parameter, the system, when receiving a request for delegation, can decide if it is permitted or not before actually delegating the permission.

The parameter that is added to permissions concerning delegations can be one of four options.

The first one is called negative delegation. This is the equivalent of a negative permission on delegation. When the system sees this, it denies permission to delegate the permission. Even if another permission elsewhere in the policy file give permission to delegate this permission, it is still denied because negative permissions have priority over positive permissions.

The second one is no delegation. This is when the administrator does not give a delegation parameter. The delegation is still denied if it is attempted but it is possible for the administrator to give a delegation permission in another identical permission.

The third one is called delegation. This gives to the user the permission to delegate this permission. More information on how this works is given later.

The fourth one is called delegation of delegation. This gives the user the permission to delegate this permission and gives the user permission to let the users who receive this delegation delegate it to other users as well. This is recursive delegation where users receive permission by delegations and then delegate those permissions.

When a delegation takes place, the entire permission is delegated. This includes any information about exceptions and constraints that go with that permission. If the permission has an exception then that exception is delegated with the permissions and this cannot be changed.

The delegation also has a time period for when it is valid. This is put in place by the administrator and is given in seconds. For example, a permission's delegation life time can be 240 seconds. When a user attemps to perform a delegation, he or she decides for how much time the user who receives this delegation will be permitted to use it. This can be anything from 1 second to the maximum permitted by the administrator, in this case 240 seconds. If the user attempts to delegate for a longer period of time, he or she is denied.

When performing delegation, the user is asked if he or she wants to add exceptions or constraints to that permission. As said earlier, the exceptions and constraints put in place by the administrator are delegated with the permission and there is no way to change this. It is however possible for a user to add new exceptions and new constraints. This enables a user holding the permission to put some additional restrictions on the permissions that he or she is delegating. With recursive delegation, all the exceptions and constraints put in place by the users since the beginning of the delegation process cannot be changed but it is always possible for new exceptions and new constraints to be added.

Also with recursive delegation, the time for which the delegation is valid is fixed. The first delegation can be valid for as long as the administrator has permitted it in the policy file. The second delegation (the actual recursive delegation) can be valid for the time left in this delegation and nothing more. For

example, the administrator gives a maximum time of 240 seconds for delegations. The first delegation is done for 240 seconds. After 60 seconds, the user decides to delegate his delegated permission. He can only delegate it for the time left in his delegation, that is 240 - 60 seconds = 180 seconds. This avoids constant recursive delegation between two users that would make the time limit on delegation useless.

# 3 Distributed and Secure Java Virtual Machine Platform

The platform of our distributed system is based on the Jini network technology. The major motivation underlying this choice is that Jini offers a flexible platform in which service providers and clients could connect themselves very easily. Furthermore, the Jini leasing mechanism garantees a good reliability of services. A service is connected for a laps of time after which the lease has to be renewed. This is not possible if the service is no more available.

The integration of our model into this technology consists on one hand, in inegrating a new policy verfier that controls the access to the system resources and in creating a new authentication module.

This section begins with a description of the Jini network technology. Then we describe how we modified the security policy administration of the Java language to implement the security properties described in the previous section.

## 3.1 Jini's Service Delivery Model

In this section we briefly describe the Jini technology and what it can do. We define the different components of a Jini network and how they are used together.

Introduced in January 1999 [7], the Jini technology is a network technology that enables developers to implement services that are centralized, meaning that they can be accessed by a lookup service that acts like a central server. It is often said that Jini federates because the services that are part of the community join it like they would join a federation; they work together and offer their services to all the other clients and services of the network. The machines and applications offer their resources to the other members of the federation. They can be implemented in either software or hardware. The resources of the services on the network appear to the clients as objects in the Java programming language. It is a set of services that are able to communicate with each other with little outside intervention. Jini provides a framework for these different services to communicate with each other.

Jini is most often defined using the hardware services as an example. A person could come into a room that he or she has never been before with a laptop and connect it to the network and the laptop would immediately know what services are available on the network. If there were a printer available on the network, the laptop would find it easily, installing and downloading just-in-time through the network all the necessary drivers, without any end-user intervention. This is the important part; the advantage of Jini is its simplicity. The clients and services can join and leave the network with little intervention and configuration from users.

**An Example**

A network may have a powerful machine that can compute mathematical equations faster than others so when a machine on the network needs something calculated it will ask that machine to do it. The machine is then a service on the network. It may also be a client when a user on this machine needs to use a printer on the network. The machine then becomes a client that will ask the network for use of the printer (service). A client can also be just a client. In this case, it needs the network to do things and never offers its services to the network because it has none to offer. All of the Jini services are also clients because they eventually all use another service at some point.

**What Makes Jini Different?**

The Jini system is a network of computers and computing devices. To the user, it looks like a single system. This is important for simplicity. Users

want to be able to use whatever they need with little complications. Also, Jini is designed to eliminate the differences between hardware and software. It does not matter if the service, the client is communicating with, is a piece of hardware or software. It should be able to communicate with both types of services with no problem. Jini has four main advantages:

1. There should be no outside intervention to bring services online or offline.

2. The Jini community can "heal" itself. It adapts when services are added to the network or when they are deleted from the network.

3. The users of Jini do not need to know anything about the implementation of the service to use it. The service should be loaded dynamically with no configuration from the user.

4. If a service registered on Jini's lookup engine has a failure, this service will not be able to renew its lease. Thus, the other participants to the Jini federation will sooner or later be aware that this service is no longer available.

The most interesting Jini feature is not that it can communicate with hardware and software without knowing which type it is communicating with, but it is the fact that it can do it in a dynamic way. One can add a printer to the network and all the other services will have access to it with no need for configuration. Later, one takes it away and all the services know that it is now gone and will stop trying to access it, again with no configuration.

**The Three Main Components of Jini**

There are three main components in Jini. They are the client, the service, and the lookup service. We give a summary of those three components here along with a description of how they communicate together.

The lookup service is the central component of a Jini network. Each client and each service participating on the network communicate with the lookup service at some point. It acts like a central server by redirecting clients towards the services that they need.

When starting, a service creates a proxy and registers it on the lookup service. The lookup service stores it and waits for client requests. It also negociates with the service on a lease. This is an amount of time that the service can guarantee its presence on the network. Upon lease expiration, the service is required to renew its lease contract or else it is terminated and the lookup service considers that the service is no longer available. This is done to lower the possibility that the lookup service will direct clients to dead services. If a service does not renew the lease, it may have been disconnected so directing clients to it would slow it down.

The lookup service eventually receives from clients the specific name or the type of service that they need. The lookup service then sends to the client the proxy of a service of the type that they requested. The client can then use it to communicate with the service.

A server aiming to offer a service creates a proxy and sends it to the lookup service. The proxy is a component of the service used by the lookup service to identify the location and type of service that it is offering. It is also used by the client to locate and then communicate with the desired service.

When the client has received this proxy from the lookup service, it can use it to directly contact the Jini-based service and send its requests for the execution of some actions.

## 3.2 Extending the Security Policy Administration

The first step in the creation of a network that uses our system is to create the security policy file. This file is a text file that helps the system to decide whether to grant an access or not.

When a service provider (server) is started, this server can receive requests by clients. When this happens, the client is required to authenticate itself to the server and the server is required to authenticate back to the client.

After the authentication has been performed, sensitive operations eventually happen. At that time,

the client sends its request to the server and the server verifies its security policy. If the security policy grants this user permission to perform that operation, the operation is performed and the result is returned. If the policy denies this user permission to perform that operation then the operation is not performed.

Our implemented JAAS (Java Authentication and Authorization Service) [11] login module is responsible for authenticating users. The client is identified as a public key. Using the SSL protocol, the client authenticates itself on the server and the server authenticates itself on the client. Only after that happens will further communication occur. Also, SSL authentication results in a symmetric encryption key known only by the two participants in the authentication protocol. That key is used to encrypt the data on the network to ensure confidentiality and data integrity.

**The Login and Authentication Modules**

We designed and implemented a JAAS Login Module that leverage the SSL protocol to authenticate users. This SSL-based login module also generates the secure sockets that are used in the communication. Those secure sockets are created using the symmetric key generated during the authentication process and known only by those two communicating parties. All the information sent between the client and the server transit through those secure sockets and is therefore encrypted and decrypted using that symmetric key.

**Generating Public and Private Keys in Java**

Before being able to use the network, a public and a private key must be generated for each participant using Keytool. Keytool is a Java program that generates keys and requests for public-key certificates. Those requests should be sent to a certification authority that will delivers public-key certificates. The keys together with their associated certifictaes are stored in a keystore repository.

**Authorization Policies Verification**

Our system uses policy files written in the XML language. A policy input tool helps the administrator to create the policy file. This tool receives the policy property in a user-friendly way. It then translates it into an XML syntax and stores it in the policy file. Those files contains permissions specifying what actions the users of the system are allowed to perform. When a sensitive operation is attempted, the system verifies those permissions and decides whether the user is allowed or not to trigger that operation.

The verification process of the permissions is the core part of the system where the security policy file is analyzed to determine if users are granted access to perform certain actions or not.

The whole architecture of the Java security management does not change except that we removed the standard Java policy provider and we replaced it with our own policy provider called XMLPolicy. In this policy provider, the verification process is more complex than the standard one because one has to check for the delegations, the constraints, the exceptions, the negative authorizations as well as the positive ones in order to grant an access or not.

Notice that, only the owner of a resource is able to grant or to deny access to this resource. The verfication related to the access request is performed on the owner site. This is how the consistency of all the policy files on the distributed system is garanteed. On the other hand, consistency of a local policy file is garanteed because negative authorizations have priorities over positive ones if they are related to the same resource.

# 4 Case Study: Secure Calculator Application

In this section, we look at an application that we developed to demonstrate the capabilities of our system. It is a simple distributed calculator application programmed in Java where the server is the calculator and the client requests from the server the answers to simple arithmetic operations.

The calculator application uses the Jini networking technology. The server creates a proxy and sends it

to a lookup service to register itself. The client then requests this proxy from the lookup service. Using the proxy, the client communicates with the service.

Our system is different from the original Jini version and the original Java policy provider and the application works under those changes. The clients and service are identified by public keys that they have generated by themselves. They have requested a signed public key certificate from a trusted third party and use it to identify each other. The public key of this trusted third party has been distributed safely on the network. Authentication of the client on the service and the service on the client is compulsory. Finally, whenever a client requests an arithmetic operation or a delegation, the service verifies that it has the required permissions to do it before performing the operation and sending the result back to the client.

The permissions work by giving to clients permissions to perform addition, substraction, multiplication, and division of either integers, long integers, floats, and double floats. For example, client Bob is permitted to perform the addition of integers. If he tries to add floats or if he tries to multiply integers, he is denied permission. Also, the permissions may or may not be delegated. The permissions are in a security policy file on the server.

The first step to use this application is to start a Jini lookup service. We use *reggie*, a simple lookup service that comes in the downloaded Jini packages. This lookup service also requires an HTTP server, which we start at the same time. The lookup service can be on a different computer or on the same one as the server.

Prior starting the server, we need an HTTP server that we start on a different port than the lookup service. It is not necessary, but it is important in case we decide to start the server and lookup service on the same computer.

Before we start the server, we must also tell to the Java virtual machine about the new Java policy provider that we want to use for this application. To do that, we change the *policy.provider* parameter in the java.security file to *XMLPolicy*, the name of our policy provider.

Then, we launch the server with the regular command but with additional specific parameters. In particular, we needed to specify that our customized policy provider must be loaded as part of the core java runtime environment (JRE) packages. Thus, the policy file needs to be loaded in the XBootclasspath of the virtual machine. Similarly, our JAAS login module, the SSLLoginModule classes needed to perform SSL authentication are also needed in the Xbootclasspath.

We then load the SSL authentication configuration file. This tells the system which classes to use for authentication as well as where to find the keystores. It also specifies that we require bidirectional authentication in our system where the client authenticates itself to the server and the server authenticates itself to the client. Next, we specify the name of the security policy file to load on the server. When the Java virtual machine starts, this file is read and stored on the server to be used to verify permissions when calculator and delegation operations are attempted by clients. In the command line, we also specify a codebase. This is where the client looks for files on the server whenever it downloads code. Finally, the command line contains the name of the application class to load.

Figure 2 shows the first step in using the calculator application. The user is invited to choose what type of numbers he or she wants to perform arithmetic operations on. The user may also choose to do a delegation of permissions. There are no permissions checks performed at this stage.

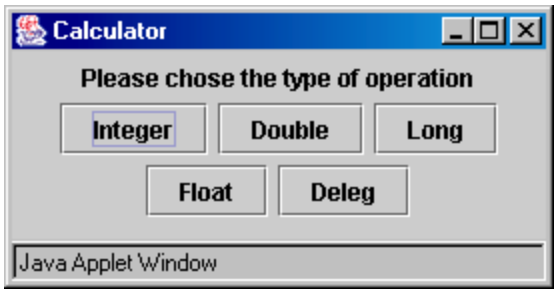


Figure 2: Secure Calculator Application Menu 1

When the user chooses a type of number, the window showed in Figure 3 appears. By using the numbers and types of operations on the calculator, the user can ask the server for the answer to simple arithmetic operations.

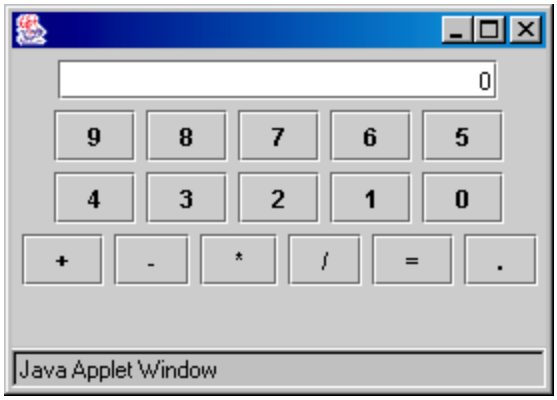


Figure 3: The Calculator

If the user has the permission to perform the requested arithmetic operations on the types of numbers chosen, the answer appears. Otherwise, the first number chosen in the arithmetic operation is displayed. That indicates that the user has tried to perform an unauthorized operation

When the user tries to do an arithmetic operation, the system checks the policy file to see if this user is allowed to perform this arithmetic operation using the desired types of numbers that he or she has chosen in the calculator.

Figure 2 shows that the user may also perform delegation. By pressing the *Deleg* button, the user is given a list of permissions that he or she is allowed to delegate. When the server receives from the client the deleg message, it looks in the policies that are valid at this point on the server for any positive permission that this user may be allowed to delegate. It creates a list of those permissions and sends them back to the client. The client displays them and the user chooses one of them to delegate. The permissions are displayed by showing the information about the permissions such as its permission class, the list of exceptions and exceptions on this permission, and its delegation information. Only permissions that can be delegated by this user are shown. The user can perform many delegations, but only one at a time.

After choosing a permission to delegate, the user is asked for several questions about how he or she wants to delegate the chosen permission. The first question is the grantee's public key. The user must give to the server the public key of whoever will receive this delegation. Then, the user must decide if the delegation can be delegated and if so, if it is a recursive one. There are three choices, *no delegation* (the delegation cannot be delegated), *delegation* (the delegation can be delegated but not recursively), and *recursive delegation* (the delegation can be delegated and it can be recursively delegated).

Next, the user gives a time limit for this delegation. This is the amount of time in seconds that the delegation is valid for the user who receives this delegation. It cannot be longer than what the administrator has specified in the policy file and if it is a recursive delegation it cannot be longer than the time left on the first delegation of this permission. The system verifies this and if the time is not valid, the delegation is unsuccessful.

The user can add constraints and exceptions to the permission before delegating it. All the constraints and exceptions specified by the administrator are delegated with the permission and this cannot be changed. However, the user can further restrict the permission for the user who receives this delegation by adding new constraints and delegations. If the user chooses to do that, he or she adds those restrictions using the same XML format used by the administrator to write the policy file. Those new constraints and exceptions are added to the ones put in place by the administrator and are delegated with this delegated permission during recursive delegation.

There is one last question that the user can answer when performing a delegation. The user can add new delegation constraints. The permissions have constraints to specify when they should be considered or not considered. The delegation permissions also have that. This means that the administrator may decide when a user should be permitted to delegate this permission. When performing a recursive delegation, a

user may restrict when the next user can delegate the permission by adding new constraints here. As with the permission's constraints and exceptions, the delegation constraints are always delegated with the permission and cannot be changed.

If the delegation is sucessful, Figure 4 appears. The receiver of this delegation now has the permission to perform the delegated operation for the time chosen by the user.

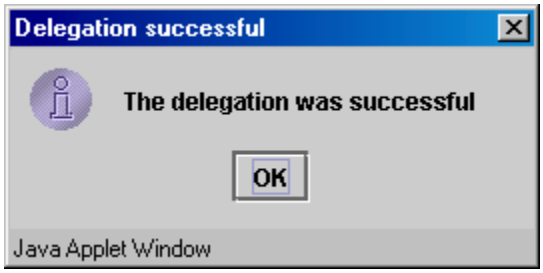


Figure 4: Successful Delegation

Policies are easily created and modified using a Policy input tool that helps the admninistrator to enter its policies correctly. Figure 5 shows the user interface of this tool.

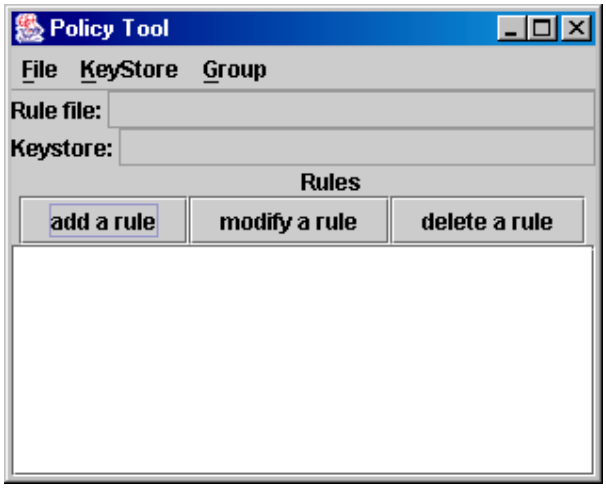


Figure 5: Policy Input Tool

# 5 Discussion

This section discusses alternatives to Jini that we could have used for our distributed system. We studied distributed systems such as CORBA (Common Object Request Broker Architecture) and Microsoft .NET. This section also looks into XACML (eXtensible Access Control Markup Language), an access control mechanism based on XML[1].

### Jini vs. CORBA

CORBA [5, 9] was a possible choice for a distributed system because it is possible to use it in Java. CORBA, developed by the Object Mangagement Group (OMG), is also a very mature distributed system that offers a wide variety of possibilities. It is independent of the programming language used by the developers. This means that a client working on a system programmed in C can communicate with a server programmed in Java.

CORBA uses a standard protocol called IIOP[2] to communicate. IIOP is the protocol that makes it possible for CORBA clients and servers to communicate using an ORB[3] regardless of the programming languages used to build them.

The reasons underlying the choice of Jini against CORBA are twofold. First, CORBA support code mobility but it is more efficient in Jini using Java. Second, Jini has built-in components like leasing that makes it easier to build stable networks.

### Jini vs. Microsoft .NET

We also looked into Microsoft .NET [3], which is a set of Microsoft Internet software technologies used to connect people and devices together [1]. Using a Common Language Runtime (CLR), it lets different applications developed in different programming languages communicate together. A programming lan-

[1] eXtensible Markup Language: A standard for writing and exchanging structured documents especially on a network.

[2] Internet Inter-ORB Protocol.

[3] Object Request Broker: A CORBA object that takes care of sending the request from the client to the server and routing the response back to the client.

guage called C# was developed specifically for use with Microsoft .NET.

In the centre of this technology are web services. Web services are applications designed to efficiently answer requests coming from the Internet.

Microsoft .NET uses standards such as XML and SOAP (Simple Object Access Protocol) to communicate information on the Internet from web services to clients. Using standard protocols makes it simpler for anyone building applications to use Microsoft .NET.

Still, we did not use Microsoft .NET because, while it supports multiple languages, it does not support Java and we were looking for a distributed system that uses the Java programming languages.

## 5.1 Ponder vs. XACML

XACML [8] is an XML-based standard for defining security policies to perfrom access control. It has become necessary to have a standard for expressing security policies because they are often stored in many different places and then read by many different systems. This section is an introduction to XACML.

A subject wants to perform a certain action that requires using a protected resource. On a network, the subject would make a request on a server that manages the resource. The server is the PEP. The PEP creates a request using various information such as the identity of the subject, the type of action that he or she wants to perform, and exactly which protected resource it needs to perform the action.

The PEP sends this request to the PDP. The PDP receives this request and compares it to the policies that are in effect at this point. It looks for policies that control the protected resources present in the request. Using those policies, it can decide if the request should be granted or not. There are actually four possible answers that can come from the PDP. They are *permit*, *deny*, *indeterminate*, and *not-applicable*. Using that response, the PEP then either grants or denies access. The PEP and PDP are separate entities, they can be distributed on a network or together on the same server.

XACML provides a policy language for writing the security policies. It provides a way to efficiently find the security policies that apply to a given resource and then using them to evaluate the request and efficiently decide whether access should be granted or not.

XACML could have been used. The only advantage it offers against our solution is that it is easier to communicate security policies over the network. In our platform, this is not mandatory because every service owner administrates its security policy locally.

## 5.2 Design Decisions

Most of our design decisions were driven by security aspects. In this case, we concentrated our efforts on security features into distributed systems but some of our results can be used in non-distributed systems as well. To attain our goal of security, we made several design decisions that we explain in this section.

For our research, we decided to use the Java programming language. Besides the fact that Java supports code mobility, Java was created with security and safety in mind. Java is strongly typed, does not support explicit pointer manipulation. Moreover, Java Virtual Machine offers a verifier and a security management engine. Furthermore, Java security mechanisms are offered in a very flexible way and could be easily extended.

Our next choice was on which distributed system we could make more secure. Since we already decided to use Java, we needed a distributed system that could use it. After studying several distributed systems, we decided to use Jini. Jini is a technology that enables the creation of distributed systems written in Java in a flexible and dynamic way. It therefore inherits all the security features found in the language. It also offers code mobility via the RMI mechanism offered by Java.

The Ponder Specification Language enables administrators to express very powerful security policies. We used Ponder to extend the authorization mechanism offered by Java.

Using a global name space as a basis for authenticating participants is very convenient when the number of participants is limited. Basing the authentication on public-keys enables us to avoid the use of global name space and to enhance the flexibility of our platform. Hence, clients and servers are free to

choose their names on a network even if the name they choose is already used elsewhere on the network, public key certificates ensuring the association between a public-key and its corresponding participant.

To perform authentication, we implemented a new JAAS login module that uses the secure sockets layer. SSL has been improved over the years. The last version SSL 3.0 could be considered as a trustable protocol [12].

### 5.3 Platform Limitations

Our security model could be easily extended with other authorization policies such as obligations. Obligations are a set of actions that a subject must perform under specific circumstances. On the other hand, the constraints we used are restricted to date and time. The policy provider, which is responsible of granting and denying accesses to resources, could be easily modified to support other types of constraints, such as the operating system on which the application is running, the resolution of the monitor, or the speed of the network connection.

Lastly, we did not test the scalability of our system. This refers to the possibility of a system running as efficiently with a big number of users as it does with a small number of users. We did test our system with a small number of users using the calculator application. We think that Jini-based platform should be able to scale very easily. The authentication based on public-keys is another feature that would improve the scalability.

## 6 Conclusion

As of today, many distributed systems co-exist, each one featuring its own strengths and weaknesses. The inherent nature of distributed platforms brings out potential communication failures, mainly due to both service brokers interoperability problems (incorrect message formats) and specific networking issues. Our experiment has shown that using Jini as the underlying broker platform is a good choice to foster the development and ease the deployment of distributed services. Indeed, in unstable environments, often service locations change and various consumer actors needs to access not a-priori known services. In this context, Jini distribution model appears as the most suitable solution to build up a ubiquitous and pervasive services platform.

Through our prototyped DSJVM we were able to demonstrate the powerful expressiveness capabilities provided by the Ponder language. This formal approach enables defining a broad range of security policy requirements. In this case, we focused more specifically on authentication and authorization features, but this could be extended as well to other security-related properties such as security audit. The DSJVM leverage Jini as the underlying service-oriented communication broker. The overall security architecture of Jini has been strengthened, and authorization policy management has been dramatically simplified. The flexibility is enhanced and strengthened by the fact that authentication is performed on public-key bases avoiding the use of a global name space.

## References


[1] James Conard, Patrick Dengler, Brian Francis, Jay Glynn, Burton Harvey, Billy Hollis, Rama Ramachandran, John Schenken, Scott Short, and Chris Ullman. *Introducing .NET.* wrox, 2000.

[2] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder specification language. *Lecture Notes in Computer Science*, 1995:18–39, 2001.

[3] David S. Platt. *Introducing Microsoft .NET, Third Edition.* Microsoft Press, 2003.

[4] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification Second Edition.* Addison-Wesley, Boston, Mass., 2000.

[5] Object Management Group. Corba specification v1.2, December 1998.

[6] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Second edition, 1999.

[7] Scott Oaks and Henry Wong. *Jini in a Nutshell*. O'Reilly & Associates, Inc., 2000.

[8] OASIS. extensible access control markup language (xacml) version 1.1, July 2003. http://www.oasis-open.org.

[9] Jeremy Rosenberger. *Sams' Teach Yourself CORBA in 14 Days*. sams, 1998.

[10] R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification and Validation*. Springer-Verlag, 2001.

[11] Inc. Sun Microsystems. Java$^{TM}$ authentication and authorization service v1.0 specification, 1999.

[12] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce, USENIX Press*, pages 29–40, November 1996.